UNIVERSIDADE FEDERAL DE OURO PRETO
INSTITUTO DE CIÊNCIAS EXATAS E BIOLÓGICAS
DEPARTAMENTO DE COMPUTAÇÃO

CARLOS EDUARDO GONZAGA ROMANIELLO DE SOUZA

ABORDAGENS MULTI-OBJETIVO PARA O PROBLEMA DE ALOCAÇÃO DE SALAS

Ouro Preto, MG
2024

UNIVERSIDADE FEDERAL DE OURO PRETO
INSTITUTO DE CIÊNCIAS EXATAS E BIOLÓGICAS
DEPARTAMENTO DE COMPUTAÇÃO

CARLOS EDUARDO GONZAGA ROMANIELLO DE SOUZA

ABORDAGENS MULTI-OBJETIVO PARA O PROBLEMA DE ALOCAÇÃO DE SALAS

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal de Ouro Preto como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Pedro Henrique Lopes Silva

Ouro Preto, MG
2024

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
INSTITUTO DE CIENCIAS EXATAS E BIOLOGICAS
DEPARTAMENTO DE COMPUTACAO

FOLHA DE APROVAÇÃO

Carlos Eduardo Gonzaga Romaniello de Souza

Abordagens multi-objetivo para o problema de alocação de salas

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal
de Ouro Preto como requisito parcial para obtenção do título de Bacharel em Ciência da Computação

Aprovada em 9 de Fevereiro de 2024.

Membros da banca:

Pedro Henrique Lopes Silva (Orientador) - Doutor - Universidade Federal de Ouro Preto
Pablo Luiz Araujo Munhoz (Examinador) - Doutor - Universidade Federal de Ouro Preto
Bárbara Letícia Rodrigues Milagres (Examinadora) - Bacharel - PPGCC UFOP

Pedro Henrique Lopes Silva, Orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 9/02/2024.

Documento assinado eletronicamente por **Pedro Henrique Lopes Silva**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 09/02/2024, às 10:01, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0665743** e o código CRC **D21D6F9D**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35402-163
Telefone: 3135591692 - www.ufop.br

Resumo

O presente trabalho tem como objetivo avaliar algoritmos que têm como finalidade resolver o Problema de Alocação de Salas (PAS) em cursos universitários. Este problema consiste em alocar turmas de disciplinas com horários predefinidos em salas de aula distribuídas em vários prédios, considerando as determinações da universidade. Dois algoritmos foram avaliados neste trabalho: (i) *Non-Dominated Sorting Genetic Algorithm* (NSGA-II) e (ii) *Multi-Objective Late Acceptance* (MOLA). A exploração do espaço de soluções no algoritmo MOLA e a mutação no algoritmo NSGA-II são feitas por meio de cinco estruturas de vizinhança. Os algoritmos foram avaliados usando dados reais e baseados nos dados do Instituto de Ciências Exatas e Biológicas (ICEB) da Universidade Federal de Ouro Preto (UFOP). Foi considerado uma versão mono-objetivo do problema, para avaliação dos algoritmos.

Palavras-chave: Otimização. Heurísticas. Meta-heurísticas. Problema de alocação de salas. PAS. Multi-objetivo. Algoritmos genéticos. Algoritmos Evolutivos.

Abstract

The present work aims to evaluate algorithms designed to solve the Classroom Allocation Problem (CAP) in university courses. This problem involves assigning class groups with predefined schedules to classrooms distributed across various buildings, taking into account the university's regulations. Two algorithms were assessed in this study: (i) *Non-Dominated Sorting Genetic Algorithm* (NSGA-II) and (ii) *Multi-Objective Late Acceptance* (MOLA). The solution space exploration in the MOLA algorithm and the mutation in the NSGA-II algorithm are performed through five neighborhood structures. The algorithms were evaluated using real data based on the information from the Institute of Exact and Biological Sciences (ICEB) at the Federal University of Ouro Preto (UFOP). A mono-objective version of the problem was considered for algorithm evaluation.

Keywords: Optimization. Heuristics. Metaheuristics. Classroom Allocation Problem. CAP. Multi-objective. Genetic Algorithms. Evolutionary Algorithms.

Lista de Ilustrações

Lista de Tabelas

Lista de Algoritmos

Lista de Abreviaturas e Siglas

DECOM	Departamento de Computação
ICEB	Instituto de Ciências Exatas e Biológicas
JSON	*JavaScript Object Notation*
LAHC	*Late Acceptance Hill-Climbing*
MIP	*Mixed-Integer Linear programs*
MOLA	*Multi-Objective Late Acceptance*
MOSA	*Multiobjective Simulated Annealing*
NSGA-II	*Non-dominated Sorting Genetic Algorithm II*
PAS	Problema de Alocação de Salas
PCA	Pavilhão Central de Aulas
RAM	*Random-access memory*
SPEA	*Strength Pareto Evolutionary Algorithm*
UFLA	*Universidade Federal de Lavras*
UFOP	Universidade Federal de Ouro Preto
UPF	*Universidade de Passo Fundo*
VNS	*Variable Neighborhood Search*

Lista de Símbolos

$\in$	Pertence
$\notin$	Não pertence
$\sum$	Somatório
$\cup$	Conjunto união
$\forall$	Para todo
$\wedge$	Condicional 'E'
$\emptyset$	Conjunto vazio
η	Letra grega minúscula eta
$\prec$	Símbolo de dominância
N	Conjunto de itens do problema da mochila
W	Capacidade máxima da mochila
v_i	Valor do item i, $i \in N$
w_i	Peso do item i, $i \in N$
x_i	Vetor binário de variáveis para o problema da mochila
E	Conjunto de encontros do Problema de Alocação de Salas (PAS)
H	Conjunto de horários do Problema de Alocação de Salas (PAS)
H^+	Conjunto de horários de cada encontro do Problema de Alocação de Salas (PAS)
S	Conjunto de salas do Problema de Alocação de Salas (PAS)
S_v	Conjunto da sala virtual do Problema de Alocação de Salas (PAS)
S^+	$S \cup S_v$
D_i	Demanda do encontro i, $i \in E$
C_j	Capacidade da sala j, $j \in S$
X_{esh}	Matriz de decisão de $\|E\| \times \|S^+\| \times \|H\|$

G	Quantidade de gerações que o NSGA-II irá fazer
P	População inicial do NSGA-II
η	Tamanho da população inicial do NSGA-II
Np	Conjunto de soluções retornadas pelo algoritmo de crossover
Fr	Conjunto de soluções ordenadas de acordo com a questão de dominância
Ng	Conjunto de soluções selecionadas de acordo com a ordenação realizada
$p1$	Uma solução pai selecionada aleatoriamente
$p2$	Uma solução pai selecionada aleatoriamente
$f1$	Uma solução filho que herdará das soluções pais
$f2$	Uma solução filho que herdará das soluções pais
e	Um encontro pertencente ao conjunto E
$f1_e$	Encontro selecionado que pertence as soluções filho.
$f2_e$	Encontro selecionado que pertence as soluções filho.
$p1_e$	Encontro selecionado que pertence as soluções pai.
$p2_e$	Encontro selecionado que pertence as soluções pai.
L	Tamanho da lista que será usada como parâmetro para avaliar as soluções geradas durante o algoritmo MOLA
S_0	Solução inicial
P	Lista de soluções não dominadas
S	Na seção 4.3 representa a solução atual
i	Iterador no algoritmo MOLA
Q	Lista de tamanho L com as soluções que serão utilizadas como parâmetro
Q_i	Solução da lista Q na posição i
S'	Solução gerada pelo algoritmo
$N()$	Função que gera uma nova solução usando os as estruturas de vizinhança
Idl	Quantidade de vagas ociosas

Dea	Quantidade de alunos desalocados
Sta	Quantidade de alunos alocados porém sem vaga
S	Para o algoritmo guloso, S representa uma solução sem encontros alocados
p	probabilidade utilizada no algoritmo guloso
MC	Vetor de cópia dos encontros
CC	Vetor de cópia das salas
e	Para o algoritmo guloso, um encontro da solução
H_e	Conjunto de horários de um encontro específico
Sa	Conjunto de salas disponíveis
$Sa^{'}$	Conjunto de salas disponíveis, incluindo uma sala virtual para os desalocados
C_{es}	Matriz bidimensional com o custo de alocação de cada encontro e em cada sala s
o_{es}	Capacidade ociosa do encontro e na sala s
d_e	Demanda não alocada do encontro e
o_{es}	Demanda alocada do encontro e que ultrapassa da capacidade da sala s
G_e	Conjunto dos horários do encontro e agrupados dois a dois
m	Modelo do problema para o método do ϵ-restrito
S	Soluções da fronteira de Pareto para o método do ϵ-restrito

Sumário

1 Introdução

Semestralmente, inúmeras instituições de ensino superior precisam realizar a alocação de turmas de disciplinas em salas de aula obedecendo a uma série de regras e restrições, como por exemplo, a capacidade de uma sala suportar a demanda de uma turma. Essa tarefa, devido a sua complexidade, é um desafio para as instituições, pois o elevado número de combinações possíveis para essas alocações inviabiliza uma solução manual, além disso, normalmente, uma solução manual não consegue contemplar todas as restrições impostas o que pode gerar insatisfação por parte dos professores e alunos.

O problema de alocação de turmas possui várias etapas, desde a montagem dos horários escolares até a alocação final das salas. Para poder montar a sua grade, cada instituição deve se atentar com conflito de horários e turmas, o que em si já é um problema muito complexo. Além disso, também é necessário evitar o conflito de horários entre professores para poder distribuir as aulas que eles ministram da melhor forma possível. Outra etapa para elaboração do problema é decidir se os alunos ou professores deverão mudar de sala durante o dia e essa decisão depende das preferências e características de cada instituição. Sendo assim, com tudo isso já decidido, a distribuição de aulas previamente definidas com horários estabelecidos, atentando-se a diversas particularidades relacionadas ao espaço físico, possibilidade de acesso, infraestrutura e recursos necessários são fatores que caracterizam o Problema de Alocação de Salas (PAS) (SCHAERF, 1999).

Na área da otimização existem algumas classificações para os problemas tratados de acordo com o número de objetivos que se busca otimizar. Caso haja apenas um objetivo para ser maximizado ou minimizado e que proporcione uma única medida de qualidade da solução o problema é classificado como mono-objetivo. Esse tipo de abordagem ocorre quando o problema pode ser reduzido para apenas uma métrica de otimização. Em contrapartida, caso o problema tente otimizar dois ou mais objetivos simultaneamente que sejam (normalmente) conflitantes entre si, ou seja, quando a otimização de um implica na piora do outro, ele é classificado como multi-objetivo. A resolução de problemas multi-objetivo é desafiadora pois não existe uma única solução como no caso mono-objetivo e sim um conjunto de soluções.

O PAS é um problema da classe NP-difícil de acordo com Even, Itai e Shamir (1975) e Carter e Tovey (1992), por isso ele tem sido normalmente resolvido por meio de técnicas heurísticas, como as metaheurísticas. Sendo assim, vários trabalhos foram apresentados para a solução do PAS por meio dessas técnicas, como o LAHC (*Late Acceptance Hill-Climbing*) (BURKE; BYKOV, 2017), Busca Tabu (SUBRAMANIAN et al., 2011) e *Simulated Annealing* (BEYROUTHY et al., 2006). Esses trabalhos utilizam de uma função objetivo para determinar a qualidade das soluções encontradas por eles e por isso esses algoritmos são classificados como

mono-objetivo.

Mesmo que o valor da solução fornecida pelos algoritmos citados anteriormente seja usado para avaliar a qualidade das soluções geradas, muitas das vezes não reflete todas as necessidades da instituição e torna difícil a avaliação individual de cada restrição. Com isso em vista, algoritmos multi-objetivos também foram utilizados na literatura para resolver esse problema como por exemplo NSGA-II (DEB et al., 2002), *Strength Pareto Evolutionary Algorithm* (SPEA) (ZITZLER; THIELE, 1998) e *Multiobjective Simulated Annealing* (MOSA) (ULUNGU et al., 1999). Esses trabalhos tem como objetivo desenvolver métodos que consigam otimizar simultaneamente n objetivos. Sendo assim, várias soluções são geradas com valores variados para cada restrição o que facilita na hora da escolha da solução pela instituição. Nenhuma solução é melhor em todos os objetivos do que outra solução. Por tentarem otimizar dois ou mais valores, esses algoritmos são chamados de multi-objetivo.

Nesta monografia foram desenvolvidos dois algoritmos multi-objetivos principais para serem comparados, são eles: (i) *Non-Dominated Sorting Genetic Algorithm* (NSGA-II) e (ii) *Multi-Objective Late Acceptance* (MOLA). O primeiro é uma adaptação de Deb et al. (2002) para o problema encontrado no Instituto de Ciências Exatas e Biológicas (ICEB) da Universidade Federal de Ouro Preto (UFOP), no primeiro semestre de 2020 (20.1). O segundo algoritmo, que também é uma adaptação, baseia-se no trabalho de Vancroonenburg e Wauters (2013) e foi adaptado para utilizar estruturas de vizinhança para exploração do espaço de solução. Além disso, os resultados de NSGA-II e MOLA são comparados com a solução encontrada pela estratégia ϵ-restrito. Utilizou-se estruturas de vizinhanças para realizar operações e mutações durante sua execução.

1.1 Justificativa

A necessidade de encontrar um método que consiga gerar soluções viáveis para o PAS até hoje é um desafio que vem sendo explorado. Cada instituição tem suas particularidades que modificam o problema, mas mantém o conceito geral. A UFOP se enquadra nesse quesito, e por isso é importante procurar um caminho para melhorar e ajudar a universidade a resolver esse impedimento visto que a sua alocação de salas ainda precisa de intervenção manual para ser concluída.

1.2 Objetivos

O objetivo geral desta monografia é a aplicação de métodos de otimização para abordar Problemas de Alocação de Salas, em particular um problema enfrentado pelo ICEB na UFOP e analisar seus resultados. Para atingir este objetivo, os seguintes objetivos específicos foram elencados:

1. Identificar objetivos e restrições ligadas à alocação de salas do ICEB.
2. Preparar um modelo de entrada de dados.
3. Propor estruturas de vizinhanças para o problema.
4. Aplicar os algoritmos multi-objetivos NSGA-II e MOLA para resolver o problema.
5. Comparar os resultados obtidos contra a estratégia do ϵ-restrito.
6. Avaliar os algoritmos desenvolvido(s).

1.3 Organização do Trabalho

O restante desta monografia está organizada da seguinte forma:

Capítulo 2: Referencial teórico e trabalhos relacionados.

Capítulo 3: Definição do problema.

Capítulo 4: Metodologia proposta.

Capítulo 5: Resultados e discussões.

Capítulo 6: Conclusão e trabalhos futuros.

2 Revisão Bibliográfica

Ao longo dos anos vários pesquisadores propuseram e estudaram diversos algoritmos e formas de se resolver o problema de alocação de salas, alguns de forma mais genérica como, por exemplo, (EVEN; ITAI; SHAMIR, 1975) que faz a análise dos algoritmos utilizados na época e outros de forma específica buscando tratar as individualidades de suas realidades como, por exemplo, (AL-YAKOOB; SHERALI, 2006) que utilizou políticas e regras da universidade de Kuwait e (KRIPKA; KRIPKA, 2012) que focou sua pesquisa para a realidade da Universidade de Passo Fundo (UPF).

Existem também trabalhos focados no problema do ICEB-UFOP como, por exemplo, (NASCIMENTO et al., 2005) que utilizou três instâncias para seu trabalho: uma fictícia, uma da universidade de Lavras (UFLA) e outra do Instituto de Ciências Exatas e Biológicas (ICEB) da Universidade Federal de Ouro Preto (UFOP). Nesse trabalho ele resolve o PAS de forma mono-objetiva por meio do algoritmo *Simulated Annealing* (AARTS; KORST; MICHIELS, 2005). Outro trabalho relacionado ao PAS do ICEB-UFOP é o (SOUZA et al., 2002) que utiliza o *Variable Neighborhood Search* (VNS) (MLADENOVIĆ; HANSEN, 1997) e neste trabalho o problema também é resolvido de forma mono-objetivo utilizando uma função objetivo para mensurar a qualidade da solução.

Por ser um problema NP-difícil de acordo com Even, Itai e Shamir (1975), conforme esse problema escala vai ficando cada vez mais difícil de encontrar a melhor solução para o problema. Além disso é possível que vários critérios sejam avaliados pela universidade ou instituição de ensino que enfrenta esse problema. Sendo assim uma abordagem multi-objetivo para o PAS é capaz de gerar várias soluções para o mesmo problema com valores variados para cada um dos objetivos o que fornece mais possibilidades para a instituição de ensino.

Neste trabalho dois algoritmos multi-objetivos foram adaptados para atender a realidade da UFOP, são eles: *Non-dominated Sorting Genetic Algorithm* (NSGA-II) Seção 2.1.3 e o *Multi-Objective Late Acceptance* (MOLA) Seção 2.1.4.

2.1 Fundamentação Teórica

Nesta seção serão explicados conceitos necessários para o entendimento do trabalho utilizando diversas referências relevantes da literatura.

2.1.1 Otimização mono-objetivo

Um problema mono-objetivo é um tipo de problema de otimização que busca encontrar o melhor valor para uma função objetivo de acordo com suas variáveis de decisão e seu conjunto

de restrições. Um exemplo clássico é o problema da mochila (SALKIN; KLUYVER, 1975) que consiste em, dado um conjunto de itens onde cada item possui um valor e peso específico, e uma mochila com uma certa capacidade, escolher quais itens inserir na mochila respeitando sua capacidade de forma que esses itens forneçam o maior valor total. Uma formulação simples do problema é a seguinte:

- Conjuntos e constantes:
  - N: conjunto dos itens;
  - W: capacidade máxima da mochila.
- Parâmetros:
  - v_i: valor do item i, $i \in N$;
  - w_i: peso do item i, $i \in N$;
- Variáveis de decisão:
  - x_i: vetor binário de tamanho $|N|$, onde 0 significa que o item não está na mochila e 1 que está na mochila
- Função objetivo:

$$Z = max \sum_{i \in N} v_i x_i \tag{2.1}$$

- Restrições:

$$\sum_{i \in N} w_i x_i \leq W \tag{2.2}$$

$$x_i \in 0, 1 \tag{2.3}$$

Essa formulação utiliza variáveis binárias (restrição representada pela Equação (2.3)) para tentar maximizar o valor total dos itens na mochila (Equação (2.1)), evitando ultrapassar o peso máximo da mochila (Equação (2.2)), otimizando os recursos disponíveis (w_i), caracterizando um problema mono-objetivo.

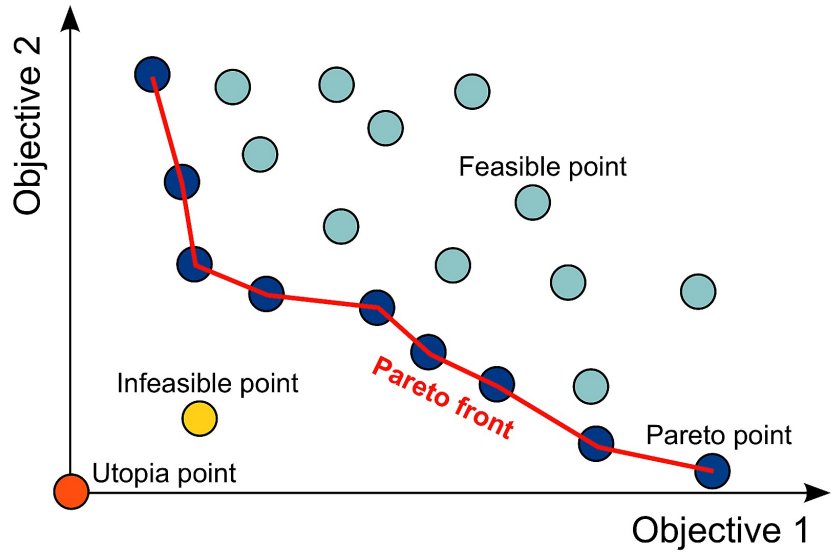
2.1.2 Otimização multi-objetivo

De acordo com Deb et al. (2002), a otimização multi-objetivo envolve a otimização simultânea de vários objetivos que, geralmente, são conflitantes e não podem ser reduzidos a um único critério de otimização. Sendo assim esse tipo de abordagem não utiliza de uma função objetivo para encontrar suas soluções, ao invés disso ele busca encontrar soluções para cada uma das variáveis do problema.

Para analisar todas as soluções geradas pelos algoritmos existem os conceitos de dominância e fronteira de Pareto. O conceito de dominância pode ser explicado da seguinte maneira: dado um conjunto de critérios de otimização, uma solução é considerada não dominada se não existe outra solução no espaço de busca que seja melhor ou igual em todos os critérios e, estritamente, melhor em pelo menos um critério. A partir disso, a qualidade das soluções geradas é avaliada. Já a fronteira de Pareto é o conjunto formado por todas as soluções não dominadas do problema e é descrito em (KLAMMER et al., 2015).

A Figura 2.1 exemplifica essas questões representando um problema de minimização. O ponto em vermelho representa uma solução ideal utópica para o problema de minimização dos objetivos *Objective 1* e *Objective 2*, o ponto em amarelo representa uma solução inviável, os pontos em cor azul escuro representam as soluções não dominadas desse problema, ou seja, não existem outras soluções que dominem essas soluções e os pontos em cor azul claro representam as soluções viáveis do problema que são dominadas pelas soluções na Fronteira de Pareto.

Figura 2.1 – Exemplo de soluções não dominadas.



Fonte: (KLAMMER et al., 2015).

2.1.3 Non-dominated Sorting Genetic Algorithm II (NSGA-II)

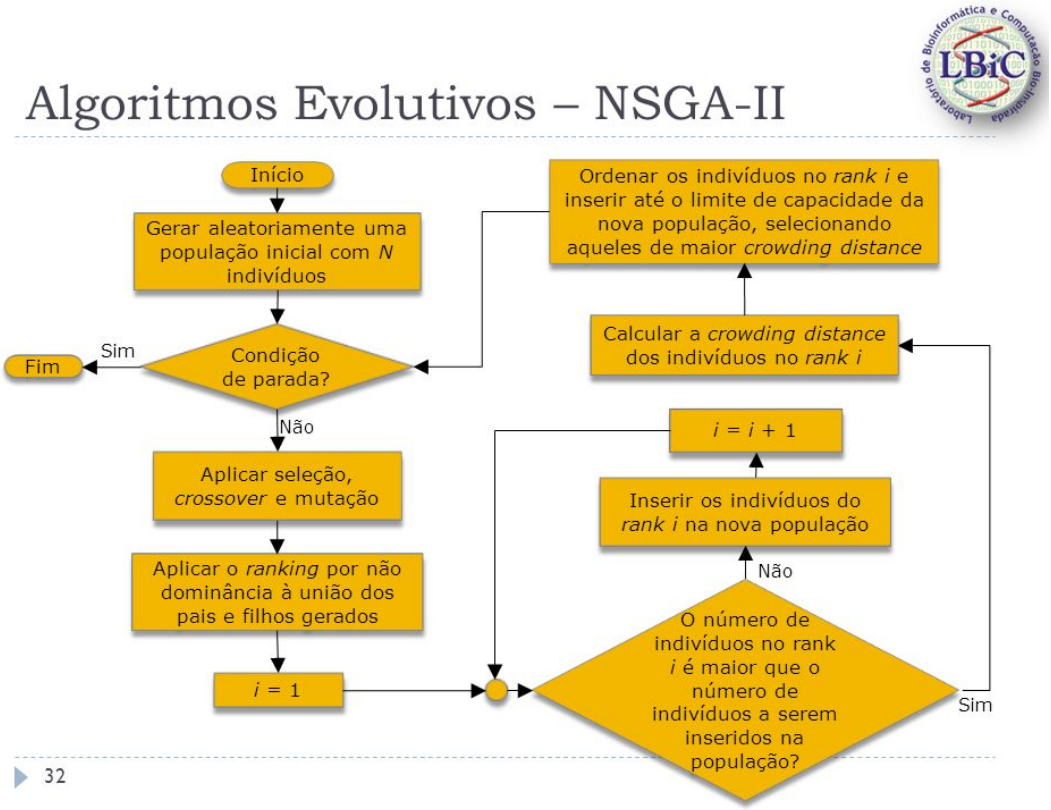
O NSGA-II, de acordo com Deb et al. (2002), foi desenvolvido para lidar com problemas de otimização multi-objetivo, onde há múltiplas funções objetivas a serem otimizadas simultaneamente como no caso deste trabalho. O algoritmo faz analogia com os processos biológicos de evolução e seleção natural onde cada indivíduo é representado por uma solução, o processo de cruzamento entre indivíduos ocorre através dos operadores de *Crossover* onde podem ocorrer mutações nos novos indivíduos e o processo de seleção utiliza da relação de dominância entre as soluções para selecionar as melhores. Sua finalidade é encontrar um conjunto de soluções não dominadas como explicado na Seção 2.1.2. Estas soluções representam soluções onde nenhuma é melhor do que outra em todos os objetivos.

O algoritmo possui cinco partes principais:

- **Inicialização**: geração de uma população inicial de soluções, muitas vezes de forma aleatória.
- **Cruzamento (*Crossover*)**: operadores de cruzamento que irão "mesclar"duas soluções "pais"para gerarem duas soluções "filhas", dessa forma as soluções "filhas"são geradas com características diferentes de cada "pai"permitindo uma melhor exploração do espaço de soluções.
- **Mutação (*Mutations*)**: alterações realizadas nas soluções criadas pela etapa do *crossover* para permitir que o algoritmo percorra novas soluções.
- **Classificação de não dominância**: classificação das soluções com base em suas relações de dominância.
- **Seleção**: seleção de soluções que priorizam aquelas que estão em camadas de não dominância.
- **Substituição**: combinação das duas gerações de indivíduos priorizando as soluções já selecionadas.

A Figura 2.2 apresenta um fluxograma com o funcionamento do algoritmo que começa gerando uma população inicial de N indivíduos. Enquanto a condição de parada não for atendida serão aplicados os operadores de seleção, cruzamento e mutação. Logo após isso, todas as soluções são classificadas em *ranks* de acordo com suas relações de dominância. Com as soluções classificadas, elas são inseridas na nova população na mesma ordem dos *ranks*. Caso um *rank* não possa ser inserido totalmente na nova população (pois ultrapassaria o limite de N indivíduos), é calculado o *crowding distance* dos indivíduos e os que tiverem os maiores valores serão inseridos. O *crowding distance* é uma métrica que calcula a densidade de soluções em torno de um ponto em específico e é utilizada para priorizar regiões menos densas do espaço de soluções para garantir cobertura mais abrangente do conjunto de soluções.

Figura 2.2 – Fluxograma do algoritmo NSGA-II.



Fonte: (COELHO, 2016).

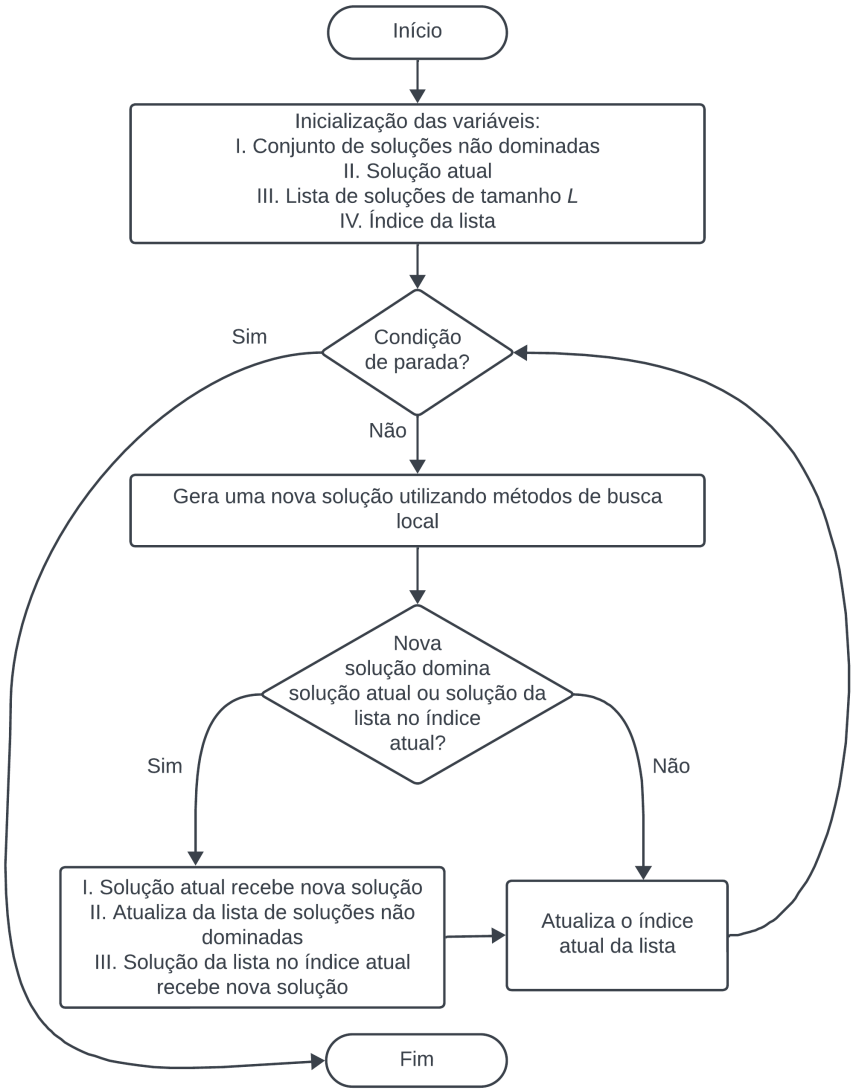
2.1.4 *Multi-Objective Late Acceptance* (MOLA)

De acordo com Liefooghe et al. (2012), em comparação a quantidade algoritmos evolucionários para a resolução de problemas multi-objetivos, a quantidade de algoritmos multi-objetivos de busca local é extremamente pequena. Através dessa constatação, o MOLA, proposta de Vancroonenburg e Wauters (2013), foi desenvolvido utilizando técnicas de busca local e foi proposto uma extensão do algoritmo *Late Acceptance Hill-Climbing* (LAHC) que foi introduzido por Burke e Bykov (2012).

O MOLA preserva a principal característica do LAHC que é comparar a solução atual com soluções geradas várias iterações anteriores através de uma lista cíclica que armazena essas soluções. A cada iteração, uma nova solução é gerada por meio de um método de busca local e comparada com tais soluções. Existem duas principais alterações na lógica do algoritmo principal, a primeira é que o método de comparação entre as soluções leva em consideração a dominância (Seção 2.1.2) e a segunda é que existe uma outra lista que armazena todas as soluções não dominadas encontradas durante a execução do algoritmo. A Figura 2.3 exemplifica o funcionamento do algoritmo que começa inicializando as variáveis que ele utilizará e logo em seguida entra em um *loop* que irá gerar uma nova solução a cada iteração e irá analisar a sua relação de dominância com as soluções da lista e com a solução atual e decidirá se ela poderá entrar na lista de soluções não dominadas. Feito isso o algoritmo irá atualizar as variáveis e

repetirá o processo até a condição de parada ser atendida.

Figura 2.3 – Fluxograma do algoritmo MOLA.

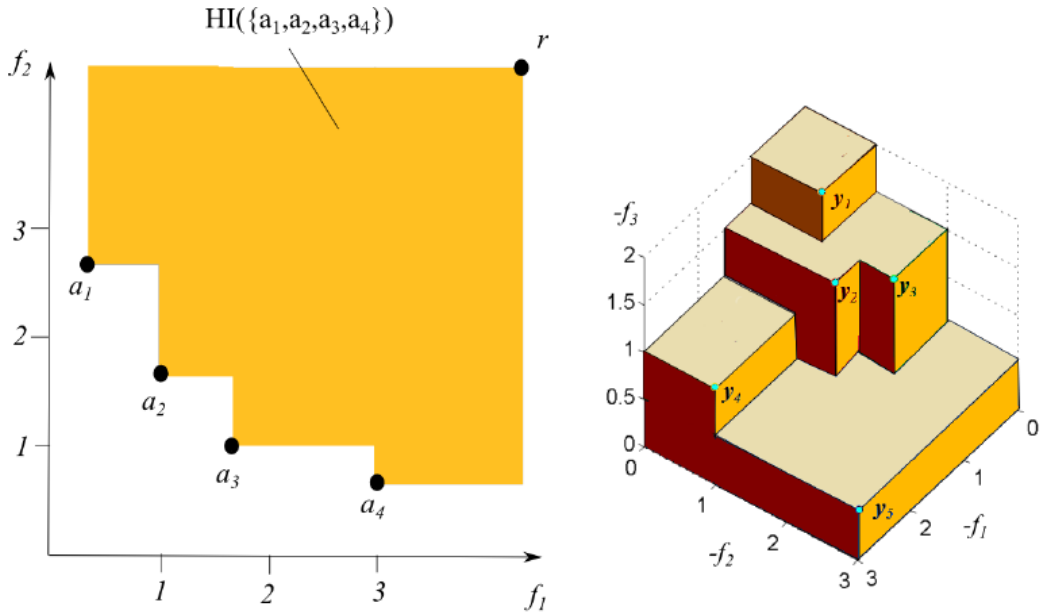


Fonte: Criado pelo autor.

2.1.5 Hiper volume

De acordo com Zitzler, Deb e Thiele (1999), no caso da otimização multi-objetivo, a definição de qualidade é substancialmente mais complexa do que para problemas de otimização mono-objetivo, porque a própria meta de otimização consiste em vários objetivos. A partir disso, várias métricas foram desenvolvidas para medir a qualidade das soluções geradas por esses algoritmos. Uma delas é o hiper volume que consiste em calcular a área do espaço de soluções dominada pela fronteira de Pareto (ZITZLER; DEB; THIELE, 1999). A Figura 2.4 apresenta um exemplo desse cálculo onde dado um ponto qualquer dominado por todas as soluções da fronteira de Pareto, o hiper volume é equivalente a área do espaço de soluções coberto por esses pontos.

Figura 2.4 – Exemplo de hiper volume em problemas com dois e três objetivos.



Fonte: (CUSTÓDIO; EMMERICH; MADEIRA, 2012).

Dessa forma quanto maior o hiper volume, melhor é o conjunto de soluções encontrada pelo algoritmo. Essa métrica será utilizada para avaliar as soluções dos dois algoritmos desenvolvidos neste trabalho

2.1.6 Programação linear inteira mista

De acordo com Schrijver (1998), se todas as variáveis em um modelo de programação linear são restritas em serem apenas valores inteiros nós temos um problema de programação inteira, como exemplo o problema da mochila citado na Seção 2.1.1. Caso um conjunto dessas variáveis de decisão assumirem valores reais esse problema se torna um problema de programação linear inteira mista como descrito por Wolsey (2007).

Para resolver esse tipo de problema, existem vários métodos como por exemplo os de decomposição, heurísticas, meta-heurísticas, algoritmos de enumeração exata, onde a escolha da técnica depende do tamanho e da complexidade do problema. Além disso existem solucionadores de otimização como CPLEX, Gurobi e SCIP que são frequentemente utilizados para resolver esse tipo de problema.

A biblioteca Python-MIP[1] implementa a maioria dessas técnicas e utiliza solucionadores de otimização. Ela tem como vantagens sua facilidade de desenvolvimento já que é possível modelar os problemas em *Python* tão facilmente quanto em linguagens de alto nível, como *MathProg*, sua performance e seus amplos recursos como gerador de cortes, restrições relaxadas

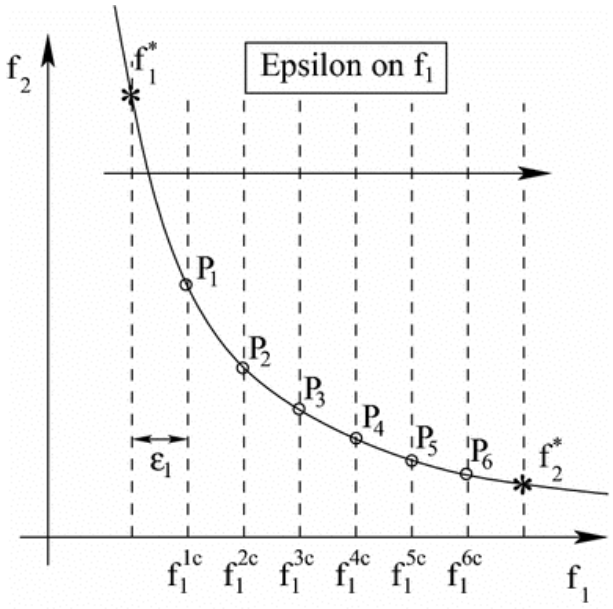
[1] Mais informações sobre em <https://www.python-mip.com>.

entre outros. Ela será usada neste trabalho para gerar uma solução exata com uma abordagem mono-objetiva apenas para fim de comparação com as soluções com abordagem multi-objetiva.

2.1.7 Método do epsilon restrito

O método ϵ-restrito, apresentado por Haimes (1971), é uma técnica utilizada em problemas de otimização multi-objetivo que tem como finalidade fornecer soluções pertencentes a fronteira de Pareto. A ideia principal dessa abordagem é transformar o problema de otimização multi-objetivo em um problema mono-objetivo focando em resolver apenas um dos objetivos e transformando os outros em restrições ϵ. A cada iteração do algoritmo uma nova solução da fronteira de Pareto é encontrada e o valor de ϵ é atualizado para restringir ainda mais o espaço de soluções que será investigado. A Figura 2.5 demonstra como esse método permite explorar diferentes espaços de soluções em busca da melhor solução para apenas um objetivo, com isso, ao final da execução do algoritmo, teremos soluções pertencentes a fronteira de Pareto. No exemplo, para cada valor de ϵ, o espaço de soluções é restringido o que permite explorar outras regiões e encontrar outras soluções da Fronteira de Pareto.

Figura 2.5 – Exemplo do método do ϵ restrito.



Fonte: (KUNDU; MANDAL, 2018).

3 Definição do problema

O presente trabalho aborda o Problema de Alocação de Salas (PAS) no contexto do Instituto de Ciências Exatas e Biológicas (ICEB) da Universidade Federal de Ouro Preto (UFOP). Este problema foi tratado na versão multi-objetivo que busca resolvê-lo mesmo por meio de todos os objetivos avaliados. O ICEB oferece vários cursos de graduação e pós-graduação para seus alunos, além de ofertar aulas do currículo de outros cursos da instituição. As aulas ocorrem em 36 (trinta e seis) salas divididas em dois prédios: o próprio ICEB e o Pavilhão Central de Aulas (PCA). Estas aulas ocorrem em 16 (dezesseis) horários diários divididos em três turnos (manhã, tarde e noite). Cada aula consiste de uma turma (conjunto de alunos), professor e disciplina, e possui ainda um conjunto de dias e horários pré-definidos. Além disso existem restrições que podem ser aplicadas nos encontros e elas podem ser classificadas como fortes ou fracas. Restrições fracas são consideradas desejáveis e a sua violação não invalida a solução, já as restrições fortes são necessárias e sua violação invalida a solução, Assim, o problema consiste em alocar os encontros às salas disponíveis, considerando situações e restrições fracas como:

- **Pré-requisitos dos encontros**: os encontros podem ter um conjunto de pré-requisitos que a sala deve atender, tais como: necessidade de projetor, quadro negro, e/ou outros equipamentos específicos;
- **Preferências** de docentes e discentes por uma determinada sala, que podem acontecer por motivos diversos. A título de exemplo, citamos que alguns docentes têm alergia a giz, e que alguns discentes possuem capacidade limitada de locomoção, preferindo, portanto, salas no andar térreo.
- **Capacidade das salas**: diferentes encontros podem ter número diferente de alunos, e a demanda das aulas deve ser atendida sempre que possível.

Há ainda outras restrições fortes que devem ser respeitadas, como: (i) cada sala pode alocar apenas um encontro em um período de tempo e (ii) cada encontro somente pode ser alocado a uma sala em cada período de tempo.

As soluções geradas por meio dos algoritmos propostos neste trabalho tiveram a finalidade de melhorar a atual utilização das salas pela UFOP. Para tal, foram levadas em consideração três características que têm como objetivo criar uma solução mais adequada para alunos e professores. Os objetivos são:

- Evitar alocar encontros de pequena demanda em salas com alta capacidade;
- Tentar alocar o máximo de encontros possíveis;

- Evitar alocar encontros em salas que possuem capacidade menor que a demanda;

Dessa forma, podemos modelar o problema para atender as nossas intenções de estudo e análise. Para isso iremos considerar três objetivos:

- Ociosidade (Idl ou *Idleness*): caso mais da metade da sala esteja com vagas não utilizadas essa quantidade é contabilizada;
- Desalocação (Dea ou *Deallocates*): a quantidade de alunos que estão desalocados;
- Em pé (Sta ou *Standing*): caso a sala não comporte a quantidade de alunos do encontro, essa diferença é contabilizada.

Para poder modelar o problema é necessário representar o caso onde um encontro não é alocado em nenhuma sala. Uma forma de resolver isso é utilizar o conceito de sala virtual que representa uma sala inexistente que "aloca" todos os encontros que não estão alocados em nenhuma sala na solução. Para melhor compreensão da modelagem do PAS neste trabalho, a seguinte notação será utilizada:

- E: conjunto de encontros;
- H: conjunto dos horários disponíveis;
- H^+: conjunto dos horários de cada encontro;
- S: conjunto de salas;
- S_v: conjunto da sala virtual;
- S^+: $S \cup S_v$;
- D_i: demanda do encontro i, $i \in E$;
- C_j: capacidade da sala j, $j \in S$;

Com todas essas informações o modelo pode ser representado por:

- Variáveis de decisão:
  - X_{esh}: matriz de decisão binária de $|E| \times |S^+| \times |H|$ que indica que o encontro e está alocado na sala s no horário h;

- Funções objetivo:

$$Z = min \sum_{i \in E} f_{Dea}(i,j), \forall j \in S^{+} \tag{3.1}$$

$$Z = min \sum_{i \in E} f_{Idl}(i,j), \forall j \in S^{+} \tag{3.2}$$

$$Z = min \sum_{i \in E} f_{Sta}(i,j), \forall j \in S^{+} \tag{3.3}$$

$$f_{Dea}(i,j) = \begin{cases} D_i, \text{ se } j \in S_v \\ 0, \text{ se } j \notin S_v \end{cases} \tag{3.4}$$

$$f_{Idl}(i,j) = \begin{cases} C_j - D_i, \text{ se } j \in S \wedge D_i < \frac{C_j}{2} \\ 0 \end{cases} \tag{3.5}$$

$$f_{Sta}(i,j) = \begin{cases} D_i - C_j, \text{ se } j \in S \wedge D_i > C_j \\ 0 \end{cases} \tag{3.6}$$

- Restrições:

$$\sum_{i \in E} X_{ijk} = 1, \forall j \in S^{+}, \forall k \in H_i^{+} \tag{3.7}$$

$$\sum_{j \in S} X_{ijk} \leq 1, \forall i \in E, \forall k \in H \tag{3.8}$$

Resumidamente o modelo tem como objetivo otimizar as funções 3.1, 3.2 e 3.3 que significam, respectivamente, a quantidade de alunos alocados em uma sala virtual S_v, que representa alunos que não foram alocados em nenhuma sala, a quantidade de vagas ociosas em uma sala que possui um encontro alocado e a quantidade de vagas que excederam a capacidade da sala. Para fazer isso é utilizado uma matriz de decisão X_{es} que indicará quando um encontro e ($e \in E$) for alocado em uma sala s ($s \in S^{+}$) contendo o valor 1 ou indicando que um encontro não está alocado em uma sala contendo o valor 0. E por fim, ele deverá respeitar as restrições 3.7 e 3.8 que significam, respectivamente, que todo encontro deverá ser alocado em alguma sala do conjunto S^{+} e que cada sala pode ter no máximo um encontro alocado por horário

As próximas seções irão abordar a organização dos dados e as estruturas de vizinhança utilizada nos seguintes tópicos:

- Organização dos dados (Seção 3.1).
- Estruturas de vizinhança (Seção 4.1).

3.1 Organização dos dados

Os dados do problema foram fornecidos por meio de um arquivo no formato *JavaScript Object Notation* (JSON) que contém informações como salas, horários, encontros, entre outros e por isso esse tipo de arquivo foi utilizado para todos os dados de entrada do problema. Por meio desse arquivo, os dados mais relevantes para o problema foram extraídos e organizados em estruturas de dados específicas para facilitar o seu acesso dentro do código. Uma decisão tomada durante o desenvolvimento do algoritmo foi optar por organizar a maioria dos dados em um *array*, visto que o acesso indexado dessa estrutura de dado é $O(1)$.

Os campos da instância são:

- Horários (*schedules*): esse campo contém os dados dos horários de aula fornecidos pela instituição;
- Prédios (*buildings*): esse campo contém os dados dos prédios da instituição;
- Salas (*classrooms*): esse campo contém os dados das salas de aula dos prédios da instituição;
- Professores (*professors*): esse campo contém os dados dos professores da instituição;
- Disciplinas (*subjects*): esse campo contém os dados das disciplinas ofertadas na instituição;
- Encontros (*meetings*): esse campo contém os dados das aulas que são ministradas na instituição;
- Preferências (*preferences*): esse campo contém as preferências dos encontros;
- Restrições (*restrictions*): esse campo contém as restrições dos encontros;
- Reservas (*reservations*): esse campo contém as reservas de horários na instituição;

Abaixo segue um exemplo da instância utilizada:

```
{
  "schedules": [ {"ID": 1, "startTime": "07:30", "endTime": "08:20"} ],
  "buildings": [ {"ID": 1, "name": "Predio-1"} ],
  "classrooms": [
     {"ID": 1, "isLab": false, "capacity": 70, "buildingID": 1,
      "description": "Predio-1-SL001", "floor": 0, "board": "N",
      "projector": true}
  ],
  "professors": [ {"code": "1.111.111", "name": "Nome"} ],
  "subjects": [ {"code": "XXX111", "name": "Disciplina 1"} ],
  "meetings": [
     {"isPractical": false, "dayOfWeek": 3, "vacancies": 50,
      "demand": 50, "subjectCode": "BEV118", "classes": ["11"],
      "schedules": [3, 2], "professors": ["2.027.719"]}
```

```
  ],
  "preferences": [
     {"category": "professor", "categoryCode": "1.111.111",
      "building": null, "floor": null, "board": "B",
      "projector": false},
     {"category": "turma", "categoryCode": "XXX111-11",
      "building": null, "floor": 0, "board": "B", "projector": false}
  ],
  "restrictions": [
     {"category": "professor", "categoryCode": "1.111.111",
     "building": null, "floor": null, "board": "W", q"projector": null},
     {"category": "turma", "categoryCode": "XXX111-11",
      "building": 1, "floor": null, "board": null, "projector": null}
  ],
  "reservations": [{"classroomID": 2, "dayOfWeek": 3, "scheduleID": 4}]
}
```

Algoritmo 3.1 – Exemplo da instância

Para melhor visualização do problema, os encontros foram separados por dia - de segunda a sábado - sendo que cada dia também contém uma matriz de tamanho *quantidade de horários* $\times$ *quantidade de salas*, que armazena as informações do encontro alocado naquela célula e o status da célula (alocado, não alocado ou reservado).

Os dados utilizados representam a alocação utilizada pela própria universidade, porém para fins de análise, foram geradas mais instâncias fictícias aleatórias que utilizam dos dados originais da UFOP.

4 Metodologia

Para poder resolver o problema de alocação de salas foram avaliados dois algoritmos multi-objetivos: o NSGA-II e o MOLA. Além deles, foram desenvolvidos mais dois algoritmos auxiliares, um gerador de soluções guloso e o algoritmo epsilon restrito que resolve o problema utilizando o Python-MIP para gerar uma solução utilizando programação inteira.

Os objetivos listados na Seção 3 serão utilizados para avaliar o critério de dominância entre as soluções para poder gerar a fronteira de Pareto, conforme explicado na Seção 2.1.2. Uma solução entrará para a fronteira de Pareto quando não existir nenhuma outra solução que domine ela, ou seja, que é melhor ou igual que ela em todos os objetivos e estritamente melhor em pelo menos um.

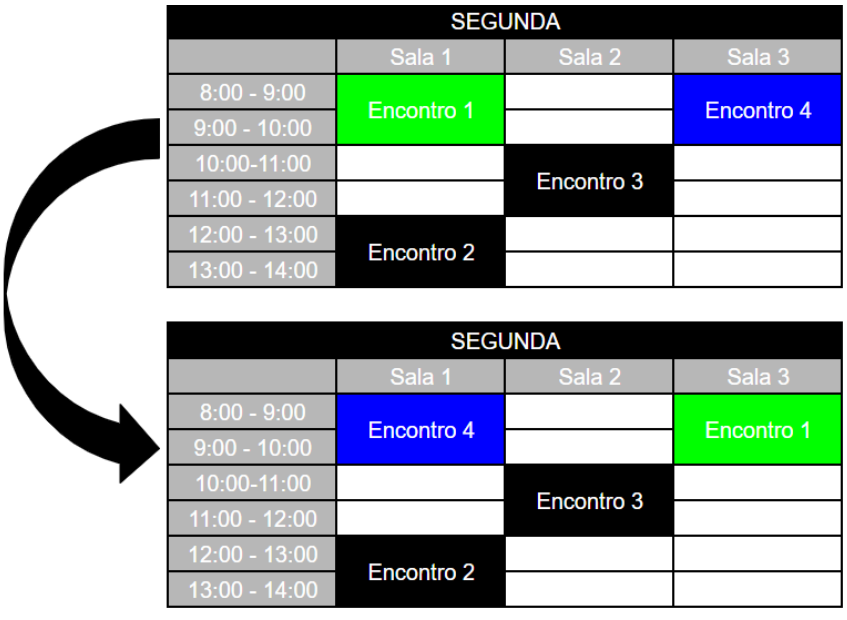
4.1 Estruturas de vizinhança

Para percorrer o espaço de soluções foram implementados cinco movimentos que são utilizados por todos os algoritmos deste trabalho.

4.1.1 Movimento trocar

Para o movimento trocar, o algoritmo procura aleatoriamente dois encontros para tentar fazer uma troca, como mostrado na Figura 4.1, onde o encontro 1 que estava na sala 1 foi trocado com o encontro 4 que estava na sala 3.

Figura 4.1 – Exemplo do movimento Trocar.



Fonte: Criado pelo autor.

Para ocorrer uma troca, os encontros precisam obedecer aos seguintes critérios:

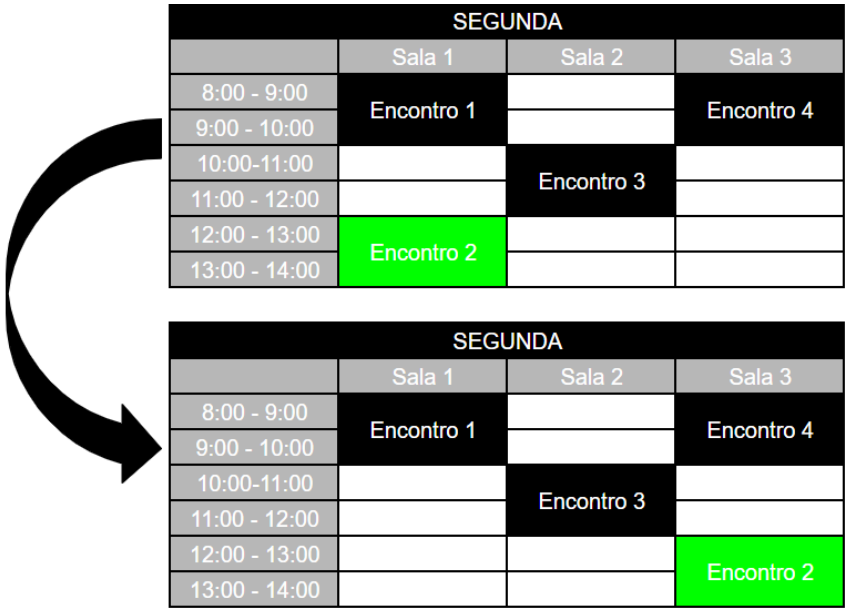
- ocorrerem no mesmo dia,
- estarem alocados,
- ter os mesmos horários de aula.

Ao localizar algum encontro que respeite esses critérios, o algoritmo calcula a alteração nos objetivos da solução ao realizar a movimentação de ambos e atualiza seus valores.

4.1.2 Movimento deslocar

Para o movimento deslocar, o algoritmo procura, aleatoriamente, uma sala para tentar deslocar o encontro escolhido, como mostrado na Figura 4.2, onde o encontro 2 que estava na sala 1 foi deslocado para a sala 3.

Figura 4.2 – Exemplo do movimento Deslocar.



Fonte: Criado pelo autor.

Para ocorrer um deslocamento, a sala precisa obedecer a alguns critérios:

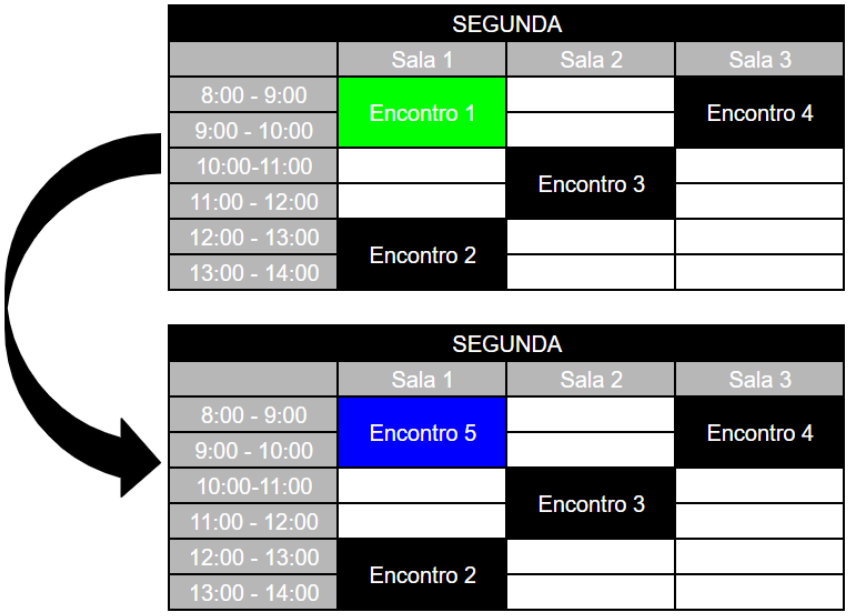
- estar situada no mesmo dia que o encontro,
- não possuir nenhum outro encontro alocado nos horários do encontro escolhido,
- não estar reservada pela instituição para outras finalidades.

Ao encontrar alguma sala que respeite esses critérios, o algoritmo calcula a alteração nos objetivos da solução ao realizar o deslocamento e atualiza seus valores.

4.1.3 Movimento substituir

Para o movimento substituir, o algoritmo procura aleatoriamente um encontro desalocado para tentar substituir um encontro que está alocado, como mostrado na Figura 4.3, onde o encontro 1 que estava alocado na sala 1 foi desalocado para que o encontro 5 pudesse ser alocado na sala 1 no mesmo horário.

Figura 4.3 – Exemplo do movimento Substituir.



Fonte: Criado pelo autor.

Para ocorrer uma substituição, os encontros precisam obedecer a alguns critérios:

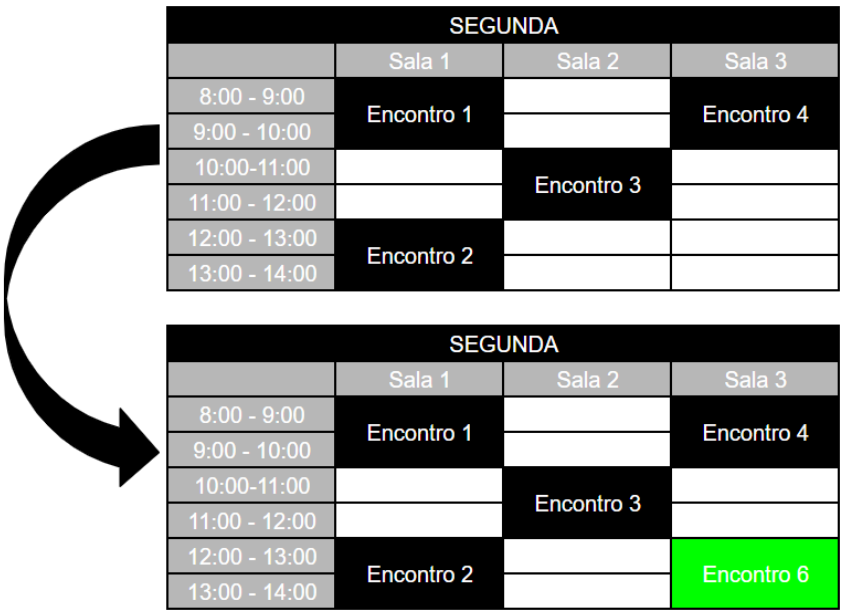
- ocorrerem no mesmo dia,
- um deles estar alocado e o outro não,
- ter os mesmos horários de aula.

Caso eles respeitem esses critérios, o algoritmo calcula a alteração nos objetivos da solução ao realizar a substituição e atualiza seus valores.

4.1.4 Movimento alocar

Para o movimento alocar, é necessário escolher um encontro que ainda não está alocado, como mostrado na Figura 4.4, onde o encontro 6 que estava desalocado foi alocado na sala 3.

Figura 4.4 – Exemplo do movimento Alocar.



Fonte: Criado pelo autor.

Para ocorrer uma alocação o algoritmo procura um encontro e uma sala aleatória que devem obedecer a esses critérios:

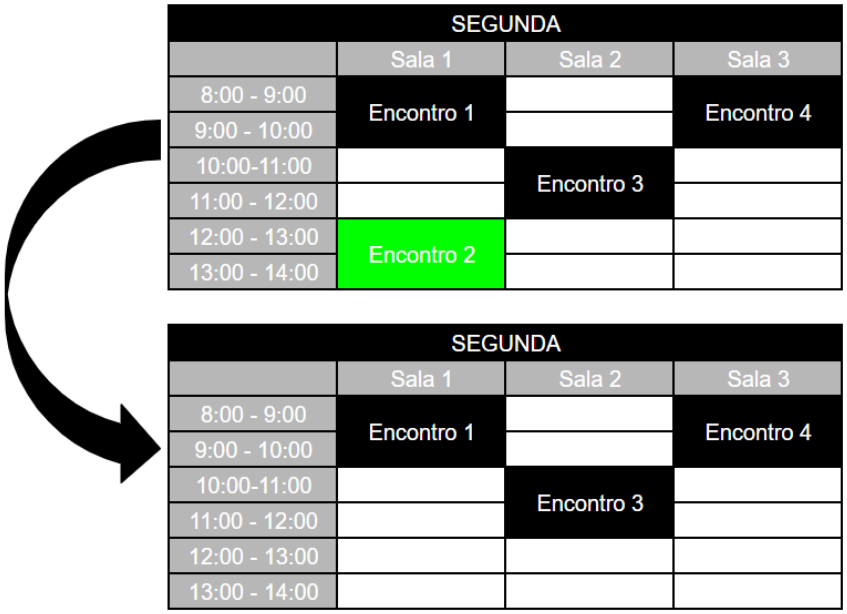
- o encontro deve estar desalocado,
- a sala deve estar vazia,
- os dois devem pertencer ao mesmo dia.

Para o encontro e a sala que respeitem esses critérios, o algoritmo calcula a alteração nos objetivos da solução ao realizar a substituição e atualiza seus valores.

4.1.5 Movimento desalocar

Para o movimento desalocar, é necessário encontrar um encontro que já está alocado, como mostrado na Figura 4.5, onde o encontro 2 que estava alocado na sala 1 foi desalocado.

Figura 4.5 – Exemplo do movimento Desalocar.



Fonte: Criado pelo autor.

Para ocorrer uma desalocação, o algoritmo procura um encontro e uma sala aleatória que devem obedecer a esses critérios:

- o encontro deve estar alocado,
- a sala deve estar vazia,
- os dois devem pertencer ao mesmo dia.

Para o encontro e a sala que respeitem esses critérios, o algoritmo calcula a alteração nos objetivos da solução ao realizar a substituição e atualiza seus valores.

4.2 *Non-dominated Sorting Genetic Algorithm* II adaptado (NSGA-II)

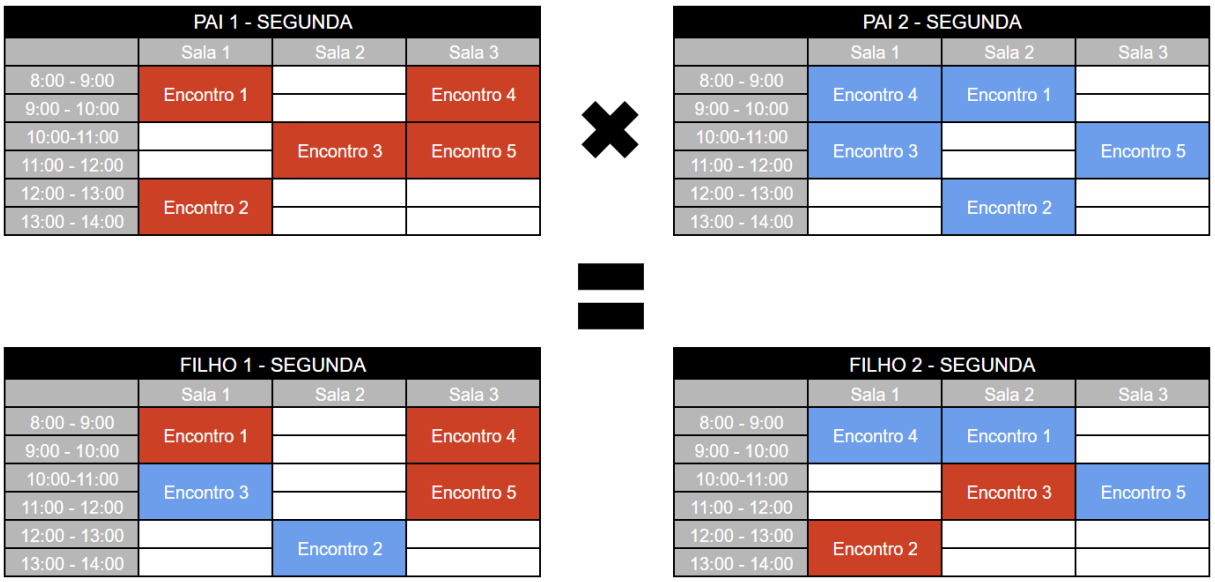
O primeiro algoritmo implementado foi o NSGA-II, algoritmo descrito na Seção 2.1.3. O algoritmo busca encontrar um conjunto de soluções não dominadas como explicado em Seção 2.1.2. Estas soluções representam as soluções não dominadas encontradas pelo algoritmo onde nenhuma é melhor do que a outra em todos os objetivos.

A ideia do algoritmo original apresentada na Seção 2.1.3 foi seguida, porém algumas adaptações foram feitas para se adequar ao problema abordado em alguns passos:

- **Cruzamento (*Crossover*)**: o problema foi separado de acordo com os dias da semana, com isso a etapa do *crossover* é realizada em cada dia da semana. Cada *crossover* realizado terá

duas soluções ‘pais’ e gerará duas soluções ‘filhas’. Para cada encontro do dia é sorteado qual ‘filho’ ficará com a alocação de qual ‘pai’ sendo que as alocações se complementam. Um exemplo pode ser visualizado na Figura 4.6 onde os encontros em vermelho são referentes do primeiro pai e os encontros em azul ao segundo.

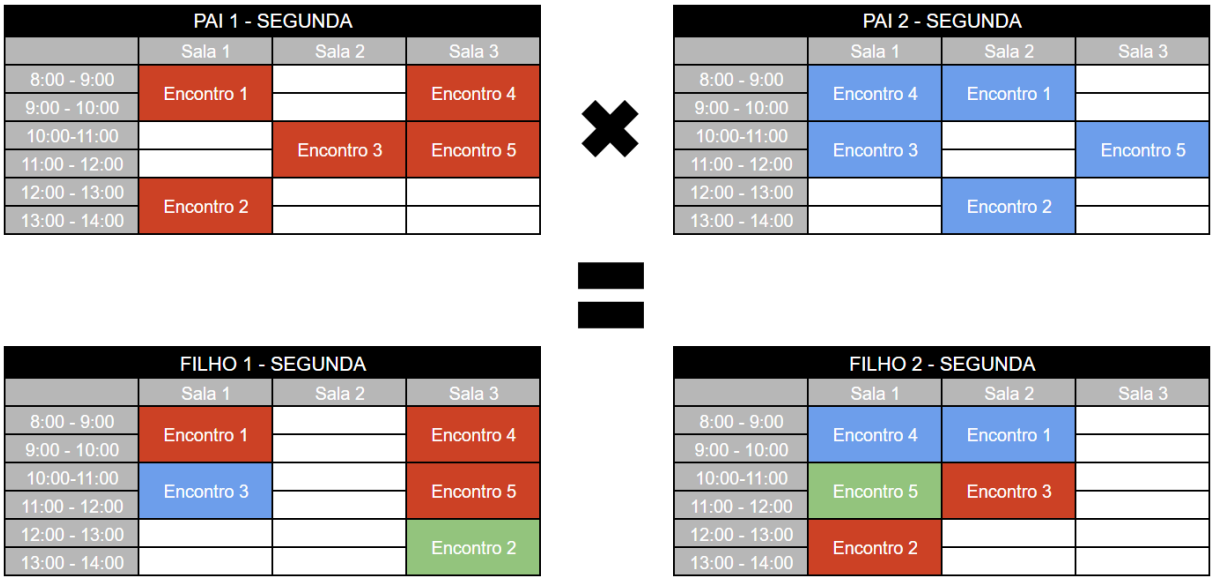
Figura 4.6 – Exemplo do operador de cruzamento.



Fonte: Criado pelo autor.

- **Mutação (*Mutation*)**: nessa etapa existe uma probabilidade da mutação acontecer e a mutação é composta por um conjunto de 5 movimentos escolhidos aleatoriamente que utilizam o conceito das estruturas de vizinhança. Um exemplo pode ser visualizado na Figura 4.7 onde os encontros em verde representam os encontros que sofreram mutações.

Figura 4.7 – Exemplo do operador de mutação.



Fonte: Criado pelo autor.

Para facilitar a compreensão dos Algoritmos 1 e 2, os seguintes termos são usados:

- $G \rightarrow$ quantidade de gerações que o NSGA-II irá fazer.
- $P \rightarrow$ população inicial.
- $\eta \rightarrow$ tamanho da população inicial.
- $Np \rightarrow$ abreviação de *new population*, o conjunto de soluções retornadas pelo algoritmo de crossover.
- $Fr \rightarrow$ abreviação de *fronts*, o conjunto de soluções ordenadas de acordo com a questão da não dominância.
- $Ng \rightarrow$ abreviação de *new generation*, o conjunto de soluções selecionadas de acordo com a ordenação realizada.
- $p1 \rightarrow$ uma solução pai selecionada aleatoriamente.
- $p2 \rightarrow$ uma solução pai selecionada aleatoriamente.
- $f1 \rightarrow$ uma solução filho que herdará das soluções pais.
- $f2 \rightarrow$ uma solução filho que herdará das soluções pais.
- $E \rightarrow$ o conjunto de todos os encontros presentes nas soluções.
- $e \rightarrow$ um encontro pertencente ao conjunto de todos os encontros presentes nas soluções.
- $f1_e, f2_e \rightarrow$ o encontro selecionado que pertence as soluções filho.
- $p1_e, p2_e \rightarrow$ o encontro selecionado que pertence as soluções pais.

O Algoritmo 1 exemplifica o código desenvolvido nesse trabalho usando como inspiração o algoritmo implementado por Deb et al. (2002). Ele inicia na linha 1 definindo o *loop* principal e para cada iteração ele irá aplicar o operador de *crossover* (linha 2) que é explicado pelo Algoritmo 2, ordena as soluções de acordo com sua dominância (linha 3) e seleciona as melhores soluções geradas para irem para a próxima iteração (linhas 4 a 7).

Algoritmo 1: *Non-dominated Sorting Genetic Algorithm II* (NSGA-II)

Entrada: Quantidade de gerações G, População inicial P e tamanho da população η

NSGA-II(G, P, η)

```
1  enquanto número de gerações G não for alcançado faça
2      Np ← crossover(P)
3      Fr ← non_dominated_sort(Np)
4      Ng ← ∅
5      enquanto |Ng| < η faça
6          Ng ← Fr.pop(0)
7      P ← Ng
8  retorne P
```

O Algoritmo 2 descreve como o passo de cruzamento (*crossover*) é feito. Para cada par de soluções pais pertencentes a população inicial (lina 1), o algoritmo cria duas novas soluções filhas vazias (linha 2) e para cada encontro ele irá sortear de qual pai o primeiro filho irá herdar a alocação e segundo filho fica com a alocação do outro pai (linhas 3 a 5). Após isso uma mutação pode ocorrer nas soluções filhas (linhas 6 a 8) e as soluções filhas são adicionadas na população (linha 9)

Algoritmo 2: *Crossover*

Entrada: População inicial P (soluções pais)

Crossover(P)

```
1  para cada (p1, p2) aleatórios ∈ P faça
2      (f1, f2) ← solução vazia
3      para cada e ∈ E faça
4          f1_e ← p1_e ou p2_e baseado em um sorteio
5          f2_e ← o que não foi escolhido para f1_e
6      se chance de mutação ocorrer então
7          f1 ← movimento aleatório
8          f2 ← movimento aleatório
9      P.append(f1, f2)
10 retorne P
```

4.3 Multi-Objective *Late Acceptance* (MOLA)

O segundo algoritmo implementado é o *Multi-Objective Late Acceptance* (MOLA) proposto por Vancroonenburg e Wauters (2013) com pequenas adaptações para se adequar ao problema abordado. O algoritmo tem a mesma finalidade do NSGA-II descrito na Seção 4.2 que é encontrar um conjunto de soluções não dominadas como explicado na Seção 2.1.2.

O algoritmo desenvolvido em (VANCROONENBURG; WAUTERS, 2013) é uma adaptação ao algoritmo criado em (BURKE; BYKOV, 2012) que permite com que ele seja capaz de encontrar soluções para problemas multiobjetivos. A ideia original consistia em criar uma lista com várias soluções que possuíam seus respectivos valores da função objetivo analisada e a cada iteração era gerada uma solução nova a partir da solução atual e caso essa nova solução fosse melhor que a atual ou a respectiva solução na lista, ela era aceita e os valores da lista e da solução atual eram atualizados. Com isso, no decorrer das iterações as soluções da lista convergem e criam um parâmetro para as próximas soluções a serem analisadas.

Para o novo algoritmo multiobjetivo, a lógica se manteve, porém na fase de análise da nova solução gerada, o valor da função objetivo não é mais relevante, para definir se ela será ou não aceita, ela será comparada com uma outra lista de soluções que armazena todas as soluções não dominadas encontradas até o momento pelo algoritmo. Caso ela não seja dominada por nenhuma delas ela é aceita e caso ela domine alguma solução dessa lista, essa solução dominada é removida.

Para facilitar a compreensão do Algoritmo 3, os seguintes termos são usados:

- $L \rightarrow$ tamanho da lista que será usada como parâmetro para avaliar as soluções geradas durante o algoritmo.
- $S_0 \rightarrow$ solução inicial.
- $P \rightarrow$ lista de soluções que armazenará as soluções não dominadas.
- $S \rightarrow$ solução que será alterada durante o algoritmo , ela é inicializada com S_0.
- $i \rightarrow$ iterador utilizado dentro do laço *while*.
- $Q \rightarrow$ lista de tamanho L com as soluções que serão utilizadas como parâmetro.
- $Q_i \rightarrow$ solução da lista Q na posição i.
- $S' \rightarrow$ solução gerada pelo algoritmo.
- $N() \rightarrow$ função que gera uma nova solução usando os as estruturas de vizinhança.

O Algoritmo 3 com a lógica do MOLA inicializa a lista P com a solução inicial que é gerada por um algoritmo guloso (Algoritmo 4), inicializa a lista Q com L soluções S_0 (linhas 1 e 2), inicializa a lista Q de tamanho L com o valor dos objetivos da solução S_0 (linhas 3 e 4) e inicializa o iterador da lista com 0 (linhas 5). Após essa inicialização, enquanto a condição de parada não for satisfeita o MOLA cria uma nova solução S' a partir da solução atual S através dos movimentos de busca local (linha 7), caso essa nova solução domine a solução atual ou a solução na posição $i\%L$, S e $Q_{i\%L}$ passam a ser S', e uma função que irá analisar a possível entrada de S na lista de preto P é chamada (linas8 a 11). Por fim o iterador que indica a posição da lista analisada na iteração é atualizado (linha 12).

Algoritmo 3: *Multi-Objective Late Acceptance* (MOLA)

Entrada: Tamanho da lista L, Solução inicial S_0
MOLA(L, S_0)
$P \leftarrow [S_0]$
$S \leftarrow S_0$
para cada $i \in L$ **faça**
  $Q_i \leftarrow \mathbf{f}(S_0)$
$i \leftarrow 0$
enquanto condição de parada não for atendida **faça**
  $S' \leftarrow N(S)$
  se $\mathbf{f}(S') \prec \mathbf{f}(S)$ or $\mathbf{f}(S') \prec Q_{i\%L}$ **então**
    $S \leftarrow S'$
    $UPDATE\_PARETO\_SET(P, S)$
    $Q_{i\%L} \leftarrow S$
  $i \leftarrow i + 1$
retorne P

4.4 Gerador de solução guloso

Para poder explicar este algoritmo os seguintes termos serão utilizados:

- $Idl \rightarrow$ quantidade de vagas ociosas
- $Dea \rightarrow$ quantidade de alunos desalocados
- $Sta \rightarrow$ quantidade de alunos alocados porém sem vaga
- $S \rightarrow$ solução com nenhum encontro alocado
- $p \rightarrow$ probabilidade
- $MC \rightarrow$ um vetor de cópia dos encontros
- $CC \rightarrow$ um vetor de cópia das salas
- $e \rightarrow$ um encontro da solução

Tanto o NSGA-II e o MOLA precisam de soluções iniciais para serem iniciados e para isso foi desenvolvido um algoritmo gerador de soluções guloso, ou seja, busca a melhor solução para o problema. Para calcular o que seria a melhor solução foi utilizada uma função objetivo simples que representa a soma de todos os objetivos da solução com peso igual a um, o qual é calculado por:

$$\min \quad Idl + Dea + Sta. \tag{4.1}$$

Como o conceito de melhor solução em um problema multi-objetivo é relativo e pelo fato de ser necessário ter soluções iniciais variadas para os algoritmos, o gerador de soluções foi

alterado para contornar essa situação. Essa mudança consiste em não escolher a melhor alocação possível para o encontro em todas as iterações, para isso uma probabilidade foi adicionada para em algumas iterações selecionar a primeira alocação disponível e não a melhor. O algoritmo de geração de soluções guloso (Algoritmo 4) inicia ordenando o vetor de encontros e de salas de maneira decrescente de acordo com a demanda de cada encontro e com a capacidade de cada sala (linha 1 e 2). Após isso, para cada encontro do problema, dependendo da probabilidade selecionada o encontro em questão será alocado na melhor opção disponível (linhas 4 e 5) ou na primeira opção disponível (linhas 6 e 7).

Algoritmo 4: Gerador de soluções guloso

Entrada: Solução com nenhum encontro alocado S, probabilidade p
Gerador de soluções(S, p)

```
1 | MC ← encontros ordenados de forma decrescente de acordo com sua demanda
2 | CC ← salas ordenadas de forma decrescente de acordo com sua capacidade
3 | para cada e ∈ MC faça
4 |     se probabilidade p então
5 |         S ← e na melhor alocação disponível
6 |     se não probabilidade p então
7 |         S ← e na primeira alocação disponível
8 | retorne S
```

Para gerar toda a população para o algoritmo NSGA-II esse algoritmo é chamado até construir uma população com a quantidade de indivíduos determinada.

4.5 Solução exata da versão mono-objetivo do problema

Para avaliar as soluções multi-objetivo encontradas, foi calculado o valor de uma solução exata para o problema mono-objetivo utilizando um método de programação inteira. Para isso foi utilizado a biblioteca Python-MIP[1]. Nessa biblioteca está implementado um algoritmo de modelagem e resolução de Programas Lineares Inteiros Mistos. Para o problema em questão foi utilizado o seguinte modelo matemático:

$$\min \sum_{e \in E} \sum_{s \in Sa'} \sum_{h \in H_e} C_{es} * X_{esh} \tag{4.2}$$

Restrições:

$$\sum_{e \in E} X_{esh} = 1, \forall s \in Sa', \forall h \in H_e \tag{4.3}$$

$$\sum_{s \in Sa} X_{esh} \leq 1, \forall h \in H, \forall e \in E \tag{4.4}$$

[1] Mais informações sobre em <https://www.python-mip.com>.

$$X_{esh} = X_{esh'}, \forall e \in E, \forall s \in Sa', \forall (h, h') \in G_e \tag{4.5}$$

onde:

- E: conjunto de encontros;
- H: conjunto de horários;
- H_e: conjunto de horários de um encontro específico;
- Sa: conjunto de salas disponíveis;
- Sa': conjunto de salas disponíveis, incluindo uma sala virtual para os desalocados;
- C_{es}: matriz bidimensional com o custo de alocação de cada encontro e em cada sala s respeitando a equação:

  $$o_{es} + d_e + d_{es} \tag{4.6}$$

  onde:

  - o_{es}: capacidade ociosa do encontro e na sala s;
  - d_e: demanda não alocada do encontro e;
  - d_{es}: demanda alocada do encontro e que ultrapassa da capacidade da sala s;
- X_{esh}: matriz binária tridimensional que assume o valor 0 quando o encontro e, na sala s, no horário h não está alocado e 1 quando está alocado;
- G_e: conjunto dos horários do encontro e agrupados dois a dois (h, h').

Resumidamente, a função objetivo da Equação (4.2) tenta minimizar o valor da solução considerando a variável de decisão X_{esh} e a matriz C_{es} que contém os valores calculados pela função na Equação (4.6). As restrições, representadas pelas funções das Equações (4.3), (4.4) e (4.5), significam, respectivamente, que todo encontro deve ser alocado em uma sala do conjunto Sa', que toda sala deve ter no máximo um encontro alocado por horário e que todos encontro em todos os seus horários devem ser alocados na mesma sala.

Através desse modelo foi possível chegar em uma solução exata para esse problema sem levar em consideração o caso multi-objetivo. A solução em questão foi apenas utilizada para comparação e obteve e os valores dos objetivos separadamente estão indicados na Tabela 4.1.

Tabela 4.1 – Tabela dos resultados da solução exata da versão mono-objetiva do problema.

Objetivo	**Valor**
Idl	0
Dea	4125
Sta	0

Fonte: Criado pelo autor.

4.6 Epsilon-restrito

Outra métrica de avaliação utilizada foi a técnica do ϵ-restrito que busca encontrar as soluções da fronteira de Pareto de um determinado problema. A estratégia utilizada consiste em transformar um problema multi-objetivo em um problema mono-objetivo que otimiza apenas um dos objetivos e criando restrições que representam os outros utilizando algoritmos exatos para encontrar a solução ótima. Com isso é possível encontrar a fronteira de Pareto como mencionado em (HAIMES, 1971).

Para executar esse método para o problema abordado nesse trabalho a seguinte estratégia foi utilizada: para uma solução vazia o objetivo de desalocação (Dea) atinge seu valor máximo, esse objetivo será escolhido para ser otimizado enquanto os outros se tornarão restrições. Com o problema modelado da mesma forma que na Seção 4.5 com a única diferença sendo que dessa vez o problema não será dividido por dias como citado na Seção 3.1 mas será modelado por completo utilizando a biblioteca do Python-MIP[2]. Com isso a ideia é começar o loop de otimização adicionando a restrição para um dos objetivos e para cada iteração outro loop será criado para o terceiro objetivo. A cada iteração desses loops um valor ϵ é retirado da restrição para o objetivo correspondente o que restringe ainda mais o espaço de busca. O valor utilizado para ϵ foi 30, o que corresponde a um valor entre 1% e 10% dos valores máximos encontrados para cada objetivo para cada uma das instâncias pelos algoritmos MOLA e NSGA-II

O Algoritmo 5 apresenta a ideia principal desse método que começa criando o modelo como descrito na Seção 4.5 com exceção da função objetivo (linha 1), após isso uma única função objetivo é adicionada para poder minimizar o objetivo Desalocado (*Dea*) (linha 2), Os outros dois objetivos do problema são adicionados ao modelo como restrições (linhas 3 e 4) e um conjunto que irá conter as soluções não dominadas é inicializado com um conjunto vazio (linha 5). O primeiro *loop* será realizado enquanto o objetivo Ociosidade (*Idl*) puder ser restringido (linha 6) e a cada iteração ele adiciona a restrição para esse objetivo (linha 7), inicia outro *loop* que será realizado enquanto o objetivo Em pé (*Sta*) puder ser restringido (linha 8), remove a restrição para o objetivo Ociosidade (*Idl*) (linha 17) e atualiza o limite para o valor do objetivo Ociosidade (*Idl*) (linha 18). Dentro do loop interno, o algoritmo adiciona a restrição para o objetivo Em pé (*Sta*) (linha 9), otimiza o modelo (linha 10) e logo após a resolução do modelo a restrição para o objeto Em pé (*Sta*) é removida para poder adicionar outra novamente na próxima iteração (linha 11). Em seguida o algoritmo verifica se o modelo atual é viável ou não, em caso positivo a solução é adicionada ao conjunto de soluções não dominadas (linhas 12 e 13) e caso contrário o *loop* interno é encerrado (linha 14 e 15) e por fim ele atualiza o limite para o valor do objetivo Em pé (*Sta*). A seguinte notação é utilizada:

- $m \rightarrow$ modelo do problema
- $S \rightarrow$ soluções da fronteira de Pareto

[2] Mais informações sobre em <https://www.python-mip.com>.

- $Sta \rightarrow$ quantidade de alunos alocados porém sem vaga
- $Idl \rightarrow$ quantidade de vagas ociosas
- $Dea \rightarrow$ quantidade de alunos desalocados

Algoritmo 5: ϵ-restrito

Entrada: Valor de ϵ

ϵ-restrito(ϵ)

```
m ← modelagem igual a da Seção 4.5 com exceção da função objetivo
m.objective ← minimize(Dea)
idl_constr_value ← maior valor encontrada para esse objetivo pelos algoritmos multi-objetivo +30
sta_constr_value ← maior valor encontrada para esse objetivo pelos algoritmos multi-objetivo +30
S ← [ ]
enquanto puder diminuir a restrição para Idl faça
    m.add_constr(Idl < idl_constr_value − ϵ)
    enquanto puder diminuir a restrição para Sta faça
        sta_constr ← m.add_constr(Sta < sta_constr_value − ϵ)
        m.optimize()
        m.remove(sta_constr)
        se modelo for viável então
            S.append(solução)
        se modelo não for viável então
            break
        sta_constr_value ← sta_constr_value − ϵ
    m.remove(idl_constr)
    idl_constr_value ← idl_constr_value − ϵ
retorne S
```

5 Resultados

Os algoritmos descritos na Seção 4 foram implementados na linguagem de programação Python[1] versão 3.11.5 sendo que o modelo da Seção 4.5 foi desenvolvido utilizando a biblioteca Python-MIP[2] utilizando seu resolvedor padrão. A instância principal é a mesma utilizada no período 2020.1 no Instituto de Ciências Exatas e Biológicas (ICEB) e Pavilhão Central de Aulas (PCA) da UFOP. Além dela foram utilizadas mais cinco instâncias geradas a partir da principal.

Para cada um dos parâmetros de entrada, tanto o NSGA-II quanto o MOLA foram executados 5 vezes totalizando 60 execuções (5 por algoritmo para a instância principal e 5 por algoritmo pra cada uma das outras instâncias geradas). Para o algoritmo MOLA a condição de parada foi estabelecida como sendo 900 segundos de execução e para o NSGA-II foram 100 gerações com 100 indivíduos. O padrão de 900 segundos foi escolhido pois com esse tempo o algoritmo começava a ter dificuldades de encontrar novas soluções não dominadas e para nivelar o tempo de execução dos dois algoritmos foram escolhidas 100 gerações e 100 indivíduos pois o tempo de execução com esses parâmetros era próximo de 900 segundos. A máquina utilizada para os testes foi um computador com processador Intel i7-9750H 2.6Ghz, 16GB de memória RAM, sistema operacional *Windows*. Dada a característica estocástica do algoritmo, para cada combinação instância-algoritmo foi utilizada uma *seed* diferente de um a cinco para geração de números pseudoaleatórios.

5.1 Descrição das instâncias

Para executar os experimentos deste trabalho foi utilizada uma instância real referente aos dados do ICEB (Instituto de Ciências Exatas e Biológicas) da UFOP (Universidade Federal de Ouro Preto). Essa instância é um arquivo JSON que separa, em campos, informações importantes para o problema como: salas disponíveis, horário de funcionamento dos prédios, os encontros a serem alocados, os professores da universidade, restrições e preferências dos encontros.

A instância original possui 888 encontros, 36 salas e 16 horários no total. Já os dados de entrada que foram gerados utilizando a instância original como base possuem 1.000 encontros, 36 salas e 16 horários no total sendo que o valor da demanda dos encontros varia entre 10 e 70. Esse pequeno aumento no número de encontros e uma maior variedade e distribuição dos valores para suas demandas foi suficiente para tornar o problema mais complexo. Para a geração dos dados foram consideradas as mesmas disciplinas, salas e horários dos dados originais, o que os difere da instância usada pela UFOP são a quantidade de encontros e a demanda de cada um deles. A criação desses novos dados de entrada teve como objetivo aumentar a dificuldade

[1] Mais informações sobre a linguagem em <https://www.python.org>.
[2] Mais informações sobre em <https://www.python-mip.com>.

do problema para poder ter uma melhor análise de seus desempenhos pois com uma maior quantidade de encontros para serem alocados mais combinações de soluções são possíveis. Além disso, a instância original gerava uma situação onde para todas as soluções o valor do objetivo *Em pé* (Idl) era 0 pois a demanda da maioria dos seus encontros era menor que a média de capacidade das salas como podemos analisar no gráfico da Figura 5.1.

Um exemplo da instância pode ser encontrado no exemplo de Código 3.1 na Seção 3.1.

5.2 Soluções

Após a execução de todos os algoritmos, a Tabela 5.1 e os gráficos das Figuras 5.1, 5.2, 5.3, 5.4, 5.5 e 5.6 foram gerados para facilitar a visualização dos resultados. Como o MIP gerou somente uma solução, ele foi removido da Tabela 5.1. Cada um dos gráficos mostram a execução de todos os algoritmos para cada uma das instâncias, sendo o primeiro para a instância original e as outras para as instâncias aleatórias. Além disso, foi calculado o hiper volume com o ponto de referência $[2, 2, 2]$ e a quantidade de soluções geradas por cada algoritmo. Todos esses dados são mostrados a seguir:

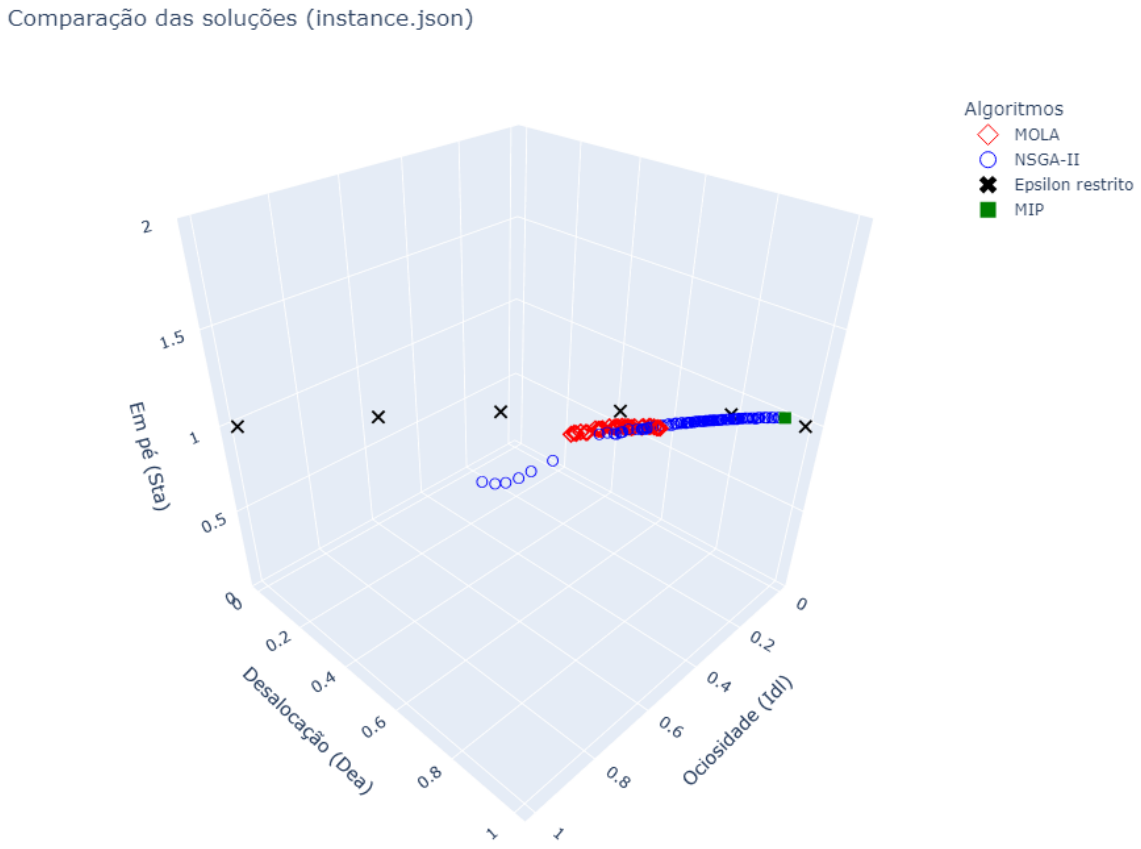
Instância	Algoritmo	Hiper volume	Soluções geradas		
			Menor	Média	Maior
Instância original	MOLA	2.405939	20	32.0	46
	NSGA-II	2.708870	49	72.4	86
	epsilon restrito	3.468343	6	6	6
Instância 1	MOLA	3.604383	11	35.0	72
	NSGA-II	5.281034	89	94.8	98
	epsilon restrito	6.903395	187	187	187
Instância 2	MOLA	3.538751	35	50.4	70
	NSGA-II	4.591808	98	99.6	100
	epsilon restrito	6.821391	181	181	181
Instância 3	MOLA	2.576858	24	37.8	48
	NSGA-II	3.794773	94	97.6	100
	epsilon restrito	6.339259	307	307	307
Instância 4	MOLA	3.810017	28	52.0	80
	NSGA-II	4.937945	95	98.2	100
	epsilon restrito	6.707994	210	210	210
Instância 5	MOLA	3.797787	17	54.0	86
	NSGA-II	5.276902	88	96.2	100
	epsilon restrito	6.914846	238	238	238

Tabela 5.1 – Tabela de métricas para todos os algoritmos para cada instância

Para a instância original, todos os algoritmos conseguiram satisfazer completamente

o objetivo Em pé (*Sta*), o que acabou gerando um conjunto *flat* de soluções como é possível visualizar na Figura 5.1. O conjunto de soluções do NSGA-II cobriu uma área mais ampla do espaço de soluções do que o conjunto de soluções do MOLA além de ter gerado mais soluções que se aproximam da fronteira de Pareto.

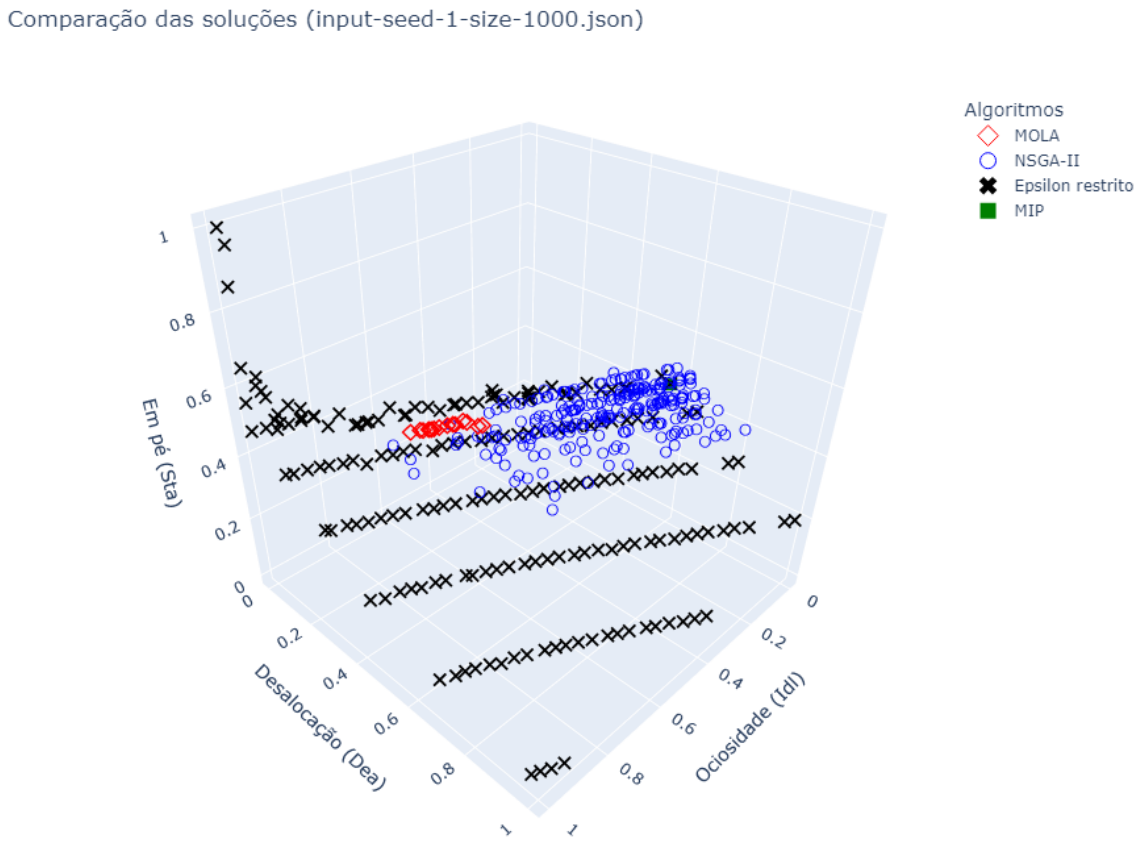
Figura 5.1 – Gráfico contendo os resultados das execuções para a instância original.



Fonte: Criado pelo autor.

Na primeira instância, a distribuição das soluções encontradas pelos algoritmos está mais distribuída no espaço de soluções, porém nenhum dos dois algoritmos conseguiu abranger uma área tão extensa quanto a fronteira de Pareto sendo o NSGA-II o que mais se aproximou como pode ser visualizado na Figura 5.2.

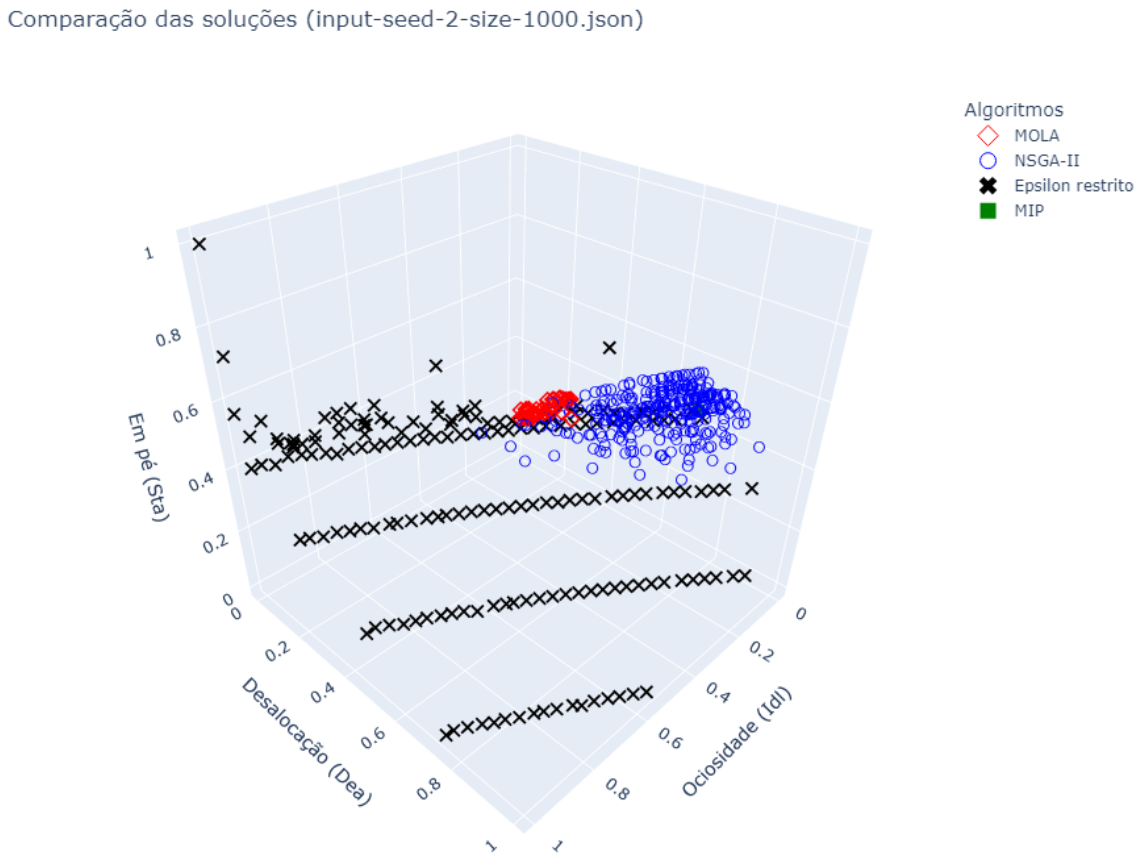
Figura 5.2 – Gráfico contendo os resultados das execuções para a primeira instância.



Fonte: Criado pelo autor.

Na segunda instância, a área coberta pelo conjunto de soluções dos algoritmos ficou mais restrita, mas mesmo assim os algoritmos conseguiram gerar uma boa quantidade de soluções e exploraram as vizinhanças da melhor forma possível como podemos visualizar na Figura 5.3.

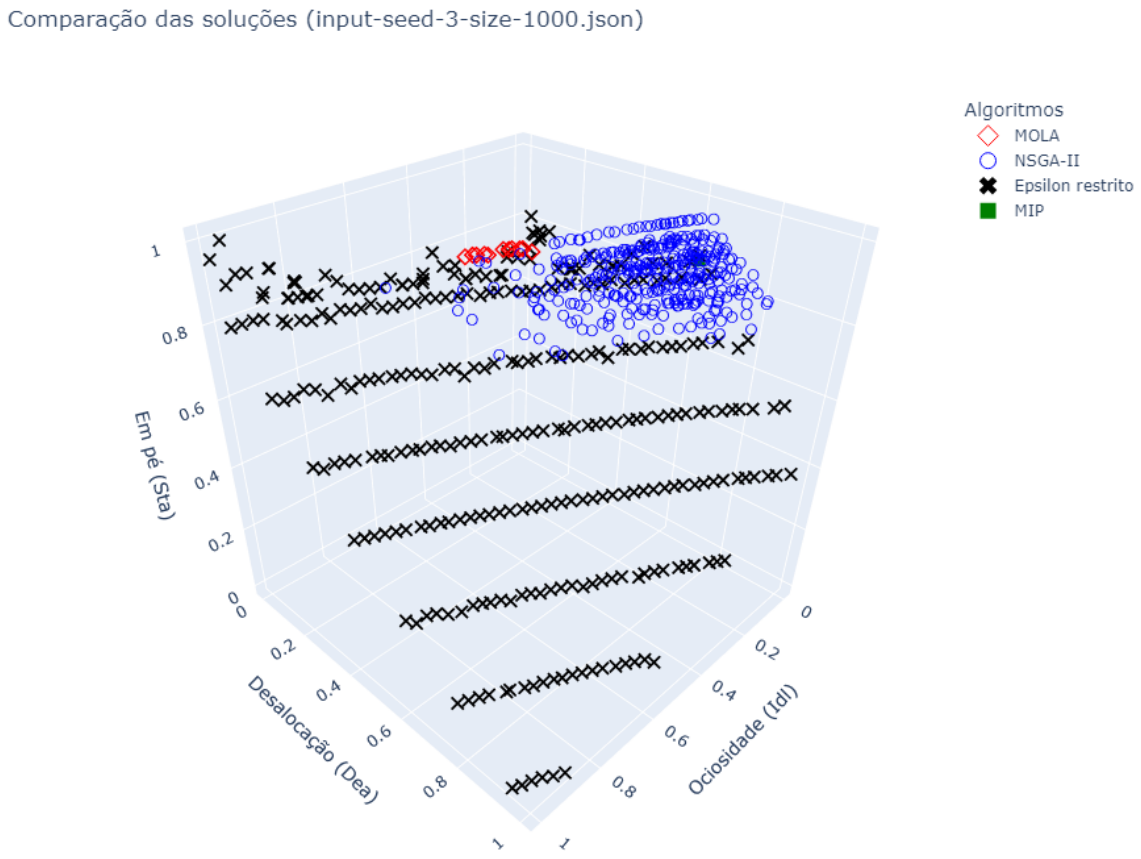
Figura 5.3 – Gráfico contendo os resultados das execuções para a segunda instância.



Fonte: Criado pelo autor.

O método do epsilon restrito teve sua melhor performance na terceira instância onde ele conseguiu encontrar uma quantidade maior de soluções da fronteira de Pareto. Já os algoritmos desenvolvidos não conseguiram explorar tão bem as soluções para esse problema ficando restritos em uma pequena área do espaço de soluções sendo o MOLA o que teve mais dificuldades de exploração com esses dados de entrada como podemos visualizar na Figura 5.4.

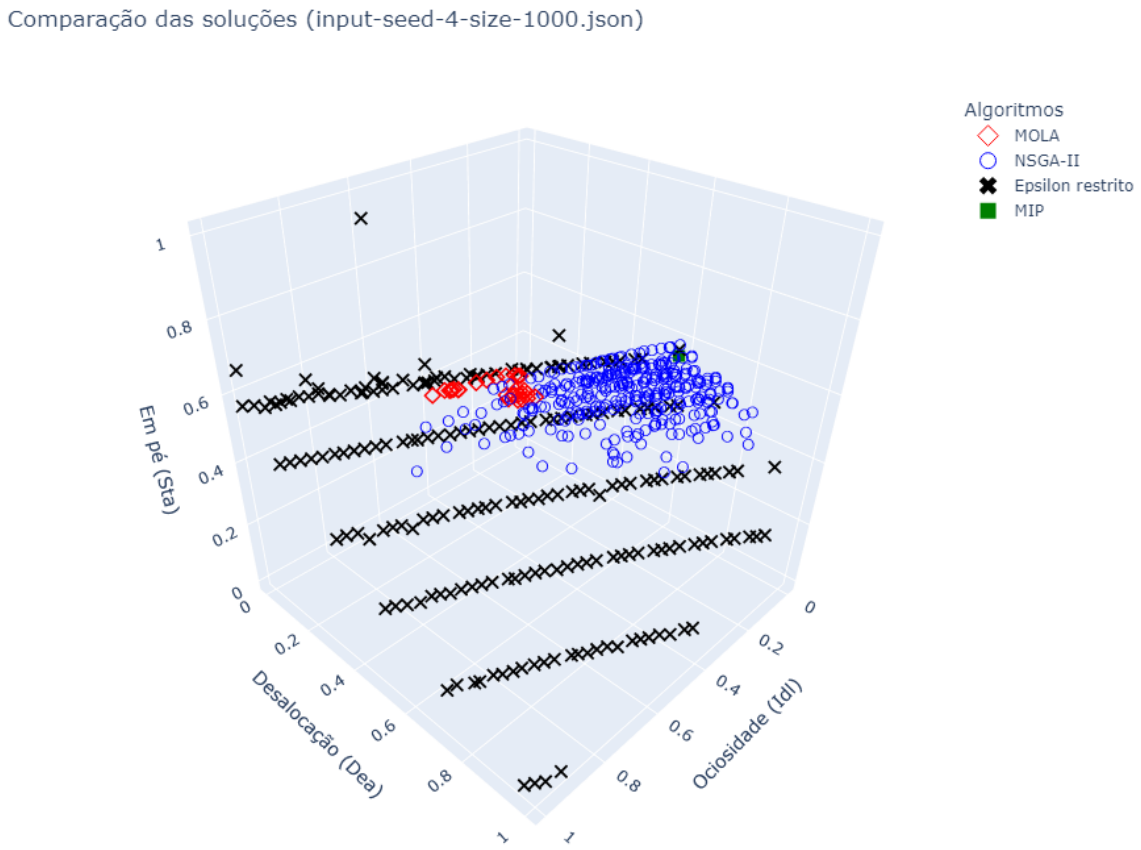
Figura 5.4 – Gráfico contendo os resultados das execuções para a terceira instância.



Fonte: Criado pelo autor.

Para a quarta instância, os algoritmos mantiveram seus comportamentos, gerando uma quantidade boa de soluções e explorando uma parte do espaço de soluções como podemos visualizar na Figura 5.5.

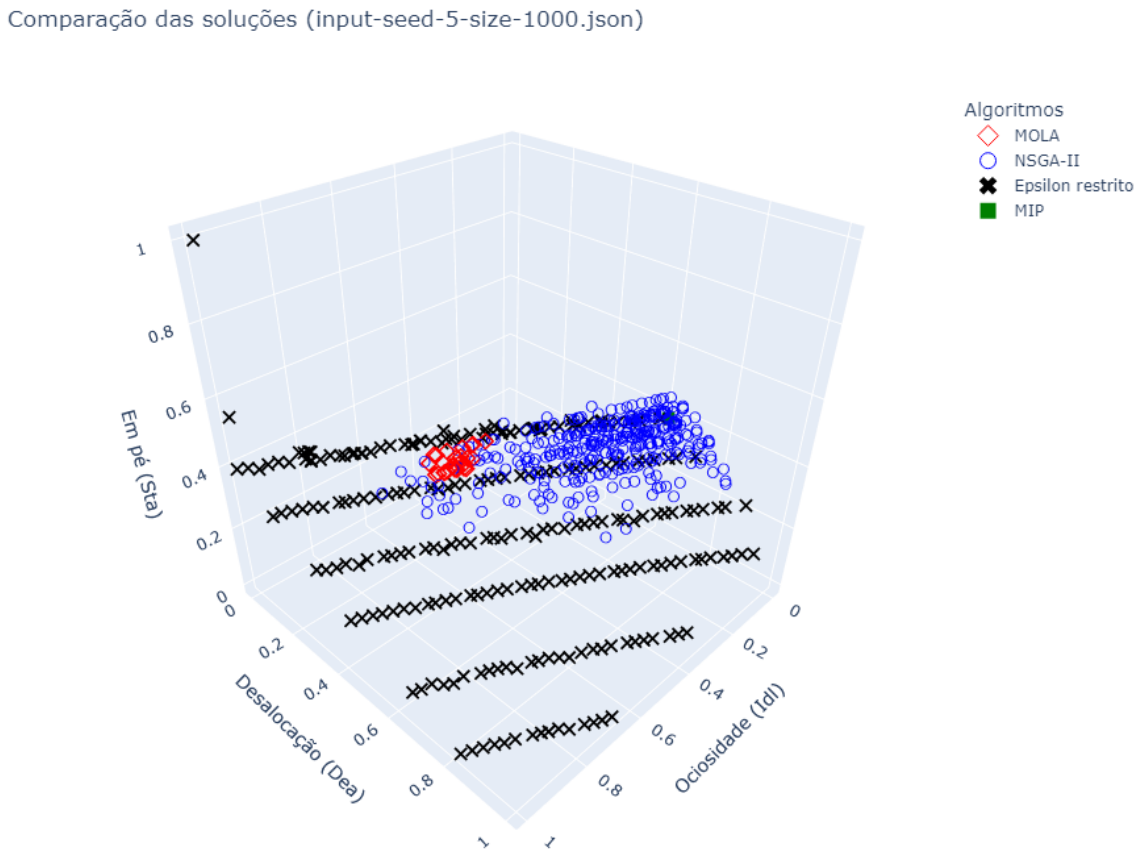
Figura 5.5 – Gráfico contendo os resultados das execuções para a quarta instância.



Fonte: Criado pelo autor.

Para a quinta instância, os algoritmos mantiveram seus comportamentos novamente como podemos visualizar na Figura 5.6.

Figura 5.6 – Gráfico contendo os resultados das execuções para a quinta instância.



Fonte: Criado pelo autor.

Analisando os gráficos das Figuras 5.1, 5.2, 5.3, 5.4, 5.5 e 5.6 e a Tabela 5.1 é possível perceber que os algoritmos conseguiram gerar uma variedade de soluções para o problema. O NSGA-II conseguiu gerar uma quantidade de soluções maiores e que cobrem um espaço maior do espaço de soluções em relação ao MOLA como pode ser observado pelos dados na Tabela de 5.1 através dos valores de média e hiper volume. Isso demonstra a versatilidade do NSGA-II que é considerado um dos melhores algoritmos para resolução de problemas multi-objetivos (ZITZLER; DEB; THIELE, 2000).

Como o MOLA executa um movimento por iteração ele tem mais dificuldades de explorar o espaço de soluções e eventualmente de sair de um ótimo local o que acaba prejudicando sua cobertura do espaço de soluções. Já o NSGA-II, por gerar várias novas soluções por iteração e promover mudanças em algumas delas através da etapa de mutação, acaba por analisar uma quantidade maior de soluções do que o MOLA durante toda sua execução, facilitando assim a exploração do espaço de soluções.

Aliado ao fator descrito no Capítulo 4 sobre a dificuldade do MOLA de explorar uma

quantidade maior de soluções, a solução inicial gerada pelo algoritmo guloso tem um grande impacto sobre qual área o MOLA irá percorrer. Outro detalhe é que para a instância original, todo o conjunto de soluções possui o mesmo valor para o objetivo *Em pé (Sta)*, isso ocorre pois a média de demanda dos encontros desses dados é inferior a média das capacidades das salas, portanto os valores desse objetivo são sempre 0. Para melhor visualização no gráfico, foi decidido que o maior valor encontrado para cada objetivo seria equivalente ao número 1, sendo assim, todas as soluções estão com o valor 1 para esse objetivo

Em relação a solução gerada pelo MIP, é possível perceber que ele foi capaz de gerar uma solução ótima para o problema mono-objetivo, conseguindo melhorar ao máximo os objetivos *Em pé (Sta)* e *Ociosidade (Idl)*. Em comparação com as soluções multi-objetivo, ela entraria no conjunto de soluções não dominadas eliminando algumas soluções dos outros algoritmos. O resultado do MIP pode ser utilizado pelas instituições, porém o método multi-objetivo fornece uma diversidade maior de soluções que podem ser analisadas para encontrar a que mais se adéqua as suas necessidades.

A fronteira de Pareto pode ser observada nos gráficos das Figuras 5.1 até 5.6 através das soluções do método ϵ-restrito que estão na cor preta. Como esperado, os algoritmos implementados nesse trabalho abrangem uma área menor em relação as soluções do método do ϵ-restrito pelo fato de serem algoritmos meta-heurísticos que não necessariamente entregam soluções ótimas, mas sim soluções aproximadas. A desvantagem de utilizar esse método é seu tempo de processamento. Para o descobrimento da fronteira da Pareto foram necessárias em média doze horas de processamento utilizando valores altos para o parâmetro ϵ que chegou a ser cerca de 10% dos valores de um dos objetivos (esse foi o responsável pela distância entre as linhas da fronteira de Pareto). Já os algoritmos multi-objetivos foram executados por apenas quinze minutos e conseguiram gerar um conjunto de soluções próximas a fronteira na maioria das vezes.

6 Considerações Finais

Os principais objetivos deste trabalho apresentados na Seção 1.2 foram alcançados. As necessidades da instituição foram identificadas e levadas em consideração na resolução do problema (Seção 2), o modelo de entrada de dados foi proposto (Seção 3.1), as estruturas de vizinhança foram elaboradas (Seção 4.1), os algoritmos foram desenvolvidos (Seção 4.2 e 4.3) e analisados (Seção 5).

Constatou-se que os algoritmos conseguem prover uma variedade de soluções não dominadas que podem ser analisadas para serem escolhidas pela instituição para aplicá-las durante o período de aulas. Porém, o algoritmo NSGA-II se mostrou melhor no quesito de soluções geradas e na cobertura do espaço de soluções buscado como exemplificado nos gráficos das Figuras 5.1 a 5.6 e na Tabela 5.1. Sendo assim o NSGA-II consegue entregar uma variedade maior de soluções para serem analisadas e então selecionadas pela instituição. Além disso essas soluções estão próximas da fronteira de Pareto gerada pelo método ϵ-restrito o que indica a qualidade das mesmas.

Já o MIP conseguiu gerar uma solução em um espaço onde nenhum dos outros algoritmos conseguiu alcançar, porém, devido a sua funcionalidade, ele apenas consegue gerar uma única solução que busca otimizar apenas uma função objetivo e não se preocupa em otimizar todos os objetivos simultaneamente. Dependendo da necessidade da instituição, a solução gerada pelo MIP pode ser uma solução viável, todavia, ela não analisará todas as demandas simultaneamente.

6.1 Trabalhos Futuros

Para trabalhos futuros, um caminho a ser seguido é implementar abordagens para sair de ótimos locais. Outra vertente que pode ser seguida é analisar como os dados de entrada influenciam na execução dos algoritmos e nas soluções, visto que, dependendo do problema, alguns objetivos podem não ser relevantes e o método de busca local pode ser ineficaz. Por fim, realizar uma análise do impacto da população inicial para os algoritmos. Para isso, o MIP pode ser usado para a geração da população ou solução inicial.

Referências

AARTS, E.; KORST, J.; MICHIELS, W. Simulated annealing. Search methodologies: introductory tutorials in optimization and decision support techniques, Springer, p. 187–210, 2005.

AL-YAKOOB, S. M.; SHERALI, H. D. Mathematical programming models and algorithms for a class–faculty assignment problem. European Journal of Operational Research, Elsevier, v. 173, n. 2, p. 488–507, 2006.

BEYROUTHY, C.; BURKE, E. K.; LANDA-SILVA, J. D.; MCCOLLUM, B.; MCMULLAN, P.; PARKES, A. J. Understanding the role of ufos within space exploitation. In: Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2006). [S.l.: s.n.], 2006. p. 359–362.

BURKE, E. K.; BYKOV, Y. The late acceptance hill-climbing heuristic. University of Stirling, Tech. Rep, 2012.

BURKE, E. K.; BYKOV, Y. The late acceptance hill-climbing heuristic. European Journal of Operational Research, Elsevier, v. 258, n. 1, p. 70–78, 2017.

CARTER, M. W.; TOVEY, C. A. When is the classroom assignment problem hard? Operations Research, INFORMS, v. 40, n. 1-supplement-1, p. S28–S39, 1992.

COELHO, F. V. Z. G. Otimização Evolutiva Multiobjetivo. 2016. Apresentação de slides. Acessado em janeiro de 2024. Disponível em: <https://www.dca.fee.unicamp.br/~lboccato/topico_13_multiobjetivo.pptx>.

CUSTÓDIO, A.; EMMERICH, M.; MADEIRA, J. Recent developments in derivative-free multiobjective optimization. Computational Technology Reviews, v. 5, n. 1, p. 1–31, 2012.

DEB, K.; PRATAP, A.; AGARWAL, S.; MEYARIVAN, T. A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE transactions on evolutionary computation, IEEE, v. 6, n. 2, p. 182–197, 2002.

EVEN, S.; ITAI, A.; SHAMIR, A. On the complexity of time table and multi-commodity flow problems. In: IEEE. 16th annual symposium on foundations of computer science (sfcs 1975). [S.l.], 1975. p. 184–193.

HAIMES, Y. On a bicriterion formulation of the problems of integrated system identification and system optimization. IEEE transactions on systems, man, and cybernetics, Institute of Electrical and Electronics Engineers (IEEE), n. 3, p. 296–297, 1971.

KLAMMER, M.; DYBOWSKI, J. N.; HOFFMANN, D.; SCHAAB, C. Pareto optimization identifies diverse set of phosphorylation signatures predicting response to treatment with dasatinib. PLoS one, Public Library of Science San Francisco, CA USA, v. 10, n. 6, p. e0128542, 2015.

KRIPKA, R. M. L.; KRIPKA, M. Alocação de Salas Objetivando a Minimização de Deslocamentos dos Alunos pelo Campus Central da Universidade de Passo Fundo. In: XXIV Congresso Nacional de Matemática Aplicada (CNMAC). [S.l.: s.n.], 2012.

KUNDU, S.; MANDAL, P. An efficient method of pareto-optimal front generation for analog circuits. Analog Integrated Circuits and Signal Processing, Springer, v. 94, p. 289–316, 2018.

LIEFOOGHE, A.; HUMEAU, J.; MESMOUDI, S.; JOURDAN, L.; TALBI, E.-G. On dominance-based multiobjective local search: design, implementation and experimental analysis on scheduling and traveling salesman problems. Journal of Heuristics, Springer, v. 18, p. 317–352, 2012.

MLADENOVIĆ, N.; HANSEN, P. Variable neighborhood search. Computers & operations research, Elsevier, v. 24, n. 11, p. 1097–1100, 1997.

NASCIMENTO, A. S.; SAMPAIO, R. M.; ALVARENGA, G. B. et al. Uma aplicação de simulated annealing para o problema de alocação de salas. INFOCOMP Journal of Computer Science, v. 4, n. 3, p. 59–66, 2005.

SALKIN, H. M.; KLUYVER, C. A. D. The knapsack problem: a survey. Naval Research Logistics Quarterly, Wiley Online Library, v. 22, n. 1, p. 127–144, 1975.

SCHAERF, A. A survey of automated timetabling. Artificial intelligence review, Springer, v. 13, p. 87–127, 1999.

SCHRIJVER, A. Theory of linear and integer programming. [S.l.]: John Wiley & Sons, 1998.

SOUZA, M. J. F.; MARTINS, A. X.; ARAÚJO, C.; COSTA, F. W. A. Métodos de pesquisa em vizinhança variável aplicados ao problema de alocação de salas. XXII Encontro Nacional de Engenharia de Produção ENEGEP, Fortaleza, Brasil, 2002.

SUBRAMANIAN, A.; MEDEIROS, J. M. F.; CABRAL, L. F.; SOUZA, M. F. Aplicação da metaheurística busca tabu ao problema de alocação de aulas a salas em uma instituição universitária. Revista Produção Online, v. 11, n. 1, p. 54–75, 2011.

ULUNGU, E. L.; TEGHEM, J.; FORTEMPS, P.; TUYTTENS, D. Mosa method: a tool for solving multiobjective combinatorial optimization problems. Journal of multicriteria decision analysis, Wiley Periodicals Inc., v. 8, n. 4, p. 221, 1999.

VANCROONENBURG, W.; WAUTERS, T. Extending the late acceptance metaheuristic for multi-objective optimization. In: Proceedings of the 6th Multidisciplinary International Scheduling conference: Theory & Applications (MISTA2013). [S.l.: s.n.], 2013.

WOLSEY, L. A. Mixed integer programming. Wiley Encyclopedia of Computer Science and Engineering, Wiley Online Library, p. 1–10, 2007.

ZITZLER, E.; DEB, K.; THIELE, L. Comparison of multiobjective evolutionary algorithms: Empirical results (revised version). TIK Report, ETH Zurich, v. 70, 1999.

ZITZLER, E.; DEB, K.; THIELE, L. Comparison of multiobjective evolutionary algorithms: Empirical results. Evolutionary computation, MIT Press, v. 8, n. 2, p. 173–195, 2000.

ZITZLER, E.; THIELE, L. An evolutionary algorithm for multiobjective optimization: The strength pareto approach. TIK report, ETH Zurich, v. 43, 1998.