Estratégias bioinspiradas aplicadas em problemas discretos com muitos objetivos

Tiago Peres França

Universidade Federal de Uberlândia
Faculdade de Computação
Programa de Pós-Graduação em Ciência da Computação

Uberlândia
2018

Tiago Peres França

Estratégias bioinspiradas aplicadas em problemas discretos com muitos objetivos

Dissertação de mestrado apresentada ao Programa de Pós-graduação da Faculdade de Computação da Universidade Federal de Uberlândia como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Ciência da Computação

Orientador: Gina Maira Barbosa de Oliveira
Coorientador: Luiz Gustavo Almeida Martins

Uberlândia
2018

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema de Bibliotecas da UFU, MG, Brasil.

A474m Sobrenome, Nome do aluno, 1979-

2014 Título do Trabalho / Nome e Sobrenome do aluno. - 2014.
81 f. : il.

Orientador: Nome do Orientador.

Dissertação (mestrado) - Universidade Federal de Uberlândia, Programa de Pós-Graduação em Ciência da Computação.
Inclui bibliografia.

1.Computação - Teses. 2. Simulação (Computadores) - Teses. I. Sobrenome, Nome do orientador. II. Universidade Federal de Uberlândia. Programa de Pós-Graduação em Ciência da Computação. III. Título.

CDU: 681.3

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Os abaixo assinados, por meio deste, certificam que leram e recomendam para a Faculdade de Computação a aceitação da dissertação intitulada "**Estratégias bioinspiradas aplicadas em problemas discretos com muitos objetivos**" por **Tiago Peres França** como parte dos requisitos exigidos para a obtenção do título de **Mestre em Ciência da Computação.**

Uberlândia, ___ de _____ de 2018

Orientadora: _____
Prof. Dr. Gina Maira Barbosa de Oliveira
Universidade Federal de Uberlândia

Coorientador: _____
Prof. Dr. Luiz Gustavo Martins de Almeida
Universidade Federal de Uberlândia

Banca Examinadora:

_____
Prof. Dr. Membro da banca 1
Instituição de Ensino Superior

_____
Prof. Dr. Membro da banca 1
Instituição de Ensino Superior

Este trabalho é dedicado
aos meus orientadores Gina Maira e Luiz Gustavo, pela dedicação;
aos meus pais Wagner e Iara, pelo apoio;
e aos meus irmãos e amigos, pelo companheirismo.

Agradecimentos

Agradeço principalmente à minha orientadora Gina Maira pela paciência e pelos ensinamentos durante esta trajetória. Agradeço ao meu co-orientador Luiz Gustavo pelas grandes ideias e correções de texto. Agradeço aos professores José Gustavo e Rita Maria pelas excelentes aulas durante o curso e à professora Márcia Aparecida, que além de ter ministrado suas reconhecidamente ótimas aulas de análise de algoritmos, sempre me incentivou a obter o título de mestre.

Meus agradecimentos também se extendem aos meus pais: Wagner e Iara; meus irmãos: Vinícius, Bruno e Felipe; e aos meus grandes amigos: Lucas, Ana Carolyne, João Marcos, Mariana, Débora e Lívia que sempre me apoiaram e, direta ou indeiretamente, contribuiram muito para a realização deste trabalho.

Agradeço também a Fundação de Amparo à Pesquisa de Minas Gerais (FAPEMIG) que providenciou o apoio financeiro para este trabalho.

"Viver é arriscar tudo. Caso contrário você é apenas um pedaço inerte de moléculas montadas aleatoriamente à deriva onde o universo te sopra."

(Rick and Morty)

Resumo

Problemas de otimização multiobjetivo são muito comuns no dia-a-dia e surgem em diversas áreas do conhecimento. Neste trabalho explora-se e compara-se várias estratégias de otimização multiobjetivo em dois problemas discretos bem conhecidos na computação: o problema da mochila e o problema do roteamento *multicast*. Dentre as estratégias para solucionar problemas multiobjetivos discretos, destacam-se os algoritmos genéticos (AGs) e os algoritmos baseados em colônias de formigas (ACOs), neste trabalho explora-se ambas as abordagens realizando diversos experimentos para 10 algoritmos diferentes. Através do estudo das vantagens e fraquezas de cada método, desenvolveu-se um novo algoritmo denominado *Many-objective Ant Colony Optimization based on Decomposed Pheromone* (MACO/D), o qual foi avaliado e comparado com todos os outros métodos aqui investigados, considerando diferentes métricas de desempenho. A partir dos resultados experimentais, foi possível comprovar a eficiência e eficácia do método proposto. Ele foi capaz de encontrar um conjunto de soluções muito superior aos produzidos pelos AG's no problema da mochila, perdendo apenas para outro algoritmo baseado em formigas, o MOEA/D-ACO. No problema do roteamento, o MACO/D foi consideravelmente mais rápido e obteve resultados muito melhores que os AG's, sendo superado apenas pelo MOACS, outro ACO.

Palavras-chave: algoritmos many-objectives; algoritmos genéticos; otimização por colônia de formigas; problema do roteamento multicast; problema da mochila multiobjetivo.

Abstract

Multi-objective optimization problems are very common in the day-to-day life and come up in many fields of knowledge. In this work, several strategies for multi-objective optimization have been explored and compared in two well known discrete problems in computer science: the knapsack problem and the multicast routing problem. Among all strategies to solve multi-objective discrete problems, genetic algorithms (GAs) and ant colony optimization (ACO) are the ones who generally provide the best results. In this work both approaches are explored through several experiments involving 10 different algorithms. As a consequence of studying the strengths and weaknesses of each method, a new algorithm has been proposed, the Many-objective Ant Colony Optimization based on Decomposed Pheromone (MACO/D), which has been evaluated and compared against all other methods investigated here, considering different performance metrics. It has been capable of finding much superior sets of solutions than the ones yielded by the GAs in the knapsack problem, losing only to another algorithm based on ant colonies, the MOEA/D-ACO. In the routing problem, MACO/D was considerably faster and obtained much better results than any GA, being surpassed only by MOACS, another ACO.

Keywords: many-objective algorithms; genetic algorithms; ant colony optimization; multicast routing problem; multi-objective knapsack problem.

Lista de ilustrações

Lista de tabelas

Lista de siglas

ACO *Ant Colony Optimization*

PSO *Particle Swarm Optimization*

AG Algoritmo Genético

AEMMT Algoritmo Evolutivo Multiobjetivo com Muitas Tabelas

AEMMD Algoritmo Evolutivo Multiobjetivo com Múltiplas Dominâncias

BACO *Binary Ant Colony Optimization*

ER Taxa de Erro

GD *Generational Distance*

HV Hiper-Volume

MOEA/D *Multiobjective Evolutionary Algorithm Based on Decomposition*

MOACS *Multi-Objective Ant Colony Optimization Algorithm*

MOEA/D-ACO *Multiobjective Evolutionary Algorithm Based on Decomposition ACO*

MACO/D *Many-Objective Ant Colony Optimization Based on Decomposed Pheromones*

NSGA *Non-Dominated Sorting Genetic Algorithm*

NSGA-II *Non-Dominated Sorting Genetic Algorithm II*

NSGA-III *Non-Dominated Sorting Genetic Algorithm III*

PMO Problema Multiobjetivo

PM Problema da Mochila

PMM Problema da Mochila Multiobjetivo

PRM Problema do Roteamento Multicast

PS *Pareto Subset*

SPEA *Strength Pareto Evolutionary Algorithm*

SPEA2 *Strength Pareto Evolutionary Algorithm 2*

SPEA2-SDE *SPEA2 with Shift-Based Density Estimation*

SDE *Shift-Based Density Estimation*

VEGA Vector Evaluated Genetic Algorithm

Sumário

Capítulo 1

Introdução

A busca e otimização é um campo de suma importância da ciência da computação que procura encontrar a melhor solução para um problema matemático. Em tais problemas, deve-se maximizar ou minimizar alguma característica, por exemplo, ao decidir um caminho entre duas cidades, é desejável minimizar a distância a se percorrer. Infelizmente, a maior parte dos problemas interessantes são também considerados impossíveis de se resolver em tempo razoável, por essa razão, ao invés de procurar pela melhor solução possível, é usualmente preferível aproximá-la através de algoritmos mais eficientes. A grande importância da pesquisa em busca e otimização está na frequência com a qual esses problemas surgem no dia-a-dia, perguntas como "qual o menor caminho para atingir um destino", "qual o melhor carro para se comprar dado um orçamento" ou "qual a forma mais rápida de se executar uma tarefa" fazem parte da vida da maioria das pessoas e resolvê-las de forma eficiente ainda representa um problema para a computação.

A única forma de garantir a obtenção da melhor solução (solução ótima) para um dado problema é por meio da avaliação de todas as soluções possíveis (busca exaustiva). Para a maioria dos problemas interessantes, essa é uma tarefa difícil e inviável. Por exemplo, combinar e testar todas as possibilidades de caminho em um grafo com muitas arestas é um problema NP-completo, ou seja, não pode ser resolvido em tempo hábil considerando o atual estado da arte da computação. Sendo assim, a pesquisa em busca e otimização se concentra em desenvolver meios para se encontrar soluções suficientemente próximas da ótima, ou seja, trabalha-se com estratégias de aproximação. Dois dos métodos mais comuns para se aproximar soluções são os algoritmos gulosos e os algoritmos de busca bio-inspirados (algoritmos genéticos, otimização por colônia de formigas e inteligência de enxames). Os algoritmos gulosos são interessantes para se resolver problemas de otimização mono-objetivo, mas existem otimizações mais complexas, onde se deve considerar mais de uma característica (problemas multiobjetivos). Por exemplo, ao escolher um carro, não se preocupa apenas com o preço, mas também com o desempenho, a durabilidade, o consumo, entre outros fatores, nesse caso, não basta otimizar um único objetivo, mas encontrar uma opção que apresente uma boa relação custo-benefício considerando

todos os fatores avaliados. A maneira mais simples de se contornar esse problema é transformando as diversas métricas em uma única função através de uma média ponderada dos objetivos. Infelizmente, essa estratégia não é ideal, pois é necessário ter um conhecimento prévio do problema para se decidir a relevância (peso) de cada objetivo. Por essa razão, normalmente deseja-se encontrar todas as soluções que são superiores às demais em pelo menos um dos objetivos. Dessa forma, existirão várias soluções que podem ser classificadas como melhores, algumas terão melhor desempenho em uma métrica enquanto outras se darão melhor em outras, esse conjunto de soluções é denominado fronteira de Pareto. Assim, para resolver os Problemas Multiobjetivos (PMOs), se faz natural a escolha dos algoritmos de busca que retornem múltiplas soluções pertencentes à fronteira de Pareto (SRINIVAS; DEB, 1994).

Muitos problemas da vida real podem tirar proveito da otimização multiobjetivo, fazendo necessária a elaboração de estratégias eficientes para resolvê-los. Portanto, vários algoritmos bio-inspirados foram propostos nos últimos anos (DEB et al., 2002; ZITZLER; LAUMANNS; THIELE, 2002; DEB; JAIN, 2014), tornando a otimização multiobjetivo um campo com alta atividade no ramo da Inteligência Artificial. O primeiro algoritmo evolutivo multiobjetivo (AEMO) proposto foi o Vector Evaluated Genetic Algorithm (VEGA) (SCHAFFER, 1985), mas os métodos mais bem consolidados na literatura são o NSGA-II (DEB et al., 2002) e o SPEA2 (ZITZLER; LAUMANNS; THIELE, 2002), considerados os métodos clássicos da otimização multiobjetivo. Entretanto, foi levantado que, devido a uma pressão seletiva fraca em direção ao conjunto de soluções ótimas em espaços de alta dimensionalidade, tais *frameworks* não funcionam bem quando aplicados a problemas com muitas funções objetivo (4 ou mais) (FRANÇA et al., 2017). Os problemas com 4 ou mais objetivos são geralmente chamados de *many-objective* e para resolvê-los foram propostas novas abordagens como decomposição, e.g. MOEA/D (ZHANG; LI, 2007), relações de dominância diferenciadas, e.g. ε-MOEA (AGUIRRE; TANAKA, 2009) e evolução baseada em indicadores, e.g. HypE (BADER; ZITZLER, 2011). Recentemente, novos algoritmos genéticos foram propostos para resolver a problemática *many-objective*, dentre eles destacam-se: NSGA-III (DEB; JAIN, 2014), MEAMT (BRASIL; DELBEM; SILVA, 2013) e MEANDS (LAFETÁ et al., 2018).

Neste trabalho, investigou-se diversos algoritmos bio-inspirados presentes na literatura multiobjetivo e propôs-se um novo método de otimização baseado em colônias de formigas (ACO), o *Many-Objective Ant Colony Optimization Based on Decomposed Pheromones* (MACO/D), em português, otimização para muitos objetivos com colônia de formigas baseada em decomposição de feromônios. O algoritmo proposto toma algumas ideias do MEANDS (LAFETÁ et al., 2018), transferindo-nas para uma aplicação em ACO, que tem uma abordagem diferente dos algoritmos genéticos tradicionais, usados na maioria dos métodos de otimização multiobjetivo. De acordo com os experimentos realizados no capítulo 8, o MACO/D mostrou-se ser um método interessante devido a sua rapidez no

problema do roteamento multicast e eficácia no problema do mochila. A fim de comparar os PMOs, utilizou-se dois problemas discretos bem conhecidos na literatura de busca e otimização multiobjetivo: o Problema da Mochila Multiobjetivo (PMM) e o Problema do Roteamento Multicast (PRM). O primeiro é uma versão multiobjetivo do clássico problema da mochila 0/1. O problema da mochila original consiste de uma mochila e um conjunto de itens, deve-se encontrar a melhor combinação de objetos para se colocar na mochila de forma que não se ultrapasse a capacidade da mesma e que se maximize o valor (lucro) dos itens carregados. O PMM é uma variação onde ao invés de um único valor de lucro, todo item possui m valores (m é o número de objetivos). O PMM é uma abordagem teórica e apesar de testar os algoritmos em seus extremos, devido a seu enorme espaço de busca, nem sempre reflete a realidade dos problemas cotidianos. O PRM, por sua vez, é um problema prático geralmente encontrado em comunicações de rede no qual se deseja transmitir uma mensagem de um dispositivo fonte para múltiplos destinos em uma rede de computadores com o objetivo de utilizar os recursos disponíveis da forma mais eficiente possível. Esse algoritmo é de extrema importância considerando o grande número de transmissões multimídia e aplicações em tempo real que se beneficiariam de algoritmos eficientes para cálculos de rota.

Em cada problema, os vários AEMOs são avaliados em diversas formulações de objetivos, variando de 2 a 6 funções, analisando a qualidade das soluções obtidas e discutindo as características presentes nos algoritmos que propicia a obtenção de determinado resultado.

A maior parte dos trabalhos em busca e otimização multiobjetivo utilizam Algoritmos Genéticos (AGs). Entretanto, outras técnicas bio-inspiradas também podem ser empregadas, tais como: os métodos baseados em colônias de formigas, em inglês *Ant Colony Optimization* (ACO), e os algoritmos inspirados em inteligência de enxame, em inglês *Particle Swarm Optimization* (PSO). Devido a seus cálculos vetoriais difíceis de se traduzir para um ambiente discreto, os PSOs são indicados especialmente para problemas contínuos e, portanto, foram deixados de fora deste estudo. Por outro lado, os ACOs lidam especialmente com problemas discretos e representações em grafos, tornando-os interessantes para os problemas estudados (PMM e PRM).

Os ACOs são pouco explorados na literatura multiobjetivo quando comparados aos algoritmos genéticos, este trabalho expande a coleção de métodos inspirados em colônias de formigas ao propor um novo *framework* ACO para problemas multiobjetivos com muitas funções de otimização (*many-objective*). Os ACOs são heurísticas criadas especialmente para problemas discretos, o que pode representar uma grande vantagem em relação aos AGs, tanto em matéria de tempo como qualidade das soluções. O MACO/D utiliza várias tabelas de feromônios para guiar a população de formigas, uma para cada combinação possível de objetivos. Cada grupo de formigas é associada a uma estrutura de feromônios diferente e, a cada passo do algoritmo, a atualiza de acordo com as novas soluções ótimas encontradas. Cada tabela de feromônio representa um subproblema e

é responsável por guiar as soluções de acordo com um conjunto de objetivos diferentes, dessa forma, o MACO/D começa explorando subproblemas mais simples passa para os mais complexos no decorrer do algoritmo. Para o PRM, os experimentos deste trabalho revelaram que o MACO/D, junto ao MOACS são os algoritmos mais rápidos e eficazes. Para o PRM, o MACO/D e MOEA/D-ACO conquistam as melhores soluções. Ou seja, o MACO/D é um método interessante para ambos os problemas, e dentre os ACOs testados é que apresentou maior estabilidade considerando ambos os problemas e suas diferentes instâncias.

1.1 Objetivos

Esta dissertação tem como objetivo estender os trabalhos de (LAFETÁ et al., 2016) e (BUENO; OLIVEIRA, 2010) sobre o problema do roteamento multicast, introduzindo um novo tipo de heurística bio-inspirada: as colônias de formigas, e um novo problema discreto: o problema da mochila multiobjetivo. O principal objetivo deste trabalho é propor um novo algoritmo ACO para otimização em problemas *many-objectives* e compará-lo com as principais estratégias presentes na literatura multiobjetivo tanto de AGs quanto de ACOs. Em linhas gerais, esta pesquisa almeja:

- Adotar um novo problema multiobjetivo discreto, diferente do roteamento multicast estudado até então, a fim de melhor entender a influência do tipo do problema no desempenho/eficiência dos algoritmos estudados. O novo problema introduzido neste trabalho é o PMM que, apesar de ter uma aplicação mais restrita, apresenta diferenças interessantes em relação ao PRM, possibilitando uma análise mais profunda sobre comportamento dos vários algoritmos.

- Empregar uma métrica não paramétrica, neste caso o hipervolume, que permita avaliar o desempenho dos algoritmos multi e *many-objective* em problemas nos quais a fronteira de Pareto não é conhecida. Essa nova métrica, até então inexplorada pelo nosso grupo de pesquisa, permitiu comprovar a vantagem dos ACOs (inclusive do método proposto, MACO/D) sobre os AGs em instâncias complexas do PMM e do PRM, onde não é possível pré-calcular as fronteiras de Pareto.

- Analisar em ambos os problemas (PMM e PRM) o comportamento de cada algoritmo em relação à complexidade do espaço de busca e ao número de objetivos a fim de guiar as decisões sobre a construção do algoritmo proposto MACO/D.

- Propor um modelo para a construção de soluções em algoritmos baseados em colônias de formigas, de acordo com as características de cada um dos problemas investigados (PMM e PRM). Tais modelos são partes essenciais para a proposição do algoritmo ACO.

- ❏ Propor o novo algoritmo baseado em ACO: O MACO/D.

1.2 Contribuições

Este trabalho contribui para o campo de busca e otimização multiobjetivo, assim como o de comunicações em rede (através do problema do roteamento multicast). os principais resultados desta pesquisa são resumidos nos seguintes tópicos:

- ❏ O Algoritmo Evolutivo Multiobjetivo com Muitas Tabelas (AEMMT) foi proposto originalmente para sequenciamento de proteínas (BRASIL; DELBEM; SILVA, 2013) e posteriormente, utilizado em (LAFETÁ et al., 2016) para resolver o problema do roteamento multicast. Neste trabalho, explora-se uma nova aplicação do algoritmo ao utilizá-lo para encontrar soluções para uma versão multiobjetivo do problema da mochila.

- ❏ O Algoritmo Evolutivo Multiobjetivo com Múltiplas Dominâncias (AEMMD) foi proposto e analisado em (LAFETÁ et al., 2016) para resolver o PRM. Sua eficácia como algoritmo multiobjetivo, separado do problema do roteamento, ainda não havia sido comprovada. Neste trabalho, mostra-se como é possível aplicar o AEMMD também ao PMM.

- ❏ Através da execução de vários algoritmos multi e *many-objective* sobre 2 problemas diferentes em diversos níveis de complexidade, foi feita uma análise sobre o comportamento desses algoritmos à medida em que se aumenta o número de objetivos e a complexidade do espaço de busca, possibilitando uma comparação entre os principais algoritmos da literatura.

- ❏ A principal contribuição desta dissertação está na proposição de um novo algoritmo denominado MACO/D. O MACO/D é uma estratégia baseada em colônia de formigas e decomposição de objetivos que resolve problemas com muitos objetivos de forma rápida e eficiente. O novo método foi aplicado aos problemas PMM e PRM e comparado com os demais algoritmos. Durante o processo de desenvolvimento desse algoritmo, diferentes estratégias para a construção de soluções em ambos os problemas (PMM e PRM) foram avaliadas e diversos experimentos foram realizados, os quais serão apresentados no decorrer do texto.

1.3 Organização do texto

Este trabalho está dividido em capítulos organizados da seguinte forma:

- ❏ Capítulo 2, otimização bio-inspirada: neste capítulo apresenta-se uma visão geral sobre os algoritmos genéticos e a otimização por colônia de formigas;

- Capítulo 3, otimização multiobjetivo: apresenta-se a definição de problemas multiobjetivos e descreve-se cada algoritmo usado neste trabalho;
- Capítulo 4, problemas de teste: introduz os dois problemas explorados nesta dissertação, o Problema da Mochila Multiobjetivo (PMM) e o Problema do Roteamento Multicast (PRM);
- Capítulo 5, estratégias evolutivas para o PMM: apresenta os modelos baseados em algoritmos genéticos e colônia de formigas, os quais foram utilizados para resolver o PMM;
- Capítulo 6, estratégias evolutivas para o PRM: apresenta os modelos baseados em algoritmos genéticos e colônia de formigas, os quais foram utilizados para resolver o PRM;
- Capítulo 7, algoritmo proposto: descreve o algoritmo MACO/D proposto nesta dissertação;
- Capítulo 8, experimentos: apresenta e discute todos os experimentos realizados no decorrer deste trabalho com o objetivo de avaliar a eficiência do algoritmo proposto, bem como analisar comparativamente o desempenho das abordagens investigadas na resolução de diferentes configurações de dois problemas discretos distintos;
- Capítulo 9, conclusão: resume as principais conclusões retiradas a partir dos experimentos e apresenta ideias para trabalhos futuros.

Capítulo 2

Otimização bio-inspirada

Grande parte dos problemas de otimização envolvem encontrar a melhor opção num conjunto de possibilidades que cresce de maneira exponencial (KANN, 1992). Tais problemas são impossíveis de serem resolvidos de modo satisfatório com a tecnologia atual e necessitam de estratégias inteligentes para se aproximarem da solução ótima em tempo hábil. Dentre essas estratégias, destacam-se os algoritmos gulosos e os algoritmos de busca bio-inspirados.

Os algoritmos gulosos (CORMEN LEISERSON; STEIN, 2009) são estratégias relativamente simples que, apesar de nem sempre obterem a solução ótima, normalmente aproximam-se bem dela. Tais métodos são geralmente utilizados quando apenas um critério é envolvido na otimização. Quando mais de um objetivo deve ser analisado, o problema fica bem mais complexo e essa abordagem deixa de ser indicada.

A otimização bio-inspirada (RAI; TYAGI, 2013) lança mão de estratégias baseadas na natureza para se encontrar boas soluções de forma eficiente. Assim como os algoritmos gulosos, não é garantida a obtenção da solução ótima, mas com uma boa modelagem do problema, é possível encontrar soluções suficientemente próximas. Na natureza, o processo de obter uma boa solução é usado a todo momento, desde a forma como as espécies evoluem, até a maneira como uma simples formiga encontra o caminho mais curto entre a colônia e uma fonte de comida. Ao observar processos comuns da natureza, estudiosos encontraram maneiras simples e eficientes de se resolver diversos problemas de otimização.

Os principais métodos bio-inspirados na literatura de otimização são os Algoritmos Genéticos (AGs), as colônias de formigas (ACO) e os enxames de partículas (PSO). Os algoritmos genéticos se inspiram na teoria da evolução de Darwin (CHARLES, 1859). Na evolução das espécies, cada indivíduo possui um material genético que é cruzado com os genes de outro representante da espécie para gerar um novo indivíduo. Durante tal cruzamento, alguns indivíduos sofrem mutações aleatórias que podem alterar parte de sua carga genética e, com isso, suas características (fenótipo). O ambiente determina a parte da população que sobrevive e a parte que perece. Os indivíduos sobreviventes, ou

seja, aqueles bem adaptados ao ambiente, possuem maiores chances de se reproduzir e espalhar suas boas características genéticas. Desta forma, a evolução das espécies nada mais é que um processo otimizador, no qual indivíduos são gerados e os melhores são selecionados. Esse processo é repetido ao longo das gerações, até que se obtenha indivíduos bem adaptados ao ambiente em questão. Os ACOs são inspirados no comportamento das formigas, organismos simples que quando analisados em conjunto (colônia) apresentam comportamento complexo (DORIGO; MANIEZZO; COLORNI, 1996). O interesse nas formigas vem da observação de que, ao buscarem por comida, acabam por encontrar o caminho mais rápido entre a fonte de alimento e o formigueiro. Como podem seres tão simples resolverem eficientemente um problema de otimização? Estudos revelaram que as formigas se baseiam no depósito de feromônios para se guiarem. Quanto mais forte o feromônio em um caminho, maior as chances dele ser utilizado pelas formigas. Tal comportamento serviu de inspiração para os algoritmos de otimização bio-inspirados, dando origem ao ACO. Os algoritmos baseados em inteligência de enxames (PSOs), assim como os ACOs são inspirados no comportamento emergente de populações de animais simples. Os PSOs se inspira na navegação de pássaros, onde cada elemento da formação se guia através dos pássaros à frente. O algoritmo se baseia na direção e velocidade de cada elemento do enxame, determinando a exploração do espaço de busca através de operações vetoriais. Portanto é uma estratégia indicada para problemas contínuos, não sendo indicada para os problemas explorados nesta dissertação.

Em geral, todo algoritmo de otimização bio-inspirado inicia sua busca por meio da geração aleatória de soluções. Após essa geração inicial, inicia-se uma etapa recorrente, na qual as melhores soluções guiam a construção de novas soluções que serão submetidas ao mesmo processo até que uma condição de parada seja atingida. Como não é necessário gerar todas as soluções possíveis, são métodos eficazes que, quando bem modelados, encontram um conjunto de boas soluções que resolvem o problema. Uma das principais diferenças entre os algoritmos bio-inspirados e as demais estratégias de otimização é o fato de que os primeiros produzem um conjunto de soluções aproximadas até o último passo do processo, quando, retorna a solução melhor avaliada. Essa é uma característica interessante para a otimização multiobjetivo, pois nela poder-se-ia retornar todas as soluções não-dominadas ao invés de apenas uma.

2.1 Algoritmos Genéticos

Os algoritmos genéticos (AGs) são métodos de busca baseados na teoria da evolução de Charles Darwin (CHARLES, 1859). A teoria de Darwin, hoje já endossada por diversas observações no campo da biologia, parte do princípio de que os organismos se adaptam ao ambiente em que vivem através de mutação, cruzamento e seleção natural. Mudanças nos genes (genótipo) de um indivíduo da população afetam suas característi-

cas genéticas (fenótipo) e podem ajudá-lo a sobreviver em seu habitat ou atrapalhá-lo. Os indivíduos com mutações favoráveis têm maiores chances de sobreviver e reproduzir, passando suas características para a geração seguinte. Dessa forma, ao longo de milhões de anos, organismos simples se tornam complexos e extremamente adaptados ao meio.

A ideia dos algoritmos genéticos é aplicar o mesmo conceito da evolução natural na computação. O algoritmo parte de um conjunto de soluções aleatórias (população inicial) e, após várias iterações de mutação, cruzamento (ou *crossover*) e seleção, obtém um conjunto de soluções (população final) que resolvem bem o problema. Nesse processo, o indivíduo na população representa uma solução, o meio representa o problema e as operações de mutação e cruzamento devem ser definidas, respectivamente, de forma a permitir uma alteração aleatória em uma solução e a combinação de duas soluções.
Conforme ilustrado na Figura 1, o funcionamento básico de um AG inicia-se com a geração da população inicial. Nessa etapa, os indivíduos da população são normalmente gerados de forma aleatória e, em seguida, avaliados quanto a sua aptidão ao meio. No caso de um problema de otimização para uma função objetivo f, a aptidão de um indivíduo x é dada por $f(x)$. Após esse processo de inicialização, parte-se para a evolução das gerações que consiste na repetição das seguintes etapas:

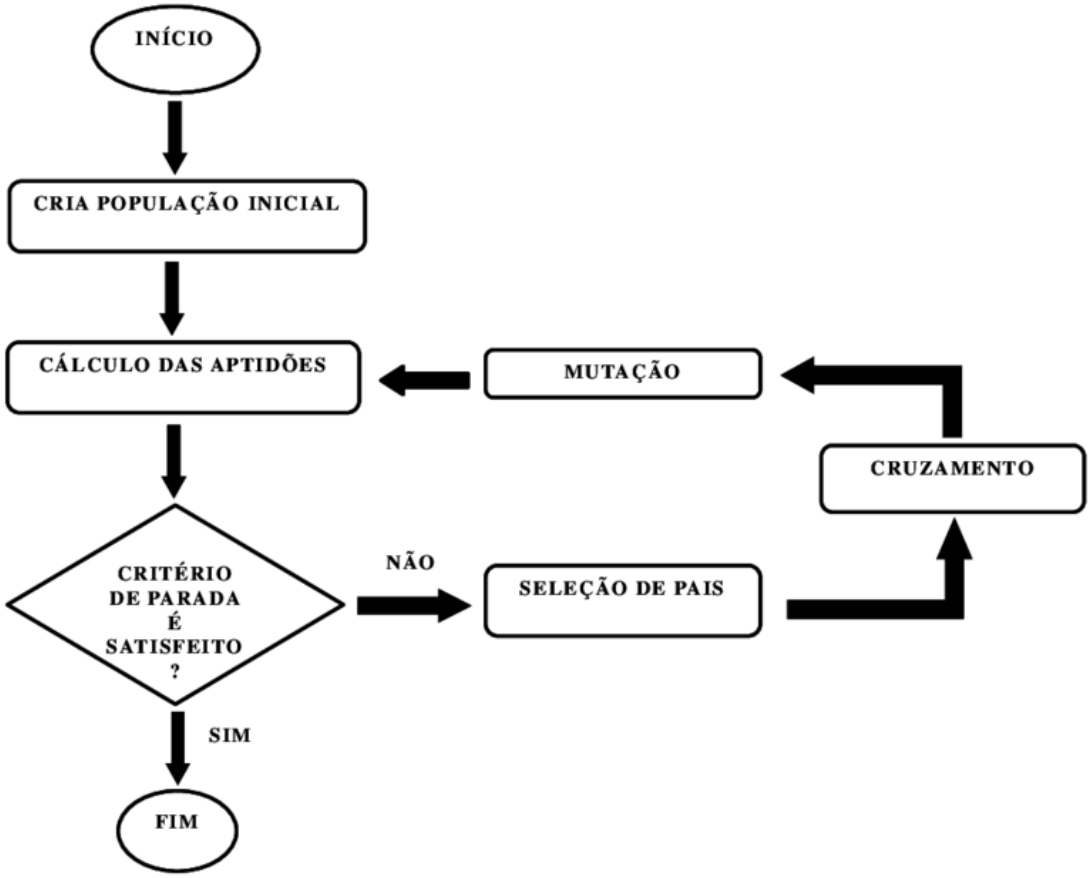


Figura 1 – Fluxograma de um AG. Retirado de (NRIA et al., 2013)

1. Sortear os pares de pais;
2. Aplicar o cruzamento em cada par e gerar os filhos;
3. Aplicar a mutação aos filhos de acordo com uma taxa de mutação pré-estabelecida;
4. Avaliar os filhos;
5. Selecionar os melhores entre pais e filhos para formar a população da iteração seguinte (elitismo).

A evolução das gerações do AG termina quando uma condição de término estabelecida pelo usuário é atingida, por exemplo, encontrar a solução ótima do problema (quando conhecida) ou atingir um número máximo de gerações. Cada etapa de execução do AG é descrita com mais detalhes a seguir.

2.1.1 Representação do indivíduo

A principal dificuldade ao se elaborar um algoritmo genético é definir a representação do indivíduo. Cada indivíduo representa uma possível solução para o problema investigado e deve ser codificado em uma estrutura que possibilite e/ou favoreça a realização das operações genéticas de mutação e cruzamento. Na proposição original do algoritmo (GOLDBERG, 1989), o indivíduo é representado de forma binária, ou seja, a solução para o problema é codificada em uma cadeia de bits, na qual cada posição pode facilmente ser invertida (mutação) ou copiada de um cromossomo para outro (cruzamento).

No problema da mochila 0/1, por exemplo, existe um conjunto de itens I com pesos e valores e uma mochila com capacidade limitada. Deve-se descobrir qual a melhor forma de se arranjar os itens de maneira que a soma dos valores de cada um seja máxima e que a capacidade da mochila não seja excedida. Para representar uma solução deste problema em um AG, basta assumir um vetor binário de tamanho igual a $|I|$, onde diz-se que o item está na mochila se sua posição correspondente no vetor binário é 1 e não está, caso contrário.

Outras representações de indivíduos também são possíveis, mas apresentam novos desafios, por exemplo em problemas de menores caminhos, por exemplo, normalmente trabalha-se com grafos. Logo, um indivíduo é um grafo e tanto a mutação quanto o cruzamento deverão ser operações em grafos.

Para elaborar a representação do indivíduo no AG, deve-se levar em consideração a facilidade de manipulação da estrutura, a possibilidade de se introduzir um fator aleatório (mutação) e, principalmente, a representação das características de ambos os pais nos filhos. Se a estrutura não permite a herança de características, não é uma boa escolha para se utilizar em um algoritmo genético. Além disso, é preciso garantir a unicidade

da forma de representação, ou seja, cada solução pode ser representada de uma única maneira e cada indivíduo deve representar uma única solução (relação um-para-um).

2.1.2 Seleção de pais

A estratégia para selecionar os pais que contribuirão para a composição genética da geração seguinte do AG deve ser escolhida de acordo com a pretensão do algoritmo. Dependendo do problema, pode ser mais desejável uma convergência rápida que uma exploração mais profunda do espaço de busca, por exemplo. Normalmente, três métodos são considerados para se fazer a seleção:

1. Seleção elitista: os pares de indivíduos com as melhores aptidões são escolhidos como pais. Dessa forma, não há chance de indivíduos ruins propagarem suas características, portanto, a população converge para indivíduos iguais (ou muito parecidos) rapidamente, sem explorar partes do espaço de busca aparentemente ruins. Isso pode prejudicar o algoritmo em problemas com muitos mínimos locais, onde o fato de existir uma solução ruim não quer dizer que as soluções próximas também o são.

2. Roleta: é um método menos rigoroso que o anterior. Cada indivíduo recebe uma probabilidade de ser selecionado de acordo com sua aptidão. Os indivíduos com maior aptidão receberão probabilidade alta e dificilmente não serão selecionados, enquanto indivíduos muito ruins raramente se tornarão pais. Apesar disso, toda solução tem chance de ser escolhida, o que melhora a exploração do espaço de busca em relação a estratégia anterior, mas diminui a velocidade de convergência. Esta estratégia é um meio-termo entre a seleção elitista e a seleção por torneio, explicada a seguir.

3. Seleção por torneio: esta é a estratégia com a menor pressão seletiva, ou seja, que dá a maior chance aos indivíduos ruins de se tornarem pais e propagarem suas características. O torneio consiste em selecionar dois indivíduos aleatoriamente e escolher como primeiro pai aquele com a melhor aptidão, da mesma forma, sorteia-se dois outros indivíduos na população, diferentes do primeiro pai, e define-se como segundo pai a solução com melhor aptidão. O torneio básico é o torneio de dois, mas para aumentar a pressão seletiva, é possível realizar o torneio com um maior número de representantes da população. Dentre as três estratégias, esta é a que melhor explora o espaço de busca. Por outro lado, é possível que, ao explorar demasiadamente o espaço, o AG não consiga convergir.

2.1.3 Operadores genéticos

Nos algoritmos genéticos destacam-se três operações principais: cruzamento, mutação e reinserção. O cruzamento e a mutação estão diretamente ligados com a forma de repre-

sentação do indivíduo, enquanto a seleção opera sobre a aptidão (função de avaliação ou *fitness*) de cada indivíduo da população.

Num algoritmo genético, cada iteração do laço principal é chamada de geração. No início de cada geração, sorteia-se pares de pais de acordo com suas aptidões para que seja gerada uma nova população de filhos. A quantidade de pares de pais sorteados é determinada pela taxa de crossover, a qual é um parâmetro de configuração do AG. Cada par é composto de duas cadeias genéticas (cromossomos), uma correspondente a cada indivíduo do par. Para gerar o filho, cruza-se os dois cromossomos a fim de se obter dois novos indivíduos que compartilhem características de ambos genitores. Esse processo é conhecido como cruzamento, ou *crossover*. Existem várias maneiras de se cruzar cadeias genéticas. A escolha depende principalmente da estrutura de dados usada para representar o cromossomo. A estrutura mais comum é a cadeia binária (GOLDBERG, 1989), onde uma solução é codificada em uma cadeia de 0's e 1's. Nesse caso, a forma mais simples de efetuar o cruzamento é gerar duas novas cadeias de bits (filhos), onde parte do material genético (posições na cadeia) pertence a um pai e o restante do material é proveniente do outro. Os filhos gerados em um cruzamento adotam materiais genéticos complementares de cada pai. Por exemplo, se o filho 1 herda os n primeiros genes do pai 1 e o restante do pai 2, o filho 2 herdará os n primeiros genes do pai 2 e o restante do pai 1. A Figura 2 ilustra este tipo de cruzamento. Existem diversas formas de se combinar duas cadeias de bits, dentre elas podem-se destacar:

- Ponto de cruzamento único: uma posição i de uma das cadeias é sorteada. O filho 1 herda os i primeiros genes do pai 1 e restante do pai 2, enquanto que o filho 2 adota o material genético complementar, como ilustrado na Figura 2.
- Dois pontos de cruzamento: ao invés de se utilizar apenas um ponto para dividir o material genético, este método divide a cadeia binária em três partes. Nesse caso, um filho herda a primeira e terceira partes de um pai e a segunda do outro.
- Cruzamento uniforme: cada bit do filho é obtido de forma aleatória, pode vir tanto do pai 1 quanto do pai 2. Em termos de implementação, sorteia-se uma máscara binária e para cada bit 0 da máscara, copia-se o gene do pai 1 para o filho. Para cada bit 1, copia-se o gene do pai 2.
- Cruzamento aritmético: realiza-se uma operação binária (ex: AND, OR, XOR, etc.) entre os cromossomos do pai 1 e do pai 2.

A fim de melhorar a diversidade entre as soluções avaliadas, uma pertubação nos genes dos indivíduos é gerada durante a busca, possibilitando a exploração de elementos ausentes na população atual. Para isso, após gerar o cromossomo de cada filho, é preciso permitir que ocorra uma mutação, ou seja, uma mudança aleatória no material genético. A chance de uma mutação ocorrer depende de um parâmetro do AG chamado de "taxa

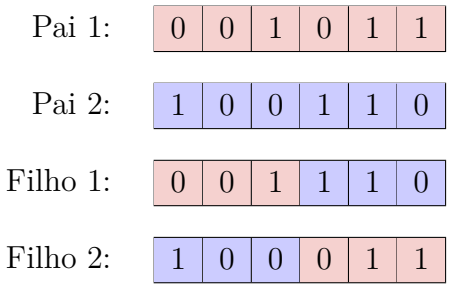


Figura 2 – Exemplo de *crossover* com cruzamento de ponto único, onde o ponto de cruzamento está na metade do material genético.

de mutação". O processo de alteração genética aleatória depende da representação do indivíduo, no caso de uma cadeia binária, por exemplo, é simples, basta sortear um bit e invertê-lo. Em árvores, a mutação pode consistir na eliminação de algum vértice e na reconexão aleatória. Em grafos ponderados, alterar os pesos de uma aresta pode ser uma boa ideia. A definição do processo de mutação dependerá do problema e pode ser feita de várias maneiras diferentes, desde que produza uma pequena diferença no indivíduo e que ele continue representando uma solução válida.

O cruzamento normalmente gera um ou dois filhos para cada pai. Em todos os métodos descritos anteriormente é possível obter um segundo filho com o material genético não utilizado. Independente do número de filhos gerados, após o processo de *crossover*, a população será maior que o limite máximo permitido, exigindo a eliminação de alguns indivíduos (seleção natural). Diversas estratégias podem ser aplicadas nesse processo de seleção, também conhecido como reinserção. Por exemplo, a seleção natural sem elitismo, simplesmente elimina a população mais velha (pais). A seleção natural com elitismo mantém um percentual da população de pais (elite), completando-a com os melhores filhos. Outro tipo de elitismo é a seleção dos melhores indivíduos. Essa estratégia é normalmente mais interessante, pois permite a sobrevivência dos indivíduos mais aptos considerando a totalidade da população, sendo assim, avalia-se todos os pais e filhos e mantém-se aqueles com melhor aptidão. A aptidão de cada indivíduo é dada por seu desempenho relativo à função de avaliação (*fitness*).

2.2 Otimização por Colônia de formigas

A otimização por colônia de formigas (ACO), proposto em (DORIGO; MANIEZZO; COLORNI, 1996), é um modelo de busca bio-inspirado que parte da ideia de que estruturas simples, com alguma espécie de comunicação, podem gerar um comportamento complexo quando operam em conjunto, de forma cooperativa. A inspiração do ACO é o forrageamento em uma colônia de formigas. Na natureza, observa-se que as formigas, mesmo sendo seres vivos simples, conseguem encontrar o melhor caminho entre o formi-

gueiro e a fonte de comida. A partir do estudo desse comportamento, descobriu-se que tal faceta é possível através de uma comunicação indireta entre os animais. Ao fazer um caminho, as formigas depositam uma substância chamada feromônio, a qual pode ser sentida por outros membros da espécie. Uma formiga ao decidir qual caminho percorrer, tem maior chance de escolher aquele com a maior quantidade de feromônios. Além disso, a substância evapora com o tempo. Dessa forma, quanto menor o caminho, maior a frequência com a qual as formigas depositarão feromônio e, portanto, maior será a chance de ser escolhido, criando um ciclo vicioso.

Ao trazer o conceito de colônia de formigas para a computação, observa-se um grande potencial para se resolver problemas em grafo. Por exemplo, para descobrir o menor caminho entre os vértices A e B em um grafo não ponderado G, basta simular várias formigas que partem de A e chegam em B, fazendo um caminho baseado na quantidade de feromônios das arestas. Ao final de cada iteração, atualiza-se o valor do feromônio em cada aresta de acordo com a evaporação e com as arestas percorridas pelas formigas. Dessa forma, espera-se que, após várias iterações, a quantidade de feromônios seja suficiente para guiar uma formiga pelo melhor caminho entre A e B. O fluxograma de um ACO é mostrado na Figura 3.

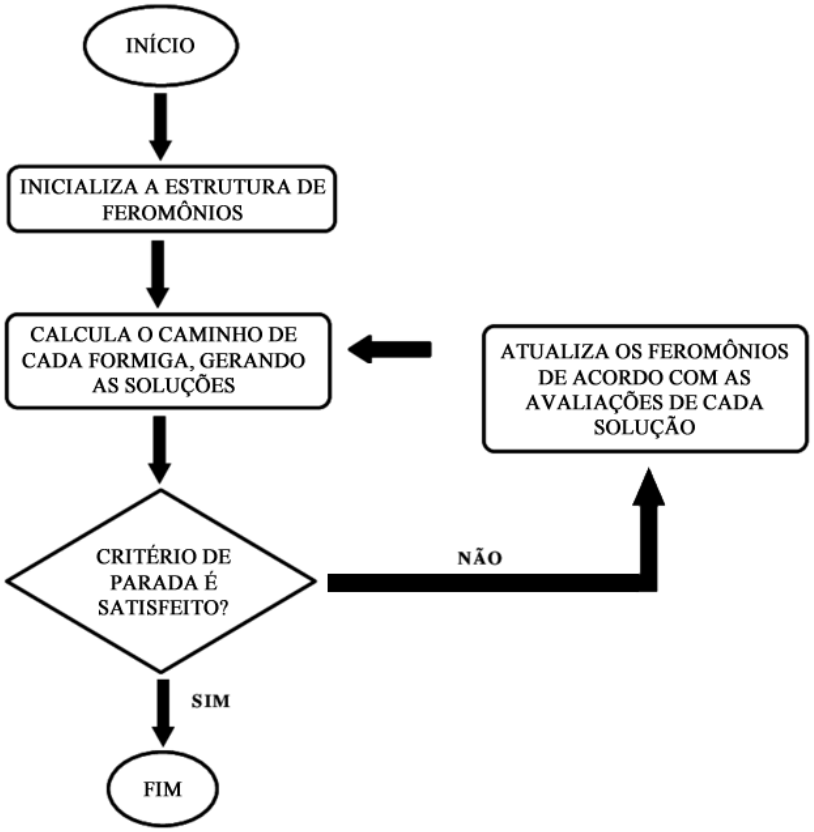


Figura 3 – Fluxograma de um ACO.

O algoritmo inicia com a limpeza dos feromônios do grafo, ou seja, é atribuído zero

para a quantidade de feromônios depositados nas arestas do grafo. Após essa limpeza, inicia-se o processo iterativo do algoritmo, onde a cada iteração, as formigas na colônia percorrem o caminho do vértice de partida (formigueiro) ao nó destino (fonte de comida). Dependendo da formulação do problema, o caminho de volta também pode ser realizado. Ao final de cada iteração, atualiza-se os feromônios em cada aresta. O algoritmo termina quando uma condição de parada pré-determinada é atingida, normalmente um número máximo de iterações. As etapas principais da execução de um ACO são descritas a seguir.

2.2.1 Representação da solução

No ACO, a solução é representada pelo caminho feito pela formiga no grafo, normalmente uma lista de vértices, uma árvore ou um subgrafo do grafo de entrada. Por exemplo, no problema de encontrar o menor caminho, a solução pode ser uma lista de vértices que representa o percurso desse caminho. Caso seja necessário encontrar caminhos entre a raiz e diversos destinos, pode-se representar a solução como uma árvore. Nem sempre é claro decidir a codificação de uma solução. No problema da mochila, por exemplo, não é trivial a visualização de um grafo no processo da construção da solução. Nesse caso, uma possível representação é a associação de feromônios diretamente aos itens, através de um vetor de feromônios ao invés de uma matriz.

2.2.2 Construção da solução

Independente da representação escolhida, deve ser possível relacionar cada parte da solução com uma quantidade de feromônios na estrutura principal. Por exemplo, no caso de grafos, as arestas escolhidas para montar a solução devem ter seus feromônios incrementados a fim de guiar as próximas formigas. Em cada época (iteração do algoritmo) um número pré-determinado de soluções é construído, onde cada formiga decide quais partes serão adotadas em sua solução, com base nos respectivos valores de feromônios.

Além dos feromônios, as formigas ainda utilizam as informações de heurística para decidir o próximo passo. Uma heurística é uma função que estima a qualidade do caminho e normalmente representa o peso de uma aresta. Estando em um vértice i de um grafo G, uma formiga tem probabilidade $p(i, j)$ de escolher a aresta que leva ao vértice adjacente j. Essa probabilidade é calculada pela seguinte equação:

$$p(i,j) = \frac{\tau_{i,j}^{\alpha} * \eta_{i,j}^{\beta}}{\sum_{v \in adj(i)} \tau_{i,v}^{\alpha} * \eta_{i,v}^{\beta}}$$

Sendo:

- $\tau_{i,j}^{\alpha}$: feromônio na aresta (i, j) elevado à constante α, que representa a importância atribuída ao valor do feromônio. Representa o conhecimento sobre o ambiente (busca global).

- $\eta_{i,j}^{\beta}$: heurística da aresta (i, j) elevada à constante β, que representa a importância atribuída à heurística. A heurística de uma aresta é dada em função do peso, como, por exemplo, $1/peso$, normalmente usado em problemas de minimização. Representa a visibilidade da formiga (busca local).

- $adj(i)$: são todos os vértices adjacentes a i, ou seja, todo vértice em G para o qual é possível construir um caminho a partir de i com apenas uma aresta.

O número de iterações (épocas), a quantidade de soluções geradas por iteração e as constantes alfa e beta são parâmetros de configuração de um algoritmo ACO. De forma geral, dado um grafo G e um nó inicial, o processo de construção da solução sempre verifica todos os movimentos possíveis para a formiga, tomando sua decisão de acordo com os feromônios e as heurísticas de cada uma das possibilidades.

2.2.3 Atualização dos feromônios

Existem duas ocasiões onde o feromônio de uma aresta pode ser atualizado: no momento em que a formiga passa pela aresta e no fim de cada iteração. A maioria das implementações considera apenas o segundo caso, pois assim, é possível avaliar as soluções e incrementar os feromônios de acordo com os desempenhos obtidos. Ao fim de cada época, a quantidade de feromônio existente na aresta que liga os vértices i e j ($\tau_{i,j}$) é atualizada por:

$$\tau_{i,j} = (1 - \rho) * \tau_{i,j} + \sum_{k \in formigas} \Delta\tau_{i,j}(k)$$

Sendo:

- ρ: parâmetro de configuração do ACO que corresponde ao coeficiente de evaporação. Determina o quão rápido o feromônio deve desaparecer das arestas após depositado.

- $formigas$: conjunto de todas as formigas na iteração.

- $\Delta\tau_{i,j}(k)$: quantidade de feromônio depositada pela formiga k na aresta entre os vértices i e j.

A quantidade de feromônio depositada por uma formiga k em uma aresta entre os vértices i e j é dada por:

$$\Delta\tau_{i,j}(k) = \begin{cases} \frac{Q}{L_k}, & \text{se } x \geq 1 \\ 0, & \text{caso contrário} \end{cases}$$

Sendo:

- Q: Quantidade máxima de feromônio que pode ser depositada por uma formiga.

- L_k: Custo da solução gerada pela formiga k.

Considerando essas características, nossa pesquisa propõe uma nova versão de ACO para tratar problemas discretos com enfoque em otimização multiobjetivo.

Capítulo 3

Otimização multiobjetivo

A otimização multiobjetivo consiste em selecionar as melhores soluções de acordo com múltiplos critérios ao invés de apenas um. Por exemplo, ao estabelecer um melhor caminho entre duas cidades pode-se não estar interessado apenas na menor distância, mas também no tráfego, segurança das vias, quantidade de pedágios, etc. Na otimização com um único objetivo, a comparação entre soluções é relativamente simples. Para que uma solução seja considerada melhor que a outra, basta que ela tenha uma melhor avaliação segundo o critério/métrica considerada. Por outro lado, quando se trabalha com mais de uma função de otimização, é preciso usar um critério de ponderação, combinando as funções em um único objetivo, como, por exemplo, o algoritmo VEGA (SCHAFFER, 1985); ou adotar uma abordagem que trabalhe com o conceito de dominância de Pareto, como os algoritmos NSGA-II (DEB et al., 2002) e SPEA2 (ZITZLER; LAUMANNS; THIELE, 2002).

A dominância de Pareto diz que uma solução A é melhor que uma solução B, ou A domina B ($A \prec B$), se, e somente se:

- A é melhor avaliado que B em pelo menos um dos objetivos;
- A não tem avaliação pior que B em nenhum dos objetivos.

Considerando um problema de minimização e F como o conjunto de funções objetivo, tem-se, matematicamente:

$$A \prec B \Leftrightarrow (\forall (f \in F) f(A) \leq f(B)) \wedge (\exists (f \in F) f(A) < f(B))$$

Em problemas de otimização multiobjetivo, o interesse está em encontrar a fronteira de Pareto, ou seja, o conjunto de todas as soluções que não são dominadas por nenhuma outra. Graficamente, a fronteira de Pareto representa a linha formada pelas soluções não-dominadas existentes para o problema. Na Figura 4 apresenta-se um exemplo da fronteira de Pareto para um problema de minimização com dois objetivos ($F1$ e $F2$). Nesse exemplo, os pontos pertencentes à fronteira de Pareto estão destacados em vermelho.

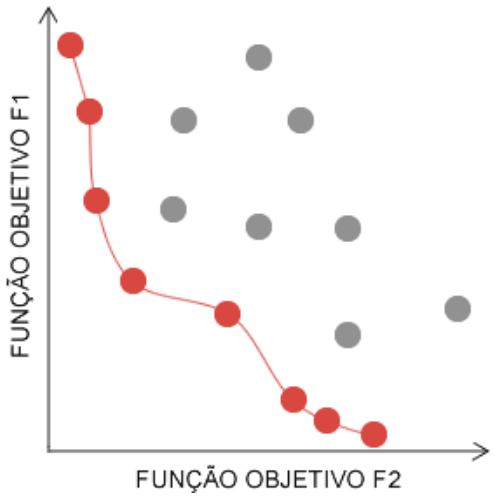


Figura 4 – Fronteira de Pareto

Observe que, todas as soluções pertencentes à fronteira de Pareto são não dominadas, ou seja, nenhum ponto da fronteira possui ambos valores menores que alguma outra solução em ambas as funções ($F1$ e $F2$). Em contra partida, toda solução acima da fronteira (pontos em cinza) é dominada, pois existe alguma solução em vermelho que possui ambos valores de F1 e F2 menores. Caso o problema em questão fosse de maximização, a fronteira de Pareto estaria acima de qualquer solução não-dominada ao invés de abaixo.

Não existe limite para o número de funções objetivo em um problema de otimização, mas quanto maior a quantidade de objetivos, mais complexa é a busca (DEB; JAIN, 2014). Os algoritmos clássicos de otimização multiobjetivo *Non-Dominated Sorting Genetic Algorithm II* (NSGA-II) e *Strength Pareto Evolutionary Algorithm 2* (SPEA2) lidam bem com até três objetivos, mas a partir de quatro critérios de otimização, ambos os métodos sofrem para encontrar soluções relevantes. Desta forma, criou-se a classificação "*many-objective*". Problemas *many-objective* (4 ou mais objetivos) apresentam um maior grau de dificuldade e precisam de novas técnicas para que sejam resolvidos eficientemente. Analisando as abordagens baseadas em algoritmos evolutivos multiobjetivos (AEMOs), (DEB; JAIN, 2014) observaram os seguintes os problemas trazidos pelo alto número de objetivos:

1. **Grande parte da população é não dominada:** a maioria dos algoritmos multiobjetivos classifica a população de acordo com a dominação de Pareto. Se existem muitas funções objetivo, é muito comum que uma solução seja melhor que outra em pelo menos uma delas. Desta forma, a maior parte das soluções se torna não-dominada, o que impede os algoritmos de evoluírem a população, já que todos os indivíduos são considerados igualmente bons.

2. **Avaliar a diversidade da população se torna computacionalmente caro:** a fim de garantir uma boa diversidade populacional, os algoritmos adotam alguma

medida de distância entre as soluções e removem aquelas consideradas mais similares. O aumento na dimensionalidade traz consequentemente um maior impacto (complexidade e custo) no cálculo da proximidade entre os indivíduos.

3. ***Crossover* ineficiente:** a alta dimensionalidade do espaço de busca faz com que os indivíduos na população sejam muito distante uns dos outros e, normalmente, o cruzamento entre duas soluções muito diferentes resultam num filho muito distante dos pais, o que prejudica a convergência da busca. Portanto, pode ser necessário redefinir os operadores de recombinação a fim de restringir as possibilidades de pareamento.

4. **População demasiadamente grande:** quanto maior o número de objetivos, maior o número de soluções na fronteira de Pareto. Portanto, para se obter bons resultados, é necessário que se manipule grandes populações de indivíduos, o que é computacionalmente caro e dificulta a análise do usuário que deverá escolher uma única solução ao final do processo.

5. **Métricas de análise de desempenho do algoritmo se tornam difíceis de calcular:** a avaliação do resultado do algoritmo (taxa de erro, distância para o Pareto, hipervolume, etc.) está diretamente relacionada ao número de objetivos, quanto maior ele for, maior será o esforço computacional necessário. A complexidade do hipervolume, por exemplo, cresce exponencialmente com o número de objetivos.

6. **Dificuldade de visualização:** é fácil representar graficamente as soluções e a fronteira de Pareto em problemas de até três objetivos. Entretanto, não existe uma forma eficiente de visualizar os resultados para problemas que lidam com maior dimensionalidade (4 objetivos em diante).

A maior parte dos algoritmos *many-objectives* mencionados neste trabalho lidam apenas com os quatro primeiros problemas. As duas últimas não são responsabilidade dos algoritmos de otimização em si.

3.1 Algoritmos Bio-Inspirados Multiobjetivos

A maior parte dos métodos de busca multiobjetivo são baseados em algoritmos inspirados na biologia. Esses algoritmos são adaptações das estratégias evolutivas, colônias de formigas e enxames de partículas tradicionais para lidar com problemas que envolvam mais de um objetivo. A principal diferença entre um algoritmo tradicional e sua variação multiobjetivo está na forma de cálculo da aptidão dos indivíduos da população. No caso dos algoritmos multiobjetivos, geralmente o conceito de dominância é usado de diferentes formas (BUENO; OLIVEIRA, 2010).

A seguir são apresentados alguns algoritmos bio-inspirados multiobjetivo encontrados na literatura. Os dois primeiros (NSGA-II e SPEA2) são algoritmos bem conhecidos e largamente utilizados em problemas multiobjetivos. Apesar de sua eficiência na resolução de problemas com até 3 objetivos, o desempenho desses AEMOs costuma cair consideravelmente com o aumento no número de objetivos (FRANÇA et al., 2017). Os demais algoritmos apresentados, são mais recentes e foram concebidos especificamente para tratar de problemas *many-objective*, ou seja, que envolvam 4 ou mais objetivos.

3.1.1 NSGA-II

O *Non-dominated Sorting Genetic Algorithm II* (NSGA-II) (DEB et al., 2002) é o algoritmo evolutivo multiobjetivo mais frequente na literatura. O processo desse algoritmo é semelhante ao do algoritmo genético comum, com diferença no cálculo de aptidão, que é feito por *ranks*, e no cálculo de distâncias, que é inexistente na proposta original do AG. A atribuição de aptidão (*fitness*) se dá pela classificação da população em categorias/classes (*rankings*) de dominância (fronteiras), de forma que o primeiro contenha todas as soluções não dominadas, o segundo todos os indivíduos não-dominados excluindo aqueles presentes na primeira fronteira, e assim por diante. Quanto melhor o *ranking* de uma solução, melhor sua aptidão e maior sua chance de sobreviver para a próxima geração. Várias soluções podem pertencer a uma mesma fronteira. A fim de diferenciá-las entre si, é utilizado um cálculo de distância (*crowding distance*), o qual confere melhor avaliação às soluções mais diferentes (maior distância), garantindo assim a diversidade da população.

O fluxo de funcionamento do NSGA-II inicia-se com a geração aleatória dos indivíduos. Em seguida, a população é classificada em *ranks* de dominância e inicia-se o processo iterativo, o qual termina assim que a condição de parada é atingida. As etapas que compõem cada iteração do NSGA-II são descritas no pseudo-código do algoritmo 1.

Algoritmo 1 Processo iterativo do NSGA-II

1: **enquanto** critério de parada não for atingido **faça**
2: $Pais \leftarrow selecao(Pop_{atual}, TX_{cross})$
3: $Filhos \leftarrow crossover(Pais)$
4: $Pop_{nova} \leftarrow Pop_{atual} \cup Filhos$
5: $ranks \leftarrow ranking(P_{nova})$
6: $crowdingDistance(P_{nova})$
7: $Pop_{atual} \leftarrow reinsecao(ranks, tam_{populacao})$
8: **fim enquanto**

O processo iterativo do algoritmo inicia-se com a seleção dos pares de pais para o cruzamento (linha 2). Na implementação considerada, a seleção de pais utiliza torneio simples com *tour* de 2, ou seja, dois elementos da população são escolhidos de forma aleatória e o indivíduo com melhor avaliação é selecionado como um dos pais. Então, esse processo é repetido para a escolha do segundo pai.

Na linha 3 do pseudo-código, através dos pares de pais, gera-se os filhos com o *crossover* e a mutação. Após a geração dos filhos, a população corrente e o conjunto de filhos são concatenados (linha 4) e submetidos à classificação em *ranks* de dominância (linha 5).

A classificação em *ranks* de dominância recebe um conjunto de soluções e verifica quais dentre elas não são dominadas. O conjunto de soluções não-dominadas forma o primeiro *rank* de dominância. Do conjunto restante (excluindo o primeiro *rank*), retira-se as soluções não dominadas para formar o segundo *rank*. Esse processo se repete até que todos os indivíduos tenham sido classificados.

Após toda a população ter sido classificada em *ranks*, na linha 6, o algoritmo calcula a distância de aglomeração (*crowding distance*) dos indivíduos em cada *rank* de dominância. O cálculo da distância, considerando cada objetivo avaliado, ordena o conjunto de soluções e faz uma relação entre as distâncias de cada indivíduo para os vizinhos adjacentes (que estão imediatamente antes ou depois). A distância de aglomeração de cada indivíduo é resultado da somatória das distâncias resultantes em cada objetivo. Quanto maior o valor, maior é a diferença entre o indivíduo e seus pares dentro do *rank*. Essa métrica é usada para manter a diversidade das soluções entre as gerações do algoritmo.

Com toda a população classificada em *ranks* (ou fronteiras) e todas as distâncias calculadas, na linha 7, uma nova população é escolhida com base nos melhores indivíduos entre pais e filhos. Para isso, analisa-se fronteira a fronteira, da melhor para a pior, até que o tamanho máximo da população seja atingido. Para cada fronteira aplica-se o seguinte processo de decisão:

- Se $tamanho(rank) + tamanho(novaPopulação) < tamPop$: adiciona-se todos os membros do *rank* à nova população.
- Caso contrário: adiciona-se à nova população os $(tamPop - tamanho(novaPopulação))$ elementos do *rank* com os maiores valores de distância.

Desta forma, ao final do algoritmo obtém-se a fronteira de Pareto aproximada na primeira fronteira da população gerada na última iteração do algoritmo.

3.1.2 SPEA2

O *Strength Pareto evolutionary algorithm 2* (SPEA2) (ZITZLER; LAUMANNS; THIELE, 2002) é um AEMO que calcula, para cada membro da população, sua força (*strength*) e densidade. A força de uma solução é dada pelo número de indivíduos que ela domina, enquanto a densidade é uma medida de distância para os vizinhos mais próximos, quanto maior a densidade mais próximo o indivíduo está das demais soluções. A aptidão (*fitness*) de uma solução é definida por sua densidade mais a soma das forças de todo indivíduo que a domina.

Os indivíduos mais aptos no SPEA2 são aqueles dominados pela menor quantidade de soluções e que possuem maior variabilidade genética. O algoritmo calcula a aptidão em três etapas: cálculo de força (*strength*), da aptidão crua (*raw fitness*) e da densidade (ZITZLER; LAUMANNS; THIELE, 2002).

A força de um indivíduo i ($s(i)$) é o número de soluções que ele domina, ou seja, considerando A o arquivo e P a população:

$$s(i) = |j| : j \in P \cup A \wedge i \prec j$$

Uma vez calculada a força de cada indivíduo, parte-se para o cálculo do *raw fitness*. O *raw fitness* de um indivíduo i ($r(i)$) é dado pela soma das forças de cada elemento que o domina, como segue:

$$r(i) = \sum_{j \in A \cup P | j \prec i} s(j)$$

Observe que, caso o indivíduo seja não-dominado, seu *raw fitness* será o menor possível (zero). Para finalizar o cálculo de aptidão, é definida a densidade de cada indivíduo ($d(i)$). Essa densidade é computada de acordo com a distância da solução para seus vizinhos e é dada por:

$$d(i) = \frac{1}{\sigma_i^k + 2}$$

Sendo, σ_i^k a k-ésima menor distância entre o indivíduo i e o restante da população; e k a raiz quadrada do tamanho do conjunto de soluções em avaliação, i.e. $k = \sqrt{|P \cup A|}$. O valor de $d(i)$ sempre está no intervalo (0,1).

Finalmente, a aptidão do indivíduo ($f(i)$) é dada pela soma do *raw fitness* e a densidade:

$$f(i) = r(i) + d(i)$$

Note que, se a solução i é não-dominada, $f(i) < 1$. Isso ocorre dado que $d(i) < 1$ para qualquer solução e, quando i é não-dominada, $r(i) = 0$.

Além do cálculo de aptidão, outra diferença importante entre o SPEA2 e um AG tradicional é a utilização de uma população extra, denominada arquivo. O arquivo é responsável por guardar as melhores soluções já encontradas até o momento, funciona como uma espécie de elitismo. Na seleção para o cruzamento, os pais são sempre escolhidos do arquivo e os filhos substituem 100% da população corrente. A cada iteração, os melhores indivíduos entre a população e o arquivo compõe o arquivo da geração seguinte. A quantidade de indivíduos no repositório de soluções não-dominadas é limitada e, quando esse tamanho máximo (tam_arq) é excedido, deve-se executar um processo de truncamento.

O processo de truncamento de um arquivo ocorre na seleção natural (reinserção), que é a última função executada na iteração do laço principal de um AG. No SPEA2, a

reinserção consiste em definir o arquivo para a próxima geração a partir das populações. Nesse processo, extrai-se do conjunto total de soluções (população e arquivo) aquelas que não são dominadas por nenhuma outra, e com esse subconjunto (n_d) constrói-se o novo arquivo através do seguinte processo de decisão:

- Se $tamanho(n_d) = tam_arq$, o novo arquivo é formado por n_d;
- Se $tamanho(n_d) < tam_arq$, o novo arquivo é formado pela união de n_d com os $tam_arq - tamanho(n_d)$ indivíduos restantes com melhor aptidão;
- Caso contrário, ($tamanho(n_d) > tam_arq$), o novo arquivo é formado por n_d e deve-se truncá-lo em ($tamanho(n_d) - tam_arq$) passos, onde em cada passo, elimina-se o indivíduo com menor variabilidade genética em relação aos demais.

O algoritmo retorna como resposta para o problema o arquivo resultante da última geração computada. Espera-se que, após as diversas iterações, o algoritmo tenha conseguido uma boa aproximação da fronteira de Pareto.

3.1.3 MOEA/D

O *Multiobjective evolutionary algorithm based on decomposition* (MOEA/D) (ZHANG; LI, 2007) é um algoritmo que avalia os objetivos através de uma função escalarizadora, se baseando na dominância de Pareto apenas para atualizar o conjunto de soluções não dominadas geradas em cada iteração (arquivo). No MOEA/D, um problema multiobjetivo é decomposto em múltiplos problemas de um único objetivo chamados de células. Cada célula é definida por um vetor de pesos gerado aleatoriamente e representa um indivíduo, ou seja, o número de células é igual ao tamanho da população. Além dos pesos, a célula, ou indivíduo, é composta de uma solução e uma vizinhança. A vizinhança é formada pelos k indivíduos mais próximos de acordo com o vetor de pesos, onde k é um parâmetro do algoritmo que representa o tamanho das vizinhanças. A aptidão (*fitness*) de uma solução é calculada de acordo com sua avaliação em cada objetivo, a função escalarizadora, e o vetor de pesos da célula. Em toda geração, uma nova solução é gerada para cada célula, onde a vizinhança é considerada nas etapas de escolha dos pais e seleção natural.

O início do algoritmo consiste na geração da estrutura de células e na definição das vizinhanças. Para isso, sorteia-se os vetores de pesos (a soma de cada vetor deve ser igual a um) e, para cada um deles, calcula-se os k vetores mais próximos (vizinhança). Essa estrutura é imutável e é utilizada no decorrer de todo o algoritmo. A geração dos vetores de pesos pode ser tanto aleatória quanto seguir uma distribuição pré-definida. Antes de começar o processo iterativo (evolução das gerações), gera-se aleatoriamente uma solução para cada célula e calcula-se a sua aptidão.

Uma parte fundamental do MOEA/D é a escolha da função escalarizadora, a qual é a principal responsável pelo cálculo de aptidão. Em todos experimentos realizados neste

trabalho, foi utilizada a soma ponderada, mas outras estratégia, como *Penalty-Based Boundary Intersection* e Tchebycheff também podem ser utilizadas (ZHANG; LI, 2007). A aptidão de uma solução é calculada através da função escalarizadora e do vetor de pesos. Por exemplo, se os valores $[2, 9, 5]$ representam a solução s no espaço de objetivos, $[0.3, 0.2, 0.5]$ é o vetor de pesos da célula c, e a soma ponderada é a função escalarizadora, então a aptidão de s em c é dada por $2 \times 0.3 + 9 \times 0.2 + 5 \times 0.5 = 4.9$.

Como em qualquer outro AG, o processo evolutivo do MOEA/D consiste basicamente da seleção de pais, da geração dos filhos e sua reinserção na população. Para cada célula c_i, dois pais são selecionados aleatoriamente em sua vizinhança. Quando um filho é gerado para um célula c_i, sua aptidão é calculada para cada uma das células na vizinhança de c_i, substituindo a solução atual caso o *fitness* do novo indivíduo (filho) seja melhor. Ao final de cada iteração (geração), atualiza-se o arquivo com as novas soluções não-dominadas.

3.1.4 NSGA-III

O *Non-dominated Sorting Genetic Algorithm III* (NSGA-III) (DEB; JAIN, 2014) é uma extensão do NSGA-II que permite o *framework* funcionar melhor para mais de três objetivos (many-objective). Ele se diferencia do original apenas na fase de reinserção ou seleção natural, onde ao invés de usar a distância de aglomeração para diferenciar soluções em uma mesma fronteira, utiliza um método de agrupamento, no qual os indivíduos são divididos em nichos de acordo com suas similaridades. O NSGA-III é caracterizado pelo processo de atribuição de nicho chamado de classificação não-dominada baseada em pontos de referência. Sua ideia é traçar uma figura geométrica de uma dimensão a menos que o número de objetivos nos pontos extremos da primeira fronteira. Um número pré-definido de pontos de referência equidistantes são distribuídos ao longo da figura, cada qual representando um respectivo nicho. Para classificar uma solução, define-se como nicho o ponto de referência mais próximo. Ao final, são selecionadas (sobrevivem) as soluções localizados nas regiões menos lotadas do espaço de busca. Maiores detalhes sobre esse processo de agrupamento podem ser obtidos no artigo original do algoritmo (DEB; JAIN, 2014).

3.1.5 SPEA2-SDE

O *SPEA2 with Shift-Based Density Estimation* (SPEA2-SDE) (LI; YANG; LIU, 2014) é uma pequena alteração no algoritmo SPEA2 que possibilita lidar com problemas que envolvam 4 ou mais objetivos de uma forma muito mais eficiente. A única alteração está no cálculo de distância entre duas soluções. Suponha que s_1 e s_2 sejam dois indivíduos na população e que seus valores no espaço de objetivo sejam, respectivamente, $[5, 10, 351, 7, 15]$ e $[6, 8, 15, 9, 14]$. No SPEA2, a distância entre s_1 e s_2 é dada pela distância euclidiana dos dois vetores ($dist$), ou seja:

$$dist(s_1, s_2) = \sqrt{(5-6)^2 + (10-8)^2 + (351-15)^2 + (7-9)^2 + (15-14)^2} = 336,0148$$

As duas soluções são bem próximas, pois só estão distantes em uma das 5 coordenadas. Entretanto, o cálculo de distância do SPEA2 não reflete isso, o que é prejudicial em problemas com muitos objetivos. A fim de resolver esse problema, (LI; YANG; LIU, 2014) introduziram o cálculo de distância *Shift-Based Density Estimation* (SDE), que nada mais faz do que transladar a coordenada mais distante do segundo ponto para o mesmo valor no primeiro ponto. Isto é, antes de calcular a distância euclidiana, identifica-se a coordenada que exibe maior diferença entre os dois pontos. No caso de s_1 e s_2, a terceira coordenada é a que possui esse comportamento. Em seguida, é realizada a translação do segundo ponto na coordenada identificada para que seja igual ao primeiro ponto, no exemplo, ($s'_2 = [6, 8, \mathbf{351}, 9, 14]$). A distância SDE é, então, determinada pela distância euclidiana entre o primeiro ponto (s_1) e o segundo ponto transladado (s'_2). No exemplo, a distância SDE obtida é:

$$dist_{SDE}(s_1, s'_2) = \sqrt{(5-6)^2 + (10-8)^2 + (351-351)^2 + (7-9)^2 + (15-14)^2} = 3,1622$$

A distância SDE descarta a coordenada mais distante ao calcular as distâncias, permitindo assim que pontos distantes em apenas uma coordenada ainda sejam considerados próximos. Todo o restante do processo do SPEA2-SDE segue os mesmos passos do algoritmo original.

3.1.6 AEMMT

O Algoritmo Evolutivo Multiobjetivo com Muitas Tabelas (AEMMT) (BRASIL; DELBEM; SILVA, 2013), assim como o MOEA/D, decompõe o problema multiobjetivo em subproblemas menores de um único objetivo. Entretanto, o AEMMT utiliza um esquema de tabelas, onde cada tabela representa uma combinação diferente dos objetivos investigados. A função que transforma os múltiplos objetivos em um valor escalar é a média aritmética e cada tabela mantém os melhores indivíduos, considerando o valor médio dos objetivos que ela representa. A cada geração, duas tabelas são selecionadas para o cruzamento. Dois pais, um de cada tabela, são sorteados aleatoriamente para gerarem um único filho. Essa nova solução (filho) é avaliada em todas as tabelas, entrando naquelas em que representar uma melhor aptidão em relação aos indivíduos atuais. Naturalmente, como um único *crossover* é realizado e um único filho é gerado a cada iteração, o AEMMT precisa de mais gerações para alcançar o mesmo número de soluções avaliadas que os algoritmos citados anteriormente.

A quantidade de tabelas é determinada pelo número de combinações possíveis entre os objetivos. Para quatro objetivos (f_1, f_2, f_3, f_4), por exemplo, serão criadas 15 tabelas

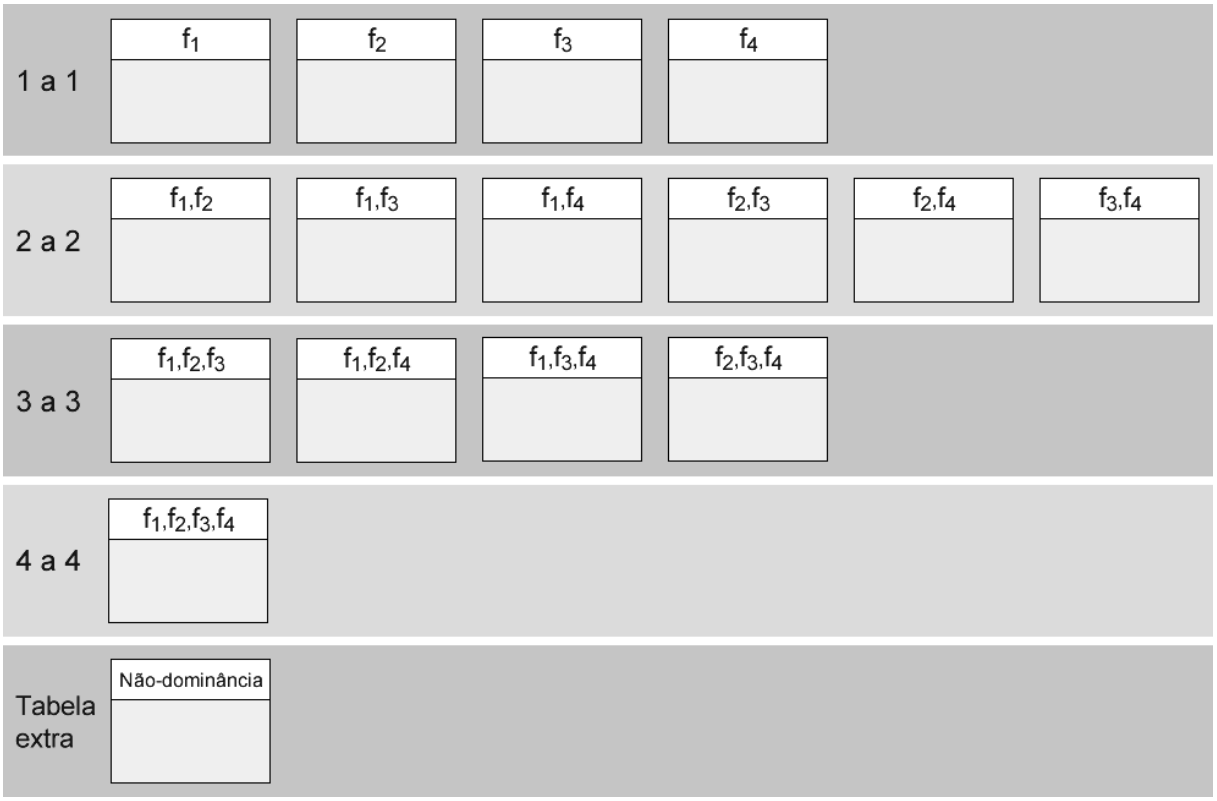


Figura 5 – Exemplo da quantidade de tabelas usadas pelo AEMMT

de combinações mais uma tabela extra, usada para guardar os indivíduos não-dominados, como ilustrado na Figura 5. Cada tabela possui um limite máximo de indivíduos. No início do algoritmo, gera-se soluções aleatórias de forma que todas as tabelas sejam completamente preenchidas. No laço principal, um indivíduo só entra em uma tabela t, se for melhor que a pior solução na população atual de t. Com relação a tabela de dominância, sempre que um filho é gerado e não-dominado por nenhum outro indivíduo da tabela, ele é incluído. A restrição no tamanho da tabela de dominância é independente das demais e, sempre que o limite for atingido, é feito um truncamento priorizando a permanência das soluções com maior valor de média aritmética entre todos os objetivos.

Após a geração e preenchimento das tabelas, inicia-se o laço principal. A cada iteração são escolhidas duas tabelas via torneio duplo de acordo com suas pontuações. A pontuação tem valor inicial zero e sempre que uma tabela gera um filho que sobrevive para a geração seguinte, sua pontuação é incrementada. As pontuações são zeradas a cada 100 gerações. Considerando as duas tabelas selecionadas, sorteia-se um indivíduo de cada e efetua-se o cruzamento entre eles. O filho gerado é, então, comparado tabela à tabela, entrando naquelas em que representar uma melhoria. Após a execução de todas as gerações, dá-se como resultado o conjunto não dominado, considerando as soluções de todas as tabelas.

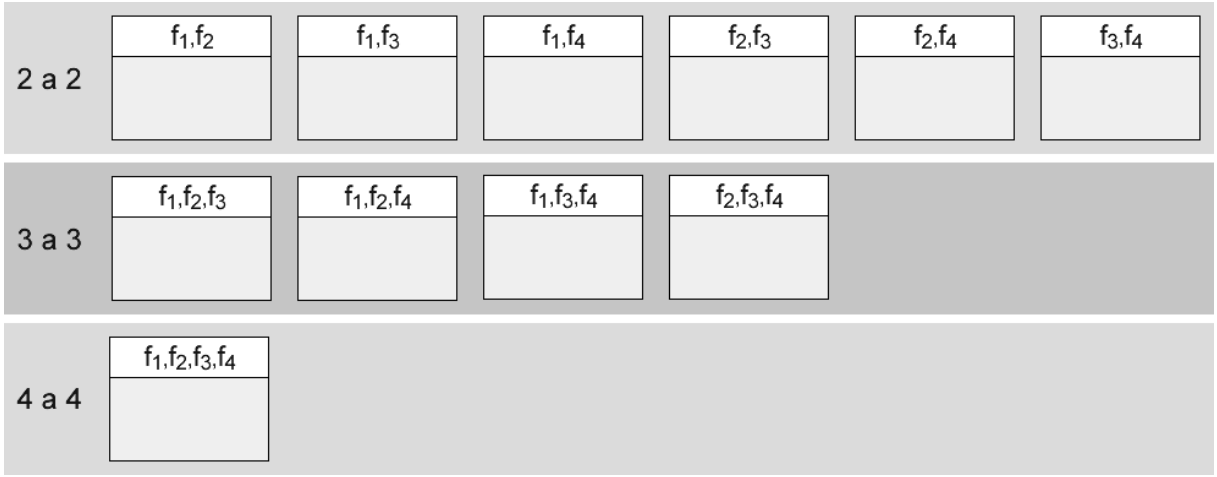


Figura 6 – Exemplo da quantidade de tabelas usadas pelo AEMMD

3.1.7 AEMMD

O Algoritmo Evolutivo Multiobjetivo com Múltiplas Dominâncias (AEMMD) (LAFETÁ et al., 2016) é uma modificação do AEMMT que, apesar de usar o mesmo processo de divisão do problema multiobjetivo, abandona a ideia de escalarização e adota o conceito de dominância também usado métodos mais antigos (e.g. NSGA-II e SPEA2). No AEMMD, ao invés de se utilizar a média dos objetivos da tabela para avaliar o indivíduo, lança-se mão da relação de dominância de Pareto. Um indivíduo novo s só entra na tabela t, se s não for dominado por nenhuma solução em t considerando apenas os objetivos da tabela. Além disso, se s entra em t, todas as soluções em t dominadas por s são removidas.

O primeiro passo do AEMMD é gerar o conjunto de tabelas, considerando todas as combinações de objetivos possíveis a partir de dois a dois. Combinações de um único objetivo não são criadas, pois o conceito de dominância é válido a partir de dois valores. Diferentemente do AEMMT, as tabelas não possuem limite de tamanho e podem crescer indefinidamente. Por exemplo, para um problema de quatro objetivos (f_1, f_2, f_3, f_4), 11 tabelas seriam geradas, como pode ser observado na Figura 3.1.7.

Com as tabelas criadas, gera-se um número pré-definido de soluções aleatórias, distribuindo-nas pelas tabelas de acordo com a relação de dominância de Pareto. Assim como no AEMMT, a cada iteração do algoritmo, é utilizado um torneio duplo para selecionar duas tabelas de acordo com suas pontuações. Entretanto, a pontuação das tabelas no AEMMD é diferente do AEMMT. Ao invés de conceder um ponto sempre que se gera um indivíduo sobrevivente, pontua-se uma tabela quando ela recebe um indivíduo. Tendo escolhido as duas tabelas, sorteia-se um representante de cada e gera-se um único filho, o qual é comparado tabela à tabela, entrando naquelas onde representa uma solução não dominada. Outra diferença em relação ao método original, é que o AEMMD não reinicia

as pontuações em momento algum do algoritmo. Espera-se que ao final das gerações, a população da tabela com todos os objetivos tenha convergido para a fronteira de Pareto.

3.2 Algoritmos multiobjetivos baseados em colônias de formigas

A maior parte dos métodos de busca multiobjetivo são baseados em algoritmos genéticos. Entretanto, uma alternativa que ainda é pouco explorada são as técnicas inspiradas em inteligência coletiva. Dentre elas, optou-se por investigar métodos de otimização baseados em colônias de formigas, os quais são particularmente adequados para lidar com problemas discretos, como aqueles utilizados em neste trabalho (problema da mochila multiobjetivo e problema do roteamento multicast). Vários ACOs multiobjetivos são encontrados na literatura, dentre eles destacam-se o MOACS (BARAN; SCHAERER, 2003) e o MOEA/D-ACO (KE; ZHANG; BATTITI, 2013).

3.2.1 MOACS

O *Multi-Objective Ant Colony Optimization Algorithm* (MOACS) (BARAN; SCHAERER, 2003) é uma adaptação do ACO original que torna possível a otimização de múltiplos objetivos utilizando uma única estrutura de feromônios, múltiplas heurísticas e um arquivo de soluções não dominadas. Esse algoritmo foi proposto originalmente para o problema de roteamento de veículos com janelas de tempo e, posteriormente, foi aplicado no problema do roteamento multicast (PINTO; BARAN, 2005). Uma variação do algoritmo original foi apresentada em (RIVEROS et al., 2016), a qual é utilizada neste trabalho. Seu funcionamento básico é descrito no Algoritmo 2.

O processo de construção de uma solução depende do problema investigado. No MOACS, os principais componentes utilizados nesse processo são:

- Feromônios (τ_{ij}): estrutura que guarda a quantidade de feromônios em cada partícula que pode formar a solução. No caso de problemas em grafos, representa a quantidade da substância em cada uma das arestas, indicando a frequência de suas utilizações;

- Heurísticas (H): conjunto de funções que estimam a qualidade de uma dada partícula que pode formar a solução. No caso de problemas em grafos, representa os vários pesos em uma aresta. Por exemplo, num grafo que representa uma rede de computadores com informações de custo, distância e tráfego, H poderia ser formado de três funções que recebem uma aresta (i, j) e devolvem, respectivamente, os valores de suas métricas custo, distância e tráfego.

Algoritmo 2 Algoritmo MOACS

```
1: Inicialize a estrutura de feromônios τ_ij com τ_0 /* τ_0 é o valor inicial */
2: Crie um conjunto vazio de soluções não-dominadas ND
3: enquanto Número máximo de iterações não for atingido faça
4:     para i ← 0 até tamanho_populacao faça
5:         Sorteie valores no intervalo [0, w_max[ para formar um vetor de pesos W com |H| posições
6:         Construa uma solução de acordo com a tabela de feromônios τ_ij, as heurísticas (H) e os pesos W
7:         Atualize ND com a nova solução
8:     fim para
9:     se ND foi modificado então
10:        Reinicie a estrutura de feromônios fazendo τ_ij = τ_0 ∀(i, j)
11:    senão
12:        Atualize a estrutura de feromônios com todas as soluções em ND
13:    fim se
14: fim enquanto
15: retorne ND
```

- Peso máximo de uma heurística (w_{max}): representa o valor máximo que o peso de uma heurística pode atingir. Em (RIVEROS et al., 2016), propõe-se $w_{max} = 3$, de forma que cada função possa ser classificada como 0 (não importante), 1 (importante), 2 (muito importante).

- Vetor de pesos (W): O vetor de pesos atribui a importância de cada heurística e normalmente é gerado de forma aleatória em cada iteração. Cada função de heurística recebe um peso variando no intervalo $[0, w_{max}[$.

Ao construir uma solução, utiliza-se o mesmo processo de decisão do ACO original, explicado na seção 2.2.2. A única diferença é que as múltiplas heurísticas do MOACS (H) são unificadas em uma única função $h(x)$, por meio de uma média ponderada. Nesse processo, utiliza-se o vetor de pesos W para definir o fator de ponderação de cada heurística. Dessa forma, a função h(x) é dada por:

$$h(x) = \frac{\sum_{i \leftarrow 0}^{size(H)} H_i(x) \times W_i}{\sum_{w \in W} w}$$

Em cada época (iteração do ACO), atualiza-se o arquivo de soluções não dominadas com as novas soluções geradas. Se o arquivo foi atualizado após criar-se todas as soluções, reinicia-se as informações de feromônio, redefinindo todos os valores na estrutura τ_{ij} para o valor inicial de feromônio τ_0. Caso o arquivo tenha se mantido estável, ou seja, nenhuma das novas soluções é não-dominada, atualiza-se as quantidades de feromônio na estrutura de acordo com as soluções no arquivo.

Considerando um problema em grafos, para atualizar a estrutura τ_{ij} com uma solução s, faz-se:

$$\tau_{ij} = (1 - \rho) \times \tau_{ij} + \rho \times \Delta\tau(s) \qquad \forall (i, j) \in s$$

Sendo:

- ρ: coeficiente de evaporação;
- $\Delta\tau(s)$: Quantidade de feromônios depositados pela solução s, que por sua vez é definida por:

$$\Delta\tau(s) = \frac{1}{performance(s)}$$

Na fórmula anterior, $performance(s)$ é dado pela soma dos valores de s no espaço de objetivos. Neste caso, considera-se um problema de minimização. Por exemplo, se os objetivos são reduzir o custo, o tráfego e o atraso (*delay*) de uma rede, então, $performance(s) = custo(s) + trafego(s) + delay(s)$. Para problemas de maximização, basta inverter a equação.

Após todas as iterações do MOACS, espera-se obter no arquivo uma boa aproximação da fronteira de Pareto.

3.2.2 MOEA/D-ACO

O *Multiobjective evolutionary algorithm based on decomposition ACO* (MOEA/D-ACO) (KE; ZHANG; BATTITI, 2013) é um método que adota o algoritmo MOEA/D (seção 3.1.3) aplicado ao *framework* de otimização por colônia de formigas (ACO). Os conceitos de células e vizinhanças são reutilizados, enquanto que a reprodução local, inexistente no ACO, é substituída por um processo de construção da solução levemente modificado. Além disso esse método introduz um novo conceito de grupos que é utilizado para agrupar as formigas de acordo com seus vetores de peso. Nesse algoritmo, o termo formiga corresponde ao conceito de célula do MOEA/D.

O primeiro passo do MOEA/D-ACO consiste em gerar os vetores de peso que, assim como no MOEA/D, são arranjos de valores entre 0 e 1, cujas somas valem 1. A geração pode ser aleatória ou seguir alguma distribuição pré-definida. No caso deste trabalho, utilizou-se a distribuição uniforme proposta no artigo original (KE; ZHANG; BATTITI, 2013). Atribui-se cada vetor a uma formiga e cria-se a estrutura de vizinhanças, onde cada formiga é associada a uma vizinhança contendo as v_{size} formigas mais próximas de acordo com os vetores de peso (incluindo ela mesma). O segundo passo do algoritmo determina os grupos, cada grupo recebe um novo vetor de pesos gerado da mesma maneira que os pesos anteriores. As formigas são então distribuídas entre os grupos de acordo com a proximidade entre seu vetor de pesos e o vetor de pesos do grupo (*agrupamento*).

Com respeito às duas principais estruturas de um ACO, heurística e feromônios, cada formiga possui uma função heurística e cada grupo mantém uma estrutura de feromônios.

Quando os grupos são criados, cada um recebe uma estrutura de feromônios com o valor máximo de feromônio em cada uma das posições. Por sua vez, quando a formiga é criada, recebe uma função heurística baseada na combinação de todas as funções heurísticas do problema e no vetor de pesos da própria formiga. O cálculo específico da heurística depende do problema.

O processo iterativo (evolução) do MOEA/D-ACO consiste em gerar as soluções, atualizar o conjunto de soluções não-dominadas ND, distribuir as novas soluções entre as formigas e atualizar as estruturas de feromônios. Primeiramente, para cada grupo G, gera-se as soluções para cada formiga $f \in G$. A solução é gerada com base nos feromônios de G, na heurística de f e na solução atual de f. Tendo gerado todas as soluções para um grupo, atualiza-se o arquivo de soluções não-dominadas ND. Para cada nova solução S que entrou no arquivo, modifica-se a estrutura de feromônios τ de G. Assim, o cálculo do feromônio de uma partícula e na iteração $i+1$ ($\tau_{i+1}(e)$) é obtida a partir do seu feromônio atual ($\tau_i(e)$), como segue:

$$\tau_{i+1}(e) = \begin{cases} \rho \times \tau_i(e) + \delta, & \text{se } e \in S \\ \rho \times \tau_i(e), & \text{caso contrário} \end{cases}$$

Onde e é uma possível partícula de uma solução para o problema investigado (qualquer aresta, PRM, ou item, PMM) da tabela de feromônios; ρ é a taxa de evaporação; e δ é dado pelo inverso da soma dos valores da função de escalarização aplicada sobre cada solução não dominada de G em relação ao vetor de pesos de G.

$$\delta = \frac{1}{\sum_{s \in S_{nd} f(s,W)}}$$

Sendo, S_{nd} o conjunto de soluções não dominadas de G, f a função de escalarização, e W o vetor pesos de G. Caso o problema seja de maximização, δ seria a soma dos valores, ao invés do inverso dela.

Cada formiga armazena duas soluções, a solução atual, que de início é nula, e a nova solução, criada em cada iteração do algoritmo. Após a geração de novas soluções a atualização de feromônios para todos os grupos e suas formigas, deve-se decidir a solução atual de cada formiga com base nas novas soluções da vizinhança. Nesse processo, para cada formiga f analisa-se as novas soluções presentes na vizinhança de f. Se alguma nova solução sn, que ainda não substituiu nenhuma outra, possui um *fitness* melhor que o da solução corrente de f (sc), substitui-se sc por sn. O *fitness* é calculado de acordo com a média ponderada dos valores da solução em cada objetivo baseadas no vetor de pesos da formiga em questão. Neste trabalho, utilizou-se a soma ponderada como função *escalarizadora*, mas qualquer uma das funções propostas em (ZHANG; LI, 2007) pode ser usada.

O processo de construção da solução depende do problema investigado. Entretanto, o MOEA/D-ACO inclui duas novas características no processo, independente do problema. São elas:

- **Influência da solução atual:** no MOEA/D-ACO, cada formiga mantém uma solução atual que influencia a construção da próxima solução com base em um parâmetro δ. Isso acontece devido à introdução de um novo termo no cálculo de feromônio na construção da solução. Ao calcular a probabilidade de uma partícula p (aresta ou item) fazer parte da solução, ao invés de adotar $feromonio(p)^{\alpha}$, considera-se $(\delta * x + feromonio(p))^{\alpha}$, sendo $x = 1$ se p pertence à solução atual e $x = 0$ caso contrário.

- **Taxa de elitismo:** a maioria dos ACOs utilizam uma espécie de roleta para decidir qual partícula fará parte da solução. Aquelas com maior probabilidade têm maior chance de serem escolhidas, mas não necessariamente serão. Em uma estratégia elitista, não se utiliza a roleta. A partícula escolhida será aquela que receber o maior valor de probabilidade. No MOEA/D-ACO, utiliza-se uma taxa de elitismo que estipula a chance de uma partícula ser escolhida por elitismo ao invés de roleta.

O processo iterativo do algoritmo é executado até que uma condição de parada pré-definida seja atingida. Normalmente adota-se um número máximo de iterações. Espera-se que, nesse momento, as soluções no conjunto ND tenham convergido para a fronteira de Pareto.

3.3 Outros algoritmos multiobjetivos

Alguns outros algoritmos multiobjetivos não foram analisados neste trabalho, mas por estarem relacionados ao tema da pesquisa, são mencionados a seguir.

VEGA (SCHAFFER, 1985), *Non-Dominated Sorting Genetic Algorithm* (NSGA) (SRINIVAS; DEB, 1994) e *Strength Pareto Evolutionary Algorithm* (SPEA) (ZITZLER; THIELE, 1999) foram os primeiros algoritmos multiobjetivos a aparecerem na literatura, apesar do NSGA-II e SPEA2 serem os mais conhecidos. Em problemas *many-objective*, uma das principais dificuldades encontradas é o excesso de soluções não-dominadas, o que dificulta a identificação de melhores indivíduos e prejudica a convergência. Para lidar com esse problema, (AGUIRRE; TANAKA, 2009) apresentam um conceito menos rigoroso de dominância, dando origem ao algoritmo ϵ-MOEA. Por sua vez, a fim de reclassificar as soluções não dominadas e resolver o mesmo problema, (BEUME; NAUJOKS; EMMERICH, 2007) propõem o algoritmo SMS-EMOA, no qual a evolução da população ocorre de acordo com indicadores de qualidade do conjunto de soluções. Nesse algoritmo, os autores usam o hipervolume para avaliar soluções consideradas igualmente boas pela dominância de Pareto. (BADER; ZITZLER, 2011) utilizam uma ideia parecida ao propor

o algoritmo Hype. Esse algoritmo também utiliza o hiper-volume como parâmetro para guiar a evolução da população.

(SOUZA; POZO, 2015) propõem o algoritmo MOEA/D-BACO que aplica uma variação do MOEA/D-ACO no problema de programação binária quadrática irrestrito (UBQP). Como para o UBQP as estruturas de feromônio crescem de forma exponencial em relação ao tamanho da entrada, torna-se inviável. A variação proposta pelos autores, algoritmo MOEA/D-BACO, visa solucionar esse problema através da substituição do ACO, no MOEA/D-ACO, pelo *Binary Ant Colony Optimization* (BACO) (KONG; TIAN, 2006), que é um algoritmo mais simples e promete menor tempo de execução em relação ao algoritmo original.

Capítulo 4

Problemas de teste

A fim de avaliar os métodos multiobjetivos, normalmente se utiliza problemas de teste. São vários os problemas em que se pode aplicar os algoritmos multiobjetivos, os quais podem ser divididos em duas categorias: contínuos ou discretos. Os problemas contínuos são funções contínuas e não necessariamente representam um problema real. Alguns exemplos de problemas contínuos (SCH, FON, POL, KUR e ZDT) podem ser encontrados no artigo original do NSGA-II (DEB et al., 2002). Os problemas discretos, por outro lado, possuem um enunciado bem definido e nem todas as soluções possíveis são válidas, ou seja, existem lacunas no contradomínio das funções. Exemplos de problemas discretos comummente usados na literatura multiobjetivo são: cacheiro viajante (LUST; TEGHEM, 2010), roteamento de veículos com janelas de tempo (OMBUKI; ROSS; HANSHAR, 2006), problema da mochila (BAZGAN; HUGOT; VANDERPOOTEN, 2009), sequenciamento de proteínas (BRASIL; DELBEM; SILVA, 2013) e problemas de roteamento em redes (LAFETÁ et al., 2018). O comportamento e a adequação ao uso de um determinado algoritmo multiobjetivo são influenciados pelo tipo do problema. Neste trabalho, dois problemas discretos foram investigados: o problema da mochila multiobjetivo (PMM) e o problema do roteamento multicast (PRM), os quais são descritos a seguir.

4.1 Problema da mochila multiobjetivo

O Problema da Mochila (PM) é um problema muito comum na computação e possui um caráter teórico. Entretanto, existem problemas reais equivalentes que podem ser resolvidos com as mesmas técnicas, como o escalonamento de tarefas em um sistema operacional.

O problema consiste em arranjar um conjunto de itens em uma mochila de forma a não exceder sua capacidade e, ao mesmo tempo, maximizar o valor (lucro) dos objetos carregados. Matematicamente, dada uma mochila de capacidade C e um conjunto de itens O, onde cada $o_i \in O$ possui um peso $peso(o_i)$ e um lucro $lucro(o_i)$, encontrar o conjunto $S \subset O$, tal que $\sum_{o \in S} peso(o) \leq C$ e $\sum_{o \in S} lucro(o)$ seja o maior possível. A

Figura 7 mostra uma instância do problema da mochila mono-objetivo e três soluções possíveis.

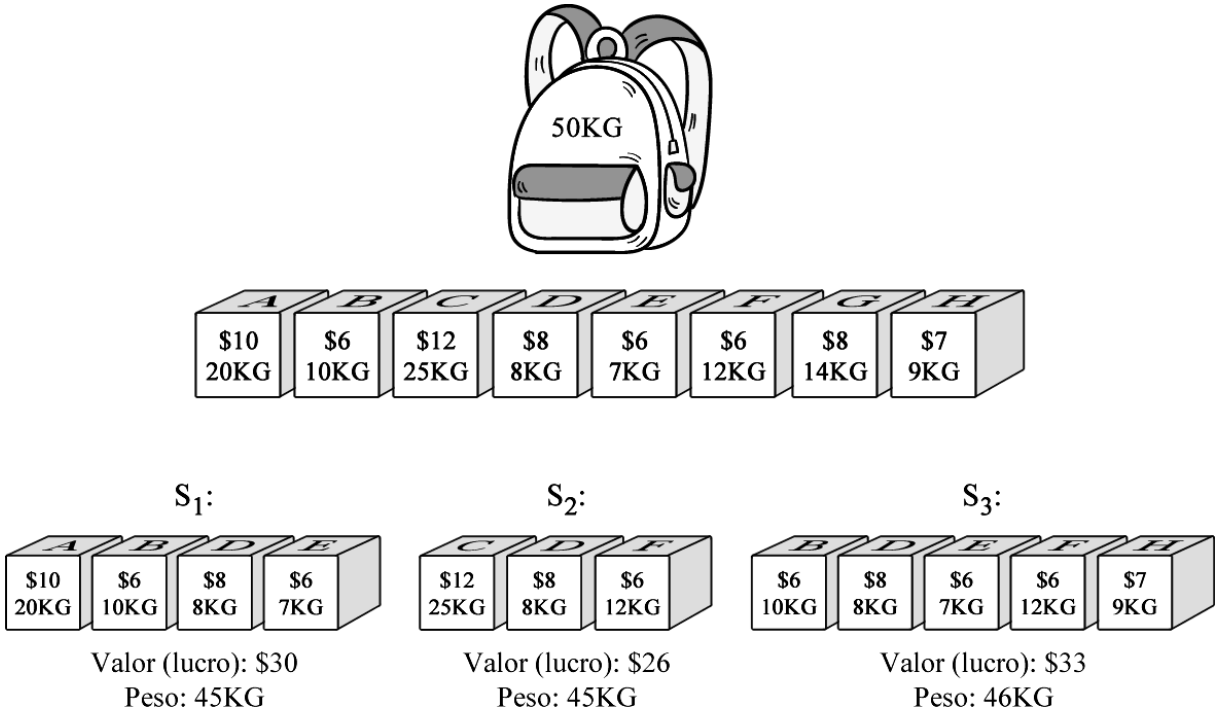


Figura 7 – Exemplo para o problema da mochila mono-objetivo.

Na Figura 7, a capacidade máxima da mochila é 50Kg e existem oito itens disponíveis para se escolher. Três soluções possíveis são mostradas (S_1, S_2 e S_3), dentre elas, a melhor é S_3, pois possui o maior valor de lucro e ao mesmo tempo é válida, ou seja, repeita a restrição de capacidade (peso menor que 50Kg). No exemplo, todas as soluções estão saturadas, em outras palavras, a adição de qualquer outro item as tornam inválidas.

Existem diversas estratégias para resolver o PM. Dentre elas, as mais usadas são os algoritmos gulosos (HRISTAKEVA; SHRESTHA, 2018), a programação dinâmica (TOTH, 1980) e os algoritmos genéticos (USLU, 2015). Os algoritmos gulosos e a programação dinâmica são os mais rápidos e eficientes para resolver o PM mono-objetivo. Em contra-partida, a complexidade adicionada ao considerar mais de um objetivo inviabiliza a utilização desse tipo de algoritmo, tornando os algoritmos genéticos e demais métodos bio-inspirados as melhores opções.

O problema da mochila multiobjetivo (PMM) é similar ao original. Sua única diferença está no fato de que cada item, ao invés de possuir um único valor (lucro), é composto de múltiplos valores. No PMM, a função $lucro(o_i)$ retorna um vetor ao invés de um único valor escalar. Cada componente (elemento) do vetor representa o valor do item o_i considerando um dos objetivos. Por exemplo, considerando um PMM de 3 objetivos, cada $o_i \in O$ possui um vetor formado por três valores de lucros (um para cada objetivo). O objetivo do problema passa a ser maximizar todos os lucros ao invés de um único valor.

O PMM já foi utilizado várias vezes para avaliar algoritmos multiobjetivos, podendo-se destacar os trabalhos de (ZITZLER; THIELE, 1999), (ZITZLER; LAUMANNS; THIELE, 2002) e (ZHANG; LI, 2007). As instâncias do PMM utilizadas no decorrer deste trabalho foram geradas de forma aleatória e compreendem problemas da mochila com 30, 40, 50, 100 e 200 itens. Para gerar as instâncias, sorteou-se valores no intervalo (0, 1000) para os lucros e para os pesos. A capacidade da mochila foi sempre definida como 60% da soma dos pesos.

4.2 Problema do roteamento multicast

O problema do roteamento multicast (PRM) aparece na engenharia de tráfego em redes de computadores e consiste em escolher a forma "mais eficiente" de transmitir uma mensagem multicast (LAFETÁ et al., 2016). Uma transmissão de rede pode ser do tipo unicast, multicast ou broadcast. Em transmissões unicast, conecta-se um ponto da rede a outro ponto qualquer (transmissão ponto-a-ponto). Para fazer isso de forma eficiente, basta usar algum algoritmo capaz de encontrar o melhor caminho entre os dois pontos (e.g. (DIJKSTRA, 1959)). As comunicações broadcast caracterizam-se pelo fato de um nó da rede (servidor) enviar o conteúdo a todos os demais. Para obter as melhores rotas para trafegar os dados, basta utilizar algum algoritmo capaz de determinar a árvore geradora de custo mínimo (e.g. (PRIM, 1957)). Em uma comunicação multicast, o objetivo é transmitir o conteúdo de um nó da rede, chamado nó de origem ou transmissor, para alguns outros nós, denominados receptores. Essa tarefa apresenta maior complexidade, pois é necessário obter uma árvore de Steiner de custo mínimo, o que é mais difícil que calcular uma única rota ou construir a árvore geradora de custo mínimo (BUENO; OLIVEIRA, 2010).

O PRM é um problema prático muito importante. Seu estudo visa o desenvolvimento de algoritmos eficientes e eficazes, proporcionando avanços na geração de rotas em redes de computadores. O objetivo desses algoritmos é encontrar soluções (rotas multicast) que permitam uma comunicação mais rápida, menos custosa e mais confiável entre dispositivos, o que é essencial em uma era onde a maioria das pessoas consomem informação e entretenimento pela Internet.

Dado que deseja-se transmitir um conteúdo via uma rede de computadores, o problema consiste em encontrar a melhor rota possível entre a fonte de dados e os vários destinos. dado um grafo $G = (V, E)$ que representa a rede de comunicação, um nó raiz $s \in V$ (nó transmissor) e um conjunto de nós destinos $D \subset V$ (nós receptores), o PRM consiste em determinar a subárvore T de G enraizada em r que inclui todos os vértices em D e apresenta o menor custo possível. A Figura 8 ilustra uma instância do problema de roteamento multicast. Nesse exemplo de rede de comunicação, o nó receptor é definido por um círculo duplo (vértice 0), os nós receptores são destacados em cinza escuro (vértices

1, 8, 12 e 13) e os nós intermediários estão em cinza claro. Os números em cada aresta determina o custo da comunicação entre os nós que ela conecta.

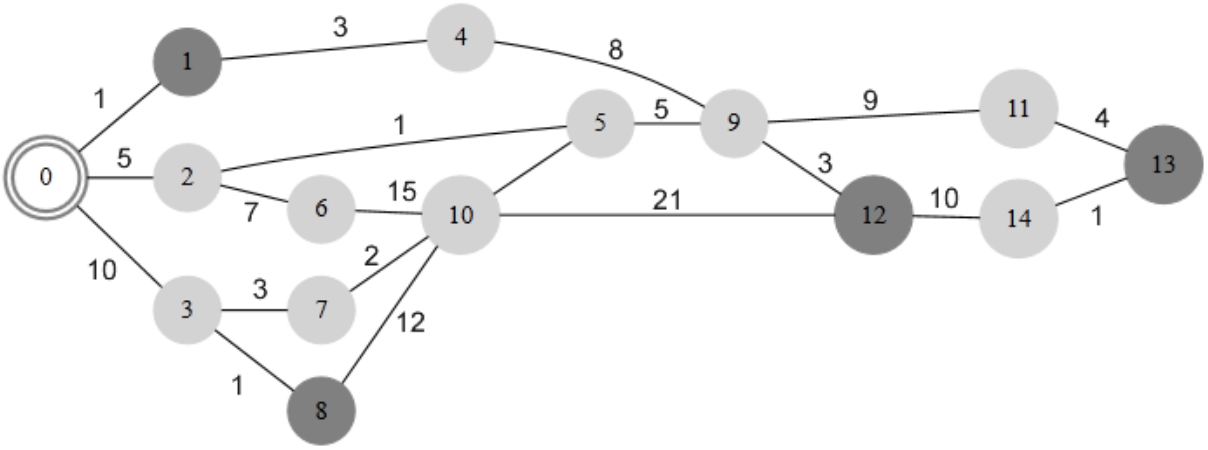


Figura 8 – Exemplo de uma rede de comunicação. Retirado de (BUENO, 2010)

Na Figura 9 são apresentados alguns exemplos de árvores multicast criadas a partir do grafo mostrado na figura Figura 8. O custo de cada árvore é dado pela soma dos custos de suas arestas. Dentre os exemplos, a árvore mais à direita possui o menor custo total: 65.

O PRM original é proposto com apenas um objetivo a se otimizar, entretanto, essa não é uma visão realista do problema. A qualidade de um enlace de rede não pode ser medida através de uma única métrica, ou seja, um custo genérico não é capaz de determinar se um link é bom ou ruim. Características como distância, atraso (*delay*), capacidade de tráfego e uso do tráfego são boas métricas da qualidade de serviço (QoS) em uma comunicação (LAFETÁ, 2016).

Nesse contexto, este trabalho investiga uma versão multiobjetivo do problema de roteamento multicast. Nesta versão do problema, as árvores apresentadas como solução devem representar o melhor compromisso entre as métricas utilizadas.

Na Figura 10 é apresentado um exemplo de rede com as métricas "custo" (primeiro valor) e "delay" (segundo valor) nas arestas. As arestas em negrito representam uma árvore multicast ótima (não-dominada) para o seguinte conjunto de objetivos (LAFETÁ et al., 2016):

1. Minimizar o custo total, que corresponde à soma dos valores de custo de todas as arestas da árvore.

2. Maximizar o atraso (*delay*) fim-a-fim atendidos, ou seja, o número de ramos da árvore em que a soma dos *delays* nas arestas não ultrapassa um valor d_{max} pré-definido, neste caso 25. Em outras palavras, quantidade de conexões cliente-servidor que mantém o limite aceitável de atraso.

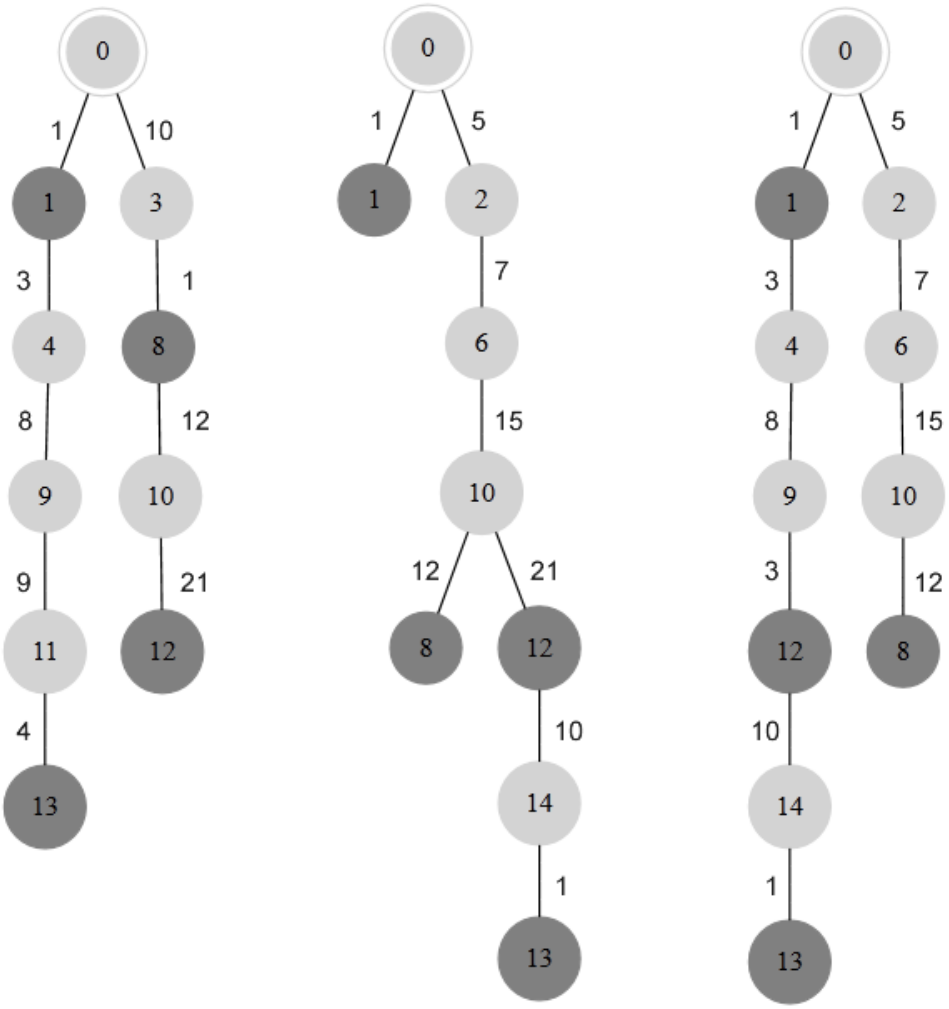


Figura 9 – Exemplos de árvores multicast relativos ao grafo da figura Figura 8. Retirado do trabalho de (LAFETÁ, 2016)

Neste trabalho considera-se até quatro valores de peso para um enlace da rede: custo, *delay*, capacidade de tráfego e tráfego corrente, representados respectivamente pelas funções: $c()$, $d()$, $z()$ e $t()$. Por meio dessas medidas são formulados os seguintes objetivos:

1. **Custo total:** soma dos valores de custo de todas as arestas da árvore.
2. **Delay fim-a-fim médio:** média da soma dos *delays* em cada ramo da árvore. Em outras palavras, média do atraso em cada uma das comunicações cliente-servidor.
3. **Delay fim-a-fim máximo:** maior valor para a soma de *delays* dentre todos os ramos da árvore, ou seja, o maior atraso dentre todas as comunicações cliente-servidor.
4. *Hops count*: número de vértices na árvore.
5. **Utilização máxima de enlaces:** considerando todas as arestas na árvore, qual delas atinge a maior utilização de banda? Matematicamente, considerando E o

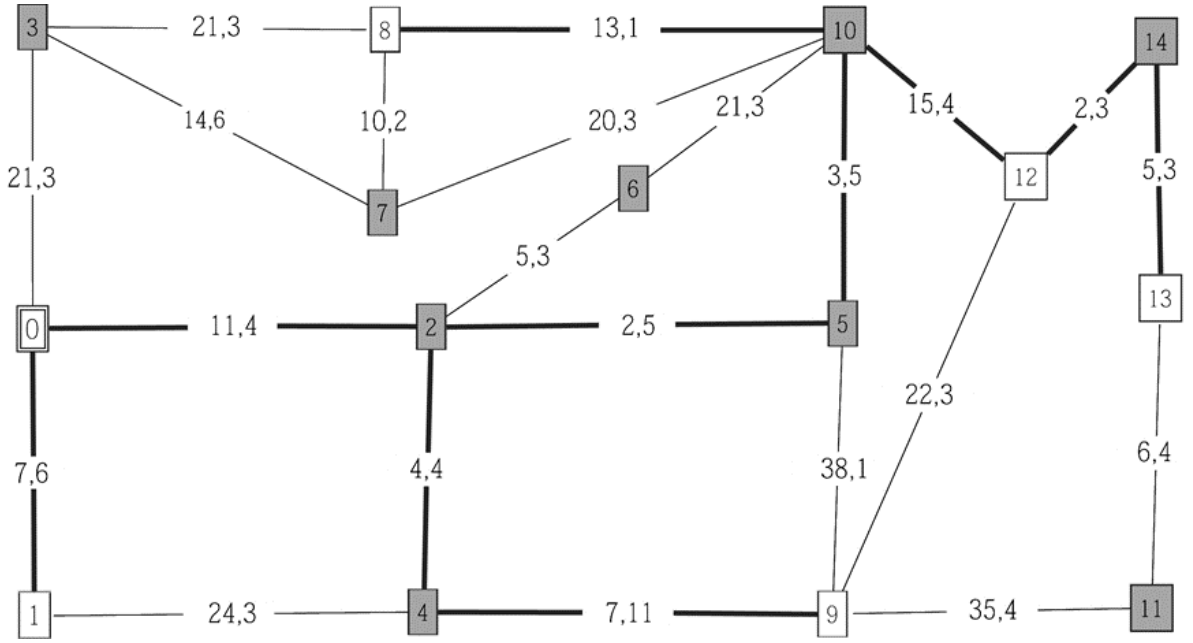


Figura 10 – Exemplo de árvore multicast no PRM multiobjetivo. Retirado de (BUENO; OLIVEIRA, 2010)

conjunto de arestas da árvore e ϕ o tamanho da mensagem, essa métrica é dada por:

$$\max_{e \in E} \frac{t(e) + \phi}{z(e)}$$

6. **Utilização média dos enlaces:** média da utilização de banda entre todas as arestas da árvore. Seu cálculo é similar ao da métrica anterior (utilização máxima de enlaces) e é dado por:

$$\frac{\sum_{e \in E} \frac{t(e)+\phi}{z(e)}}{|E|}$$

A fim de possibilitar diversos cenários de teste para o PRM, os objetivos acima podem ser combinados de diversas maneiras, criando vários ambientes multi-objetivos. Neste trabalho foram utilizadas 5 formulações (combinações) desses objetivos:

1. P_2: formado pelos objetivos 1 e 3.
2. P_3: formado pelos objetivos 1, 3 e 4.
3. P_4: formado pelos objetivos 1, 3, 4 e 5.
4. P_5: formado pelos objetivos 1, 3, 4, 5 e 6.
5. P_6: formado pelos objetivos 1, 2, 3, 4, 5 e 6.

O PRM foi trabalhado sobre 5 redes diferentes variando a complexidade em termos de quantidade de nós destinos, vértices e arestas. Essas redes foram retiradas do trabalho de (LAFETÁ et al., 2016) e suas caraterísticas são apresentadas na tabela 1.

Tabela 1 – Definições das redes utilizadas no PRM

Nome	Destinos	Vértices	Arestas
Rede 1 (R1)	10	33	106
Rede 2 (R2)	18	75	188
Rede 3 (R3)	37	75	188
Rede 4 (R5)	12	75	300
Rede 5 (R5)	16	100	250

Capítulo 5

Estratégias evolutivas para o PMM

Uma das partes mais importantes de um algoritmo de busca bio-inspirado é a modelagem da solução. Para os algoritmos genéticos é necessário definir a representação da solução, a geração aleatória de indivíduos, o cruzamento e a mutação. Para os ACOs, é possível utilizar a mesma representação de solução do AG, mas deve-se desenvolver um algoritmo que constrói a solução a partir de uma estrutura de feromônios e uma heurística.

A implementação de um AG para o problema da mochila mono-objetivo é trivial, pois a solução é representada por um vetor binário e a literatura está repleta de exemplos que podem ser resolvidos dessa maneira (USLU, 2015; HRISTAKEVA; SHRESTHA, 2013). A versão *many-objective* do problema não requer nenhuma modificação no modelo, fazendo com que o mesmo processo de cruzamento e mutação possam ser utilizados. No entanto, a implementação de um ACO para essa versão do problema pode ser desafiador, já que as colônias de formigas esperam trabalhar com grafos e não vetores de bits. Um estudo extensivo relativo aos trabalhos encontrados na literatura que adotam ACOs para a resolução do problema da mochila foi realizado (KE et al., 2010; KONG; TIAN; KAO, 2008; CHANGDAR; MAHAPATRA; PAL, 2013; ALAYA; SOLNON; GHEDIRA, 2004; ALAYA; SOLNON; GHEDIRA, 2007). Nas seções a seguir, detalhe-se a representação da solução, os operadores genéticos e a construção de soluções para o PMM usados neste trabalho.

5.1 Representação da solução

Em ambas as estratégias bio-inspiradas, AGs e ACOs, uma solução para o PMM é representada por um vetor binário. Se a instância do problema da mochila apresenta 10 itens ao total, por exemplo, um vetor com 10 bits representa a solução. As posições do vetor representam a ausência ou a presença de cada item. Se a posição i do vetor é 0, então o i-ésimo item não faz parte da solução. Caso contrário, se o vetor na posição i vale 1, então o i-ésimo item está presente na mochila. Por exemplo, em uma solução representada pelo vetor [1,0,0,1,0,1,1,0,0,0], apenas os itens 0, 3, 5 e 6 são colocados na

mochila, os outros itens (1, 2, 4, 7, 8 e 9) são descartados.

Nesse cenário, a geração aleatória de uma solução consiste em sortear os valores 0 ou 1 para cada posição do vetor. Tanto a criação aleatória, quanto a combinação de vetores (cruzamento) não garantem a formação de uma solução válida, ou seja, é possível que se crie um vetor, cuja soma dos pesos dos itens ultrapasse a capacidade da mochila. Portanto, após a geração de qualquer solução, seja aleatória ou proveniente de um *crossover*, é necessário executar um processo de validação da solução. Para validar um vetor binário, basta verificar a soma dos pesos associados aos itens representados por 1. Caso esse valor seja maior que a capacidade da mochila, remove-se um item aleatório. Esse processo é repetido sucessivamente, até que a solução se torne válida, ou seja, até que a soma dos pesos dos itens respeite a capacidade da mochila (ISHIBUCHI; AKEDO; NOJIMA, 2015).

5.2 Cruzamento e mutação (AG)

O cruzamento entre duas soluções binárias, como explicado na Seção 2.1, pode ser efetuado de diversas maneiras. Neste trabalho, foi utilizado *crossover* uniforme, ou seja, o filho herda de forma aleatória os bits do pai 1 ou do pai 2. Esse processo é ilustrado na Figura 11.

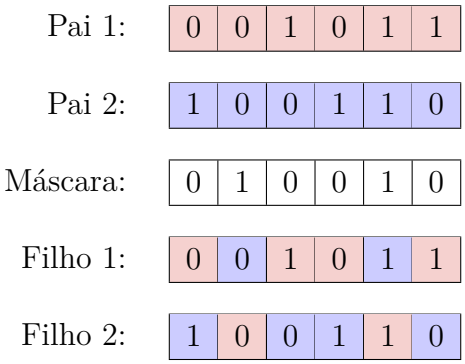


Figura 11 – Exemplo de crossover uniforme

Como pode ser visto na figura 11, o *crossover* uniforme pode ser implementado com uma máscara, que é um vetor aleatório de bits que controla os genes herdados de cada filho. Se o bit na posição i da máscara vale 0, então o gene do filho na posição i herda o valor do pai 1, caso contrário, herda o valor do pai 2. Dessa forma, ainda é possível gerar 2 filhos: um usando a regra na qual o bit 0 da máscara representa o pai 1 e o bit 1 representa o pai 2 (filho 1 na figura); e outro usando os genes complementares (filho 2).

Após o *crossover*, existe uma chance, determinada pela taxa de mutação definida para o AG, do indivíduo sofrer alguma mutação genética. Neste trabalho, foi utilizado o método mais simples de mutação para vetores binários, ou seja, a inversão de bit. Esse

processo consiste em sortear uma posição aleatória do vetor e se o valor for 0, troca-se para 1, caso contrário, troca-se para 0.

5.3 Construção da solução (ACO)

As colônias de formigas foram propostas inicialmente para problemas em grafos, e, portanto, não é intuitivo sua adaptação para outros tipos de problemas, como o problema do mochila. Entretanto, como mostrado em (KE et al., 2010), não é necessário ter um grafo para se utilizar a técnica, sendo possível manipular os feromônios de diversas outras formas. Foram encontradas na literatura três diferentes formas de lidar com os feromônios para o problema da mochila multiobjetivo (PMM). A primeira abordagem consiste em depositar feromônios em cada um dos itens (LEGUIZAMON; MICHALEWICZ, 1999). Sempre que um item é escolhido para compor a solução, incrementa-se a quantidade de feromônios presente no item. Dessa forma, a quantidade de feromônio em cada item representa sua preferência em relação aos demais. A segunda abordagem propõe a criação de um grafo direcionado que representa a preferência em incluir um item b após a inclusão do item a (FIDANOVA, 2003). Dessa forma, sempre que se escolher um item a e posteriormente um b, deposita-se feromônio na aresta (a, b) do grafo. Ao construir uma solução, analisa-se o último item selecionado e todas as arestas no grafo que partem dele. O destino com a maior quantidade de feromônios representa o item preferível. A terceira estratégia foi proposta por (ALAYA; SOLNON; GHEDIRA, 2004) utiliza um conceito semelhante a abordagem anterior. Entretanto, ao invés de depositar feromônios apenas em pares consecutivos, eles são depositados para todos os pares de objetos existentes na solução. Por exemplo, se na solução os itens a, d e f estão na mochila e no passo atual adiciona-se o item c, o depósito de feromônios é feito nas arestas (a, c), (d, c) e (f, c), de forma que o grafo represente a preferência de se escolher um item dado que algum outro item já tenha sido selecionado. Assim, ao construir a solução, deve-se analisar todas as arestas com origem em algum item já presente na solução, o destino da aresta com maior quantidade de feromônios representa o item mais desejável. Neste trabalho utilizou-se a primeira estratégia que deposita os feromônios diretamente nos itens.

Além dos feromônios, outra importante fonte de informação para se construir uma solução no ACO é a heurística. Ela é responsável por estimar o quão vantajoso é escolher um item em relação a outro. Tomando como inspiração o estudo de (KE et al., 2010), nossas implementações do ACO utilizam como heurísticas o seguinte modelo de funções:

- Para cada objetivo k (lucro do item), cria-se uma função de heurística h_k que recebe dois parâmetros, a capacidade restante (cr) e o item que se deseja incluir. A capacidade restante é a diferença entre a capacidade da mochila e a soma dos pesos

dos itens que já foram incluídos. A heurística é então dada por:

$$h_k(item, cr) = lucro_k(item) \times (1 - \frac{peso(item)}{cr})$$

.

- Uma heurística extra é usada exclusivamente para se referir ao peso do item, a qual é dada por:

$$h_{peso}(item) = 1 - \frac{peso(item)}{peso\_maximo}$$

.

Logo, o número de heurísticas no PMM é sempre o número de objetivos + 1. Tendo definido a estrutura de feromônios e as heurísticas, para construir uma solução, o algoritmo recebe a função heurística (h) e um vetor de feromônios (τ). Inicialmente, a lista de exploração Exp contém todos os itens possíveis, a probabilidade p de cada item $exp_i \in Exp$ é calculada de acordo com a heurística de exp_i ($h(exp_i)$) e a quantidade de feromônios no item tau_i, como na fórmula seguinte:

$$p(exp_i) = \frac{\tau_i^\alpha + h(exp_i)^\beta}{\sum_{j \leftarrow 0}^{tamanho(Exp)} \tau_j^\alpha + h(exp_j)^\beta}$$

O próximo item na solução é escolhido de acordo com as probabilidades calculadas em uma espécie de roleta, onde, quanto maior o valor de probabilidade ($p(exp_i)$), maior a chance do item ser escolhido. Após ter selecionado um item, recalcula-se a lista Exp para que contenha apenas itens válidos para a solução, isto é, remove-se qualquer elemento de Exp que, se incluído na solução, excede a capacidade da mochila. O processo é repetido até que Exp seja uma lista vazia.

A fim de reduzir a complexidade do algoritmo de construção da solução para o PMM, ao invés de se utilizar a lista completa de itens possíveis (Exp), utiliza-se uma amostra desse conjunto (Exp'). Exp é calculado normalmente, mas ao submeter-se o conjunto para o cálculo de probabilidades e roleta, no lugar de Exp, é utilizado Exp', uma amostra aleatória de tamanho fixo Exp'_{size} (parâmetro do algoritmo) da lista completa Exp.

Capítulo 6

Estratégias evolutivas para o PRM

A modelagem de um algoritmo genético para o problema do roteamento multicast não é trivial, pois a solução não pode ser representada por um vetor. De fato, como deve representar os caminho entre o servidor e os múltiplos destinos em uma rede de computadores, a solução para o PRM é melhor representada por uma árvore. Dessa forma, é preciso desenvolver o processo de crossover e de mutação de acordo com essa estrutura. O operador de cruzamento deve receber duas árvores pais e gerar uma nova árvore (filha) que compartilha características de ambos os pais. O operador de mutação precisa criar uma pequena alteração na árvore que permita explorar diferentes regiões do espaço de busca, mas que não a descaracterize completamente.

Em contrapartida, o PRM se aproxima da definição original do ACO, pois trabalha com grafos. A diferença está no fato de que a solução é representada por uma árvore ao invés de um simples caminho, fazendo com que o depósito de feromônio e a escolha de arestas sejam desenvolvidas conforme a nova estrutura.

6.1 Representação da solução

Como mostrado na seção 4.2, considerando que em cada nó da rede a mensagem pode ser replicada e enviada aos próximos nós conectados, o PRM deseja encontrar a árvore que representa o processo de transmissão de menor custo que parte do nó fonte (servidor) e atinge todos os destinos. Existem duas maneiras de se representar uma solução:

1. **Representação em árvore:** (BUENO; OLIVEIRA, 2010) o AG evolui a própria árvore que se deseja encontrar como solução. É um processo mais complicado que elimina a necessidade de pós-processamento. A Figura 9, na seção 4.2, mostra alguns exemplos de árvores multicasts.

2. **Representação em conjunto:** (CRICHIGNO; BARAN, 2004) o AG evolui um conjunto de caminhos C, ou seja, para cada nó destino d, deve existir uma sequência de nós $L \in C$ que contém o caminho que leva do nó servidor (emissor) até o nó d.

A representação em conjuntos de caminhos é mais fácil de se gerenciar, mas exige a transformação (conjunção) dos caminhos pertencentes ao conjunto em uma árvore ao final do processo. Como diferentes árvores podem ser formadas a partir de um único conjunto de caminhos, essa representação não é tão eficiente quanto a anterior ao explorar o espaço de busca.

Neste trabalho, optou-se por utilizar a representação em árvores.

6.2 Inicialização dos indivíduos

Considere um grafo da rede G, um nó raiz (também chamado de servidor ou emissor) r e um conjunto de nós de destino D, para criar uma solução aleatória no PRM, inicia-se um grafo S apenas com o vértice r, o passo seguinte é extrair um destino aleatório $d \in D$ e, com base no grafo G, criar um caminho em S entre qualquer um de seus vértices atuais e d. Após construir o caminho, d com certeza será atingível a partir de qualquer vértice de S. O processo se repete até que todos os nós destinos estejam no grafo S. Por fim, o grafo S é transformado em uma árvore a partir da remoção dos ciclos presentes em S e da aplicação de podas dos ramos desnecessários.

Para criar o caminho aleatório entre um vértice de S e um nó destino d, considera-se um vetor de exploração Exp que, inicialmente, contém todos os nós de S. Enquanto o destino d não é encontrado, algum nó $exp_i \in Exp$ é escolhido e todos os vértices adjacentes (conectados) a exp_i são incluídos em Exp. Quando d é encontrado, adiciona-se a S o caminho necessário para sair de algum vértice de S e chegar ao destino d.

6.3 Cruzamento (AG)

A estratégia de cruzamento utilizada neste trabalho para o PRM é chamada de cruzamento por caminho e foi proposta em (LAFETÁ et al., 2016) como alternativa ao cruzamento por similaridade utilizado em trabalhos anteriores (BUENO; OLIVEIRA, 2010). Esse tipo de cruzamento é realizado entre duas árvores P_1 e P_2 e produz um único filho F. O processo consiste em separar cada um dos pais em ramos e, então, para cada nó destino d, acrescentar a F o ramo de P_1 ou de P_2 que leva a d. A escolha entre P_1 e P_2 é feita de forma aleatória. Assim que todos os nós destinos são atingíveis em F, a etapa de seleção de ramos é interrompida, os ciclos que possivelmente foram criados são removidos, e um processo de poda é realizado a fim de remover qualquer nó folha que não seja destino.

A figura 6.3, retirada do trabalho de (LAFETÁ et al., 2016), representa o processo de cruzamento por caminho entre duas árvores (Pai 1 e Pai 2). No exemplo, o nó raiz é o vértice 0 e os nós destinos são o conjunto $\{1, 8, 11\}$. As setas em "ramos do pai 1" e "ramos do pai 2" representam os caminhos escolhidos em cada um dos pais para compor a

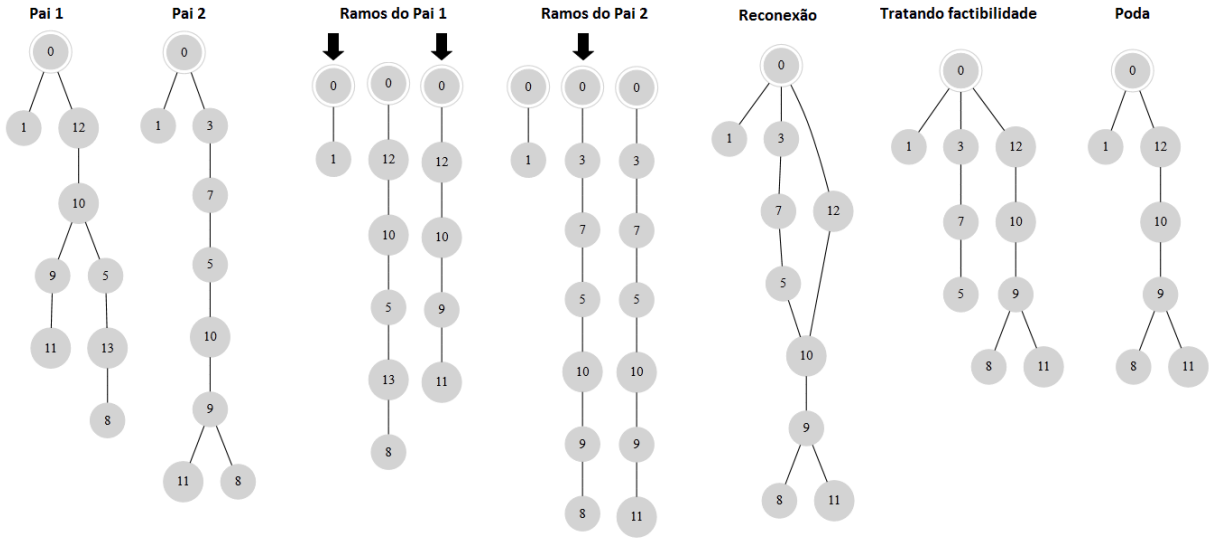


Figura 12 – Exemplo de cruzamento por caminho. Retirado de (LAFETÁ, 2016)

árvore filha. O grafo nomeado "reconexão" representa o filho após a inclusão de todos os ramos. Como foi gerado um ciclo, o mesmo deve ser removido a fim de obter uma árvore válida. Em "tratando factibilidade", apresenta-se o filho após a remoção dos ciclos. Como o nó folha 5 não é um destino, ele é removido durante o processo de poda, resultando na última árvore ("poda") que é o filho resultante do processo.

Para remover os ciclos, percorre-se a árvore em largura removendo qualquer aresta que adicione ciclo. No processo de poda, verifica-se todas as folhas. Se algum nó folha não for um destino, ele é removido e o processo repetido até que todos os nós folhas sejam destinos.

Após o cruzamento, realiza-se um processo de filtragem, onde todas as soluções repetidas são substituídas por novos indivíduos gerados aleatoriamente.

6.4 Mutação (AG)

A mutação em uma árvore que representa uma solução para o PRM consiste em remover parte dos nós da árvore e então reconectá-los de maneira aleatória utilizando o grafo correspondente à rede em questão. As etapas desse processo estão descritas no o algoritmo 3.

O algoritmo recebe como entrada a árvore que sofrerá mutação (A), o grafo da rede (G), a quantidade de arestas a se remover na mutação ($qte_{arestas}$), o vértice raiz (r), e o conjunto de destinos (D). Na linha 1, desconecta-se a árvore através da remoção aleatória de $qte_{arestas}$ arestas. Entre as linhas 2 e 4, cria-se a estrutura para a construção da árvore. A partir da linha 5 começa o laço para reconectar todos os nós destinos à arvore. A linha 6 seleciona um destino aleatório d para reconectar. Na linha 7, a componente conexa

Algoritmo 3 Mutação para uma árvore $(A, G, qte_{arestas}, r, D)$

```
 1: desconectar(A, qte_arestas)
 2: C ← extraiComponente(A, r)
 3: M ← novoGrafo()
 4: M ← mesclar(M, C)
 5: enquanto |D| > 0 faça
 6:     d ← removeAleatorio(D)
 7:     C ← extraiComponente(A, d)
 8:     se d ∉ V(M) então
 9:         P ← caminhoAlearorio(C, M, G)
10:         M ← mesclar(M, P)
11:     fim se
12:     M ← mesclar(M, C)
13: fim enquanto
14: removerCiclos(M)
15: podarArvore(M)
16: retorne M
```

contendo d é extraída de A e chamada de C. Se não existe componente conexa com o vértice d, C será um grafo com um único nó (d) e nenhuma aresta. Caso o vértice d não exista na componente conexa M, cria-se um caminho aleatório em M (com base no grafo G) que a conecta a C (linhas 8 a 10). A construção do caminho aleatório é feita nó a nó até se encontrar uma sequência de arestas entre as duas componentes. Na linha 11, inclui-se em M todas as arestas de C. Ao final, o mesmo pós-processamento do cruzamento por caminho é realizado: remoção de ciclos e poda da árvore (linhas 14 e 15).

6.5 Construção da solução (ACO)

O processo de construção da solução por um ACO no PRM deve gerar a árvore *multicast* com base no grafo da rede, da estrutura de feromônios e da heurística. Existem diversas maneiras de se criar tal árvore. A fim de estudar o comportamento das diferentes estratégias possíveis, aprimorá-las e determinar o melhor modelo para se utilizar no PRM, analisou-se o comportamento de cada ideia no caso mais básico possível: o PRM mono-objetivo. As ideias consideradas neste trabalho são listadas a seguir:

1. A primeira estratégia, apresentada em (PINTO; BARAN, 2005), pode ser vista como uma formiga que caminha pelo grafo até encontrar todos os destinos. A análise é feita passo a passo, considerando apenas a vizinhança do vértice corrente como possibilidades para compor a solução. O processo inicia com uma lista de exploração (Exp) que contém um único elemento: a raiz. Sorteia-se um vértice $i \in Exp$ e atribui-se probabilidades a todas as arestas (i, j), onde j é qualquer vértice não-visitado na adjacência de i. i é removido de Exp caso não possua vizinhos factíveis. Um vértice v é escolhido de acordo com as probabilidades calculadas (conforme à

seção 2.2.2), a aresta (i, v) é adicionada à solução e v é inserido na lista de exploração Exp. O processo é repetido até que todos os destinos tenham sido alcançados. Como etapa final, poda-se a árvore, eliminando as folhas que não representam destinos.

2. Uma solução pode ser vista como os caminhos entre a raiz e cada um dos destinos. Portanto, outra estratégia é imaginar que $|D|$ formigas partirão da raiz e cada uma deve encontrar um vértice $d \in D$ diferente, sendo D o conjunto de nós destinos. Com os caminhos para todos os nós destinos definidos, monta-se a árvore evitando a ocorrência de ciclos. Por fim, realiza-se a poda da árvore para excluir qualquer vértice folha que não seja um destino.

3. Uma terceira estratégia é proposta neste trabalho. Ela adota uma representação de formiga fictícia que pode estar em vários locais do grafo ao mesmo tempo (formiga com sobreposição quântica). Dessa forma, é possível analisar as probabilidades de todas as arestas factíveis ao mesmo tempo, ao invés de sempre escolher a composição da solução de acordo com uma vizinhança local. Ao considerar todas as possibilidades ao mesmo tempo, obtém-se um processo melhor de decisão que não apresenta qualquer tendência (viés) para algum dos ramos da árvore, já que a todo momento, qualquer vértice factível pode ser incluído no resultado. O processo é iniciado a partir de uma lista de exploração (Exp) que contém todas as arestas conectadas ao nó raiz. A cada passo, calcula-se as probabilidades para toda aresta de Exp e, de acordo com os valores obtidos, escolhe-se $exp_i \in Exp$ para compor a solução. exp_i é então removida de Exp, adicionada à solução e tem todas suas arestas adicionadas à lista de exploração, desde que ainda não tenham sido incluídas. O processo é repetido até que todos os destinos sejam atingidos. Uma poda é realizada ao final do processo.

4. A última estratégia consiste em construir a solução de forma inversa. Ao invés de partir da raiz e chegar aos destinos, este modelo propõe que se utilize $|D|$ formigas, onde cada uma tem como posição inicial um nó destino $d \in D$ diferente. A ideia é que as formigas escolham seus caminhos localmente com base nas probabilidades das arestas em suas vizinhanças. Sempre que uma formiga encontrar o caminho que já foi explorado por outra formiga, ela para de explorar e segue os mesmos passos realizados pelo outro agente. O processo termina quando o nó raiz foi encontrado por alguma formiga e todas as formigas tiverem se encontrado em algum vértice. Em outras palavras, o processo inicia com uma componente conexa para cada $d \in D$. Em todo passo do algoritmo, cada componente conexa é explorada a partir do último nó adicionado. Nesse processo, calcula-se as probabilidades das arestas na vizinhança do nó explorado e escolhe-se, de acordo com os valores obtidos, um novo vértice v para ser adicionado à componente. Se não há arestas factíveis na vizinhança, deve-se retroceder para um vértice anterior. Caso o nó incluído v pertença a uma outra

componente conexa, elas se unem. O processo termina quando existe apenas uma componente conexa e o vértice raiz foi encontrado. Por fim, uma poda é realizada como pós-processamento da árvore.

Cada uma das estratégias mencionadas foi implementada e testada a fim de determinar aquela que produz as soluções de melhor qualidade para o PRM. Para obter um valor como parâmetro de qualidade, também foi implementada uma versão modificada do algoritmo de Prim (PRIM, 1957) que aproxima a árvore *multicast* de menor custo. Como esperado, o algoritmo de Prim é uma opção melhor que o ACO quando se trata do PRM mono-objetivo. Entretanto, a intenção deste estudo não é produzir um algoritmo melhor para o problema, mas avaliar as estratégias de construção de solução com o objetivo de escolher aquela que será empregada em versões mais complexas (multiobjetivo) do PRM.

Os resultados dos testes, que podem ser encontrados no capítulo de experimentos (seção 8.2), mostram que nossa estratégia (formiga com sobreposição quântica) representa a melhor relação entre qualidade do resultado e tempo de execução. Portanto, para todos os demais experimentos no PRM, foi o método de construção da solução utilizado.

Durante a implementação da estratégia, foram adotadas algumas técnicas de amostragem a fim de agilizar o algoritmo, sem afetar significativamente a qualidade das soluções. A descrição detalhada dessa implementação é apresentada no Algoritmo 4.

Algoritmo 4 Geração de solução no ACO $(G, r, D, \tau, h, Exp'_{size})$

1: Inicie uma árvore vazia T
2: Inicie Exp com todas as arestas que possuem alguma extremidade em r
3: Marque r como visitado
4: **enquanto** T não incluir todos os vértices em D **faça**
5: Crie uma amostra aleatória Exp' a partir da lista Exp com Exp'_{size} elementos
6: Calcule as probabilidades de todas as arestas em Exp' de acordo com τ e h
7: Escolha uma aresta $e = (i, j) \in Exp'$ de acordo com as probabilidades
8: Inclua e em T
9: Marque j como visitado
10: Calcule a vizinhança V do vértice j
11: **para** $v \in V$ **faça**
12: **se** já existe aresta a em Exp que leva a v **então**
13: Remova a de Exp
14: Calcule as probabilidades de a e de (j, v) de acordo com τ e h
15: Sorteie uma das duas arestas de acordo com as probabilidades e adicione a vencedora em Exp
16: **senão se** v não tiver sido visitado **então**
17: Inclua (j, v) em Exp
18: **fim se**
19: **fim para**
20: **fim enquanto**
21: Pode a árvore T
22: **retorne** T

O Algoritmo 4 recebe como parâmetros de entrada:

- G: o grafo que representa a rede;
- r: o nó raiz, servidor de onde parte a mensagem;
- D: conjunto de nós destinos;
- τ: estrutura de feromônios;
- h: função heurística;
- Exp'_{size}: tamanho da amostra.

Trabalhar com todas as arestas possíveis é demasiadamente caro e inviável para um algoritmo em que se deseja bom desempenho em termos de tempo de execução. Por isso, trabalha-se com uma amostra da lista de exploração (linha 5 do algoritmo). Outro processo de simplificação também é empregado para evitar o crescimento exagerado de Exp durante a etapa de exploração, nas linhas 12 a 16. Nesse processo, caso um novo vértice descoberto já seja atingível a partir de alguma aresta em Exp, mantém-se em Exp apenas uma das arestas. Para escolher qual das arestas manter, calcula-se as probabilidades de acordo com os feromônios (τ) e a heurística (h).

Com relação às heurísticas no PRM, uma função é construída para cada métrica de rede presente na formulação de objetivos. Por exemplo, no problema P_4, cujos objetivos envolvem as métricas custo, *delay*, tráfego e capacidade, quatro heurísticas são criadas:

$$\begin{cases} h_1(e) = 1 - custo(e) \\ h_2(e) = 1 - delay(e) \\ h_3(e) = 1 - trafego(e) \\ h_4(e) = capacidade(e) \end{cases}$$

Onde $custo(e)$, $delay(e)$, $trafego(e)$ e $capacidade(e)$ correspondem, respectivamente, ao valor normalizado entre 0 e 1 para o custo, *delay*, tráfego e capacidade da aresta e. Na heurística h_4 não é feito o complemento da função, pois essa é a única métrica que deve ser maximizada.

Capítulo 7

Algoritmo proposto

O algoritmo proposto neste trabalho é baseado em colônia de formigas (ACO) e foi chamado de *Many-objective Ant Colony Optimization based on Decomposed Pheromone* (MACO/D), em português, otimização em colônia de formigas para muitos objetivos baseada em decomposição de feromônios (FRANÇA; MARTINS; OLIVEIRA, 2018). A proposição envolve os seguintes módulos:

- *Framework* ACO: algoritmo geral desenvolvido para se trabalhar com qualquer problema discreto.
- Modelo PMM: construção da solução para o problema da mochila multiobjetivo, apresentado na seção 5.3.
- Modelo PRM: construção da solução para o problema do roteamento multicast, apresentado na seção 6.5.

Este capítulo apresenta o módulo principal do algoritmo, o *framework*. framework. Nela são discutidos alguns conceitos gerais sobre a implementação multiobjetivo do ACO, bem como uma descrição mais detalhada das etapas de construção das soluções e atualização dos feromônios.

7.1 Visão geral do algoritmo

O MACO/D se baseia na ideia do AEMMD de se criar tabelas de dominância para diferentes combinações possíveis de objetivos, adaptando-na para um modelo de colônia de formigas.

A multiplicidade de objetivos impõe que algumas mudanças sejam feitas na ideia original do ACO. Isso envolve responder as seguintes perguntas:

- Como representar os diferentes objetivos no depósito de feromônio?

- Como compor as múltiplas heurísticas em uma única função?

De acordo com (ALAYA; SOLNON; GHEDIRA, 2007), essas questões podem ser resolvidas através de uma das seguintes opções:

- **Várias colônias e múltiplos feromônios** $(m+1, m)$**:** considerando m o número de objetivos, esse modelo utiliza $m+1$ colônias de formigas e m estruturas de feromônios. Cada colônia é associada a um único objetivo e otimiza apenas esse objetivo através de sua própria estrutura de feromônios. Uma colônia adicional, que representa o conjunto de todos os objetivos, utiliza os feromônios das outras colônias de forma aleatória. Ao criar uma solução, sorteia-se alguma das estruturas de feromônios para se utilizar a cada passo. Outra proposta, no modelo $(m+1, m)$ é utilizar a soma de todas as estruturas de feromônios ao criar uma solução na colônia extra. Quanto as heurísticas, cada colônia utiliza a função referente a seu objetivo, enquanto a colônia extra utiliza o somatório de todas as funções de heurística.

- **Colônia e feromônio únicos** $(1, 1)$**:** este modelo utiliza apenas uma colônia de formigas e uma estrutura de feromônios. Assim como no modelo anterior, as heurísticas são somadas para montar uma solução. A única estrutura de feromônios representa todos os objetivos. A atualização dos feromônios se dá através de um arquivo que mantém todas as soluções não-dominadas encontradas. Toda solução não-dominada contribui com a mesma quantidade de feromônios, já que, de fato, são indiferenciáveis em termos de qualidade.

- **Única colônia e vários feromônios** $(1, m)$**:** considerando m o número de objetivos, esse modelo utiliza uma colônia de formigas e m estruturas de feromônios. Assim como nos demais modelos, as heurísticas são somadas para montar uma solução. A cada passo da construção da solução, sorteia-se aleatoriamente a estrutura de feromônios a se utilizar. Cada estrutura de feromônios representa um objetivo e sua atualização é feita a cada iteração do algoritmo, de acordo com a melhor solução encontrada considerando aquele objetivo.

Em (ALAYA; SOLNON; GHEDIRA, 2007), os experimentos mostraram que o terceiro modelo $(1, m)$ gera os melhores resultados quando aplicado ao problema da mochila multiobjetivo. Dentre os ACOs multiobjetivos investigados, o MOACS se encaixa na segunda abordagem (1,1), adaptando apenas o modo de lidar com a heurística, enquanto o MOEA/D-ACO adota a primeira estratégia (m,m), pois trabalha com múltiplas formigas e várias estruturas de feromônios, apesar da quantidade exata de formigas e tabelas de feromônios não ser ditada pelo número de objetivos.

Nosso algoritmo (MACO/D), não pertence a nenhuma das categorias apresentadas por (ALAYA; SOLNON; GHEDIRA, 2007), mas se aproxima da estratégia (1,m), pois

lida com uma única colônia e múltiplos feromônios. A grande diferença é que o número de estruturas de feromônio não é igual a quantidade de objetivos, mas ao número de possíveis combinações entre os objetivos, de forma semelhante ao que ocorre no AEMMD (LAFETÁ et al., 2018). Além disso, o processo de cálculo da heurística e de atualização dos feromônios se difere substancialmente da ideia original do modelo $(1, m)$. O MACO/D, se baseia na ideia de decomposição, ou seja, ao invés de trabalhar diretamente com todos os objetivos, o problema é atacado em várias frentes com quantidades reduzidas de funções. Desta forma, evita-se o problema onde não é possível classificar a população devido ao alto número de soluções não-dominadas em espaços de dimensionalidade alta (DEB; JAIN, 2014). O Algoritmo 5 apresenta uma visão em alto nível do MACO/D.

Algoritmo 5 Algoritmo geral do MACO/D

1: $P \leftarrow criarEstruturasDeFeromonios()$
2: $P_{ativo} \leftarrow \{P[0], P[1], ..., P[4]\}$
3: **enquanto** condição de parada não for atingida **faça**
4: $\quad S \leftarrow criarSolucoes(P_{ativo})$
5: $\quad atualizarArquivo(P[tamanho(P) - 1].arquivo, S)$
6: $\quad S_{nd} \leftarrow P[tamanho(P) - 1].arquivo$
7: $\quad atualizarEstruturasDeFeromonios(P_{ativo}, S_{nd})$
8: **fim enquanto**
9: **retorne** S_{nd}

O primeiro passo do MACO/D (linha 1 do Algoritmo 5) é criar as estruturas de feromônio $P = \{p_1, p_2, ..., p_i, ..., p_n\}$, uma para cada combinação possível de objetivos (análogo ao processo do AEMMD apresentado na seção 3.1.7). Para um problema de seis objetivos, por exemplo, são criadas 57 estruturas ($|P| = 57$). Cada $p_i \in P$ é responsável por um subproblema e guarda as seguintes informações:

- **Valores:** corresponde aos valores dos feromônios em si, os quais são inicializados com o menor valor possível.
- **Objetivos:** determina os objetivos do subproblema em questão. É um vetor binário de tamanho m (número de objetivos), onde cada o bit 1 representa um objetivo que faz parte do problema e o bit 0 indica aqueles que não pertencem ao problema.
- **Arquivo:** conjunto de soluções não-dominadas que apareceram até o momento para o subproblema em questão.
- **Convergência:** indica a convergência do arquivo. Começa em 0 e incrementa em 1 sempre que o arquivo sofre alterações durante uma iteração (época).
- β**:** importância da heurística no cálculo de probabilidades ao construir uma solução. Sempre inicializado com o valor do parâmetro β passado ao MACO/D.

As estruturas de feromônios são utilizadas no processo de construção da solução e devem ser atualizadas em toda iteração do algoritmo. Manipular todas as estruturas de feromônios de P (57 estruturas, no caso de seis objetivos) é computacionalmente caro e inviável. Por essa razão, define-se um número máximo de estruturas para se trabalhar em um dado momento. Em ambos os problemas utilizados neste trabalho (PMM e PRM), foi utilizado um limite de cinco estruturas de feromônios ativas simultaneamente. A ordem de ativação dos elementos de P é determinada pela sua ordem em P. Por isso, é importante a sequência em que as estruturas são criadas: os primeiros subproblemas instanciados são aqueles que representam um menor número de objetivos, ou seja, combinações 2 a 2. Em seguida, instancia-se os subproblemas de 3 objetivos e, assim por diante, até que a última estrutura de feromônios com o número total de objetivos seja criada.

Na linha 2 do Algoritmo 5, os cinco subproblemas mais simples são ativados. Na atualização dos feromônios, assim que algum dos subproblemas passa a não contribuir satisfatoriamente para o conjunto de soluções, é desativado em favor do próximo problema mais complexo ainda não utilizado. Dessa forma, o conjunto de soluções do MACO/D cresce gradualmente, partindo das decomposições mais simples em direção às mais complexas.

O laço principal do ACO, iniciado na linha 3 do algoritmo, consiste em gerar soluções e atualizar as estruturas de feromônios ativas. Na linha 4, é construído o conjunto de soluções S de acordo com os feromônios em P_{ativo}, esse processo é detalhado na seção 7.2. Nas linha 5 e 6, o arquivo de soluções não dominadas geral (considerando todos os objetivos) é atualizado e chamado de S_{nd}. Por último, na linha 7, as soluções de S_{nd} são utilizadas para atualizar cada uma das estruturas em P_{ativo}. Após o término do laço principal, o algoritmo MACO/D retorna todas as soluções não dominadas encontradas (S_{nd}).

O processo de se atualizar as estruturas de feromônio a partir das soluções não dominadas encontradas (linha 7) é mostrado com mais detalhes no Algoritmo 6, que recebe como entrada o conjunto P_{ativo} e o conjunto de soluções S_{nd}

O Algoritmo 6 mostra a atualização das estruturas de feromônios no MACO/D. O processo iterativo (linha 1) passa por todas as estruturas em P_{ativo}. Nas linhas 2 a 5, a estrutura atual é chamada de a e seu arquivo é atualizado de acordo com as soluções no conjunto S_{nd}. Além disso, todas as soluções adicionadas ao arquivo são guardadas na lista A, enquanto todas as removidas são colocadas em R. As linhas 7, 8 e 9 são executadas apenas se o arquivo de a foi atualizado. Na linha 7, o valor *beta* de a é reinicializado para o valor original (parâmetro β do algoritmo); na linha 8, os valores de feromônio de a são incrementados de acordo com as soluções em A; e na linha 9, os valores de feromônios de a são decrementados de acordo com as soluções em R. As operações das linhas 8 e 9 são detalhadas na seção 7.3, sobre a atualização dos feromônios. Os valores de feromônios são alterados apenas se o arquivos tiver sido modificado, por outro lado, caso o arquivo não

Algoritmo 6 Atualização das estruturas de feromônios

```
1: para i ← 0atétamanho(P_ativo) faça
2:     a ← P_ativo[i]
3:     dif = atualizarArquivo(a, S_nd)
4:     A ← dif.added
5:     R ← dif.removed
6:     se |A| > 0 então
7:         a.β ← β
8:         depositarFeromonios(a, A)
9:         evaporarFeromonios(a, R)
10:    senão
11:        a.convergencia ← a.convergencia + 1
12:        a.β ← a.β × β_inc
13:        se a.convergencia > MAX_convergencia então
14:            P_ativo[i] ← proxima_estrura_feromonios,
15:            a.convergencia ← 0
16:        fim se
17:    fim se
18: fim para
```

tenha sofrido atualização, as linhas 11 a 16 são executadas. Na linha 11, a convergência de a é incrementada em 1. Na linha 12, o valor β de a é modificado de acordo com a constante pré-definida β_{inc}, a intenção é que se diminua a importância das heurísticas sempre que a busca parecer estagnada. Assim que novas soluções são encontradas, o valor β de a volta ao normal (linha 7). No caso da implementação utilizada neste trabalho, para diminuir a importância da heurística, deve-se aumentar o valor de β. A quantidade em que se deve alterar esse valor dependerá do problema. Nos nossos experimentos (descritos no capítulo 8), o valor de β aumenta em 10% tanto no PMM quanto no PRM. Por último, nas linhas 13 e 14, se a convergência de a atingiu o limite máximo $MAX_{convergencia}$, substitui-se a estrutura a em P_{ativo} pelo próximo $p_i \in P$, reiniciando o valor da convergência de a para caso seja necessário utilizá-la novamente no decorrer do algoritmo. O valor da constante $MAX_{convergencia}$ utilizado neste trabalho foi 10.

7.2 Construção das soluções

As soluções em colônias de formigas são construídas a partir do grupo de feromônios ativo (P_{ativo}), das heurísticas (H) e dos valores de α e β. Os feromônios são criados e atualizados no decorrer do algoritmo, enquanto os demais são parâmetros de entrada. A heurística é uma função que estima a qualidade de uma parcela da solução, ex: arestas em problemas que envolvam grafos, como o PRM, e itens em problemas representados por vetores, como o PMM. Enquanto o parâmetro α determina a importância do feromônio ao tomar uma decisão a respeito da composição da solução, β representa a importância

da heurística.

No MACO/D existem múltiplas estruturas de feromônios e heurísticas. Portanto, para que se possa construir uma solução, é necessário antes escolher quais valores de feromônio e qual função de heurística serão utilizados.

As estruturas de feromônios são recebidas pelo processo de construção das soluções através da lista de estruturas ativas do MACO/D (P_{ativo}), ou seja, 5 conjuntos de feromônios são utilizados por vez. De maneira circular, cada solução utiliza uma única estrutura de feromônios para ser construída, ou seja, a primeira solução é criada a partir da primeira estrutura de P_{ativo}, a segunda a partir da segunda e, assim por diante, até a sexta solução que utiliza novamente a primeira estrutura de P_{ativo}.

Com relação às heurísticas, o MACO/D utiliza o mesmo processo proposto em (RIVEROS et al., 2016). Admite-se um grau de importância para cada função: 0 (não importante), 1 (importante) ou 2 (muito importante). O valor de importância é sorteado para cada heurística e funciona como um peso. A função única utilizada para construir a solução é a soma ponderada das heurísticas considerando os pesos sorteados.

O processo geral para se construir as soluções é exposto no Algoritmo 7.

Algoritmo 7 Construção das soluções

1: $S \leftarrow \emptyset$
2: **para** $i \leftarrow 0$ até nº máximo de soluções **faça**
3: $\quad W \leftarrow \emptyset$
4: $\quad$ **para** $j \leftarrow 0$ até $|H| - 1$ **faça**
5: $\quad\quad W[j] \leftarrow random(0, 2)$
6: $\quad$ **fim para**
7: $\quad h(x) \leftarrow \sum_{i \leftarrow 0}^{|H|-1} \frac{H[i](x) * W[i]}{\sum_{w \in W} w}$
8: $\quad p \leftarrow P_{ativo}[i \pmod{|P_{ativo}|}]$
9: $\quad s \leftarrow gerarSolucao(p.feromonios, h, \alpha, p.\beta)$
10: $\quad S \leftarrow S \cup \{s\}$
11: **fim para**
12: **retorne** S

O Algoritmo 7 apresenta o processo geral para se construir um conjunto de soluções. Na linha 1, S é inicializado como um conjunto vazio. A partir da linha 2, um laço é executado para gerar todas as soluções necessárias. Nas linhas 3 a 6, $|H|$ valores entre 0 e 2 (inclusive) são sorteados para um vetor W. Na linha 7, é criada a função heurística correspondente à combinação de todas as heurísticas de acordo com as importâncias sorteadas. Na linha 8, chama-se de p a estrutura de feromônio atual. A nova solução é criada na linha 9 de acordo com os feromônios de p, a função heurística h, o parâmetro do α do algoritmo e o valor β de p. Na linha 10, a nova solução é incluída no conjunto de soluções S. Por fim, na linha 11, retorna-se o conjunto S de soluções.

Na linha 6 do algoritmo 7 constrói-se a solução em si. Esse processo depende exclusivamente do problema em questão e representa a principal parte na elaboração do

modelo para um algoritmo baseado em colônia de formigas. No caso do problema da mochila multiobjetivo, a estratégia utilizada é aquela descrita em 5.3. Para o problema do roteamento multicast, a estratégia de construção da solução foi apresentada em 6.5.

7.3 Atualização dos feromônios

A atualização dos feromônios no MACO/D pode ser de dois tipos:

1. **Depósito:** a partir de uma solução, adiciona-se uma quantidade δ ao feromônio correspondente à cada partícula da solução. No caso de um problema em grafos, por exemplo, para cada aresta da solução, incrementa-se em δ o feromônio daquela aresta.

2. **Evaporação:** similar ao depósito, mais ao invés de incrementar o feromônio em δ, decrementa-se.

O depósito de feromônio ocorre quando novas soluções não-dominadas são encontradas. A evaporação no MACO/D, diferentemente da maioria dos algoritmos baseados em ACO, não ocorre para todos os feromônios em todas as iterações. Ao invés disso, o feromônio só é decrementado quando uma solução deixa de ser não-dominada (ver algoritmo 5).

Dada uma solução s do problema do roteamento multicast (PRM), se s deve ser reforçada (depósito), cada aresta e da árvore s incrementa o valor correspondente na estrutura de feromônios em um fator δ. Matematicamente, esse processo de depósito é definido por:

$$\delta(e) = (1 - pesos(e)) \times \rho$$

Sendo, $pesos(e)$ é a média dos pesos normalizados na aresta e, e ρ é a taxa de evaporação (parâmetro do MACO/D). Se s deve ser desencorajada (evaporação), os valores na matriz de feromônios correspondentes às arestas de s devem ser decrementados em δ.

Dada uma solução s do problema da mochila multiobjetivo (PMM), se s deve ser reforçada (depósito), cada item i de s incrementa o valor correspondente na estrutura de feromônios em um fator δ. O cálculo do depósito no PMM é dado por:

$$\delta(i) = lucros(i) \times (1 - peso(i)/peso\_max) \times \rho$$

Onde $lucros(i)$ é a média dos valores normalizados de lucro do item i, e $peso\_max$ é o maior peso dentre todos os itens. Se s deve ser desencorajada (evaporação), os valores no vetor de feromônios correspondentes aos itens de s devem ser decrementados em δ.

Capítulo 8

Experimentos

Os experimentos realizados neste trabalho compreendem os algoritmos:

- *Non-Dominated Sorting Genetic Algorithm II* (NSGA-II) (DEB et al., 2002);
- *Non-Dominated Sorting Genetic Algorithm III* (NSGA-III) (DEB; JAIN, 2014);
- *Strength Pareto Evolutionary Algorithm 2* (SPEA2) (ZITZLER; LAUMANNS; THIELE, 2002);
- *SPEA2 with Shift-Based Density Estimation* (SPEA2-SDE) (LI; YANG; LIU, 2014);
- *Multiobjective Evolutionary Algorithm Based on Decomposition* (MOEA/D) (ZHANG; LI, 2007);
- Algoritmo Evolutivo Multiobjetivo com Muitas Tabelas (AEMMT) (BRASIL; DELBEM; SILVA, 2013);
- Algoritmo Evolutivo Multiobjetivo com Múltiplas Dominâncias (AEMMD) (LAFETÁ et al., 2016);
- *Multiobjective Evolutionary Algorithm Based on Decomposition ACO* (MOEA/D-ACO) (KE; ZHANG; BATTITI, 2013);
- *Multi-Objective Ant Colony Optimization Algorithm* (MOACS) (RIVEROS et al., 2016)
- e o ACO proposto: *Many-Objective Ant Colony Optimization Based on Decomposed Pheromones* (MACO/D).

Todos os métodos são testados em dois problemas discretos multiobjetivos: o PMM e o PRM. A fim de verificar o comportamento dos algoritmos em relação ao número de objetivos, diversas formulações foram consideradas, avaliando-se problemas desde dois até seis objetivos. Assim como o número de funções objetivos, a topologia da rede (PRM)

e a quantidade de itens (PMM) também afetam a complexidade, portanto elaborou-se instâncias diferentes para cada problema: no PRM, são consideradas seis redes de diferentes complexidades e no PMM variou-se a quantidade de itens em 30, 40, 50, 100 e 200.

Os experimentos podem ser divididos em quatro etapas distintas:

1. Teste dos AG's multiobjetivos NSGA-II, NSGA-III, SPEA2, MOEA/D e AEMMT nos problemas da mochila e do roteamento multicast. O número de objetivos varia entre dois e seis e, no PRM, três redes são testadas, enquanto o PMM lida com instâncias de 30, 50 e 100 itens. Esses experimentos compuseram o primeiro artigo gerado neste trabalho intitulado *"A Comparative Analysis of MOEAs Considering Two Discrete Optimization Problems"* (FRANÇA et al., 2017), em português, uma análise comparativa entre algoritmos evolutivos multiobjetivos considerando dois problemas de otimização discretos. Além dos cinco algoritmos mencionados, avalia-se também uma pequena modificação no AEMMT chamada de AEMMT-f que remove o limite no tamanho do arquivo de soluções não-dominadas. As avaliações dos algoritmos são feitas com base em métricas relacionadas ao Pareto conhecido e permitem um melhor entendimento sobre o comportamento dos algoritmos, o que facilita a identificação de pontos fortes e fracos de cada estratégia, ajudando na elaboração de um novo modelo melhor adequado aos problemas em questão (PMM e PRM).

2. Com um novo algoritmo ACO em mente, o MACO/D, fez-se necessário um estudo sobre os métodos para a construção das soluções no PRM. Além disso, foi também preciso testar modelos de atualização de feromônio e ideias sobre os parâmetros de entrada α e β do algoritmo. A fim de propor o novo método de otimização *many-objective* e um modelo de aplicação para o PRM, realizou-se a segunda etapa de experimentos, onde várias configurações possíveis do algoritmo e do modelo foram testadas.

3. Teste dos métodos de otimização many-objective NSGA-III, MOEA/D, AEMMT, AEMMD, MOACS, MOEA/D-ACO e do algoritmo proposto, MACO/D, nos problemas da mochila e do roteamento multicast. São testadas formulações com 4, 5 e 6 objetivos, avaliando-se 3 redes no PRM e problemas com 30, 40 e 50 itens no PMM. Esses experimentos foram responsáveis pelo artigo ainda não publicado *"MACO/D: Many-objective Ant Colony Optimization based on Decomposed Pheromone"*, em português, MACO/D: otimização por colônia de formigas com muitos objetivos baseada em decomposição de feromônios. Os experimentos colocam à prova pela primeira vez o algoritmo proposto neste trabalho e revelam suas vantagens e fraquezas, possibilitando a identificação dos casos em que se é uma boa ideia

utilizá-lo e as formas em que se pode melhorá-lo em cenários onde que os resultados foram desfavoráveis.

4. Teste dos algoritmos de otimização many-objective NSGA-III, MOEA/D, AEMMT, AEMMD e MACO/D em instâncias complexas do PMM e PRM utilizando a métrica hipervolume, independente do Pareto. Esses experimentos são necessários, pois as etapas anteriores testaram apenas problemas com complexidade razoável, onde se é possível estimar o Pareto. A inclusão de mais itens no PMM ou o uso de redes mais complexas no PRM torna inviável a obtenção da fronteira de Pareto e faz necessária a utilização do hipervolume para avaliar os algoritmos. Nesses experimentos são testadas duas novas redes para o PRM e uma nova instância do PMM, com 200 itens.

Considerando as quatro etapas de experimentos, foram usadas 5 métricas diferentes para se avaliar os algoritmos, são elas:

- ❏ Taxa de Erro (ER): porcentagem das soluções encontradas que não fazem parte do Pareto aproximado;
- ❏ *Generational Distance* (GD): distância das soluções incorretas para a solução mais próxima no Pareto aproximado. Mede o quão distantes as soluções erradas encontradas estão de uma solução correta.
- ❏ *Pareto Subset* (PS): número absoluto de soluções encontradas que fazem parte do Pareto aproximado.
- ❏ Hiper-Volume (HV): Única métrica, além do tempo, utilizada que independe de um Pareto pré-conhecido. Mede o volume da figura geométrica m-dimensional (m é o número de objetivos) formada pelas distâncias entre as soluções encontradas e um ponto de referência p_{ref}. As coordenadas de p_{ref} são diferentes para cada cenário (problema, formulação de objetivos e instância) e são determinadas pelo pior valor encontrado em cada um dos objetivos considerando a união das soluções dos Paretos obtidos em cada execução. Se o PMM de 5 objetivos e 100 itens é executado 10 vezes, por exemplo, extrai-se os 10 resultados e coloca-se soluções em um único conjunto S_{todos}. Varre-se S_{todos} procurando pelo pior valor em cada uma das 5 coordenadas e cria-se uma solução fictícia p_{ref} para os valores encontrados. p_{ref} é então utilizado como ponto de referência para o PMM de 5 objetivos e 100 itens. Para os experimentos deste trabalho foi utilizada a implementação oficial do cálculo de hiper-volume descrito em (WHILE; BRADSTREET; BARONE, 2012).
- ❏ Tempo: tempo em segundos necessário para se executar o algoritmo.

As seções a seguir apresentam os experimentos e seus resultados em cada uma das etapas citadas acima.

8.1 Etapa 1: AG's multiobjetivos

Neste etapa testou-se os algoritmos NSGA-II, NSGA-III, SPEA2, MOEA/D e AEMMT. Ao todo, 30 cenários de teste foram considerados:

- PRM: 5 formulações de objetivos (P_2, P_3, P_4, P_5 e P_6) e 3 redes (R_1, R_2 e R_3). Tanto as formulações quanto às redes foram descritas na seção correspondente ao problema do roteamento multicast (4.2).
- PMM: 5 formulações de objetivos (2 a 6) e 3 instâncias (30, 50 e 100 itens).

Para cada um dos cenários foi extraído um Pareto através de múltiplas execuções dos 5 algoritmos testados. A tabela 2 mostra a cardinalidade de cada Pareto encontrado.

Tabela 2 – Cardinalidade dos Paretos encontrados para a primeira etapa de experimentos

	PRM			**PMM**		
Objetivos	R1	R2	R3	30 itens	50 itens	100 itens
2	14	9	6	15	67	170
3	30	18	17	106	501	6288
4	122	72	60	425	986	88374*
5	424	326	551	1765	5213	176868*
6	1196	657	1078	5800	35760*	248198*

Na tabela 2, a quantidade de elementos nos paretos do PMM é demasiadamente grande para as formulações de objetivos com 100 itens, isso acontece, pois o espaço de busca do problema da mochila cresce exponencialmente com o número de itens. Além disso, a situação é pior quando o número de objetivos é alto, pois, naturalmente, quanto maior a quantidade de funções objetivos, maior o número de soluções que serão não-dominadas. O asterisco ao lado de alguns valores no PMM significa que não foi possível estabilizar o Pareto, ou seja, cada rodada de execuções dos algoritmos encontrava novas soluções. Principalmente por essa razão, julgou-se necessária a execução da etapa 4 de experimentos, onde não se usa um Pareto pré-calculado para se avaliar os algoritmos. Apesar de não serem perfeitos, os resultados para o problema de 100 itens ainda são relevantes, pois ainda que os Paretos não sejam estáveis, compara-se as execuções de todos algoritmos contra as melhores soluções encontradas por todos eles, o que indica o algoritmo com melhor potencial para encontrar boas soluções em problemas multiobjetivos.

Para compilar os resultados através das medidas de desempenho ER, GD e PS, foram feitas 100 execuções de cada algoritmo com os parâmetros listados na tabela 3. As figuras 13, 14 e 15 mostram respectivamente os resultados para o PMM de 30, 50 e 100 itens. As figuras 17, 18 e 19 revelam respectivamente os resultados para o PRM aplicado às redes R_1, R_2 e R_3. Uma análise conjunta, através de uma média entre as três instâncias de cada problema, é apresenta nas figuras 20 (PRM) e 16 (PMM).

Tabela 3 – Parâmetros utilizados pelos algoritmos no PRM e PMM na etapa 1 de experimentos.

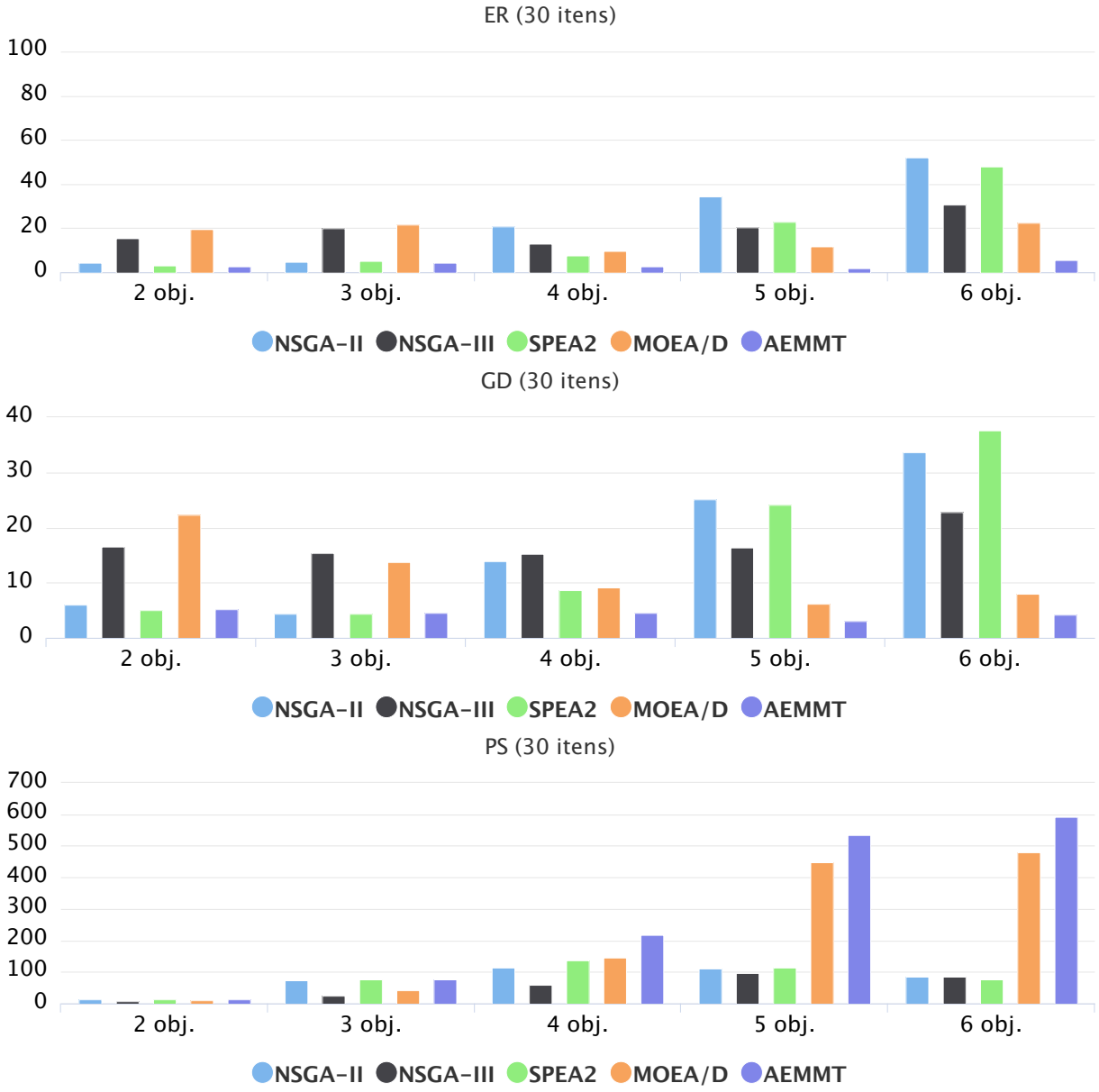
Parâmetro	**(A) PRM**	**(B) PMM**
Tamanho da população	90	150
Número de gerações*	100	100
Taxa de crossover	100%	100%
Taxa de mutação	20%	variável
Tamanho da vizinhança (MOEA/D)	10	10
Tamanho das tabelas (AEMMT)	30	50
Tamanho da tabela de dominância (MEAMT)	90	150
Número de subdivisões (NSGA-III)	8	8

Na tabela 3, o asterisco em "número de gerações" é para dizer que nem todos os algoritmos seguem esse parâmetro. o AEMMT executa 9 mil gerações para o PRM e 7500 para o PMM. Isso acontece, pois esse algoritmo gera apenas 1 filho por ciclo no PRM e apenas 2 no PMM necessitando, portanto, de mais gerações para fazer o mesmo número de comparações. No problema da mochila com 100 itens, devido à complexidade do problema, dobrou-se a quantidade de gerações. Ainda na tabela 3, "variável" quer dizer que a taxa de mutação foi de 6% para o PMM de 30 itens, 4% para o de 50 itens e 2% para o de 100 itens, similar aos valores utilizados em (ISHIBUCHI; AKEDO; NOJIMA, 2015).

O PMM com 30 itens (figura 13) é um problema simples e geralmente qualquer método de otimização multiobjetivo atinge bons resultados. Até 4 objetivos, apenas o MOEA/D passou a marca de 20% de erro. Para 2 e 3 objetivos, o NSGA-II, o SPEA2 e o AEMMT são indistinguíveis e possuem erro muito baixo. A partir de 4 objetivos é possível notar uma piora considerável nos algoritmos clássicos NSGA-II e SPEA-2. Considerando todas as formulações de objetivo, os melhores resultados foram obtidos pelo AEMMT que manteve o erro abaixo de 6% em todos os casos. O GD acompanha as conclusões tiradas a partir do ER, o AEMMT gera, globalmente, os melhores resultados enquanto, apesar de terem ótimo desempenho nos problemas de 2 e 3 objetivos, o NSGA-II e o SPEA2 sofrem a partir de 4 objetivos. O tamanho da fronteira de Pareto encontrada, medida pelo PS, é maior nos algoritmos MOEA/D e AEMMT, o que é esperado, pois diferentemente dos demais, esses dois algoritmos não aplicam uma limitação muito grande sobre o tamanho do Pareto. Para o PMM de 30 itens está claro que, em qualquer situação, o AEMMT é o melhor dentre os métodos testados.

O PMM com 50 itens (figura 14) é um problema bem mais complexo que o PMM de 30 itens e já permite visualizar bem o comportamento de cada algoritmo nos diferentes cenários. Considerando as três métricas (ER, GD e PS) os algoritmos clássicos (NSGA-II e SPEA2) são imbatíveis nas formulações com 2 e 3 objetivos. A partir de 4 objetivos

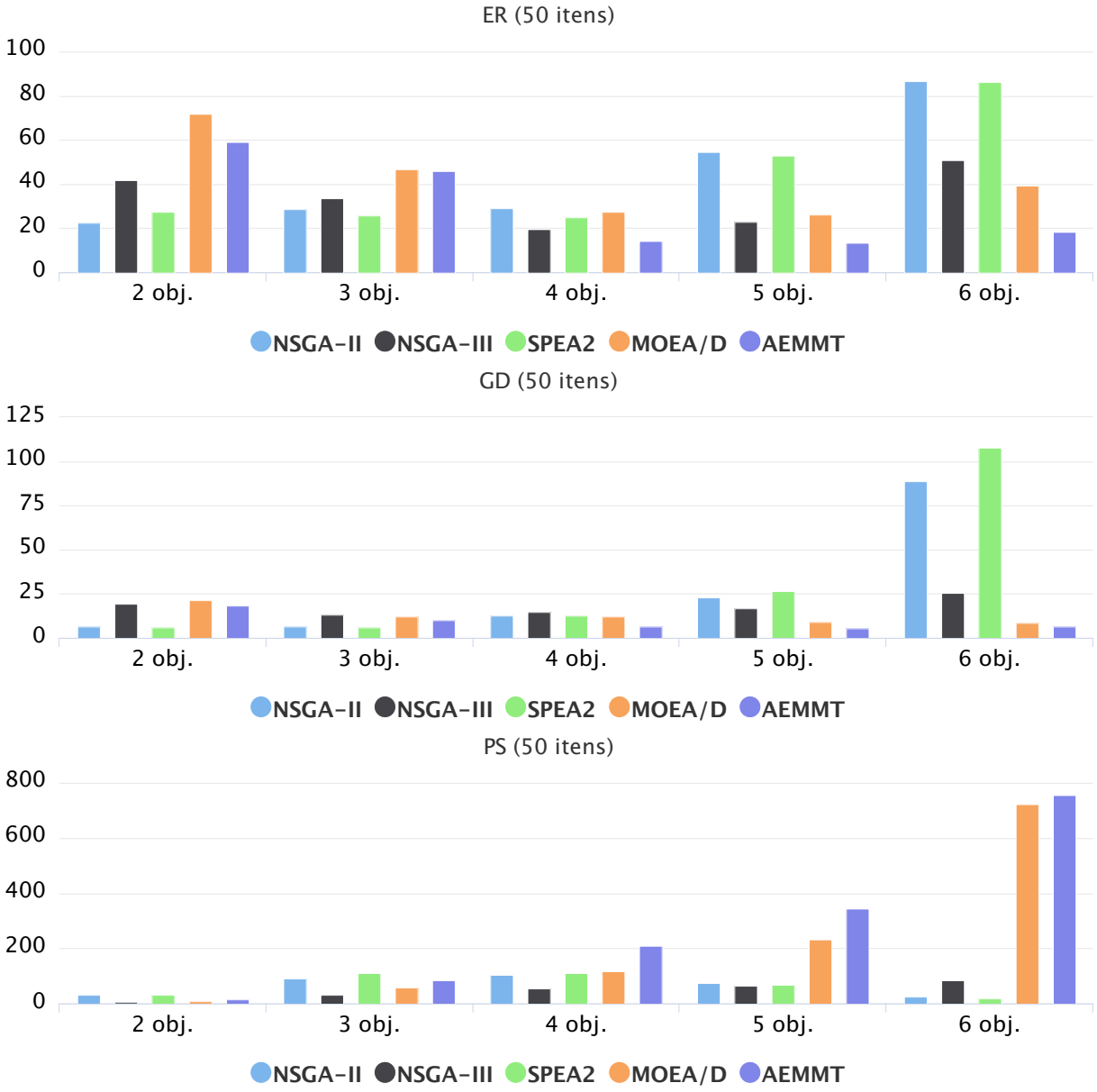
Figura 13 – Etapa 1: resultados para o PMM com 30 itens



a situação muda e o AEMMT passa a apresentar melhores resultados, ao aumentar os objetivos, o NSGA-II e o SPEA2 pioram enquanto o AEMMT mantém o ER e o GD estáveis e melhora o PS. O MOEA/D é o segundo melhor algoritmo para problemas *many-objectives* e o NSGA-III é o terceiro. Para o PMM de 50 itens, o AEMMT é claramente a melhor opção.

O tamanho do espaço de busca no problema da mochila pode ser medido através da equação 2^n, onde n é o número de itens. Portanto, a complexidade do PMM com 100 itens (figura 15) é absurdamente maior que os anteriores. Por essa razão, não foi possível encontrar um Pareto estável para as formulações de 4, 5 e 6 objetivos. Dessa forma, apesar de as métricas ER, GD e PS serem um bom indicativo da performance entre os algoritmos, a melhor forma de avaliação seria o hiper-volume. Um experimento

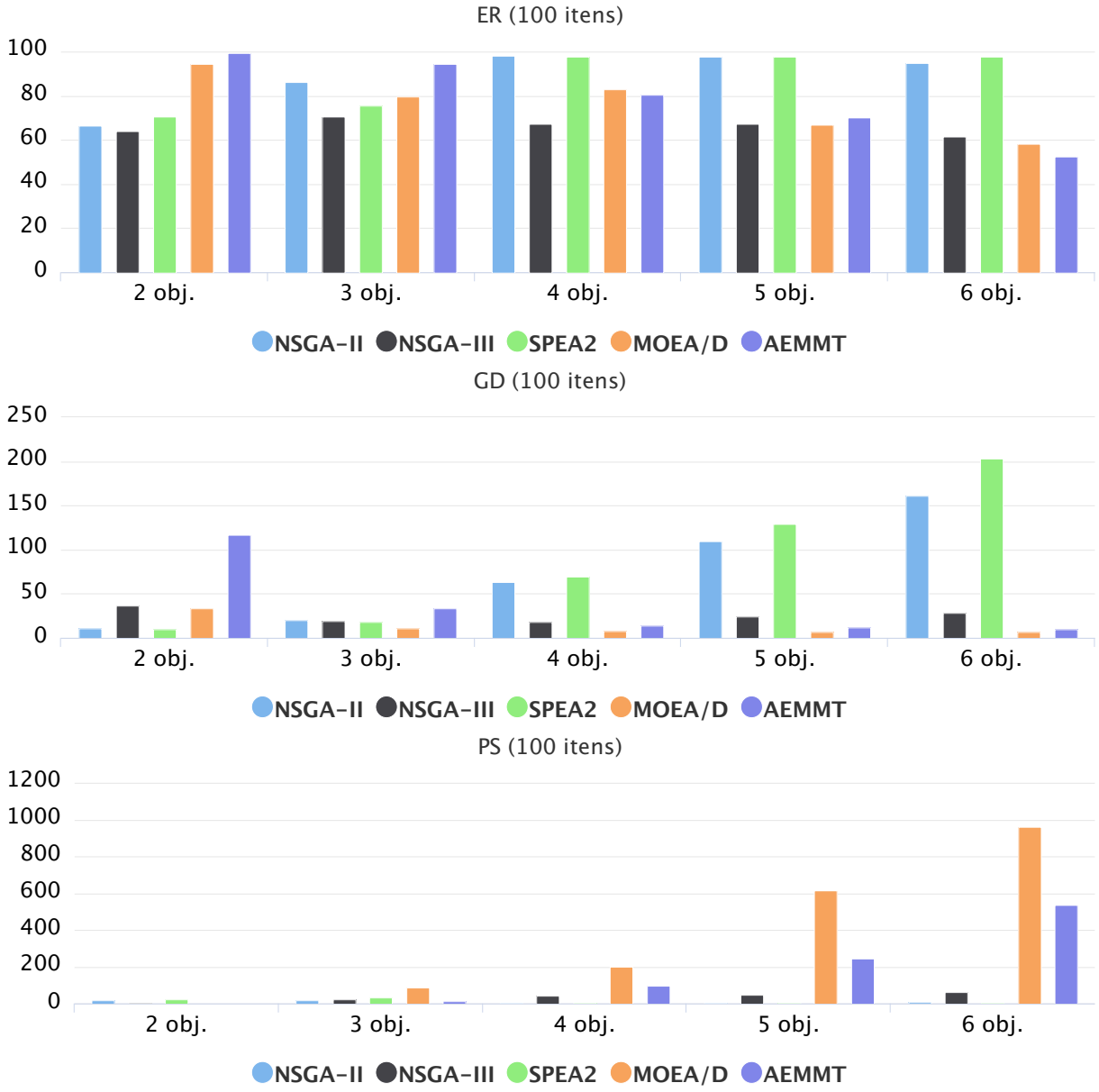
Figura 14 – Etapa 1: resultados para o PMM com 50 itens



parecido com este, mas utilizando o hiper-volume é apresentado na seção 8.4. É difícil avaliar o problema de 100 itens, pois, já no erro é possível perceber que poucos algoritmos conseguiram encontrar boas soluções, a menor taxa de erro está acima de 50%. No PMM de 100 itens o NSGA-III apresentou o menor ER nos cenários de 2, 3, 4 e 5 objetivos, e na formulação de 6, foi ultrapassado pelos AEMMT e MOEA/D. Com relação ao GD, para 2 objetivos, o NSGA-II e o SPEA2 apresentaram os melhores resultados. Na formulação de 3 objetivos em diante, o MOEA/D foi o algoritmo com menor GD, sendo que em 4, 5 e 6 objetivos o AEMMT obteve desempenho similar. A situação se repete ao analisar o PS: a partir de 3 objetivos, o MOEA/D traz um conjunto maior de soluções na fronteira de Pareto, seguido pelo AEMMT a partir de 4 objetivos.

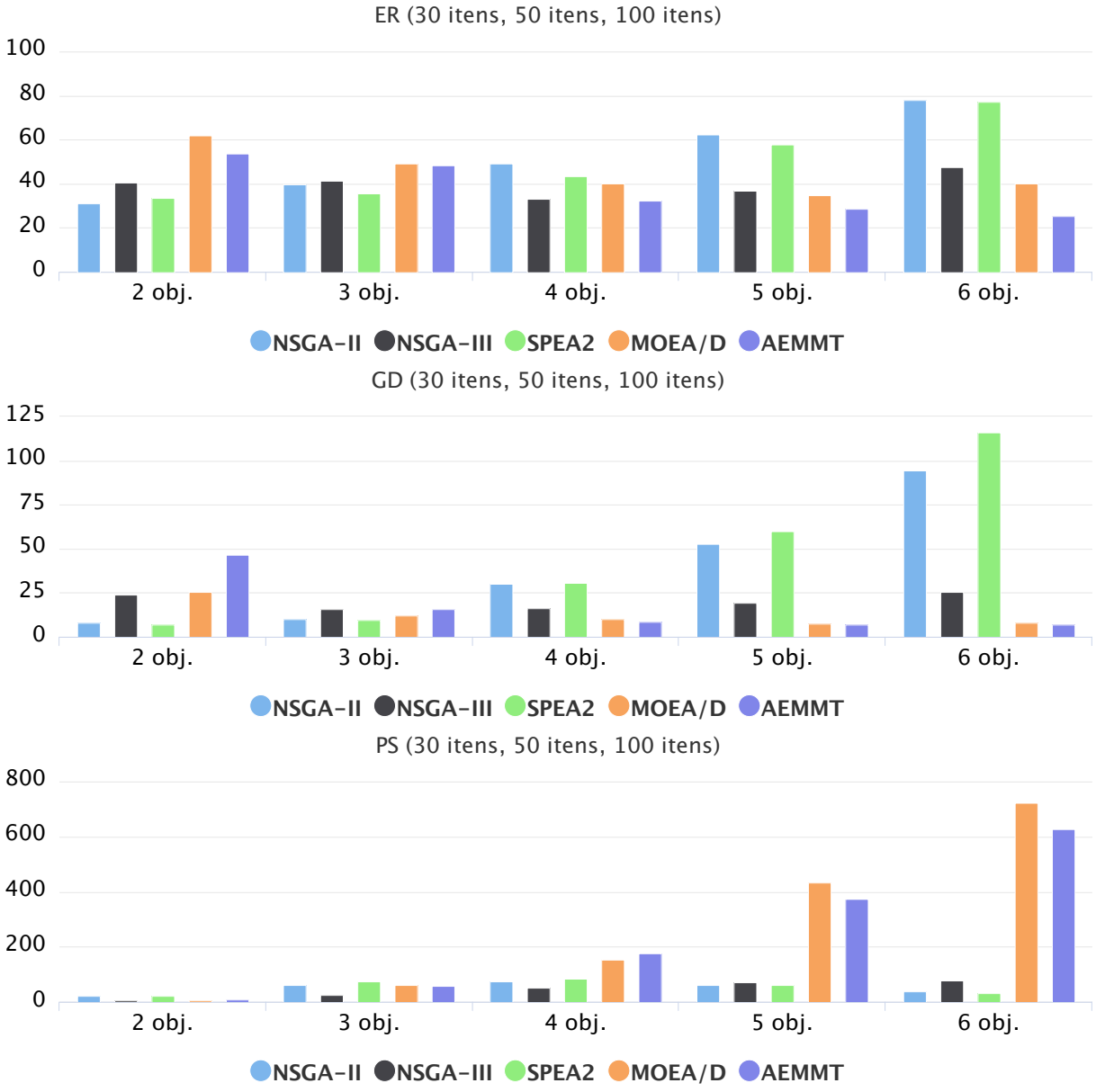
A figura 15 apresenta os resultados do PMM de 30, 50 e 100 itens de forma condensada

Figura 15 – Etapa 1: resultados para o PMM com 100 itens



para que se possa ver, de forma geral, o comportamento dos algoritmos nas diferentes formulações de objetivo. Os gráficos representam as médias entre os três cenários de dificuldade (30, 50 e 100 itens). Como esperado, considerando a literatura correlata, o NSGA-II e o SPEA2 são os melhores algoritmos para as formulações de 2 e 3 objetivos, apresentam melhor ER, GD e PS. Por outro lado, a partir de 4 objetivos, o desempenho de ambos os algoritmos cai consideravelmente, enquanto o AEMMT assume a liderança. O NSGA-III, no problema de 4 objetivos, apresenta um erro quase tão baixo quanto o AEMMT, mas seu GD e PS são piores. o MOEA/D, para 5 e 6 objetivos, apresenta o segundo melhor resultado em qualquer uma das métricas. Em resumo, o NSGA-III não parece uma boa opção em nenhum dos casos, pois sempre há outro algoritmo que o supera. O NSGA-II e o SPEA2 são igualmente bons e os melhores em problemas

Figura 16 – Etapa 1: resultados agrupados para o PMM com 30, 50 e 100 itens

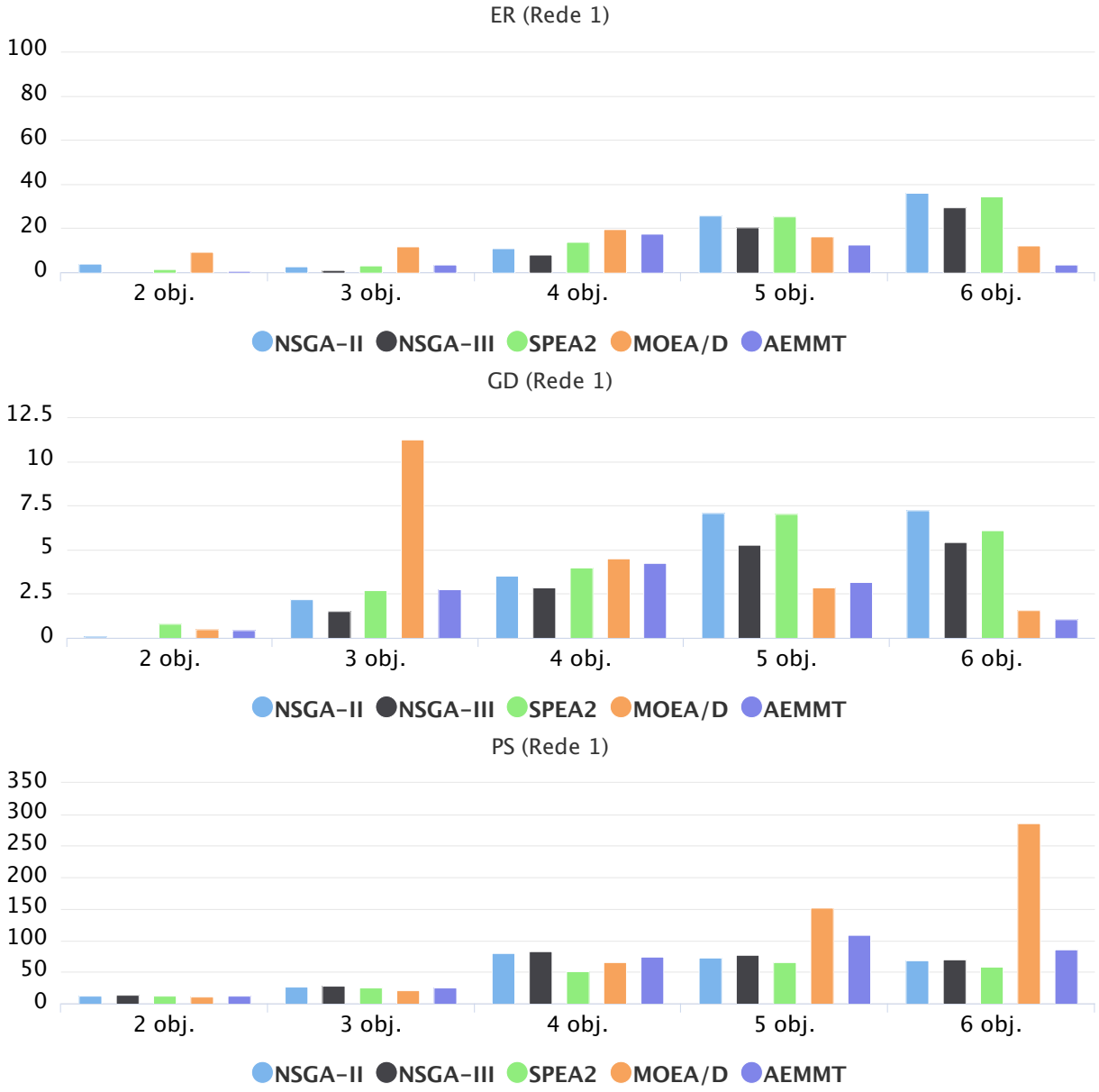


com poucos objetivos. O AEMMT e o MOEA/D são ótimas opções para problemas a partir de 4 objetivos, sendo que o MOEA/D confere um melhor PS enquanto o AEMMT providencia menor taxa de erro.

O PRM sobre a rede 1 (figura 17), a mais simples dentre elas, mostrou bons resultados para todos os algoritmos. Diferente do esperado, o NSGA-III mostrou o melhor resultado (ER, GD e PS) para os problemas com 2, 3 e 4 objetivos. A partir de 5 objetivos, o AEMMT e o MOEA/D são os dois melhores métodos, o primeiro apresenta uma menor taxa de erro, enquanto o segundo obtém um Pareto de maior cardinalidade. Para poucos objetivos, o NSGA-III é claramente o melhor método, para 5 ou mais critérios de otimização, ambos AEMMT e MOEA/D são boas opções.

Na rede 2 (figura 18), o NSGA-III é o melhor algoritmo nos problemas com 2, 3, 4

Figura 17 – Etapa 1: resultados para o PRM na rede R_1

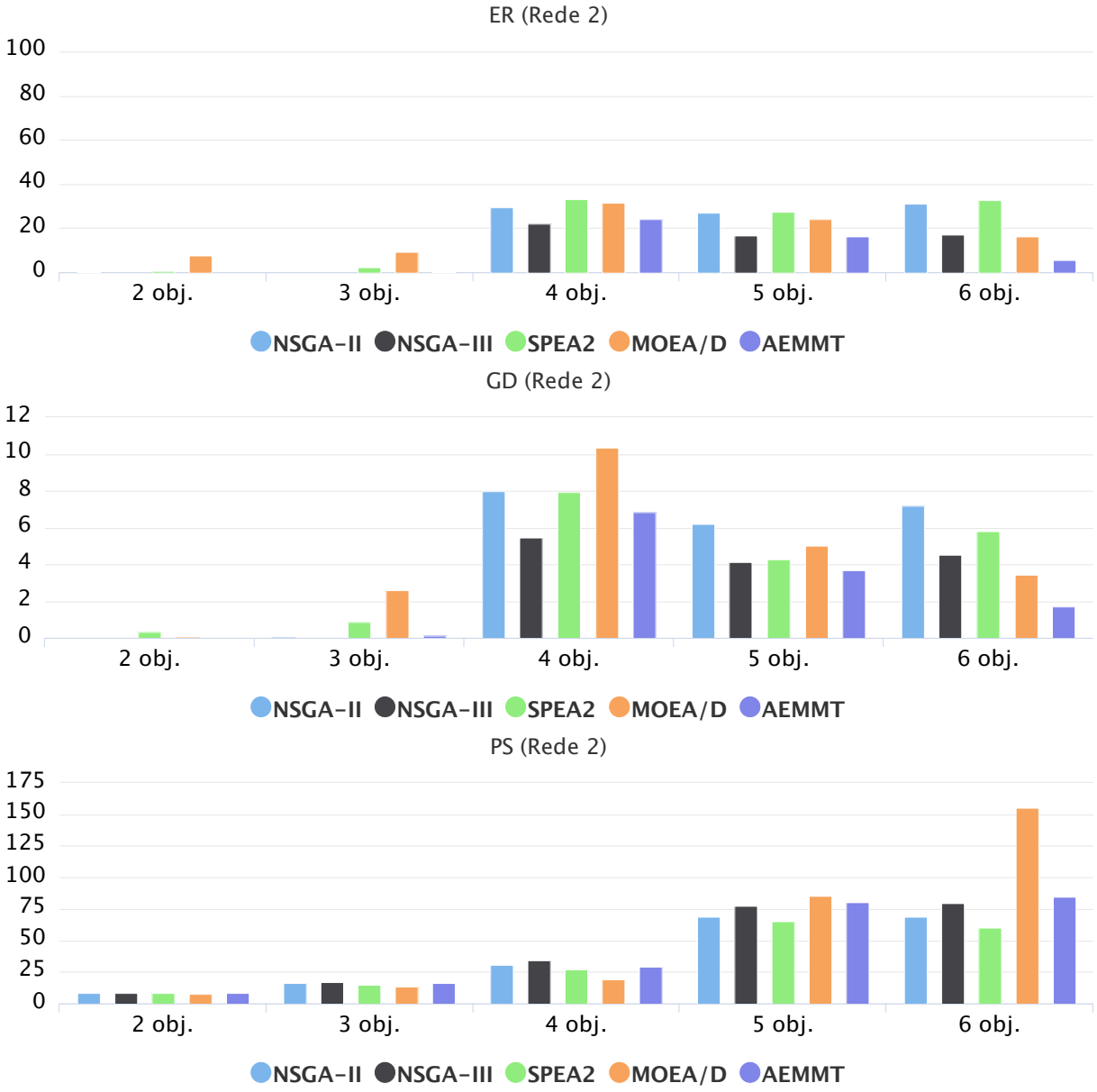


e 5 objetivos, perdendo, por pouco, apenas em PS para o AEMMT e o MOEA/D no problema de 5 objetivos. Com 6 objetivos, o AEMMT é o melhor algoritmo quando se considera o erro e o GD, mas se um maior PS é mais desejável, então o MOEA/D é o método mais adequado.

A rede 3 (figura 19) é a mais complexa analisada nesta etapa dos experimentos. Nela, a tendência já observada do NSGA-III de ser o melhor método para poucos objetivos continua. Até 4 objetivos, em todas as métricas, o NSGA-III apresenta os melhores resultados. Para 5 e 6 critérios de otimização, o AEMMT apresenta menor ER e GD, enquanto o MOEA/D consegue maior PS.

Afim de fazer uma análise geral do PRM, na figura 20, todos os cenários (redes 1, 2 e 3) são combinados num único gráfico através de uma média aritmética dos resultados.

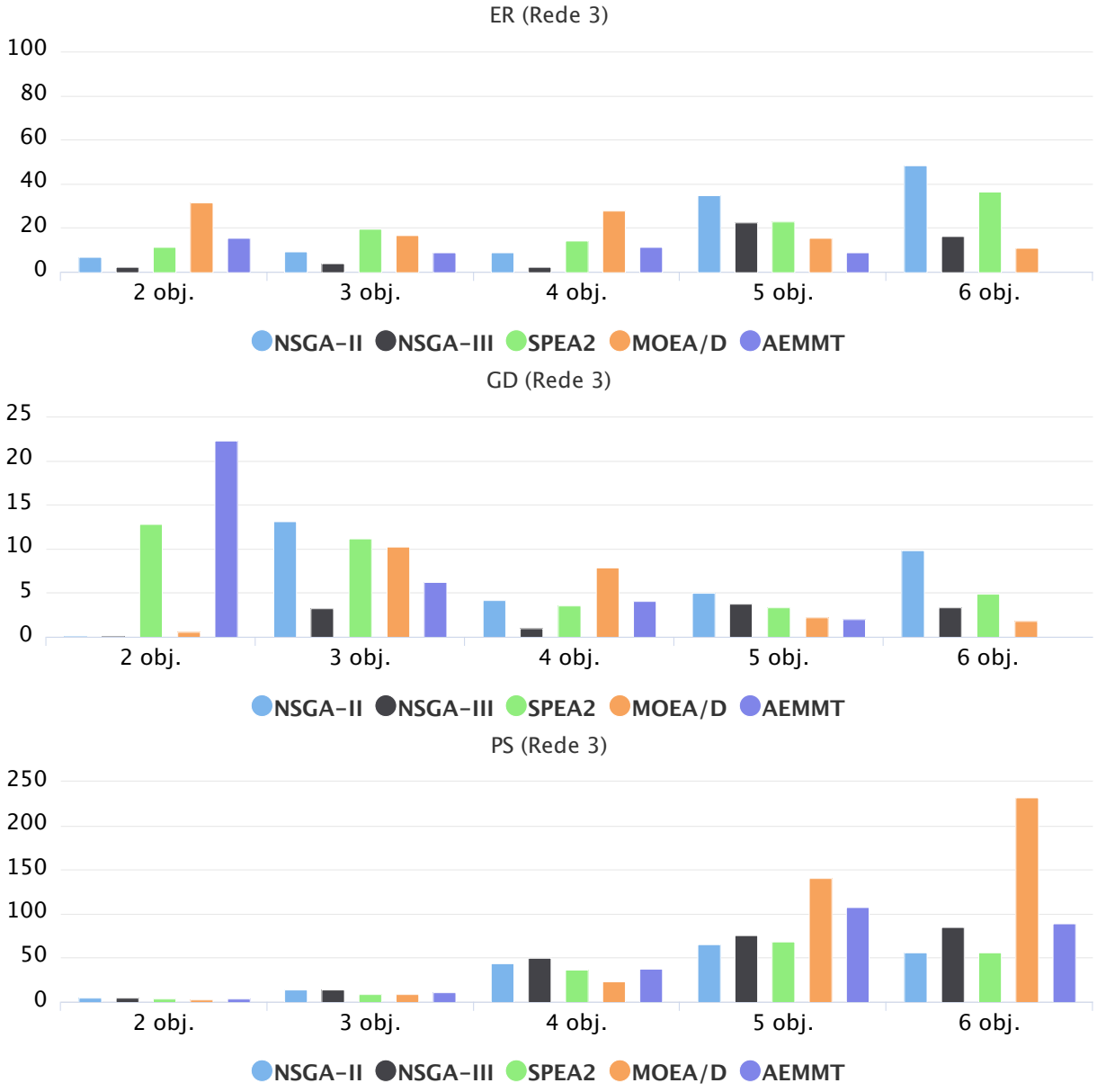
Figura 18 – Etapa 1: resultados para o PRM na rede R_2



Observa-se que, apesar de ser esperado que o NSGA-II e o SPEA2 fossem os melhores métodos para poucos objetivos, na verdade, o NSGA-III foi o algoritmo que obteve melhor resultado para problemas com, 2, 3 e 4 objetivos. O NSGA-II também produz bons resultados para problemas com 2 e 3 objetivos, mas o SPEA2 apresenta um GD relativamente ruim. Considerando problemas de 5 e 6 objetivos, a escolha do algoritmo dependerá do intuito da busca, se é preferível uma maior quantidade de soluções, o MOEA/D é mais indicado, caso contrário, se um menor erro é preferível, então o AEMMT é a melhor opção.

Considerando ambos os problemas, PMM e PRM, os algoritmos NSGA-II, SPEA2 e NSGA-III são os que geram melhores resultados para problemas com poucos objetivos. A performance dos algoritmos clássicos (NSGA-II e SPEA2) cai consideravelmente a medida

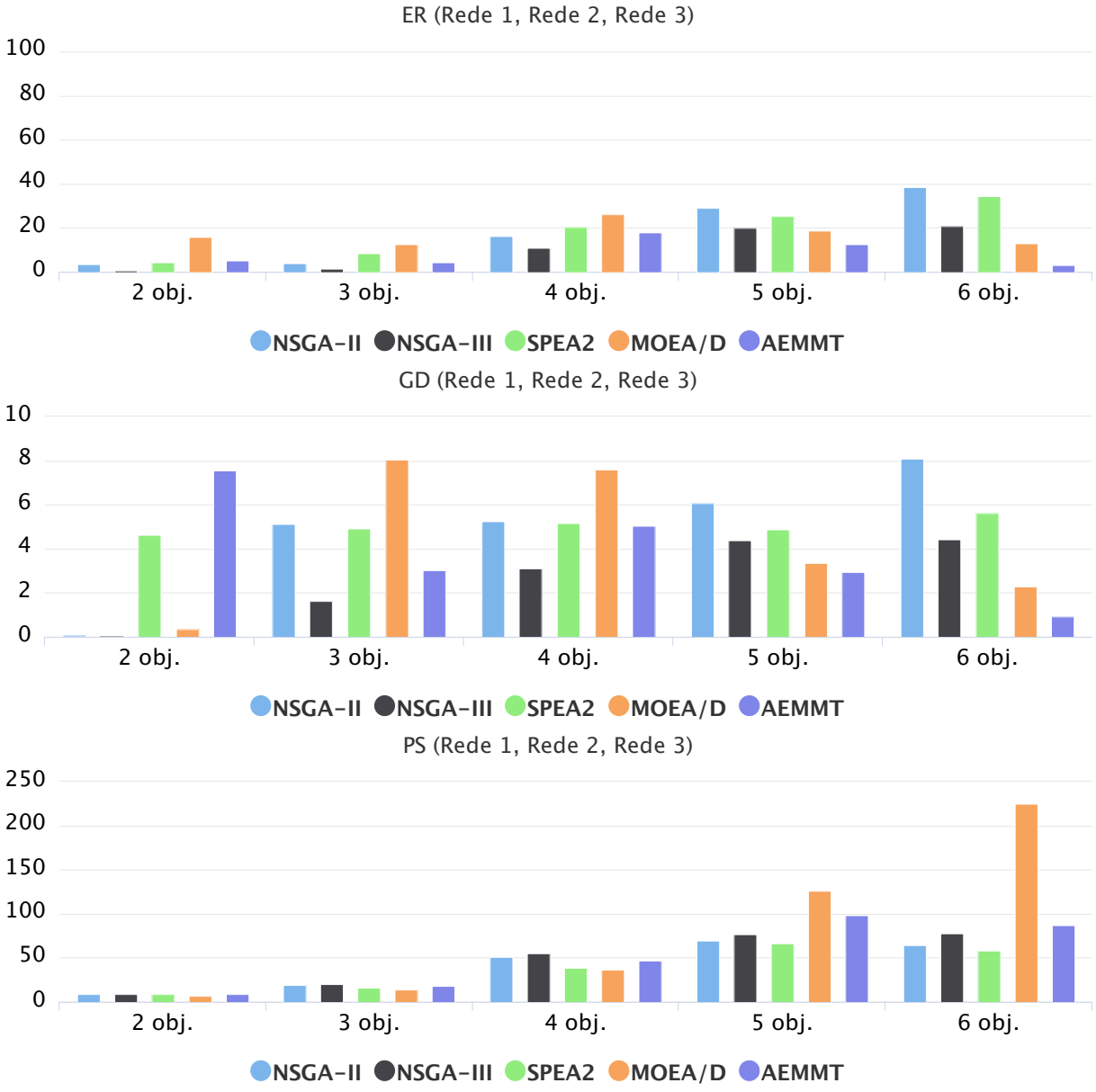
Figura 19 – Etapa 1: resultados para o PRM na rede R_3



que se aumenta o número de objetivos, enquanto o desempenho dos métodos AEMMT e MOEA/D melhora a partir de quatro objetivos, tornando-nos os mais indicados para problemas *many-objectives*.

Uma outra observação que pôde ser feita a partir dos experimentos nesta etapa é que o AEMMT perde apenas em PS para o MOEA/D. Uma das características do AEMMT é a limitação no tamanho do arquivo, por isso surge a dúvida: se não houvesse um limite, seria possível que o AEMMT obtivesse um melhor resultado em todas as métricas? Pensando nisso, executou-se os mesmos testes para uma variação do AEMMT (AEMMT-F), onde o limite não foi aplicado. Na maioria dos resultados, o AEMMT-F apresentou erro maior que sua versão original, mas ainda sim ER e PS melhores que o MOEA/D. Dessa forma, através de um teste de hipótese z-teste com 0,1% de significância ($\alpha = 0.1$), confirmou-se

Figura 20 – Etapa 1: resultados agrupados para o PRM nas redes R_1, R_2 e R_3



a superioridade do AEMMT-F em relação ao MOEA/D nos cenários com 5 e 6 objetivos.

8.2 Etapa 2: MACO/D

O algoritmo responsável pela construção das soluções do PRM no MACO/D foi concebido de acordo com os experimentos realizados nesta etapa. O mesmo processo não foi realizado para o PMM, pois já havia um modelo bem conceituado que funcionava bem com o algoritmo MACO/D. Além disso, aproveitou-se esse conjunto de experimentos para variar aspectos sobre a atualização de feromônios e parâmetros de entrada no ACO, resultando em particularidades do algoritmo descrito no capítulo 7.

Como visto na seção 6.5, foram estudadas quatro estratégias para se construir soluções no PRM. Primeiramente, foram implementados os dois algoritmos mais simples: estratégia 1 (formiga única) e 2 (múltiplas formigas). Infelizmente ambos os métodos não produziram resultados satisfatórios no problema do roteamento multicast multiobjetivo. Portanto, com a finalidade de se obter um problema mais simples e testar de maneira isolada o processo de construção da solução, decidiu-se elaborar um cenário do PRM com um único objetivo. As estratégias de construção da solução são apresentadas, em resumo, a seguir. Consulte a seção 6.5 para mais detalhes.

1. Formiga única: o espaço de busca é explorado de forma aleatória, mas sempre considerando apenas a vizinhança da posição atual da formiga.

2. Múltiplas formigas: uma formiga para cada destino. Une-se os caminhos produzidos por cada agente em uma árvore.

3. Formiga com super-posição: uma formiga explora o espaço de busca de forma aleatória, mas pode estar em vários nós ao mesmo tempo, excluindo o problema de localidade na busca.

4. Formigas invertidas: uma formiga para cada destino, mas ao invés de percorrerem o caminho da raiz ao destino, fazem o contrário, partem do destino e tentam encontrar a raiz.

O problema mono-objetivo criado consiste em minimizar o valor de $custo{*}delay$ de uma árvore, portanto, quanto menor esse valor, melhor a árvore obtida. As quatro estratégias testadas são aquelas mencionadas na seção 6.5. Além dos quatro algoritmos, é incluído um resultado novo, obtido a partir de uma modificação do algoritmo de Prim (PRIM, 1957), para servir como referência de uma boa solução possível de ser encontrada. Cada estratégia foi executada cinco vezes e a tabela 4 mostra o resultado da melhor execução de cada estratégia.

Na tabela 4, a coluna "resultado" representa a soma dos valores de $custo * delay$ das arestas da árvore obtida como solução, ou seja, quanto menor esse valor, melhor a solução obtida. Dessa forma, a estratégia número 3, que usa a ideia de formiga com super-posição, obteve melhor performance. Em termos de tempo, ela infelizmente leva mais tempo que

Tabela 4 – Resultados para as estratégias de construção de solução do PRM

Estratégia	Rede	Resultado	Tempo (s)
Prim	R_1	3.285714286	0.021
1	R_1	3.047619048	2.496
2	R_1	3.031746032	5.94
3	R_1	3.007936508	3.88
4	R_1	3.007936508	3.16
Prim	R_2	3.134920635	0.016
1	R_2	3.341269841	4.947
2	R_2	3.261904762	13.22
3	R_2	3.134920635	10.694
4	R_2	3.301587302	5.549
Prim	R_3	7.968253968	0.024
1	R_3	8.134920635	4.212
2	R_3	8.238095238	25.606
3	R_3	7.484126984	9.821
4	R_3	8.111111111	6.939
Prim	R_4	1.801587302	0.025
1	R_4	2.341269841	4.715
2	R_4	2.325396825	12.471
3	R_4	1.857142857	11.015
4	R_4	1.976190476	4.935
Prim	R_5	6.341269841	0.015
1	R_5	6.126984127	6.87
2	R_5	6.333333333	17.38
3	R_5	5.857142857	14.767
4	R_5	5.857142857	8.423

a maioria das demais. Por essa razão propõe-se a ideia de amostragem explicada na seção 6.5. Ao construir a solução, ao invés de se utilizar a totalidade do conjunto de exploração, toma-se uma amostra desse. Em nossos testes para o PRM, a amostragem é sempre de 10 elementos. A tabela 5 mostra uma comparação da estratégia 3 com e sem a amostragem.

Tabela 5 – Comparação entre da estratégia 3 na técnica de amostragem

Amostragem	Rede	Resultado	Tempo (s)
s/ amostragem	R_1	3.007936508	3.88
c/ amostragem	R_1	3.007936508	3.41
s/ amostragem	R_2	3.134920635	10.694
c/ amostragem	R_2	3.134920635	7.602
s/ amostragem	R_3	7.484126984	9.821
c/ amostragem	R_3	7.484126984	7.26
s/ amostragem	R_4	1.857142857	11.015
c/ amostragem	R_4	1.785714286	7.28
s/ amostragem	R_5	5.857142857	14.767
c/ amostragem	R_5	5.76984127	10.037

Como pode ser observado na tabela 5, a amostragem não só diminuiu o tempo do algoritmo como também melhorou o resultado. A melhora na qualidade da solução pode ser atribuída ao maior grau de aleatoriedade dada ao algoritmo, similar ao que acontece nos algoritmos genéticos quando se lança mão de operações como a mutação ou seleções por torneio.

Ao implementar a estratégia 1 (uma formiga e super-posição) com amostragem no PRM many-objetive, percebeu-se que o AEMMD conseguia resultados muitos superiores (tabela 6) ao novo algoritmo. A fim de reduzir essa diferença, propôs-se as seguintes mudanças para o MACO/D:

- Depósito de feromônio baseado na qualidade da aresta (ou do item, no caso do PMM): num problema multi-objetivo, ao depositar feromônios sobre as arestas correspondentes às soluções não-dominadas, sempre deposita-se a mesma quantidade independente da solução e da aresta, pois, segundo a relação de não-dominância, ambas opções são igualmente boas. Esta proposta muda esse conceito ligando a quantidade de feromônios depositados à qualidade da aresta, i.e., a quantidade passa a ser inversamente proporcional à soma dos pesos da aresta (considerando um problema de minimização).

- Dinamização do parâmetro de entrada β: β é o valor que controla a importância da heurística ao calcular as probabilidades de cada aresta (ou item, no PMM) de fazer parte da solução. A proposta é diminuir um pouco a importância da heurística, dando maior peso à informação de feromônio sempre que, após uma iteração, não for encontrada uma nova solução. Assim que uma nova solução é encontrada, o valor de beta é reiniciado para o padrão.

Tomando como referência o problema P_6, com seis objetivos, construiu-se a tabela 6 que mostra as diferenças entre os desempenhos dos algoritmos AEMMD, MACO/D antes das alterações no depósito de feromônios e do parâmetro β (MACO/D-pré) e o algoritmo final proposto no capítulo 7 (MACO/D).

Na tabela 6 pode-se observar que na maioria dos casos as duas alterações propostas melhoraram o resultado do MACO/D, portanto, o algoritmo final proposto inclui essas duas pequenas alterações. O AEMMD ainda apresenta melhores resultados, mas como será visto nas etapas 3 e 4 de experimentos, o MACO/D leva até 5 vezes menos tempo para executar.

Tabela 6 – Desempenho do AEMMD, MACO/D pré-alterações e MACO/D final

Algoritmo	Rede	ER	GD	PS
AEMMD	R_1	6.59	0.38	502.8
MACO/D-pré	R_1	18.42	0.30	337.6
MACO/D	R_1	11.57	0.36	424.2
AEMMD	R_2	7.58	0.49	296.6
MACO/D-pré	R_2	11.75	0.47	284.4
MACO/D	R_2	11.18	0.37	300
AEMMD	R_3	11.73	0.19	388.6
MACO/D-pré	R_3	36.47	0.16	206
MACO/D	R_3	30.20	0.25	232.4
AEMMD	R_4	35.88	0.17	234
MACO/D-pré	R_4	54.48	0.11	150.2
MACO/D	R_4	50.57	0.15	186.6
AEMMD	R_5	32.67	0.20	181.2
MACO/D-pré	R_5	32.95	0.22	168.6
MACO/D	R_5	32.67	0.30	160.8

8.3 Etapa 3: AG's *many-objectives* vs. MACO/D

Na terceira etapa de experimentos descartam-se os AEMOs clássicos NSGA-II e SPEA2 e inclui-se o AEMMD e o MACO/D, algoritmo proposto neste trabalho. Uma nova métrica de desempenho é utilizada, o tempo, e as demais continuam sendo o erro (ER), a distância (GD) e o número de soluções corretas (PS). Assim como na etapa 1, o Pareto aproximado foi pré-calculado a partir de múltiplas execuções dos algoritmos e é apresentado na tabela 7. Enfim, os resultados (com exceção do tempo) foram obtidos através das médias entre 100 execuções dos 30 cenários descritos na lista a seguir. À medida de tempo considerou apenas 3 execuções de cada algoritmo em cada cenário.

- PRM: 3 formulações de objetivos (P_4, P_5 e P_6) e 3 redes (R_1, R_2 e R_3). Tanto as formulações quanto às redes foram descritas na seção correspondente ao problema do roteamento multicast 4.2.

- PMM: 3 formulações de objetivos (4 a 6) e 3 instâncias (30, 40 e 50 itens).

Tabela 7 – Cardinalidade dos Paretos encontrados para a primeira etapa de experimentos

	PRM			**PMM**		
Objetivos	R1	R2	R3	30 itens	40 itens	50 itens
4	122	553	1349	425	1199	1012
5	75	372	712	1769	3862	5467
6	60	660	1283	5828	6491	55471

A tabela 8 apresenta os parâmetros dos algoritmos utilizados neste experimento. Note que o número de gerações (marcado com asterisco) deve ser multiplicado pelo tamanho

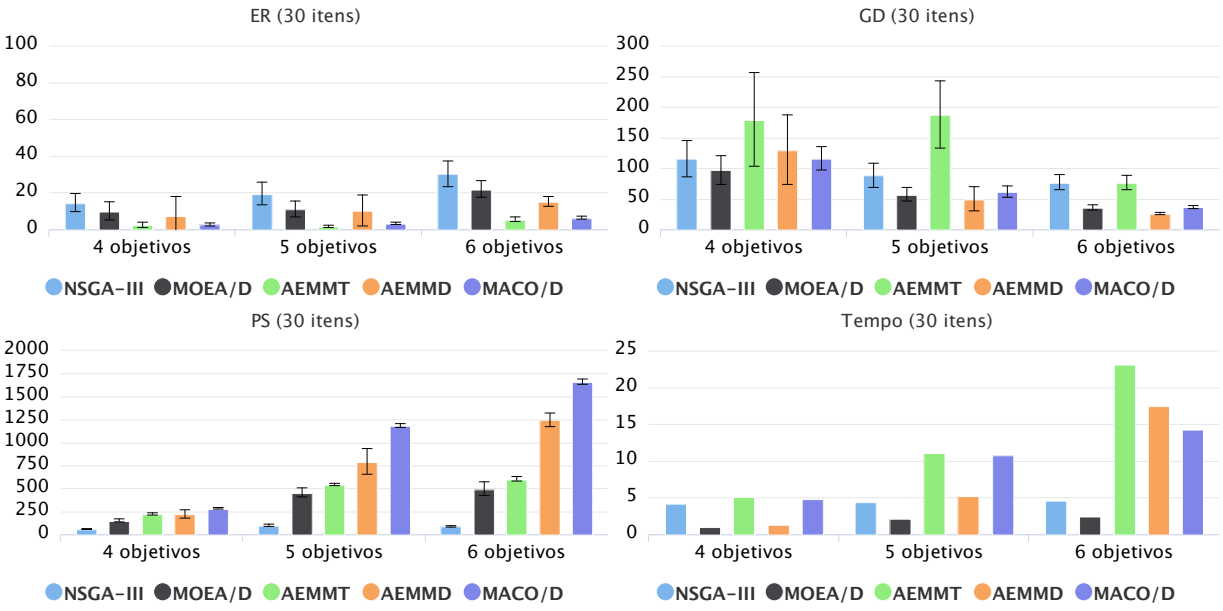
da população no AEMMT e AEMMD devido ao fato de realizarem apenas um crossover por geração.

Tabela 8 – Parâmetros utilizados para o PRM e o PMM na etapa 3 de experimentos.

Parâmetro	**PRM**	**PMM**
Tamanho da população	90	150
Número de gerações*	100	100
Taxa de crossover	100%	100%
Taxa de mutação	20%	5%
Tamanho da vizinhança (MOEA/D)	10	10
Tamanho das tabelas (MEAMT)	30	50
Tamanho da tabela de dominância (MEAMT)	90	150
Número de divisões (NSGA-III)	8	8
α, β, ρ (MACO/D)	1, 2, 0.3	1, 4.3, 0.3
Intervalo de valores para os feromônios (MACO/D)	[0.1, 0.9]	[0.1, 0.9]
Tamanho das amostras (MACO/D)	10	25% of the number of items
Tamanho do grupo de estruturas ativas (MACO/D)	5	5

As figuras 21, 22 e 23 mostram respectivamente os resultados para o PMM de 30, 40 e 50 itens. As figuras 25, 26 e 27 revelam respectivamente os resultados para o PRM aplicado às redes R_1, R_2 e R_3. Uma análise conjunta, com uma média entre as três instâncias de cada problema é apresenta nas figuras 28 (PRM) e 24 (PMM).

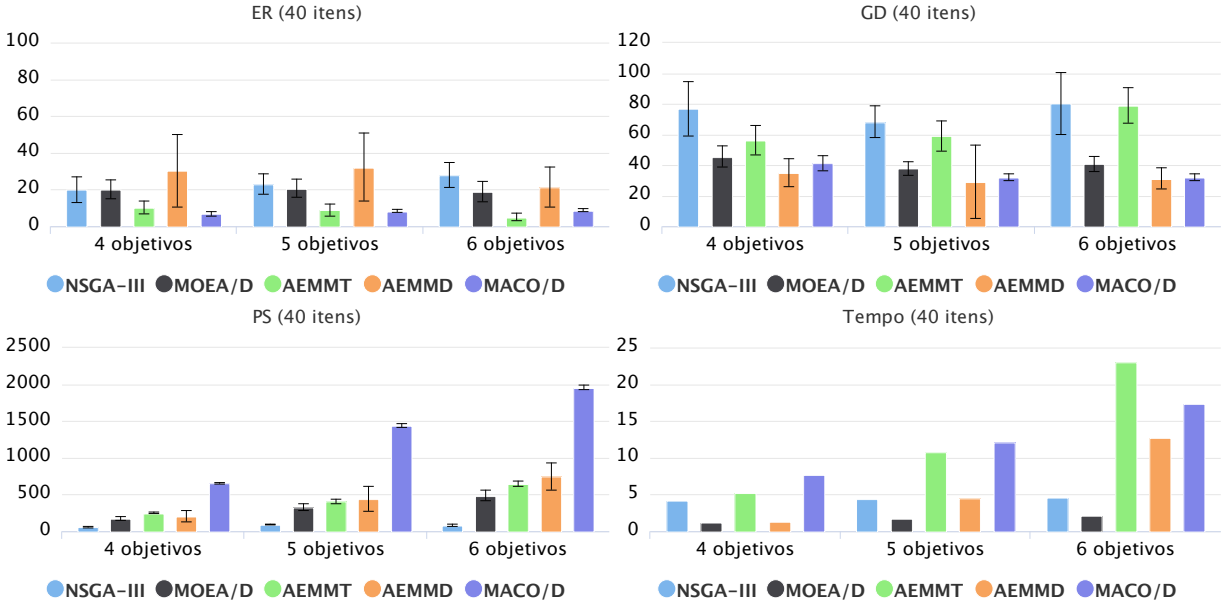
Figura 21 – Etapa 3: resultados para o PMM com 30 itens



No problema da mochila mais simples, de 30 itens (figura 21), o AEMMT apresenta a menor taxa de erro dentre os algoritmos *many-objectives*, em segundo lugar está o MACO/D e em terceiro o AEMMD. Com relação ao *GD*, com 4 objetivos, os melhores

resultados são encontrados pelo MOEA/D seguido pelo MACO/D. O Em 5 e 6 objetivos, o AEMMD produz o menor GD, e o MACO/D, bem próximo, aparece em segundo lugar. Na métrica PS, o MACO/D é melhor que os demais em todos as formulações de objetivo, seguido, em ordem pelo AEMMD, AEMMT, MOEA/D e NSGA-III. Destre esses algoritmos, o único com um limite fixo no tamanho do Pareto é o NSGA-III, portanto, é esperado que possua um valor de PS menor. Quanto ao tempo, o MOEA/D é o algoritmo mais rápido dentre os 5.

Figura 22 – Etapa 3: resultados para o PMM com 40 itens



O PMM de 40 itens é analisado na figura 22. O AEMMT e o MACO/D apresentam a menor taxa de erro, sendo que o MACO/D é melhor para 4 e 5 objetivos, enquanto o AEMMT obtém o melhor resultado para 6 objetivos. Os melhores valores de GD foram encontrados pelo AEMMD e MACO/D, sendo que o AEMMD é um pouco melhor no problema com 4 objetivos. Uma característica negativa do AEMMD em relação ao MACO/D no que se refere ao GD é sua alta variação nos resultados, o que diz que algumas execuções produz soluções muito próximas do Pareto enquanto outras nem tanto. A respeito do tamanho dos Paretos encontrados, novamente o MACO/D lidera independente da formulação de objetivos. O MOEA/D é o algoritmo mais rápido, enquanto o AEMMT e o MACO/D são os mais lentos.

O PMM com 50 itens (figura 23) apresenta comportamento similar às instâncias anteriores. O MACO/D e o AEMMT se revezam em menores taxas de erro, o MACO/D consegue menor ER em 4 e 5 objetivos, enquanto o AEMMT apresenta melhor resultado em 6 objetivos. O MACO/D obtém os melhores valores de GD e PS, e o MOEA/D é o algoritmo mais rápido.

A fim de se fazer uma análise conjunta dos resultados, a média entre os 3 cenários é apresentada nos gráficos da figura 24. As taxas de erro são muito baixas para ambos

Figura 23 – Etapa 3: resultados para o PMM com 50 itens

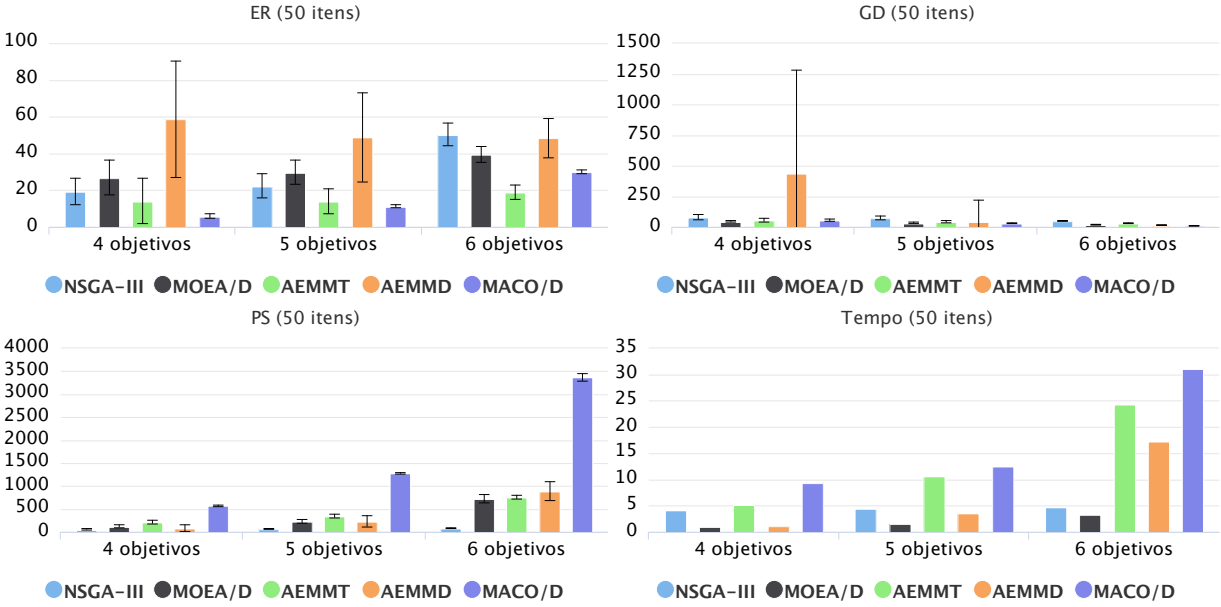
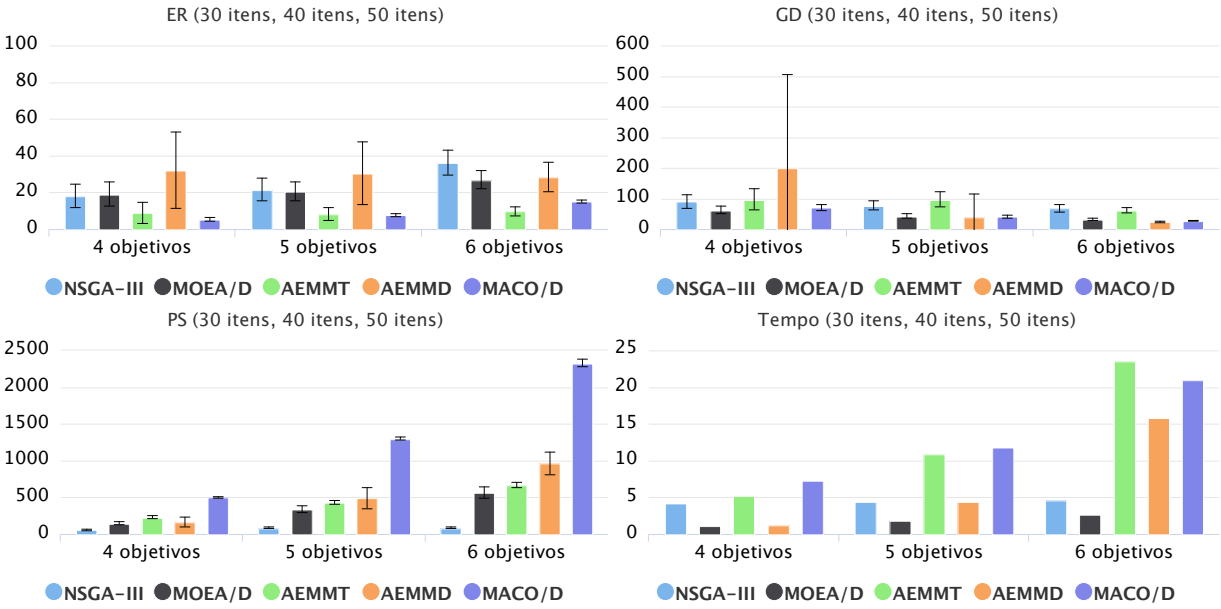
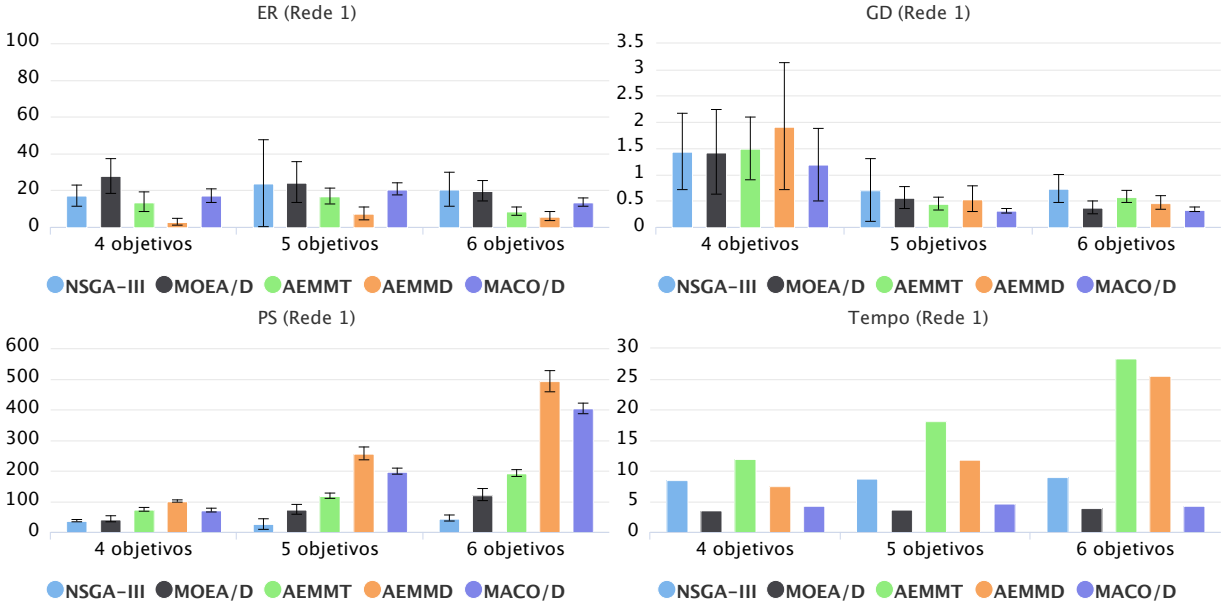


Figura 24 – Etapa 3: resultados agrupados para o PMM com 30, 40 e 50 itens



AEMMT e MACO/D quando comparados aos demais algoritmos. Os resultados em GD são bons para a maioria dos métodos, apenas o AEMMD apresenta um valor ruim de GD no problema de 4 objetivos. O MOEA/D produz o melhor GD no problema de 4 objetivos, enquanto que em 5 e 6 objetivos, o MACO/D e o AEMMD consegue valores bem similares. É importante notar que os desvio padrões no AEMMD são altos, o que pode representar uma certa inconsistência do método em gerar boas soluções. O PS é , de longe, dominado pelo MACO/D. Em questão de tempo de execução, o AEMMT, o AEMMD e o MACO/S variam bastante com o número de objetivos, enquanto os demais são estáveis. O MOEA/D é o algoritmo mais rápido entre os avaliados nesta etapa dos experimentos. O NSGA-III não é pior método em nenhuma das formulações de objetivo, mas também não se destaca em nenhum dos critérios de avaliação.

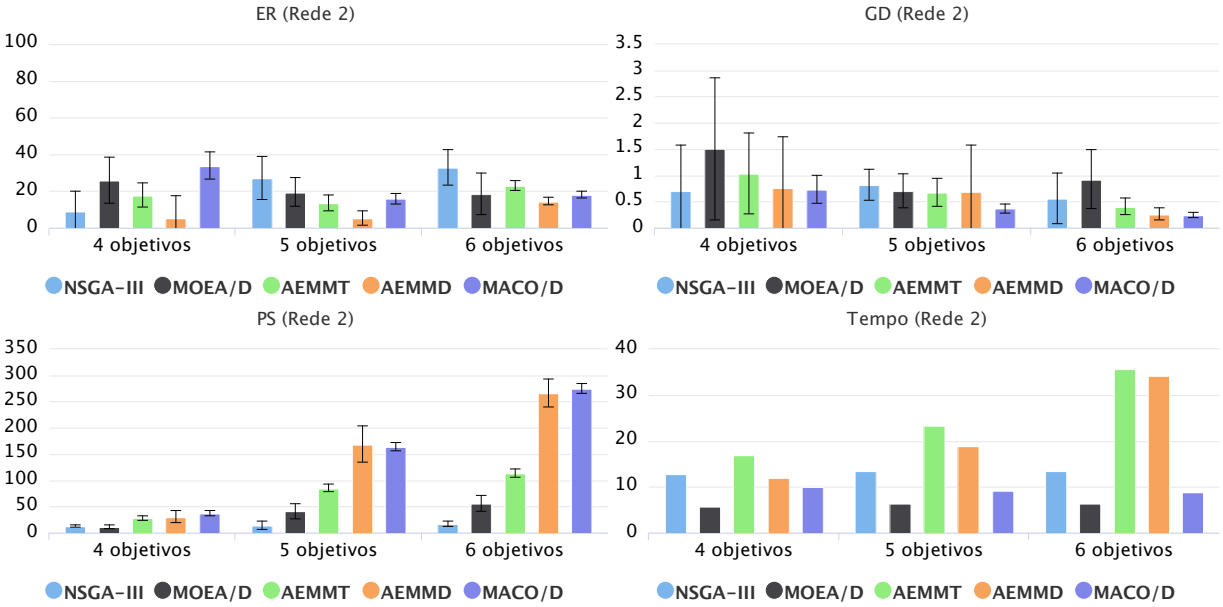
Figura 25 – Etapa 3: resultados para o PRM na rede R_1



Os gráficos correspondentes ao PRM na rede 1 (figura 25) mostram uma vantagem em relação à taxa de erro pelo algoritmo AEMMD. O AEMMT e o MACO/D aparecem, respectivamente, em terceiro e quarto lugar, enquanto o MOEA/D apresenta o maior ER. Com relação ao GD, é possível observar que o MACO/D e o MOEA/D atingem desempenhos bem próximos, sendo o MACO/D o melhor entre os dois. Considerando-se o tamanho das fronteiras de Pareto encontrada (PS), o AEMMD é o melhor algoritmo, seguido pelo MACO/D, AEMMT, MOEA/D e NSGA-III, nessa ordem. O MACO/D e o MOEA/D são os algoritmos mais rápidos, enquanto o AEMMT e o AEMMD são os mais lentos.

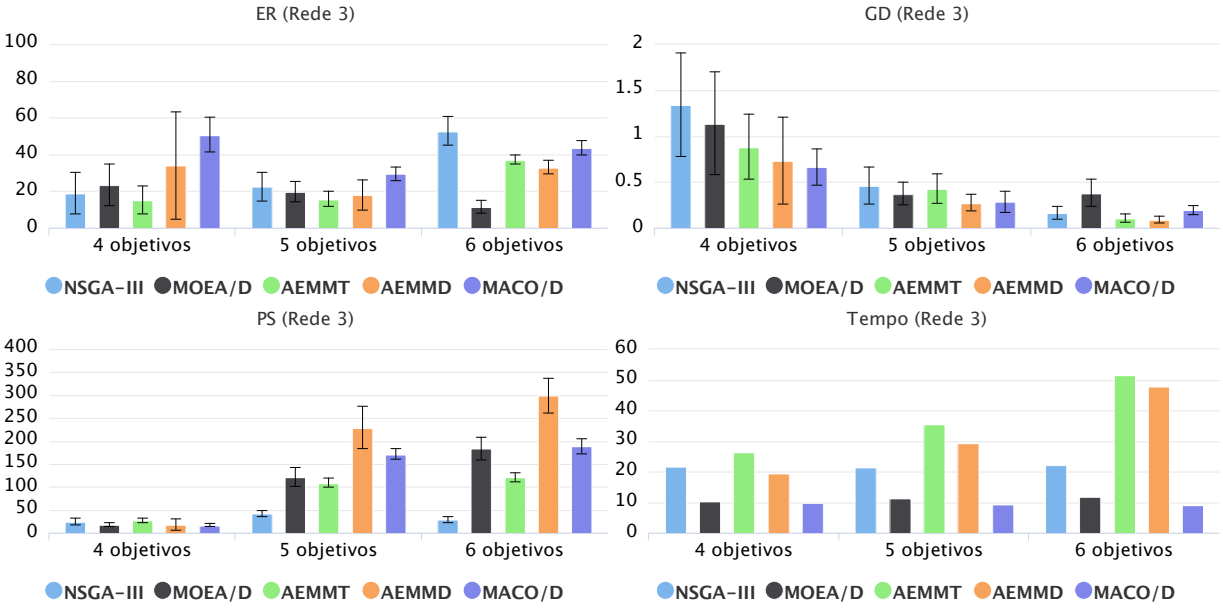
A rede 2 (figura 26), apesar de mais complexa que a primeira, apresenta comportamento bem parecido à instância anterior. O AEMMD produz as menores taxas de erro, seguido pelo AEMMT. Dentre as soluções incorretas encontradas, o MACO/D é o algoritmo que chega mais perto das soluções corretas (GD), sendo que os AEMMT e

Figura 26 – Etapa 3: resultados para o PRM na rede R_2



AEMMD conseguem resultados bem próximos. Com relação ao PS, os melhores valores foram encontrados pelo AEMMD, mas com vantagem muito pequena para o segundo lugar: MACO/D. Em último, aparece o NSGA-III, que é um algoritmo limitado no crescimento do Pareto. O método mais rápido é o MOEA/D e o mais lento é o AEMMT.

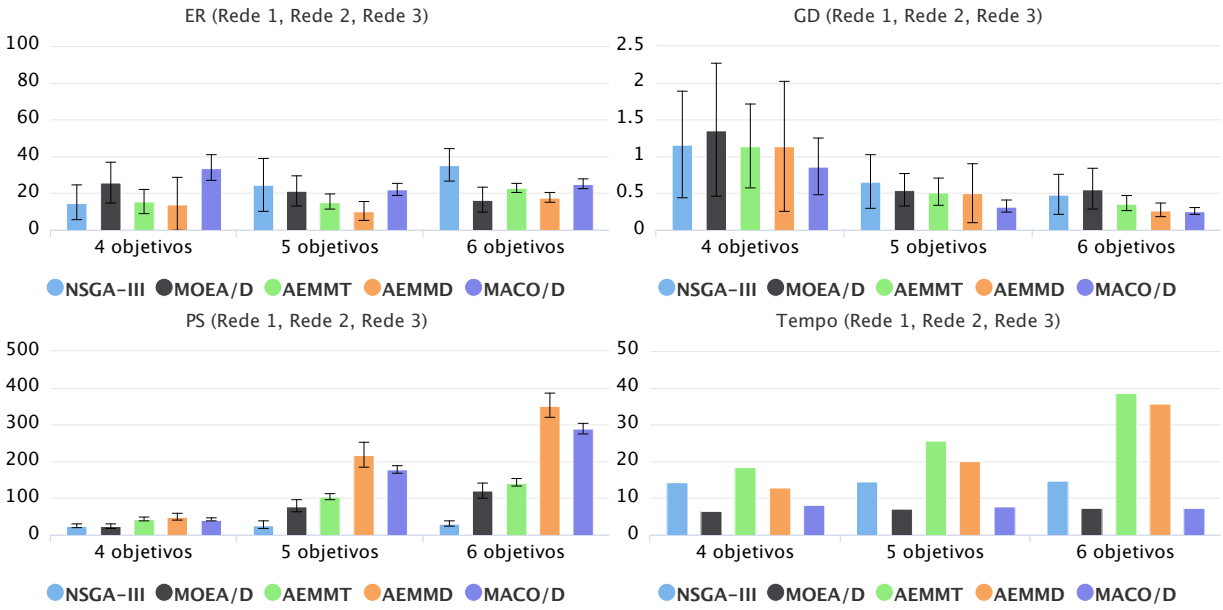
Figura 27 – Etapa 3: resultados para o PRM na rede R_3



No PRM aplicado à rede 3 (figura 27), o AEMMT encontra as melhores taxas de erro para 4 e 5 objetivos, enquanto que para 6 objetivos, o MOEA/D consegue o melhor resultado. O NSGA-III produz o segundo menor ER no problema de 4 objetivos, mas é o segundo pior no de 5 e o pior no de 6. O menor GD no problema com 4 objetivos é dado pelo MACO/D, enquanto nos problemas com 5 e 6 objetivos, o AEMMT obtém o melhor

GD. As maiores fronteiras de Pareto são encontradas pelo AEMMD e MACO/D, sendo o AEMMD o melhor entre os dois. O algoritmo mais rápido é o MACO/D, executando em tempo 5 vezes menor que o método mais lento: AEMMT.

Figura 28 – Etapa 3: resultados agrupados para o PRM nas redes R_1, R_2 e R_3



Para analisar de forma conjunta os resultados das três redes, na figura 28 compila-se todos os experimentos através da média aritmética dos resultados. De maneira geral, o NSGA-III consegue soluções de qualidade razoável, mas apresenta baixo PS e apesar de não ser um algoritmo lento, também não é o mais rápido. O MOEA/D obtém as melhores taxas de erro no problema de seis objetivos e tem desempenho razoável nos problemas de 4 e 5 objetivos, seu GD é levemente mais alto que os demais métodos e seu PS é baixo. A velocidade de execução do MOEA/D é tão boa quanto a do MACO/D, juntos são os dois algoritmos mais rápidos, tornando o MOEA/D uma boa opção se o objetivo da busca é conseguir uma baixa taxa de erro em um curto tempo de execução, sem se preocupar com o PS. Os melhores resultados foram obtidos pelo AEMMD e o método proposto: MACO/D. Com relação ao erro, o AEMMD consegue as melhores taxas, o MACO/D não fica muito atrás. Analisando o GD, observa-se que não há muita diferença entre os dois métodos, mas o MACO/D gera melhores soluções que o AEMMD, além disso, o desvio padrão do MACO/D é consideravelmente menor, o que o revela ser um método mais estável no que se refere à qualidade das soluções. O AEMMD produz as maiores fronteiras de Pareto, mas o MACO/D apresenta resultado bem próximo. O mais interessante nesta análise é o tempo de execução, por mais que o AEMMD gere soluções razoavelmente melhores que o MACO/D considerando as métricas ER e GD, o segundo é consideravelmente mais rápido, chegando a executar em quase 5 vezes menos tempo. Considerando o fato de que o PRM é um problema sensível ao tempo de execução, pelo menos nos cenários analisados, o MACO/D é a melhor opção entre os algoritmos testados.

A primeira diferença que se nota entre os resultados dos dois problemas é o tempo de execução do algoritmo proposto, o MACO/D. considerando os gráficos da figura 24, o MACO/D é o segundo algoritmo mais lento e seu tempo de execução está altamente relacionado ao número de objetivos. Na figura 28, que representa o comportamento geral dos algoritmos no PRM, vê-se o oposto: o MACO/D é o algoritmo mais rápido e seu tempo de execução se mantém estável independente da formulação de objetivos. O processo de maior custo computacional no MACO/D é a atualização dos feromônios, onde é necessário recalcular o conjunto de soluções não-dominadas *nd*. A cada iteração, para toda solução criada, é necessário passar por todos elementos em *nd* verificando a relação de não-dominância. Naturalmente, quanto maior o conjunto *nd* mais caro se torna o processo. No PRM, foram encontradas fronteiras de Pareto de tamanho razoavelmente pequenos, todas com menos de 350 elementos, ou seja, foi necessário, para cada solução criada, fazer no máximo 350 comparações. No caso do PMM, as fronteiras de Pareto são muito grandes, no problema com 6 objetivos e 50 itens, por exemplo, a cardinalidade do conjunto *nd* chega a ser maior que 4500. Quanto maior a quantidade de objetivos do problema, maior o número de soluções no Pareto e mais lento será a classificação de soluções não dominadas. Se o conjunto *nd* é pequeno até mesmo para a maior quantidade de objetivos (caso do PRM), o processo será rápido e o tempo de execução não será muito diferente entre as formulações de objetivos. Por outro lado, se a cardinalidade de *nd* é muito grande e cresce muito com o aumento da quantidade de objetivos, o tempo necessário para se atualizar os feromônios será muito alto e terá grande variação conforme a formulação de objetivos.

A fim de melhor comparar o MACO/D aos algoritmos AEMMT e AEMMD realizou-se o teste de hipótese z-teste com 5% de significância ($\alpha = 0.05$). Os resultados são mostrados nas tabelas 9 e 10. Em ambas as tabelas, uma célula de fundo verde representa uma ocasião onde o MOACS obteve melhor resultado que seu adversário, vermelho significa pior e branco empate.

Tabela 9 – Testes de hipótese para: MACO/D vs. AEMMT nos problemas PMM and PRM

	4 objectives			**5 objectives**			**6 objectives**		
Instance	**ER**	**GD**	**PS**	**ER**	**GD**	**PS**	**ER**	**GD**	**PS**
30 items	=	<	>	>	<	>	>	<	>
40 items	<	<	>	<	<	>	>	<	>
50 items	<	=	>	<	<	>	>	<	>
Rede 1	>	<	<	>	<	>	>	<	>
Rede 2	>	<	>	>	<	>	<	<	>
Rede 3	>	<	<	>	<	>	>	>	>

No PMM, se o tempo de execução é uma preocupação, a melhor alternativa é o algoritmo MOEA/D, que executa em tempo recorde e produz bons resultados. Se a rapidez do algoritmo não é tão importante e deseja-se encontrar um conjunto de soluções mais

Tabela 10 – Testes de hipótese para: MACO/D vs. AEMMD nos problemas PMM and PRM

	4 objectives			5 objectives			6 objectives		
Instance	**ER**	**GD**	**PS**	**ER**	**GD**	**PS**	**ER**	**GD**	**PS**
30 items	<	<	>	<	>	>	<	>	>
40 items	<	>	>	<	>	>	<	>	>
50 items	<	<	>	<	<	>	<	<	>
Rede 1	>	<	<	>	<	<	>	<	<
Rede 2	>	=	>	>	<	<	>	<	>
Rede 3	>	<	<	>	=	<	>	>	<

próximo do Pareto real, o MACO/D é o algoritmo mais indicado. No PRM, o método que representa melhor relação entre qualidade e tempo é o MACO/D, mas se tempo não é importante, o AEMMD pode ser preferível. Ao mesmo tempo, seria interessante testar se, dada a mesma quantidade de tempo, o MACO/D não supera o AEMMD.

8.4 Etapa 4: Análise com hiper-volume

A fim de testar propriamente o comportamento dos algoritmos em espaços de busca mais complexos que os utilizados nos experimentos das etapas 1 e 3, lançou-se mão de duas novas redes (redes 4 e 5) e duas novas instâncias do problema da mochila (100 e 200 itens). Como não é possível extrair Paretos estáveis para as redes R_3, R_4 e R_5, nem para os problemas da mochila com 50, 100 e 200 itens, não é interessante basear-se em métricas dependentes de tais Paretos para se tirar conclusões. Por isso, nesta etapa, testa-se unicamente as métrica hiper-volume e tempo, independentes do Pareto.

O hiper-volume, junto ao *inverse generational distance* (IDG), são as métricas mais utilizadas na literatura para se avaliar algoritmos many-objectives. O hiper-volume, como explicado no início deste capítulo, calcula o volume da figura geométrica formada pelas distâncias das soluções a um ponto de referência pré-definido. Os pontos de referência utilizados em cada cenário são apresentados na tabela 12.

Neste conjunto de experimentos, assim como na etapa 3, comparou-se apenas os cenários many-objectives, com 4, 5 e 6 objetivos. Por esse motivo, ficaram de fora dos testes os algoritmos clássicos NSGA-II e SPEA2. Em contra-partida incluiu-se 3 novos métodos many-objectives na comparação: SPEA2-SDE (AG), MOACS (ACO) e MOEA/D-ACO (ACO). Os parâmetros utilizados para cada método podem ser encontrados na tabela 11.

Sobre os parâmetros marcados com "*" na tabela 11, a quantidade de formigas depende do parâmetro H, do artigo original do algoritmo (KE; ZHANG; BATTITI, 2013), enquanto o tamanho dos grupos depende de outro parâmetro, K, também descrito no mesmo artigo. Ambos os parâmetros H e K servem como guia para gerar os pesos aleatórios das formigas e dos grupos, quanto maior o valor de H, maior será o número de

Tabela 11 – Parâmetros utilizados para o PRM e o PMM na etapa 4 de experimentos.

Parâmetro	**PRM**	**PMM**
Tamanho da população	90	150
Número de comparações	9000	15000
Taxa de crossover	100%	100%
Taxa de mutação	20%	5%
Tamanho da vizinhança (MOEA/D e MOEA/D-ACO)	10	10
Tamanho das tabelas (MEAMT)	30	50
Tamanho da tabela de dominância (MEAMT)	90	150
Número de divisões (NSGA-III)	8	8
α, β, ρ (ACO's)	1, 2, 0.3	1, 4.3, 0.3
Intervalo de valores para os feromônios (ACO's)	[0.1, 0.9]	[0.1, 0.9]
δ (MOEA/D-ACO)	0.2	0.2
Número de formigas e grupos (MOEA/D-ACO)*	6	variável
Número de grupos de formigas (MOEA/D-ACO)*	3	3
Taxa de elitismo (MOEA/D-ACO)	0.9	0
Tamanho das amostras (MACO/D)	10	10
Tamanho do grupo de estruturas ativas (MACO/D)	5	5

formigas, o mesmo vale para K e a quantidade de grupos. De toda maneira, o número de iterações no laço principal é ajustado para que sempre se tenha o mesmo número de comparações (9000 no PRM e 15000 no PMM). Os valores de H usados no PMM foram: $H = 8$ para 4 objetivos, $H = 6$ para 5 objetivos e $H = 5$ para 6 objetivos.

Com relação aos algoritmos NSGA-III e SPEA2-SDE, na etapa 4, tomou-se uma outra atitude em relação às limitações no tamanho do Pareto. Ao invés de impedir o conjunto de soluções não-dominadas de crescerem além do tamanho máximo da população, ajustou-se esse limite para coincidi-lo com o tamanho médio dos Paretos encontrados pelos algoritmos que retornaram os maiores conjuntos de soluções. Dessa forma, ambos NSGA-III e SPEA2-SDE podem encontrar Paretos tão grandes quanto os demais algoritmos. A tabela 12 mostra os limites utilizados para cada cenário de teste.

Na tabela 12, os valores para o PMM são negativos, pois todos os algoritmos foram implementados para lidar com problemas de minimização. Dessa forma, como no problema da mochila deseja-se maximizar o valor de lucro carregado na mochila, todos os valores são multiplicados por -1 antes de se iniciar a busca.

Com relação aos algoritmos baseados em colônias de formigas (MOEA/D-ACO, MOACS e MACO/D), afim de testar o *framework* da forma mais isolada possível, foi utilizada a mesma estratégia de construção da solução para os três métodos. Isto é, as estratégias para se criar uma solução a partir da tabela de feromônios e das heurísticas dos artigos originais do MOEA/D-ACO e do MOACS foram ignoradas em favor das estratégias utilizadas no MACO/D, algoritmo proposto nesta dissertação. A construção da solução para o ambos os problemas foi explicada na seção 2.2.2. No MOEA/D-ACO, cada formiga pos-

Tabela 12 – Ponto de referência e limitações no tamanho do Pareto usados para cada cenário de teste

Instância	**Obj.**	**Ponto de referência**	**Limite**
PMM 50 itens	4	[-15665, -17464, -19122, -16978]	600
	5	[-15948, -15980, -14696, -14800, -14610]	1400
	6	[-16094, -14354, -12511, -14240, -17865, -11801]	4500
PMM 100 itens	4	[-30442, -25071, -30870, -29800]	1300
	5	[-30673, -30266, -30171, -30922, -28821]	3200
	6	[-29389, -27406, -30824, -32040, -30531, -30171]	3400
PMM 200 itens	4	[-64608, -58090, -61540, -59399]	1500
	5	[-62513, -63014, -58939, -64477, -65814]	3000
	6	[-59835, -60434, -65232, -60525, -60843, -60753]	3200
PRM Rede 3	P_4	[0.758449304, 283, 115, 52]	90
	P_5	[0.47215566, 0.776046738, 304, 159, 56]	250
	P_6	[0.471416081, 0.776046738, 310, 159, 56, 83.459]	400
PRM Rede 4	P_4	[0.717830882, 185, 107, 33]	90
	P_5	[0.454221232, 0.776046738, 256, 157, 42]	200
	P_6	[0.457194502, 0.776046738, 239, 157, 40, 80.667]	350
PRM Rede 5	P_4	[0.776046738, 259, 146, 43]	90
	P_5	[0.458729196, 0.776046738, 296, 166, 49]	100
	P_6	[0.458729196, 0.776046738, 287, 177, 48, 101.938]	250

sui uma solução corrente que interfere nas probabilidades ao construir uma nova solução, por isso o processo de construção foi adaptado para suportar essa característica, sempre, ao calcular o feromônio no MOEA/D-ACO, soma-se um novo termo correspondente à presença ou ausência da aresta (ou item) na solução atual da formiga.

Nesta etapa testou-se os algoritmos NSGA-III, SPEA2-SDE, MOEA/D, AEMMT, AEMMD, MOEA/D-ACO, MOACS, e MACO/D. Foram 3 formulações de objetivo para cada problema (4, 5 e 6 objetivos) e 3 instâncias, totalizando 18 cenários de teste (problema, instância e objetivo). Foram realizadas 30 execuções para cada algoritmo em cada cenário e os resultados foram calculados através das médias dos hiper-volumes de cada execução. O tempo foi avaliado através da média de três execuções em uma máquina i7-3770k@4.36GHz.

Foi possível executar os 8 algoritmos 30 vezes cada apenas para o PRM, os gráficos a seguir, para o PMM, possuem apenas 7 algoritmos, sendo que os cenários com 6 objetivos do NSGA-III foram feitos a partir da média de 5 execuções ao invés de 30. O espaço de busca do problema da mochila é muito maior que o do roteamento *multicast*, o que encarece muito o processo e o torna inviável em algumas situações. No PRM, o SPEA2-SDE foi, claramente, o algoritmo mais caro em termos de tempo, mas no PMM, sua execução foi inviável. No cenário mais simples do PMM, com 50 itens e 4 objetivos, o SPEA2-SDE levou 5,4 minutos para executar. Com 50 itens e 5 objetivos, o algoritmo precisou de 1 hora e 11 minutos. Para o próximo cenário, o computador começou a apresentar problemas de falta de memória. Por essa razão, decidiu-se excluir o SPEA2-

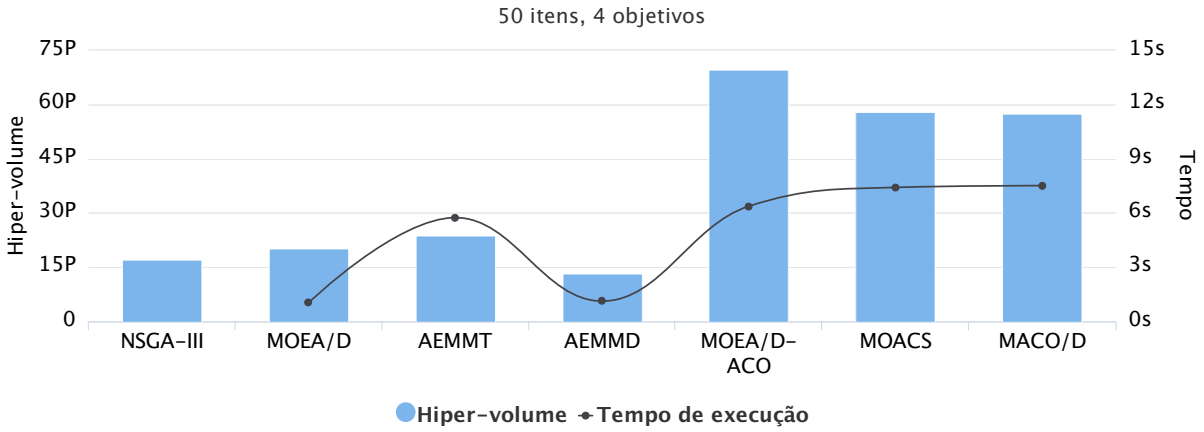
SDE dos experimentos para o problema da mochila. Com relação ao NSGA-III no mesmo problema, o tempo de execução foi muito superior aos demais algoritmos, chegando a mais de uma hora em uma das situações. Por essa razão, nos cenários com 6 objetivos, tomou-se a média de apenas 5 execuções ao invés de 30. Além disso, para facilitar a visualização do tempo de execução nos gráficos, excluiu-se essa informação referente ao NSGA-III, ao invés disso, o tempo do NSGA-III é informado na forma de tabela para cada um dos cenários do PMM (tabela 13).

Tabela 13 – Tempos de execução para o NSGA-III no PMM

Instância	**Obj.**	**Tempo de execução**
PMM 50 itens	4	1m 1s
	5	5m 34s
	6	1h 9m 5s
PMM 100 itens	4	4m 39s
	5	30m 34s
	6	35m 10s
PMM 200 itens	4	6m 21s
	5	26m 52s
	6	30m 49s

As figuras 29 a 37 mostram os resultados para os 9 cenários do PMM, enquanto as figuras 38 a 46 traz os gráficos com os resultados dos 9 cenários do PRM. O hiper-volume é indicado pelas barras e o eixo vertical do lado esquerdo, enquanto o tempo de execução é mostrado pela linha e o eixo vertical do lado direito. No eixo do hiper-volume, as unidades k, M, P e E significam, respectivamente, 10^3, 10^6, 10^15 e 10^18. Para o PMM, como os resultados foram muito similares uns aos outros, fez-se uma análise geral. O PRM, por sua vez, mostrou diferenças significativas entre um cenário e outro e portanto foi analisado caso a caso.

Figura 29 – Resultados do PMM com 50 itens e 4 objetivos



No problema da mochila todos os algoritmos tiveram comportamento parecido ao variar os cenários. Todos os AG's obtiveram hiper-volumes ruins quando comparados aos

Figura 30 – Resultados do PMM com 50 itens e 5 objetivos

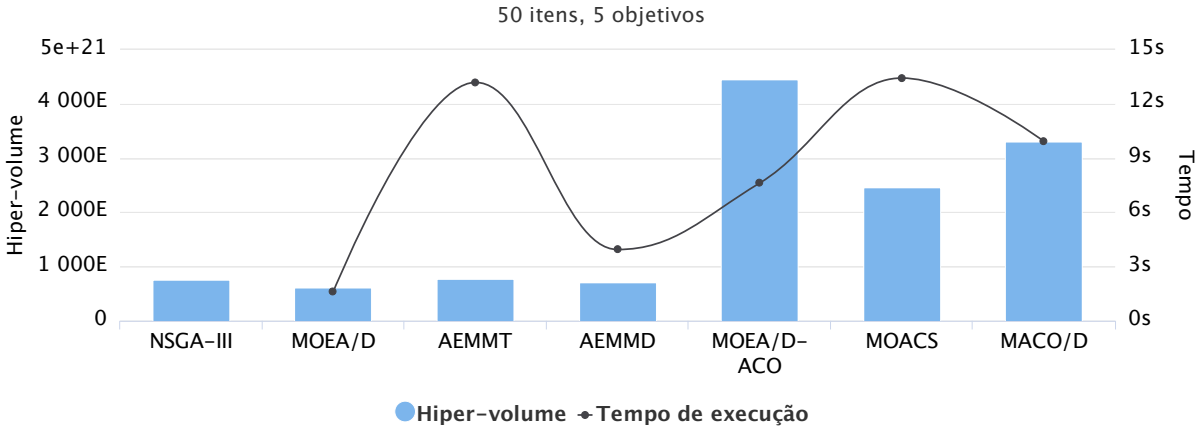


Figura 31 – Resultados do PMM com 50 itens e 6 objetivos

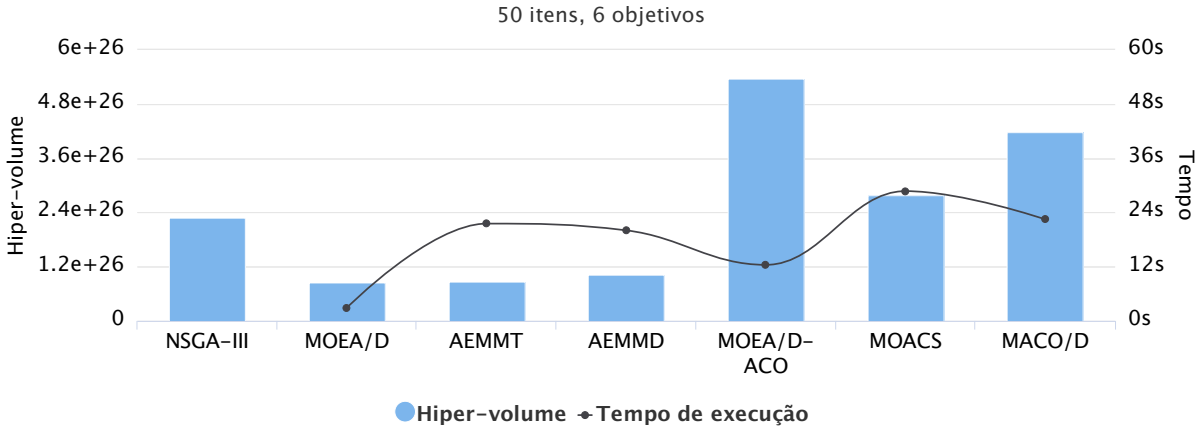


Figura 32 – Resultados do PMM com 100 itens e 4 objetivos

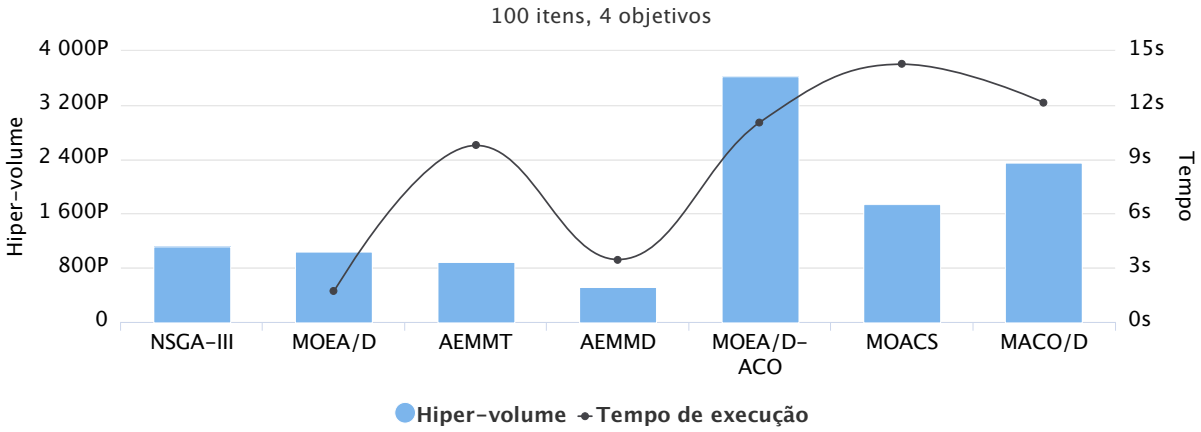


Figura 33 – Resultados do PMM com 100 itens e 5 objetivos

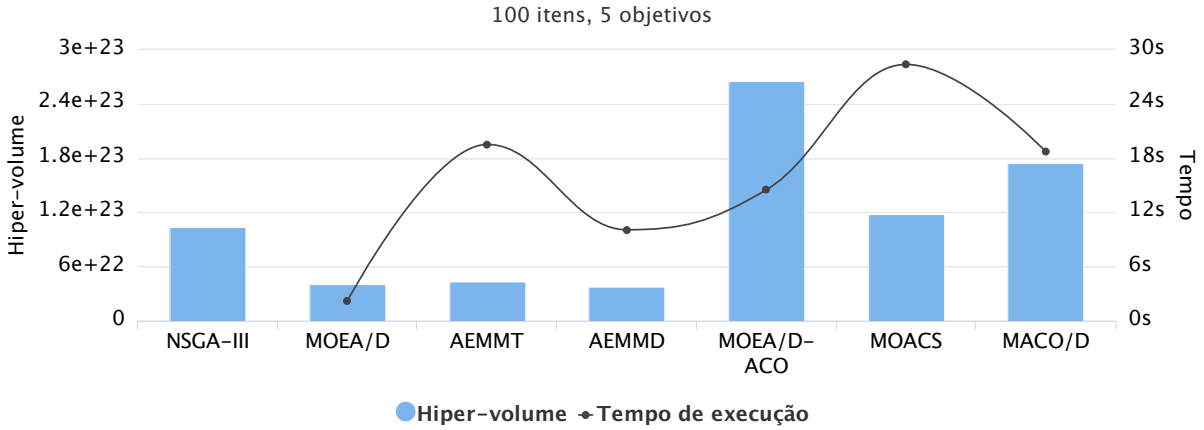


Figura 34 – Resultados do PMM com 100 itens e 6 objetivos

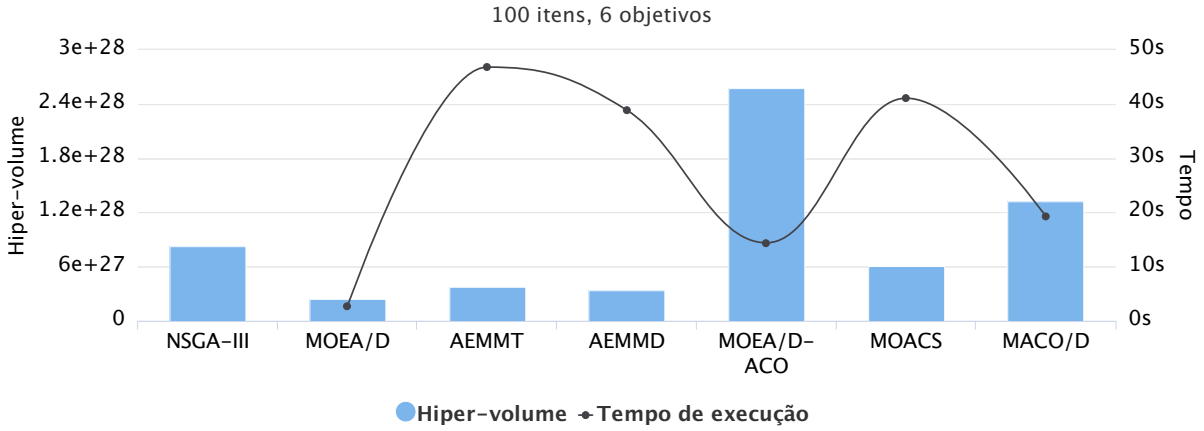


Figura 35 – Resultados do PMM com 200 itens e 4 objetivos

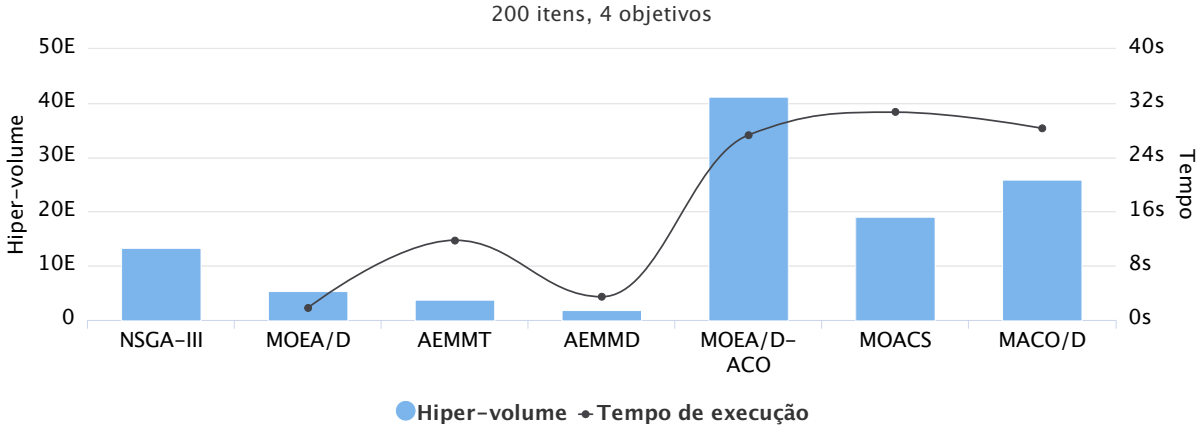


Figura 36 – Resultados do PMM com 200 itens e 5 objetivos

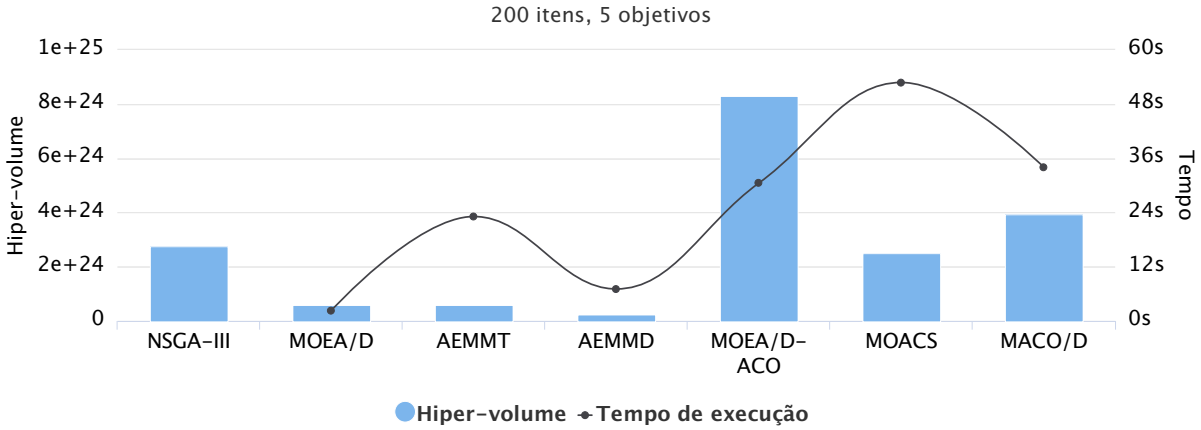
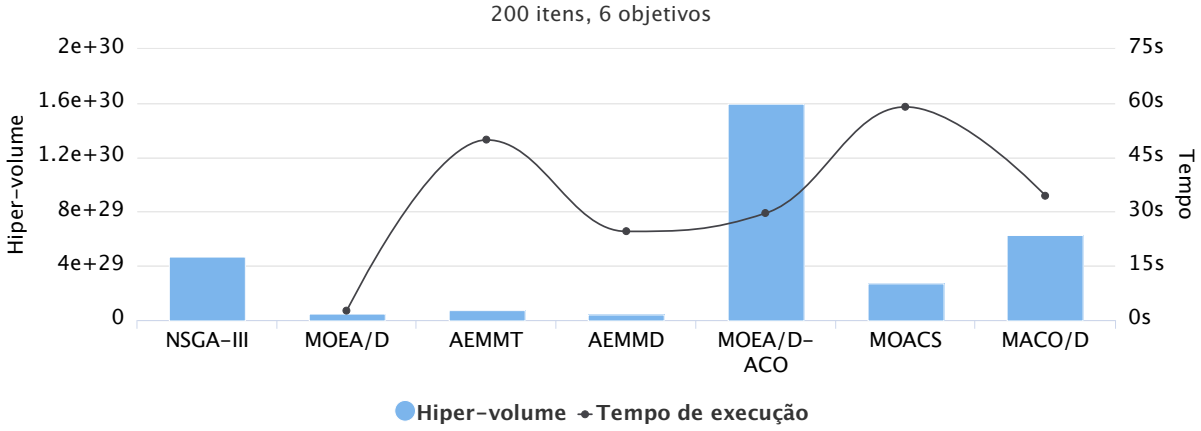


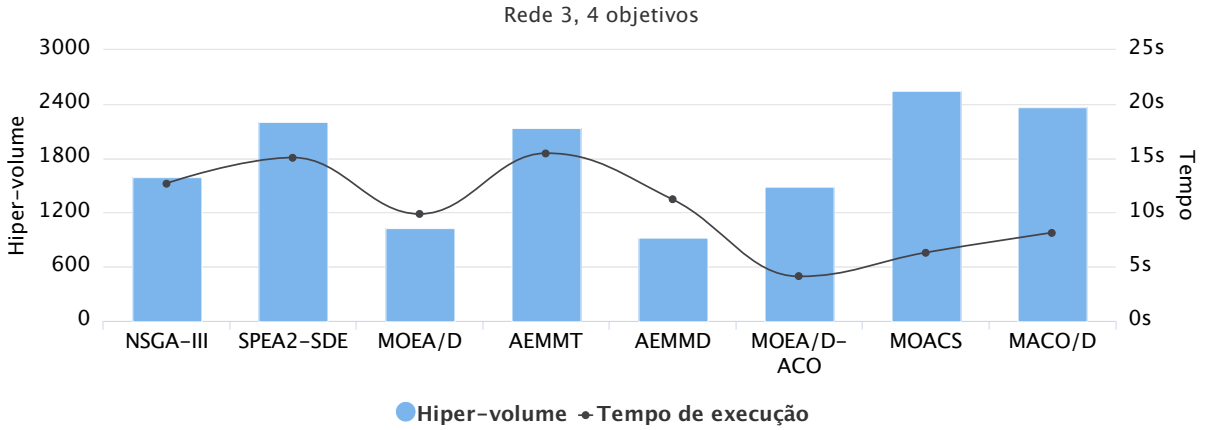
Figura 37 – Resultados do PMM com 200 itens e 6 objetivos



ACO's. Em termos de tempo, o MOEA/D é de longe o mais rápido dentre todos os métodos e, caso essa seja uma grande preocupação, ele pode ser a melhor opção de algoritmo. Além disso, dentre os AG's, o MOEA/D sempre obtém soluções de qualidade similar ou superior ao AEMMT e ao AEMMD. O NSGA-III, apesar de conseguir o melhor hiper-volume dentre os AG's, é muito mais lento que qualquer outro método e apresenta hiper-volume sempre menor que os ACO's, por esse motivo, não é uma boa opção em nenhum dos cenários aqui analisados. A custo de mais tempo, mas ainda se mantendo abaixo da marca de 1 minuto, os ACO's MOEA/D-ACO e MACO/D produziram conjuntos de soluções de qualidade muito superior aos AG's. Dentre os três ACO's, o MOACS demorou mais a executar e obteve os piores resultados, enquanto o MOEA/D-ACO superou os dois outros dois métodos tanto em hiper-volume quanto tempo. Finalmente, os gráficos das figuras 29 a 37 permitem concluir que o MOEA/D-ACO é o melhor algoritmo para o problema da mochila multiobjetivo com 4, 5 e 6 objetivos, a única exceção é caso seja muito importante obter um baixo tempo de execução, nessa situação, o MOEA/D é o melhor método.

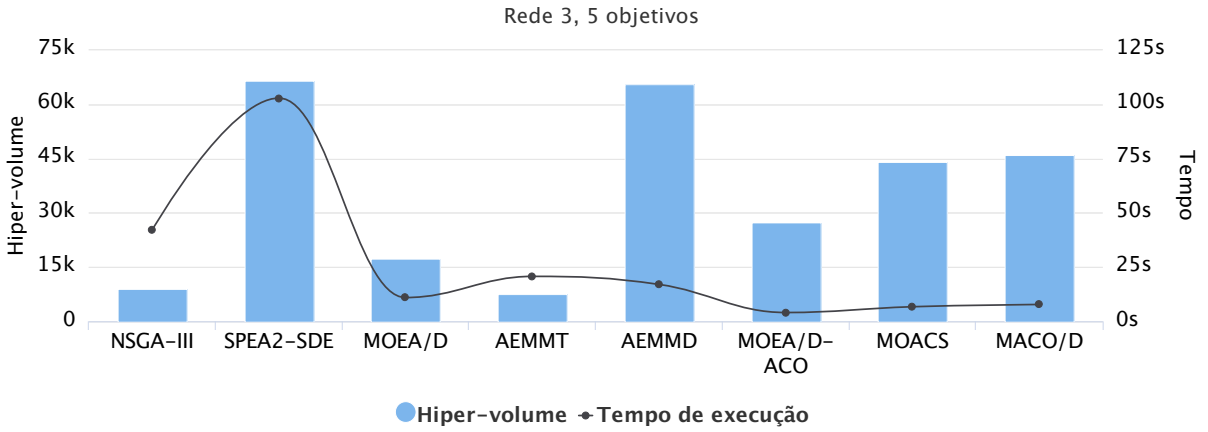
No PRM, inicia-se a análise pelo cenário mais simples: a rede 3 com 4 objetivos. Nesse caso o AEMMD apresenta o pior hiper-volume, enquanto os ACO's MOACS e MACO/D

Figura 38 – Resultados do PRM na rede 3 com 4 objetivos



se destacam, sendo o MOACS o melhor entre os dois, tanto em termos de hiper-volume quanto tempo. O AEMMT atinge hiper-volume quase tão bom quanto o MACO/D mas leva consideravelmente mais tempo para executar. Neste cenário, o MOACS é claramente a melhor estratégia para se resolver o problema.

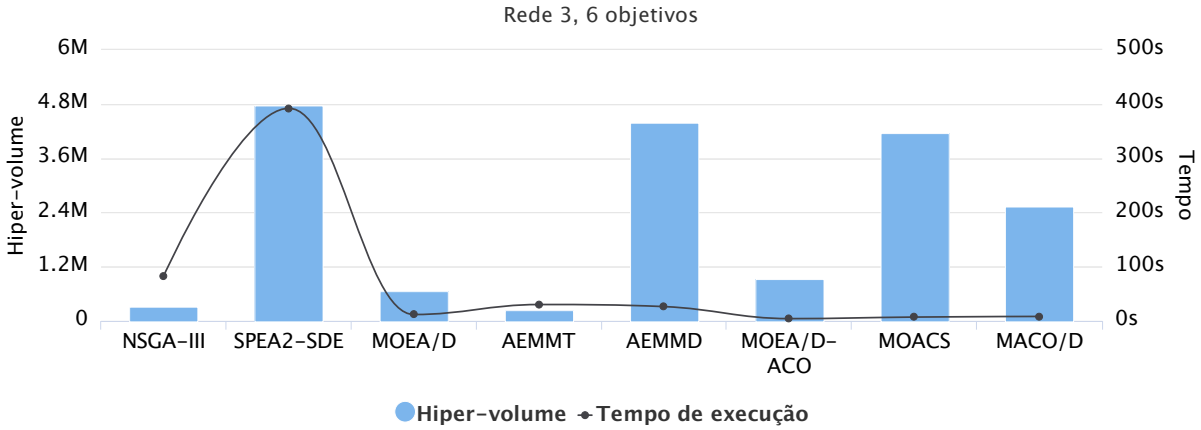
Figura 39 – Resultados do PRM na rede 3 com 5 objetivos



Para a rede 3 com 5 objetivos, a simples modificação que o SDE traz para o SPEA2 o permite obter o melhor hiper-volume entre todos os algoritmos. Infelizmente, um dos piores problemas do SPEA2, o custo do algoritmo, não é resolvido no SPEA2-SDE e, dessa forma, leva-se muito tempo para executá-lo quando comparado às demais estratégias. Os algoritmos NSGA-III, MOEA/D, AEMMT e MOEA/D-ACO obtiveram todos performance ruins. O AEMMD apresentou ótimo valor de hiper-volume com tempo de execução razoável, enquanto os ACO's MOACS e MACO/D precisaram de menos tempo para executar e conseguiram hiper-volumes bons, mas inferiores ao AEMMD. Neste cenário, não é claro qual é o melhor algoritmo. Se um bom hiper-volume é mais desejável que um rápido tempo de execução, o AEMMD é a melhor opção, caso seja de extrema importância a rapidez do algoritmo, ambos MOACS e MACO/D fariam bem o trabalho.

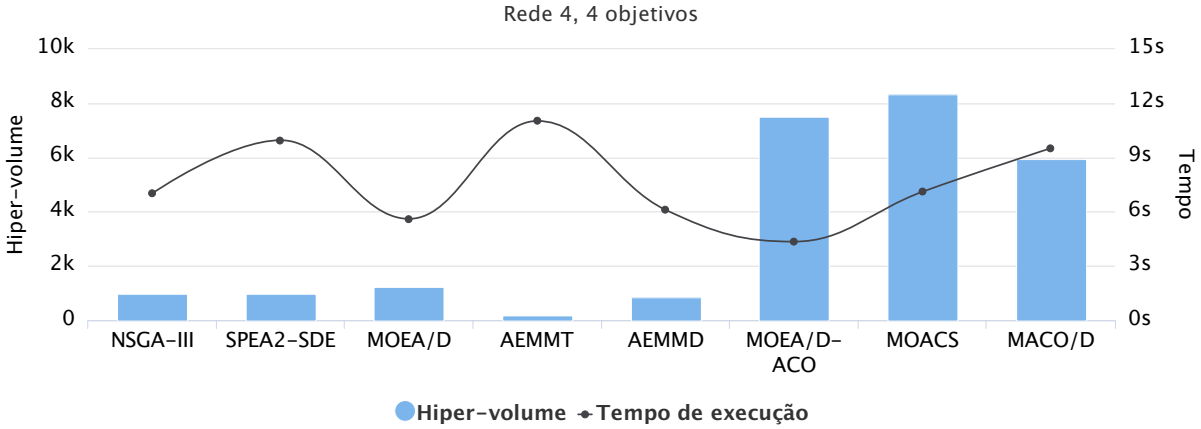
Na rede 3 com 6 objetivos, o SPEA2-SDE repete o comportamento do cenário anterior,

Figura 40 – Resultados do PRM na rede 3 com 6 objetivos



ou seja, consegue um resultado de hiper-volume muito bom, mas a um custo muito alto de tempo. Dentre os demais algoritmos, o único que se destaca é o MOACS, que possui ótimo resultado de hiper-volume e um custo muito pequeno em tempo: 7,3 segundos. O AEMMD consegue hiper-volume pouco melhor que o MOACS, mas a um custo superior em mais de três vezes: 26,6 segundos.

Figura 41 – Resultados do PRM na rede 4 com 4 objetivos



Na rede 4 com 4 objetivos todos os algoritmos genéticos tiveram performance bem fraca, os únicos métodos que apresentaram bom desempenho foram aqueles baseados em colônias de formigas. Dentre os ACO's, destacaram-se o MOEA/D-ACO, que executa em menor tempo e obtém um bom hiper-volume, e o MOACS, que leva mais tempo para executar, mas em compensação produz hiper-volume levemente melhor.

Com 5 objetivos, na rede 4, o SPEA2-SDE volta a mostrar seu potencial quanto ao hiper-volume, mas seu alto custo o torna um algoritmo menos interessante em aplicações práticas. Os algoritmos AEMMD e MOACS são os que apresentam melhor desempenho, o AEMMD consegue maior hiper-volume e leva 10 segundos para executar, enquanto o MOACS garante um resultado levemente pior e executa em 7,3 segundos.

Para 6 objetivos, na rede 4, o custo em tempo do SPEA2 dispara, o que inclusive

Figura 42 – Resultados do PRM na rede 4 com 5 objetivos

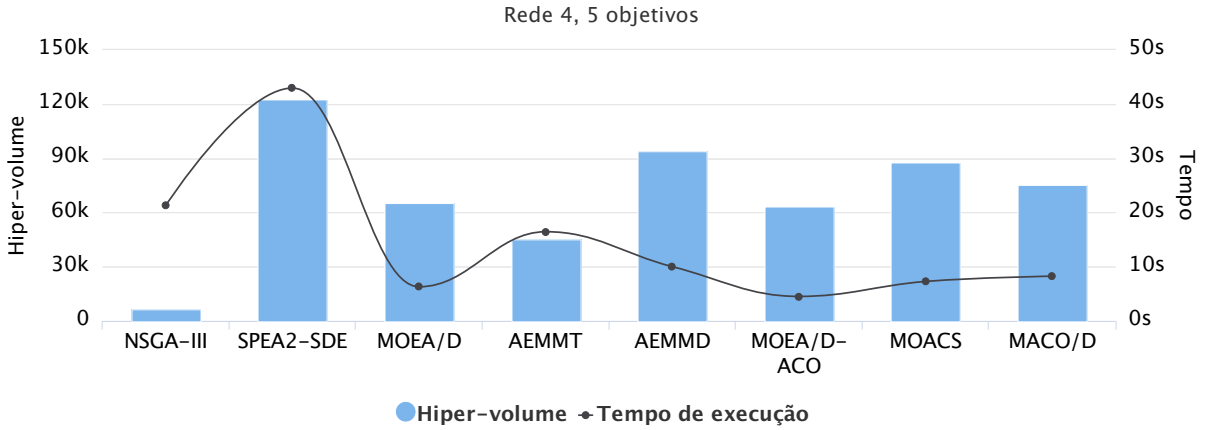
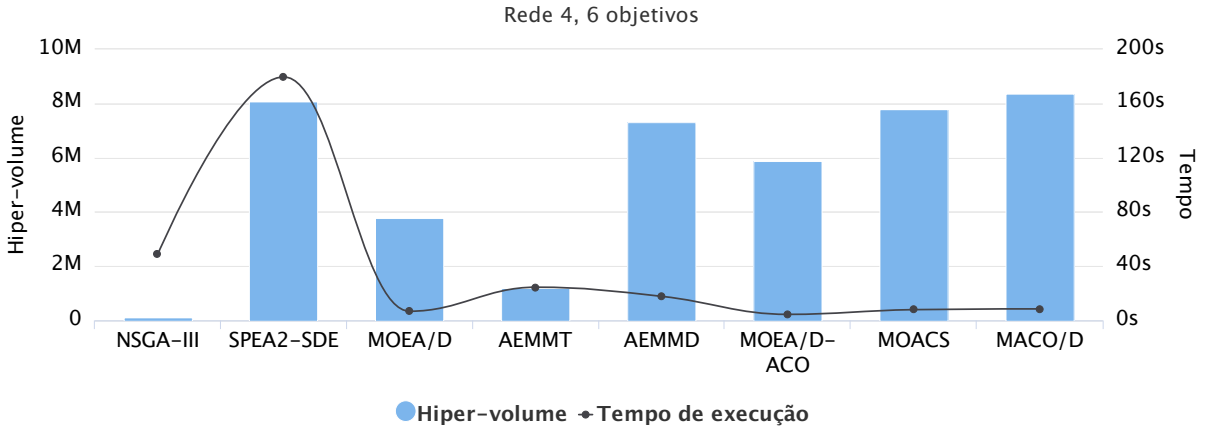
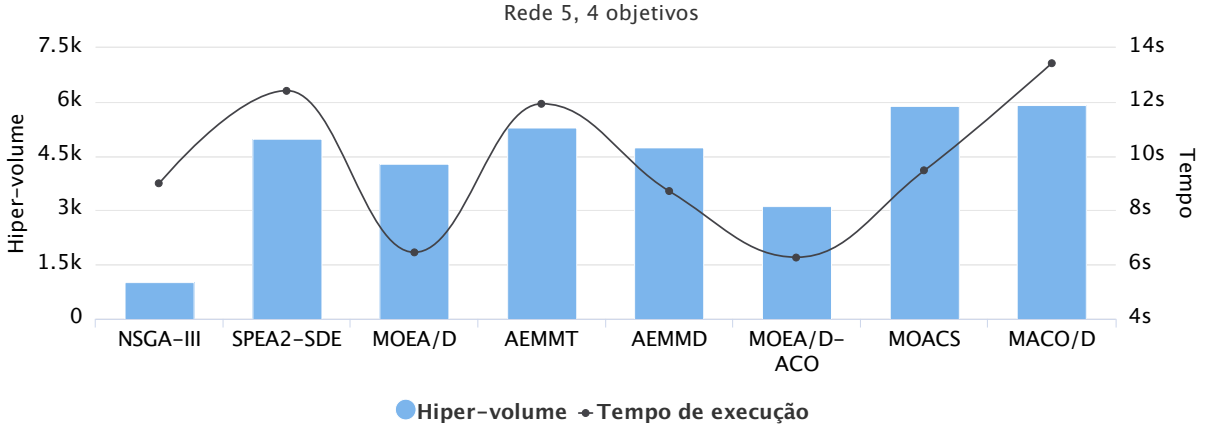


Figura 43 – Resultados do PRM na rede 4 com 6 objetivos



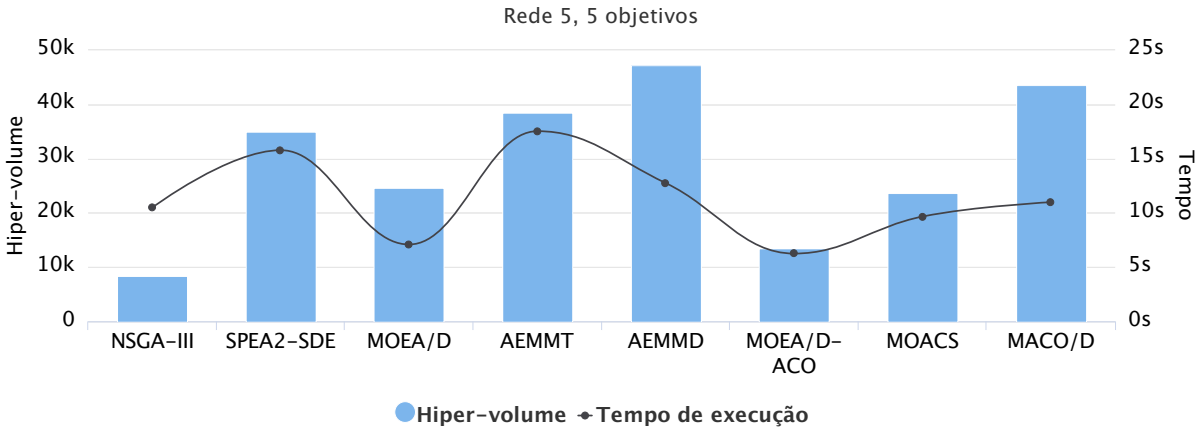
dificulta a visualização do tempo nos demais algoritmos. O melhor hiper-volume é dado pelo MACO/D, o segundo pelo SPEA2-SDE e o terceiro pelo MOACS, sendo a diferença entre os três muito pequena. O SPEA-SDE possui péssima relação custo/benefício quando comparado aos dois ACO's. Entre o MOACS e o MACO/D, o segundo consegue um conjunto de soluções de qualidade levemente superior ao primeiro e leva apenas um segundo a mais para executar, portanto é preferível.

Figura 44 – Resultados do PRM na rede 5 com 4 objetivos



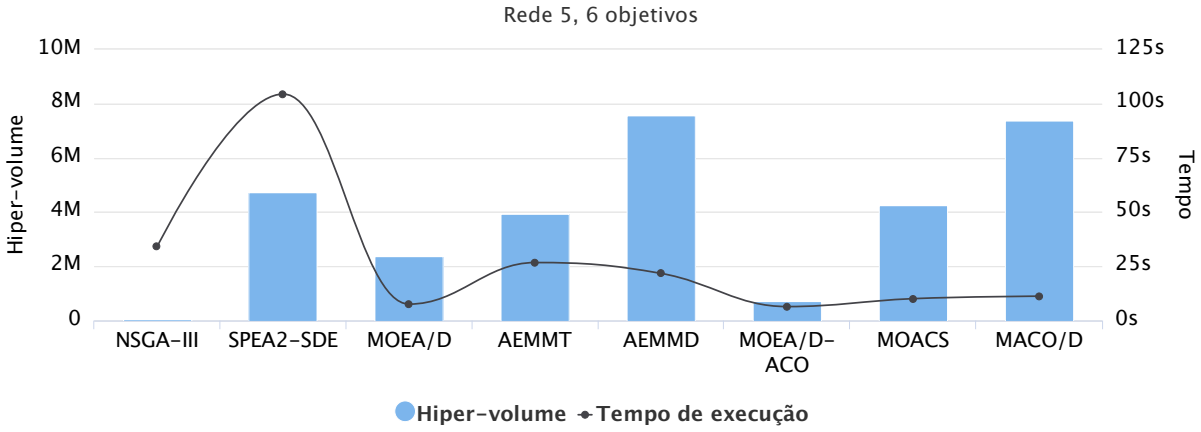
A rede 5 é a mais complexa utilizada em nossos experimentos, para 4 objetivos, o NSGA-III apresentou o pior resultado. O MOEA/D-ACO levou praticamente o mesmo tempo que o algoritmo original MOEA/D, mas obteve um hiper-volume consideravelmente pior; e os demais algoritmos não variaram muito em questão de hiper-volume. A melhor relação custo/benefício foi obtida pelo MOACS, que apresentou o segundo maior hiper-volume e um tempo de execução razoável.

Figura 45 – Resultados do PRM na rede 5 com 5 objetivos



Na rede 5 com 5 objetivos, os melhores hiper-volumes são dados pelo AEMMD e MACO/D. O AEMMD apresenta um conjunto de soluções de melhor qualidade, mas leva um pouco mais de tempo para calculá-las, enquanto as soluções encontradas pelo MACO/D são levemente piores e o tempo necessário para obtê-las é um pouco menor.

Figura 46 – Resultados do PRM na rede 5 com 6 objetivos



O cenário mais complexo do PRM nesta etapa é dado pela rede 5 com 6 objetivos. Nele, os piores resultados foram encontrados pelo NSGA-III e MOEA/D-ACO. O SPEA2-SDE e o AEMMT obtiveram bons hiper-volumes, mas levaram muito tempo para executar. Novamente, repetindo o ocorrido no cenário anterior, o AEMMD e MACO/D foram os algoritmos que atingiram os melhores resultados, ambos com diferenças mínimas um para

o outro: o AEMMD com hiper-volume melhor e tempo pior, e o MACO/D com hiper-volume pior e tempo melhor.

De forma geral, percebe-se que o SPEA2-SDE encontra ótimos resultados, mas a um custo muito alto de tempo, o que o torna inviável para aplicações como o PRM, onde é importante que se tenha uma resposta rápida sobre qual rota tomar. Dentre os demais AG's, o melhor método é o AEMMD que na maioria dos casos consegue o melhor hiper-volume sem tomar muito mais tempo. A grande vantagem dos ACO's em relação aos AG's está no tempo de execução, além disso, em 4 de 9 cenários, os algoritmos baseados em formigas ultrapassam os AG's também em hiper-volume. Para o PRM, parece ser mais adequado utilizar um ACO que um AG, pois assim é possível encontrar soluções de ótima qualidade em um curto espaço de tempo. Dentre os três ACO's o MOACS na maioria das vezes apresentou melhores resultados que o MACO/D, tanto em termos de tempo quanto hiper-volume, mas é interessante notar que, na rede 5, o MACO/D supera o hiper-volume MOACS em todas as formulações de objetivos, o que indica que o MACO/D possa ser o melhor método para redes mais complexas. É importante também lembrar que o MOACS utilizado neste experimento utiliza a abordagem do MACO/D para construir as soluções, o que permitiu que ele obtivesse melhores resultados que o algoritmo original.

Capítulo 9

Conclusão

Este trabalho estendeu a pesquisa de (BUENO; OLIVEIRA, 2010) e (LAFETÁ et al., 2016), adicionando um novo problema de teste, o problema da mochila multiobjetivo (PMM), e 6 novos algoritmos many-objectives, sendo 3 deles AG's (MOEA/D, SPEA2-SDE e NSGA-III) e 3 deles ACO's (MOACS, MOEA/D-ACO e MACO/D). Dentre os ACO's incluídos no trabalho, um deles, o *Many-objective Ant Colony Optimization based on Decomposed Pheromone* (MACO/D), foi proposto pelo autor juntamente com uma nova estratégia de construção de solução para o problema do roteamento multicast (PRM).

O primeiro objetivo do trabalho, representado pela etapa 1 de experimentos (seção 8.1), foi incluir o novo problema PMM e comparar os algoritmos NSGA-II, SPEA2, MOEA/D, NSGA-III e AEMMT utilizando as métricas baseadas em Pareto: taxa de erro (ER), *generational distanc*e (GD) e *pareto subset* (PS). Todos os métodos foram executados para ambos os problemas (PRM e PMM) em diversos cenários variando a complexidade das instâncias e as quantidades de objetivo. O estudo permitiu verificar o comportamento de cada estratégia multiobjetivo em relação tanto à complexidade da entrada quanto ao número de funções de otimização. De maneira geral, confirmou-se a expectativa de se encontrar melhores resultados para os algoritmos clássicos NSGA-II e SPEA2 nos cenários com 2 e 3 objetivos e a decadência em performance dos mesmos a partir de 4 objetivos. Com 4 ou mais objetivos, os algoritmos MOEA/D e AEMMT obtiveram resultados bem melhores que os demais. O NSGA-III, por sua vez, se mostrou o método mais estável: não é o pior nem o melhor na maioria das situações. Para os problemas *many-objectives*, foco deste trabalho, o AEMMT foi o algoritmo que se mostrou mais interessante.

Os AG's *multi* e *many-objectives* estão massivamente presentes na literatura e já foram explorados de diversas maneiras possíveis, por outro lado, os algoritmos baseados em colônias de formigas, apesar do potencial, não são utilizados com a mesma frequência. Os ACO's, por utilizarem uma estratégia de busca diferente, podem encontrar soluções até então inexploradas. Por esse motivo, esta pesquisa estudou vários ACO's na literatura, implementou dois deles e propôs um novo algoritmo, o MACO/D. Um dos principais

aspectos de um ACO é a construção da solução, assim como num AG um dos principais fatores é o processo de *crossover*. Dessa forma, este trabalho também traz uma revisão dos métodos para se construir uma solução no PMM (seção 5.3) e uma nova estratégia para se criar as árvores do PRM (seção 6.5). Afim de se desenvolver o algoritmo proposto para a construção da solução no PRM, efetuou-se a segunda etapa de experimentos (seção 8.2) que comprovou a superioridade do modelo proposto.

Como os problemas multi-objetivos com 2 e 3 funções de otimização são considerados bem resolvidos e através da etapa 1 de experimentos comprovou-se a ineficácia dos algoritmos NSGA-II e SPEA2 para problemas com 4 ou mais objetivos, a partir da etapa 3 de experimentos descartam-se os cenários com 2 e 3 objetivos e deixa-se de incluir os métodos clássicos NSGA-II e SPEA2. Assim, na terceira etapa de experimentos, analisa-se pela primeira vez o comportamento do MACO/D contra os algoritmos genéticos NSGA-III, MOEA/D, AEMMT e AEMMD nos três cenários mais simples de cada problema. A análise é feita através das métricas baseadas em Pareto (ER, GD e PS) e do tempo de execução. Em geral, no PRM, o AG AEMMD e o método proposto, MACO/D, obtiveram os melhores resultados. O AEMMD consegue soluções de qualidade um pouco melhor que seu oponente, mas ao mesmo tempo leva até quatro vezes mais tempo para executar. O MACO/D, apesar de não conseguir o melhor conjunto de soluções entre os dois métodos, chega bem próximo e é um algoritmo muito rápido, característica essencial para um problema de roteamento em redes. Com relação ao problema da mochila, devido ao tamanho muito grande das fronteiras de Pareto, o MACO/D não exibe um comportamento tão bom em relação ao tempo, mas ao mesmo tempo, é de longe o melhor algoritmo em respeito ao PS, atingindo valores muito bons de ER e GD.

Tendo comprovado a eficácia do MACO/D em relação aos algoritmos genéticos utilizando as instâncias mais simples de cada problema, na última etapa de experimentos (seção 8.4), concentra-se apenas nas 3 instância mais complexas do PMM e do PRM, além disso, para que seja possível aferir a qualidade do MACO/D em relação a outras estratégias baseadas em colônias de formigas, incluiu-se os algoritmos MOEA/D-ACO e MOACS. Um novo algoritmo genético também foi incluído nas comparações, o SPEA2. Como forma de avaliação dos resultados, utilizou-se o hiper-volume devido à dificuldade de se extrair um Pareto aproximado para as redes 4 e 5 e os problemas da mochila com 100 e 200 itens. No PRM, foi possível verificar que todos ACO's, normalmente, levam menos tempo para executar que os AG's. Dentre os algoritmos genéticos, o único método com bons resultados foi o AEMMD. O SPEA2 consegue ótimas soluções, mas o alto custo do algoritmo em espaços de alta dimensionalidade faz com que ele seja uma opção inviável para o PRM. Em geral, os ACO's apresentaram uma melhor relação de custo/benefício e dentre eles, o MOACS obteve melhor hiper-volume e tempo na maioria dos casos enquanto o MACO/D mostrou uma tendência de obter o melhor conjunto de soluções à medida que cresce a complexidade da entrada. Quanto ao problema da mochila, os algoritmos se

comportaram de maneira mais estável ao variar os cenários de testes. Para todos os casos do PMM, o MOEA/D-ACO apresentou o melhor custo benefício entre hiper-volume e tempo, fazendo com que o único outro algoritmo considerável seja o MOEA/D, quando realmente é necessário um tempo de execução muito curto.

Desta forma, as principais contribuições deste trabalho para o campo de busca e otimização multi-objetivo foram:

- ❏ Comparação entre AG's: foram comparados 7 algoritmos genéticos multi-objetivos em dois problemas discretos diferentes, o que oferece uma grande gama de dados para que se possa tomar decisões a respeito de qual algoritmo utilizar em determinadas situações.

- ❏ Proposição de um novo ACO e modelo para o PRM: este trabalho propôs uma nova ideia para se implementar ACO's multiobjetivos, o que contribuiu para o meio acadêmico apresentando um algoritmo eficaz que lida bem com problemas de muitos objetivos em um campo relativamente pouco explorado (colônias de formigas multiobjetivo). O novo algoritmo de construção da solução apresentado para o PRM, não só é parte da proposição do MACO/D, como também pode ser utilizado em outros *frameworks* ACO já estabelecidos, melhorando o desempenho do algoritmo.

- ❏ Comparação entre AG's e ACO's: outra grande contribuição desta pesquisa foram as comparações feitas entre os algoritmos genéticos e as colônias de formigas possibilitadas pelos experimentos das etapas 3 e 4 (seções 8.3 e 8.4).

9.1 Trabalhos Futuros

Algumas duvidas surgiram no decorrer deste trabalho e seria muito interessante abordá-las no futuro. Em próximas pesquisas pretende-se investigas os seguintes tópicos:

- ❏ Nos cenários onde se conhece a fronteira de Pareto, seria interessante utilizar a métrica de avaliação *inverse generation distance* (IGD), mais comum que o ER, GD e PS nos trabalhos mais recentes sobre otimização multiobjetivo.

- ❏ O tempo de execução do MACO/D no PMM é muito alto, talvez seja possível criar algum tipo de limitação no tamanho do Pareto que elimine soluções de forma eficiente, diminuindo o tempo sem afetar muito a qualidade das soluções.

- ❏ O MACO/D e o MOACS executam muito mais rápido que o AEMMD em alguns cenários, mas produzem um conjunto de soluções com qualidade levemente inferior. Qual dos três algoritmos obteria o melhor resultado se fossem executados todos durante o mesmo espaço de tempo?

- Nos cenários mais complexos do PRM, o MACO/D apresenta resultados significativamente melhores que o MOACS, essa tendência continuaria ao testar instâncias de redes mais complexas? E quanto a um maior número de objetivos?
- O problema da mochila multiobjetivo aqui testado inclui múltiplos objetivos, mas apenas uma restrição. Algumas variações do PMM trabalham com múltiplos objetivos e múltiplas restrições. O comportamento dos algoritmos mudaria muito com essa outra abordagem? Com uma menor quantidade de elementos no Pareto, os algoritmos levariam muito menos tempo para executar?
- Foram testados 3 ACO's neste trabalho, como pesquisa futura seria interessante investigar outras ideias de ACO's adaptando-nas aos problemas da mochila e do roteamento.
- O MACO/D foi até agora testado em apenas dois problemas, para validá-lo como *framework*, seria de extrema importância uma pesquisa envolvendo sua implementação em outros problemas discretos.

9.2 Contribuições em Produção Bibliográfica

A produção bibliográfica originada neste trabalho compreende os seguintes artigos:

1. "*A Comparative Analysis of MOEAs Considering Two Discrete Optimization Problems*" por Tiago Peres França, Thiago Fialho de Q. Lafetá, Luiz G. A. Martins e Gina M. B. de Oliveira. Publicado e apresentado no evento *Brazilian Conference on Intelligent Systems* (BRACIS) de 2017. Este artigo compreende os experimentos realizados na etapa 1 (seção 8.1). (FRANÇA et al., 2017).
2. "*MACO/D: Many-objective Ant Colony Optimization based on Decomposed Pheromone*" por Tiago Peres França, Luiz G. A. Martins e Gina M. B. de Oliveira. Aprovado no evento *Congress on Evolutionary Computation* (CEC) de 2018 e pendente de publicação. Este artigo propõe o algoritmo MACO/D (capítulo 7) e compreende os experimentos realizados na etapa 3 (seção 8.3).

Referências

AGUIRRE, H.; TANAKA, K. Many-objective optimization by space partitioning and adaptive ε-ranking on mnk-landscapes. p. 407–422, 2009.

ALAYA, I.; SOLNON, C.; GHEDIRA, K. Ant algorithm for the multi-dimensional knapsack problem. p. 63–72, 2004.

_____. Ant colony optimization for multi-objective optimization problems. In: **19th IEEE International Conference on Tools with Artificial Intelligence(ICTAI 2007)**. [S.l.: s.n.], 2007. v. 1, p. 450–457. ISSN 1082-3409.

BADER, J.; ZITZLER, E. Hype: An algorithm for fast hypervolume-based many-objective optimization. **Trans. Evol. Comput.**, v. 19(1), p. 45–76, 2011.

BARAN, B.; SCHAERER, M. A multiobjective ant colony system for vehicle routing problem with time windows. v. 21, p. 97–102, 01 2003.

BAZGAN, C.; HUGOT, H.; VANDERPOOTEN, D. Solving efficiently the 0-1 multi-objective knapsack problem. **Comput. Oper. Res.**, Elsevier Science Ltd., Oxford, UK, UK, v. 36, n. 1, p. 260–279, jan. 2009. ISSN 0305-0548. Disponível em: <http://dx.doi.org/10.1016/j.cor.2007.09.009>.

BEUME, N.; NAUJOKS, B.; EMMERICH, M. Sms-emoa: Multiobjective selection based on dominated hypervolume. **European Journal of Operational Research**, v. 181, n. 3, p. 1653 – 1669, 2007. ISSN 0377-2217. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0377221706005443>.

BRASIL, C. R. S.; DELBEM, A. C. B.; SILVA, F. L. B. da. Multiobjective evolutionary algorithm with many tables for purely ab initio protein structure prediction. **Comput. Chem.**, v. 34(20), p. 1719–1734, 2013.

BUENO, M. L. d. P. **Heurísticas e algoritmos evolutivos para formulações mono e multiobjetivo do problema do roteamento multicast**. 2010.

BUENO, M. L. P.; OLIVEIRA, G. M. B. Multicast flow routing: Evaluation of heuristics and multiobjective evolutionary algorithms. In: **Congress Evol. Comput.** [S.l.: s.n.], 2010. p. 1–8.

CHANGDAR, C.; MAHAPATRA, G.; PAL, R. An ant colony optimization approach for binary knapsack problem under fuzziness. **Applied Mathematics**

and Computation, v. 223, p. 243 – 253, 2013. ISSN 0096-3003. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0096300313008278>.

CHARLES, D. **On the origin of species by means of natural selection, or preservation of favoured races in the struggle for life**. [s.n.], 1859. Disponível em: <https://search.library.wisc.edu/catalog/9934839413602122>.

CORMEN LEISERSON, R.; STEIN. Greedy algorithms. In: FAGERBERG, J.; MOWERY, D. C.; NELSON, R. R. (Ed.). **Introduction to Algorithms, Third Edition**. MIT: MIT Press, 2009. cap. 16.

CRICHIGNO, J.; BARAN, B. Multiobjective multicast routing algorithm. v. 3124, p. 1029–1034, 08 2004.

DEB, K.; JAIN, H. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints. **Trans. Evol. Comput.**, v. 18(4), p. 577–601, 2014.

DEB, K. et al. A fast and elitist multiobjective genetic algorithm: Nsga-ii. **Trans. Evol. Comput.**, v. 6(2), p. 182–197, 2002.

DIJKSTRA, E. W. A note on two problems in connexion with graphs. **Numerische Mathematik**, v. 1, p. 269–271, 1959.

DORIGO, M.; MANIEZZO, V.; COLORNI, A. Ant system: optimization by a colony of cooperating agents. **IEEE Trans. on Systems, Man, and Cybernetics, Part B**, v. 26, n. 1, p. 29–41, 1996.

FIDANOVA, S. Aco algorithm for mkp using various heuristic information. In: DIMOV, I. et al. (Ed.). **Numerical Methods and Applications**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003. p. 438–444. ISBN 978-3-540-36487-0.

FRANÇA, T. P. et al. A comparative analysis of moeas considering two discrete optimization problems. In: **2017 Brazilian Conference on Intelligent Systems (BRACIS)**. [S.l.: s.n.], 2017. p. 402–407.

FRANÇA, T. P.; MARTINS, L. G. A.; OLIVEIRA, G. M. B. de. Maco/nds: Many-objective ant colony optimization based on non-dominated sets. In: **2018 IEEE Congress on Evolutionary Computation (CEC)**. [S.l.: s.n.], 2018.

GOLDBERG, D. E. **Genetic Algorithms in Search, Optimization and Machine Learning**. [S.l.]: Addison-Wesley Longman, 1989.

HRISTAKEVA, M.; SHRESTHA, D. Solving the 0-1 knapsack problem with genetic algorithms. 2013.

_____. Different approaches to solve the 0/1 knapsack problem. 04 2018.

ISHIBUCHI, H.; AKEDO, N.; NOJIMA, Y. Behavior of multiobjective evolutionary algorithms on many-objective knapsack problems. **Trans. Evol. Comput.**, v. 19(2), p. 264–283, 2015.

KANN, V. On the approximability of np-complete optimization problems. 01 1992.

KE, L. et al. An ant colony optimization approach for the multidimensional knapsack problem. v. 16, p. 65–83, 2010.

KE, L.; ZHANG, Q.; BATTITI, R. Moea/d-aco: A multiobjective evolutionary algorithm using decomposition and antcolony. **IEEE Transactions on Cybernetics**, v. 43, n. 6, p. 1845–1859, Dec 2013. ISSN 2168-2267.

KONG, M.; TIAN, P. Introducing a binary ant colony optimization. In: DORIGO, M. et al. (Ed.). **Ant Colony Optimization and Swarm Intelligence**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. p. 444–451. ISBN 978-3-540-38483-0.

KONG, M.; TIAN, P.; KAO, Y. A new ant colony optimization algorithm for the multidimensional knapsack problem. **Computers and Operations Research**, v. 35, n. 8, p. 2672 – 2683, 2008. ISSN 0305-0548. Queues in Practice. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0305054806003236>.

LAFETÁ, T. F. d. Q. **Algoritmos evolutivos many objectives aplicados ao problema de roteamento Multicast com qualidade de serviço**. 2016.

LAFETÁ, T. F. d. Q. et al. Many-objective evolutionary algorithms for multicast routing with quality of service problem. In: **Braz. Conf. on Intelligent Systems**. [S.l.: s.n.], 2016. p. 187–192.

LAFETÁ, T. F. de Q. et al. Meands: A many-objective evolutionary algorithm based on non-dominated decomposed sets applied to multicast routing. **Appl. Soft Comput.**, v. 62, p. 851–866, 2018.

LEGUIZAMON, G.; MICHALEWICZ, Z. A new version of ant system for subset problems. In: **Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)**. [S.l.: s.n.], 1999. v. 2, p. 1464 Vol. 2.

LI, M.; YANG, S.; LIU, X. Shift-based density estimation for pareto-based algorithms in many-objective optimization. **IEEE Transactions on Evolutionary Computation**, v. 18, n. 3, p. 348–365, June 2014. ISSN 1089-778X.

LUST, T.; TEGHEM, J. The multiobjective traveling salesman problem: A survey and a new approach. In: _____. **Advances in Multi-Objective Nature Inspired Computing**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. p. 119–141. ISBN 978-3-642-11218-8. Disponível em: <https://doi.org/10.1007/978-3-642-11218-8_6>.

NRIA, S. et al. Roteamento de veículos utilizando otimização por colônia de formigas e algoritmo genético. p. 219–238, 05 2013.

OMBUKI, B.; ROSS, B. J.; HANSHAR, F. Multi-objective genetic algorithms for vehicle routing problem with time windows. **Applied Intelligence**, v. 24, n. 1, p. 17–30, Feb 2006. ISSN 1573-7497. Disponível em: <https://doi.org/10.1007/s10489-006-6926-z>.

PINTO, D.; BARAN, B. Solving multiobjective multicast routing problem with a new ant colony optimization approach. In: **Int. Latin American Conf. on Networking**. [S.l.: s.n.], 2005. p. 11–19.

PRIM, R. C. Shortest connection networks and some generalizations. **The Bell System Technical Journal**, v. 36, n. 6, p. 1389–1401, Nov 1957. ISSN 0005-8580.

RAI, D.; TYAGI, K. Bio-inspired optimization techniques: A critical comparative study. **SIGSOFT Softw. Eng. Notes**, ACM, New York, NY, USA, v. 38, n. 4, p. 1–7, jul. 2013. ISSN 0163-5948. Disponível em: <http://doi.acm.org/10.1145/2492248.2492271>.

RIVEROS, F. et al. A many-objective ant colony optimization applied to the traveling salesman problem. **J. of Computer Science & Technology**, v. 16(2), p. 89–94, 2016.

SCHAFFER, J. D. Some experiments in machine learning using vector evaluated genetic algorithms. 1985.

SOUZA, M. Z. d.; POZO, A. T. R. Multiobjective binary aco for unconstrained binary quadratic programming. In: **2015 Brazilian Conference on Intelligent Systems (BRACIS)**. [S.l.: s.n.], 2015. p. 86–91.

SRINIVAS, N.; DEB, K. Multiobjective optimization using nondominated sorting in genetic algorithms. **Trans. Evol. Comput.**, v. 2(3), p. 221–248, 1994.

TOTH, P. Dynamic programming algorithms for the zero-one knapsack problem. **Computing**, v. 25, n. 1, p. 29–45, Mar 1980. ISSN 1436-5057. Disponível em: <https://doi.org/10.1007/BF02243880>.

USLU, F. S. Solving knapsack problem with genetic algorithm. In: **2015 23nd Signal Processing and Communications Applications Conference (SIU)**. [S.l.: s.n.], 2015. p. 1062–1065. ISSN 2165-0608.

WHILE, L.; BRADSTREET, L.; BARONE, L. A fast way of calculating exact hypervolumes. **IEEE Transactions on Evolutionary Computation**, v. 16, n. 1, p. 86–95, Feb 2012. ISSN 1089-778X.

ZHANG, Q.; LI, H. Moea/d: A multiobjective evolutionary algorithm based on decomposition. **Evol. Comput.**, v. 11(6), p. 712–731, 2007.

ZITZLER, E.; LAUMANNS, M.; THIELE, L. Spea2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. 2002.

ZITZLER, E.; THIELE, L. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. **Trans. Evol. Comput.**, v. 3(4), p. 257–271, 1999.