DAYANNE GOUVEIA COELHO

Estruturas de Memória e Operadores Adaptativos em Meta-Heurísticas

Belo Horizonte
2016

UNIVERSIDADE FEDERAL DE MINAS GERAIS
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

DAYANNE GOUVEIA COELHO

Estruturas de Memória e Operadores Adaptativos em Meta-Heurísticas

Tese de Doutorado submetida à banca examinadora designada pelo Colegiado do Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Minas Gerais, como requisito para a obtenção do título de Doutora em Engenharia Elétrica.

Orientador: Prof. Dr. Eduardo Gontijo Carrano.

Belo Horizonte - MG
Agosto de 2016

Estruturas de Memória e Operadores Adaptativos em Meta-Heurísticas

DAYANNE GOUVEIA COELHO

Tese defendida no Programa de Pós–Graduação em Engenharia Elétrica da Universidade Federal de Minas Gerais como requisito parcial para obtenção do título de Doutora em Engenharia Elétrica, aprovada em 05 de Agosto de 2016, pela Banca Examinadora constituída pelos professores:

Prof. Eduardo Gontijo Carrano
Departamento de Engenharia Elétrica – (UFMG) – Orientador

Profa. Dra. Elizabeth Fialho Wanner
Departamento de Computação (CEFET-MG)

Prof. Dr. Felipe Campelo Franca Pinto
Departamento de Engenharia Elétrica (UFMG)

Prof. Dr. Lucas de Souza Batista
Departamento de Engenharia Elétrica (UFMG)

Prof. Dr. Rodney Rezende Saldanha
Departamento de Engenharia Elétrica (UFMG)

Prof. Dr. Rodrigo Tomás Nogueira Cardoso
Departamento de Física e Matemática (CEFET-MG)

Agradecimentos

Ao meu orientador Eduardo Gontijo Carrano pela parceria, paciência, críticas e ensinamentos transmitidos durante todas as etapas desta pesquisa. Além do apoio e amizade que foram fundamentais para a conclusão desta etapa tão importante da minha vida.

Á Professora Elizabeth Fialho Wanner pela amizade, incentivo e sugestões valiosas que contribuíram com meu crescimento pessoal e acadêmico. Obrigada por acreditar em mim e por sempre ter me encorajado a seguir em frente.

Aos meus amigos do Departamento de Computação da UFOP, em especial à Amanda, Elton, Guilherme, José Américo, Marcelo, Mariana e Viviane, pela amizade e por me apoiarem nesta fase final do trabalho.

Ao Marcelus e a Vanessa pelas dicas e contribuições além de todo carinho e amizade. E as minhas amigas Luana e Thais que sempre me deram força e conselhos nos momentos em que mais precisei.

Aos amigos que ganhei no PPGEE, por dividir comigo todas as conquistas e angústias deste percurso, em especial a Graziela e ao Marcus Rene.

Aos meus pais, por toda dedicação e amor incondicional. Por sempre me motivarem a seguir em frente e alcançar meus objetivos. À minha irmã, pelas confidências e momentos de carinho. E a todos os meus familiares pelas orações e torcida.

Por fim, agradeço a Deus por ter colocado cada uma dessas pessoas especiais em minha vida.

Resumo

Este trabalho propõe o uso de novos operadores adaptativos em algoritmos de otimização multiobjetivo e meta-heurísticas de busca local. No primeiro estudo, é proposto um operador baseado em técnicas de controle em malha fechada para a realização dos ajustes dos parâmetros dentro dos operadores. Foram desenvolvidos dois operadores de controle por esferas: o operador de redução de arquivo (*archive-set reduction*) e um operador de cruzamento *surface-filling*. Os dois operadores propostos foram incluídas em uma versão padrão do NSGA-II, gerando o Algoritmo Genético Multiobjetivo com Controle por Esferas (SCMGA). Este algoritmo foi utilizado para resolver um problema de otimização multiobjetivo que consiste em obter um conjunto de configurações para um sistema de extrusão de polímeros. No segundo estudo, é proposto um operador de vizinhança adaptativo baseado na ideia de memória, para evitar reavaliações de função de soluções duplicadas e aumentar a exploração pelo espaço de busca. As técnicas desenvolvidas foram incorporadas ao VNS padrão e utilizadas para resolver um problema de *scheduling* com data de entrega comum em uma única máquina. Para verificar a eficiência dos métodos propostos, foram realizados testes com problemas e algoritmos da literatura. Verificou-se, no primeiro estudo, que o algoritmo SCMGA proposto pode ser uma boa escolha na resolução de problemas multiobjetivo, por fornecer as informações detalhadas sobre o conjunto Pareto-Ótimo. Os resultados obtidos na aplicação de um sistema extrusão de polímeros foram satisfatórios. Para o segundo estudo, os resultados indicam que o uso de estratégias de memória e vizinhança adaptativa, aplicadas em meta-heurística para resolver o problema de *scheduling* reduziram o número de avaliações de função e conduziram a melhores soluções: novas melhores soluções da literatura foram encontradas em 79 das 80 instâncias testadas.

Palavras–chave: Algoritmos evolutivos multiobjetivo, meta-heurística de busca local, operadores adaptativos, memória.

Abstract

This thesis proposes the use of new adaptive operators for evolutionary multi-objective optimization and local serach metaheuristics. In the first study, it is proposed an algorithm based on feedback control theory for making adjustments of parameters within the operators. Two sphere-control operators were developed: the archive-set reduction and the surface-filling crossover operator. Both the operators were included in a standard version of the NSGA-II, generating the Multi-objective Genetic Algorithm with Control for Spheres (SCMGA). This algorithm was used to handle a problem of multi-objective optimization which consists in obtaining a set of settings for a polymer extrusion process. In the second study, an adaptive neighborhood operator based on the idea of memory is proposed to avoid the revaluation of duplicate solutions and to increase search space exploration. The techniques developed were incorporated into the VNS algorithm, which was applied to solve a scheduling problem with common due date. In order to check the efficiency of proposed methods, tests with problems and literature algorithms were carried out. It was found in the first study that the proposed SCMGA algorithm can be a good choice to solve multiobjective problems by providing detailed information on the set of Pareto-optimal solutions. The results obtained in the application of polymer extrusion process were satisfactory. For the second study, the results indicate that the use of memory and adaptive neighborhood strategies reduced the number the number of solution evaluations and improved solution quality: new best known solutions were found for 79 of the 80 instances tested.

Keywords algorithm evolutionary multi-objective, metaheuristics in local search, adaptive operators, memory.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Lista de Abreviaturas e Siglas

AEs	Algoritmos Evolucionários
AGs	Algoritmos Genéticos
ANOVA	*Analysis of Variance*
CMA-ES	*Covariance Matrix Adaptation Evolution Strategy*
CMSA-ES	*Covariance Matrix Self-adaptation Evolution Strategy*
BSP	*Binary Space Partitioning*
GAS	*Guided Anisotropic Search*
GRASP	*Greedy Randomized Adaptive Search Procedure*
HdEA	*History Driven Evolutionary Algorithm*
ILS	*Iterated Local Search*
MOEAs	*Multi-Objective Evolutionary Algorithms*
MOGA	*Multi-Objective Genetic Algorithm*
MOP	Problemas de Otimização Multiobjetivo
NrGA	*Non-Revisiting Genetic Algorithm*
NSGA-II	*Nondominated Sorting Genetic Algorithm II*
NrPSO	*Non-Revisiting Particle Swarm Optimization*
NrSA	*Non-Revisiting Simulated Annealing*
PDF	*Probability Distribution Function*
PI	Proporcional-Integral
PSO	*Particle Swarm Optimization*
PRBMO	*Pareto Rank Based Mutation Operator*
RPSGA	*Reduce Pareto Set Genetic Algorithm*
SCMGA	*Sphere-Control Multiobjetive Genetic Algorithm*
SPEA	*Strength Pareto Evolutionary Algorithm*
VND	Descida em Vizinhança Variável
VNS	*Variable Neighborhood Search*
VNS-CM	VNS com Memória
VNS-CMVA	VNS com Memória e Vizinhança Adaptativa

Capítulo 1

Introdução

Otimização (matemática), também conhecida como programação matemática, é uma área da Matemática Aplicada que consiste na elaboração de um planejamento estratégico para resolver problemas de diversas áreas. Resolver um problema de otimização consiste em encontrar as melhores configurações que retornam uma solução para este problema. Em outras palavras, otimizar refere-se a encontrar uma solução que satisfaça todas as restrições que definem o espaço de busca e atenda, ao mesmo tempo, a função objetivo que precisa ser minimizada ou maximizada.

Em geral, dependendo da natureza do problema tratado, encontrar uma boa solução ou a solução ótima não é uma tarefa trivial. Em problemas com características difíceis, para os quais não se encontram disponíveis algoritmos determinísticos eficientes (problemas não-lineares multimodais, não diferenciáveis, que envolvem múltiplas escalas, problemas combinatórios, entre outros) o custo computacional envolvido no processo de otimização, por exemplo, é um fator que precisa ser considerado. Por serem facilmente adaptadas a qualquer tipo de problema, as meta-heurísticas tornaram-se importantes ferramentas para o tratamento de problemas com esse perfil.

Apesar de se mostrarem eficientes, uma desvantagem das meta-heurísticas em relação aos métodos exatos, é o fato de não garantirem que a solução final encontrada seja a solução ótima do problema. Em alguns casos, o algoritmo pode ser interrompido mesmo que ainda não se tenha obtido uma solução aceitável. Além disso, algumas destas técnicas, como por exemplo os algoritmos evolucionários, necessitam de um número elevado de avaliações de função durante sua execução. Nestes casos, o uso das meta-heurísticas podem se tornar restritas quando o custo computacional que envolve uma avaliação de função é bem relevante.

Uma boa convergência das meta-heurísticas dependem de uma solução inicial de boa qualidade, dos valores dos parâmetros do algoritmo apropriados para o problema, das estruturas de vizinhanças e operadores adequados e que levem em consideração as características do problema. A dependência existente entre o desempenho dos algoritmos em relação a esses fatores faz com que muitos pesquisadores concentrem esforços para melhorar estas ações. Nos últimos anos a ideia da calibração dos parâmetros nos

componentes dos algoritmos, tais como operadores, mecanismos de seleção ou tamanho da população vem ganhando destaque [23, 24]. Dependendo dos valores escolhidos para estes parâmetros, o algoritmo pode ou não encontrar de forma eficiente a solução ótima do problema.

De forma geral, as configurações dos parâmetros dos algoritmos são obtidas por meio de dados empíricos baseados nas experiências anteriores do otimizador ou a partir de dados obtidos na literatura. Além disso, os operadores e os valores de seus parâmetros são escolhidos antes do início do processo de otimização e fixos até o final deste. Assim, determinar a melhor configuração destes parâmetros e obter um conjunto de valores que se aplique a qualquer tipo de problema, não é uma tarefa simples e pode consumir muito tempo. Por estas razões cada vez mais, a aplicação de operadores que fazem a adaptação dos parâmetros durante a execução nas meta-heurísticas, tem se tornado um recurso promissor no que envolve a melhoria do desempenho dos algoritmos de otimização.

A adaptação nas meta-heurísticas parte do princípio de desenvolver operadores que sofrem modificações em seu comportamento durante a execução do algoritmo, dependendo do desempenho do mesmo. A adaptação pode ser definida como alterações nas ações dos operadores, incluindo até os ajustes de seus parâmetros [27]. Seguindo a mesma ideia, o termo auto-adaptação é usado para definir um sistema que seja capaz de adaptar-se de forma autônoma [43]. Na Computação Evolutiva estes termos são utilizados principalmente, para denotar técnicas de ajustes ou controle de parâmetros, como por exemplo, o ajuste do tamanho do passo do operador de mutação. A auto-adaptação também é usada durante a execução dos algoritmos para fazer o controle das suas propriedades, tais como os operadores genéticos, os valor dos parâmetros destes operadores ou o tamanho da população.

O conceito de auto-adaptação dos operadores nas Estratégias Evolutivas foi introduzido inicialmente nos trabalhos de [39, 71]. Como exemplo de algoritmos que fazem uso da auto-adaptação podem ser citados o *Covariance Matrix Adaptation Evolution Strategy* (CMA-ES) proposto por [37] e o *Covariance Matrix Self-adaptation Evolution Strategy* (CMSA-ES) proposto por [4]. Ambos os algoritmos são variantes das Estratégias Evolutivas clássicas que tem por finalidade obter um maior direcionamento de busca através de uma técnica de auto-adaptação da matriz de covariância, que por sua vez é usada para auto-ajustar os valores dos parâmetros do operador de mutação.

Neste contexto, visto que o uso de operadores adaptativos nas meta-heurísticas podem contribuir com a melhoria do desempenho desses algoritmos, o presente trabalho tem por objetivo estudar e propor novos operadores adaptativos aplicados em meta-heurísticas para a resolução de problemas de otimização. A ideia principal dos operadores aqui propostos é direcionar o algoritmo com base nas informações extraídas da população atual e do processo de busca local do algoritmo. Assim, neste trabalho são feitos dois estudos de caso que seguem linhas distintas.

No primeiro estudo é proposto um operador adaptativo, que realiza o auto-ajuste dos seus parâmetros, para um algoritmo genético multiobjetivo (do inglês, *multi-objective evolutionary algorithms* (MOEAs)) e a metodologia proposta é aplicada a um problema real. São desenvolvidos novos operadores adaptativos para o MOEA baseados em controle do raio de esferas. Esses operadores utilizam as informações sobre as distâncias entre os indivíduos de cada amostra do conjunto de estimativas de Pareto para melhorar a qualidade da descrição desse conjunto e gerar novas soluções nas regiões do espaço de busca que estão menos povoadas, garantindo assim, um conjunto de soluções diversificadas e de boa qualidade.

Diversos problemas reais são modelados como um problema de otimização multiobjetivo, e uma das técnicas mais usadas para resolver problemas com esta configuração são os Algoritmos Genéticos (AGs). Esses algoritmos, por trabalharem com um conjunto de soluções em todas as suas iterações, apresentam a convergência mais lenta em comparação às demais meta-heurísticas. Por esta razão, seus operadores necessitam de um tratamento específico. Na literatura, são encontrados trabalhos que desenvolvem mecanismos para diversificar o conjunto de soluções. Alguns deles envolvem a criação de novos operadores ou adaptações aos operadores usuais de mutação, cruzamento e seleção. Uma das principais preocupações no estudo dos algoritmos evolutivos multiobjetivo é garantir a qualidade e diversidade do conjunto de Pareto. Por ser responsável em manter a diversidade do conjunto de soluções, a maioria dos trabalhos se dedicam a fazer adaptações no operador de mutação, como pode ser visto em [13, 56, 91, 92].

Fazer esse ajuste dos valores dos parâmetros dos algoritmos pode ser desafiador dependendo do problema, dos operadores e da quantidade de parâmetros envolvidos. Neste intuito, vários trabalhos consideraram relevante o estudo a respeito da auto-adaptação dos parâmetros nos algoritmos, para impedir que esse ajuste seja feito manualmente. Uma proposta que vem se destacando como uma ferramenta para a criação de parâmetros adaptativos é inspirada em controle realimentado. O conceito de controle realimentado permite o uso de procedimentos de adaptação de parâmetros e de recuperação de informações. Alguns estudos neste sentido podem ser vistos nos trabalhos de [41, 53, 60, 73, 76, 80].

No segundo caso é proposto um operador adaptativo, baseado em técnicas de memória, para uma meta-heurística de busca local convencional e os métodos propostos são aplicados a um problema discreto de otimização combinatória. Nesta abordagem, as técnicas propostas fazem uso de um arquivo de memória, que é utilizado para evitar reavaliações de soluções já visitadas e, um operador de vizinhança adaptativo, que realiza a exploração no espaço de busca por regiões ainda não exploras, garantindo um melhor desempenho da meta-heurística em relação a exploração do espaço de soluções.

Na literatura, novas técnicas são desenvolvidas para a melhoria da exploração do espaço factível de soluções e para a busca por regiões que sejam mais promissoras neste

espaço, o intuito é garantir uma convergência mais rápida do algoritmo. De acordo com [86], uma região promissora é definida como um subconjunto do espaço de busca que contenha uma quantidade significativa de soluções de boa qualidade para o problema. Em trabalhos como os de [49, 58, 92] são desenvolvidas técnicas para identificar quais são as regiões mais promissoras do espaço de busca.

O número de avaliações da função objetivo também é considerado um fator que pode afetar o desempenho das meta-heurísticas. Nos problemas em que uma avaliação da função requer um tempo computacional expressivo, um número de avaliações elevado gera um custo computacional muito alto. Na literatura são encontradas duas abordagens para diminuir a quantidade de avaliações de função efetuadas. Na primeira vertente, são propostas metodologias para que a avaliação da função seja realizada por aproximação, enquanto na segunda vertente são estudadas técnicas para evitar reavaliações de uma mesma solução. Exemplos da primeira perspectiva são obtidos nos trabalhos de [13], onde as soluções são armazenadas em uma árvore binária e a avaliação da função acontece considerando os valores de aptidão das soluções que estão armazenadas em uma mesma região desta árvore, e nos trabalhos [73, 77], onde são usadas as informações do raio de uma esfera para representar um domínio da solução contida no centro dessa esfera representativa, de tal forma que não são necessárias avaliações da função para os demais pontos concentrados no interior desta esfera. Nos trabalhos [8, 88, 92] são encontrados exemplos da segunda vertente, que propõem operadores para impedir que o algoritmo avalie mais de uma vez as soluções que são replicadas.

Operadores que fazem uso de uma estratégia de memória para armazenar todas as soluções geradas pelos algoritmos são proposto em [8, 13, 88, 92]. Nestes trabalhos, a partir do histórico de soluções, o algoritmo verifica se uma nova solução foi ou não avaliada, evitando caso necessário, novas reavaliações. No caso específico de [13], a avaliação acontece de acordo com a localização da solução no histórico.

Além da reavaliação de soluções idênticas, a memória também pode armazenar as informações a respeito das operações que foram realizadas pelo algoritmo, a localização de uma solução no espaço de busca, a quantidade de replicações geradas de uma mesma solução e outras informações que sejam relevantes. Em [8, 13, 91], essas informações extras armazenadas no histórico são utilizadas como mecanismo para conduzir a exploração do espaço de busca e auxiliar o algoritmo na orientação durante a procura por novas soluções. As estrutura de memória indica quais as regiões que possuem soluções mais promissoras para o problema e quais regiões possuem soluções de pior qualidade.

Baseado nos trabalhos [73, 76] e considerando relevante o estudo a respeito dos operadores adaptativos e do auto-ajuste dos parâmetros para manter a qualidade e diversidade do conjunto de estimativas de Pareto em todas as iterações do algoritmo, o estudo e proposta por novos operadores adaptativos aplicados em algoritmos multiobjetivo na primeira parte deste trabalho é considerado relevante. Além disso, fundamentado nos tra-

balhos [8, 13, 75, 92] e sabendo da existência de problemas de otimização que possuem o cálculo da função objetivo custoso, justifica-se o interesse na segunda parte deste trabalho pelo estudo de mecanismos que reduzem a quantidade de avaliações de função e, com isso, o esforço computacional dos algoritmos de otimização. Nestes últimos artigos referenciados, o principal interesse é o de se evitar reavaliações de uma mesma solução e utilizar operadores adaptativos para melhorar a exploração no espaço de busca. É interessante destacar que estudos reforçam a ideia de que a redução da quantidade de avaliações da função e uma boa orientação na exploração do espaço de busca, pode reduzir o esforço computacional e garantir uma convergência mais rápida do algoritmo [89]. Assim, o estudo de estruturas de memória vinculadas às meta-heurísticas pode contribuir para a resolução de diversos problemas de otimização.

Neste contexto, o objetivo desta tese é estudar e propor novos operadores adaptativos para as meta-heurísticas clássicas a fim de contribuir com o melhor desempenho destas técnicas. Para o algoritmo multiobjetivo, os operadores propostos são baseados em controle realimentado e no controle do raio de esferas, utilizados para melhorar a qualidade e garantir a diversidade do conjunto de amostras de Pareto, e para a meta-heurística de busca local, as estratégias usadas são baseadas em técnicas de memória e têm como finalidade reduzir o número de avaliações da função e auxiliar na exploração do espaço de objetivos, evitando o desperdício de recursos computacionais com reavaliações de uma mesma solução. Os métodos propostos se mostram eficientes quando comparados com as versões originais das meta-heurísticas escolhidas.

1.1 Objetivo

O presente trabalho apresenta uma metodologia baseada em operadores adaptativos para duas classes de meta-heurísticas diferentes, uma baseada em busca local e outra em busca populacional. Assim, os principais objetivos desta tese são:

- Desenvolver novos operadores adaptativos para algoritmos multiobjetivo, com o intuito de melhorar a qualidade, espalhamento e distribuição das soluções, ao longo do conjunto de estimativas de Pareto.
- Desenvolver novos operadores, baseados em técnicas de memória para as meta-heurísticas de busca local, a fim de evitar revisitações de soluções e melhorar a exploração pelo espaço de objetivos.

1.2 Contribuições deste Trabalho

As principais contribuições desta tese consistem em:

- Proposição de operadores baseados em controle realimentado para solução de problemas de otimização multiobjetivo utilizando algoritmos evolucionários.
- Desenvolvimento do algoritmo SCMGA (*Sphere-Control Multiobjetive Genetic Algorithm*) usando operadores baseados em controle do raio de esferas.
- Aplicação do SCMGA em um problema real e complexo, de um sistema de extrusão de polímeros, com bons resultados.
- Proposição de uma nova função de *hash* para problemas permutacionais e inteiros, com bom espalhamento e baixo índice de colisões.
- Proposição de um operador de vizinhança adaptativo baseado em memória para problemas permutacionais.
- Construção de duas variantes do algoritmo VNS (*Variable Neighborhood Search*) para solução do problema de *common due date scheduling*.
- Obtenção de novos limitantes superiores para 79 das 80 instâncias testadas do problema de *common due date scheduling*.

1.3 Organização do Trabalho

Este texto é dividido em 5 capítulos e está organizado da seguinte forma: o Capítulo 2 apresenta uma revisão bibliográfica sobre as meta-heurísticas e os trabalhos que abordam a ideia de operadores adaptativos como novas técnicas de melhorias para estes algoritmos. São apresentados estudos que utilizam a ideia de controle e o auto-ajuste dos parâmetros para manter a diversidade do conjunto de soluções nos algoritmos multiobjetivos, além de estudos que aplicam o conceito de memória para melhorar o desempenho na exploração do espaço de busca em meta-heurísticas de busca local.

O Capítulo 3 apresenta os operadores baseados em controle realimentado para algoritmos evolucionários multiobjetivos e desenvolve o algoritmo SCMGA, usando operadores baseados em controle do raio de esferas. Esses operadores utilizam a ideia de distância para aumentar a capacidade do MOEA em representar o conjunto de estimativas de Pareto. Ainda neste capítulo é apresentado uma aplicação da metodologia desenvolvida para o problema de otimização dos parâmetros operacionais e geométricos de um sistema de extrusão de polímeros de parafuso único.

O Capítulo 4 apresenta um novo operador de vizinhança adaptativo baseado em memória para a meta-heurística VNS. Neste capítulo, a metodologia proposta é aplicada ao problema de *scheduling* com data de entrega comum. Testes estatísticos são realizadas para gerar uma boa configuração do conjunto de parâmetros e do conjunto de estruturas

de vizinhança do algoritmo, e os resultados computacionais obtidos pelas metodologias são apresentados e comparados com a versão original do VNS.

Por fim, no Capítulo 5, são apresentadas as considerações finais e algumas propostas de continuidade deste trabalho.

Capítulo 2

Revisão Bibliográfica

Este capítulo apresenta uma revisão da literatura a respeito do tema abordado no presente trabalho. Para contextualizar a aplicação dos operadores adaptativos como técnicas de melhoria para as meta-heurísticas mono e multiobjetivo, uma introdução sobre as principais meta-heurísticas existentes são apresentadas e em seguida são abordados trabalhos e ferramentas disponíveis no estado da arte que propõem o uso de operadores adaptativos e técnicas de memória.

2.1 Introdução

As heurísticas são procedimentos computacionais baseadas em aproximações que englobam estratégias com o objetivo de encontrar uma boa solução, mesmo que não seja a solução ótima do problema, explorando o espaço de busca de forma inteligente e com baixo esforço computacional. Também é definida como um atalho mental que permite às pessoas resolverem problemas e tomar decisões de forma rápida e eficiente [32].

As heurísticas podem ser divididas em heurísticas construtivas ou de busca local. As heurísticas construtivas constroem uma solução, adicionando elementos a ela, passo a passo, por meio de uma sequência de decisões, enquanto as heurísticas de busca local utilizam a ideia de percorrer os "vizinhos" da solução atual para explorar o espaço de soluções.

Diferente dos métodos exatos, que procuram obter uma solução por meio de uma pesquisa por todas as soluções possíveis, as heurísticas são construídas a partir de um conhecimento prévio do problema e realizam a busca por uma quantidade menor de soluções. Este fato faz com que estas técnicas sejam cada vez mais usadas para resolver os problemas que não podem ser revolvidos por métodos exatos, devido ao alto esforço computacional desses métodos. Vários problemas de otimização combinatória apresentam essas características, como por exemplo os problemas de *scheduling* e roteamento de veículos.

Ainda em relação aos métodos exatos, as heurísticas possuem como desvantagem o fato de não assegurarem que a solução obtida seja uma solução factível ou a solução ótima do problema. Geralmente também não é possível medir com exatidão a qualidade desta solução e nem determinar exatamente qual a distância que ela se encontra da solução ótima. No entanto, vale destacar que as heurísticas têm como vantagem a flexibilidade de tratamento das características do problema e, dependendo da técnica escolhida, ao invés de obter uma única solução o algoritmo pode ser capaz de apresentar um conjunto de soluções, permitindo ampliar as possibilidades de decisão do otimizador. Ao abrir mão de encontrar a solução ótima, as heurísticas devem oferecer pelo menos o compromisso de serem computacionalmente eficientes [34].

Novas estratégias, denominadas meta-heurísticas, surgiram devido a necessidade de tornar os métodos heurísticos mais gerais. O termo meta-heurística foi introduzido por [33] para definir uma nova metodologia voltada a direcionar e modificar o processo de construção e os operadores heurísticos básicos. O principal objetivo das meta-heurísticas é aumentar a eficiência das heurísticas na exploração do espaço de busca, utilizando procedimentos para intensificar e diversificar essas regiões. O objetivo desta classe de algoritmos é escapar de ótimos locais ainda distantes dos ótimos globais e obter uma solução final de melhor qualidade.

Por serem facilmente adaptáveis para diferentes tipos de problemas, as meta-heurísticas vêm sendo cada vez mais exploradas como ferramentas de otimização. Alguns exemplos de meta-heurísticas que se destacam no estado da arte são:

Otimização mono-objetivo: *Greedy Randomized Adaptive Search Procedure* (GRASP) [26], *Simulated Annealing* (SA) [42], *Iterated Local Search* (ILS) [57], *Variable Neighborhood Search* (VNS) [62] e Busca Tabu [33, 38].

Otimização multiobjetivo: *Non-dominated Sorting Genetic Algorithm* II (NSGA-II) [21], *Non-dominated Sorting Genetic Algorithm* III (NSGA-III) [20], *Strength Pareto Evolutionary Algorithm* (SPEA) [101], *Strength Pareto Evolutionary Algorithm 2* (SPEA2) [98], *Preference-inspired Co-evolutionary Algorithm* (PICEA-g) [84], *Grid-based Evolutionary Algorithm* (GrEA) [87] e *Knee Point Driven Evolutionary Algorithm* (KnEA) [95].

No geral, o que diferencia esses algoritmos são a forma como uma solução é obtida; os mecanismos utilizados para escapar dos ótimos locais e a forma com que o espaço de busca é explorado.

As meta-heurísticas podem ser agrupadas em duas categorias: algoritmos de busca local e algoritmos populacionais. Nas meta-heurísticas de busca local, a exploração do espaço de soluções é realizada através de operações chamadas de movimentos. Um movimento é uma modificação que transforma uma solução s em outra s', que seja vizinha a ela. Esses métodos dependem necessariamente de alguma estratégia de variação

da solução para escapar de ótimos locais. GRASP, VNS e SA são exemplos de meta-heurísticas de busca local. As meta-heurísticas populacionais mantêm a cada iteração um conjunto de soluções que são de alguma forma combinadas para gerar as soluções novas da próxima iteração. Nesse caso, a convergência prematura tende a ser evitada pela diversidade da população utilizada na busca. Algoritmos Genéticos, Algoritmos de Colônia de Formigas e a Estratégia Evolutiva EE(1+1) são exemplos de meta-heurísticas populacionais.

Diferentes estratégias podem ser incorporadas às meta-heurísticas usuais para aprimorar o desempenho destas técnicas. Uma tática que vem sendo utilizada em alguns trabalhos é acrescentar às meta-heurísticas uma estrutura de memória. A memória pode exercer várias funções dentro do algoritmo, dentre elas armazenar todas as soluções encontradas e evitar reavaliações de uma mesma solução. Exemplos de aplicações desta estratégia são abordadas nos trabalhos [88, 92]. Outra tática que também vem ganhando destaque é o desenvolvimento de operadores adaptativos para ajudar o algoritmo a melhorar a exploração do espaço de busca e a qualidade do conjunto de soluções, e operadores de controle para fazer a auto-ajuste dos parâmetros do algoritmo [7, 8, 85, 92].

No caso específico da otimização multiobjetivo, em que vários objetivos precisam ser otimizados ao mesmo tempo e que solução final é um conjunto de soluções eficientes, as técnicas usadas para resolver estes problemas devem ter um tratamento diferente, principalmente no que tange a comparação das soluções. As técnicas desenvolvidas são usadas para manter a qualidade, diversidade e espalhamento do conjunto de soluções no espaço dos objetivos [6, 77] e procedimentos de auto-ajuste dos valores dos parâmetros nos operadores e recuperação das informações das regiões que foram exploradas [13, 73, 80, 91].

2.2 Estratégias de Memória em Meta-heurísticas

Em relação ao armazenamento de informações, nos seres humanos, a estrutura cognitiva é dividida em três tipos de memória: a memória sensorial, que retém por alguns segundos as informações recebidas pelos órgãos do sentido; a memória de curto prazo, que retém os dados por alguns minutos ou horas, e a memória de longo prazo, que grava de forma definitiva todos os dados, permitindo que mais tarde essas informações possam ser recuperadas ou evocadas [2]. Seguindo o mesmo ponto de vista, o conceito de memória é empregado nos algoritmos.

Nas meta-heurísticas, a memória, quando existente, tem por finalidade reter informações a respeito das soluções encontradas e avaliadas pelo algoritmo. Essas informações podem ser armazenadas por algumas iterações ou durante toda a execução do algoritmo, sendo usadas por seus operadores para manter a diversidade do conjunto de soluções e impedir que sejam gerados indivíduos duplicados. Evitar duplicações e reava-

liações de uma mesma solução é interessante principalmente nos problemas em que uma avaliação de função é considerada custosa. A memória do algoritmo também pode ser utilizada para criar orientações dentro do espaço de busca, ou seja, as informações armazenadas podem indicar são as regiões mais promissoras ou quais as regiões que ainda não foram exploradas. A memória também é capaz de informar quando o algoritmo estiver preso em um ótimo local, indicando por exemplo, quantas vezes uma mesma solução está sendo pesquisada dentro da memória ou quando são geradas soluções localizadas em uma mesma região da memória.

Com exceção da meta-heurística Busca Tabu, originada nos trabalhos independentes de [33] e [38] e que é amplamente utilizada para resolver diversos problemas de otimização, a maioria dos algoritmos estocásticos não armazenam informações à respeito das soluções ou das regiões exploradas. A Tabela 2.1 apresenta alguns trabalhos recentes que fazem uso desta técnica para resolver problemas clássicos da literatura.

Tabela 2.1: Aplicações da meta-heurística Busca Tabu em problemas de otimização combinatória.

Referência	**Problema de aplicação**
(Alvarez-Valdés, Parajón e Tamarit, 2002) [1]	Problema de corte guilhotinado bidimensional
(Lü e Hao, 2010) [47]	Problema de *timetabling*
(Venditti, Pacciarelli, e Meloni, 2010) [81]	Problema de *scheduling* aplicado a industria farmacêutica
(Misevicius,2015) [61]	Problema do caixeiro viajante
(Talbi e Belarbi, 2015) [78]	Controladores fuzzy para a estabilização de um simulador de helicóptero
(Lai, Demirag, e Leung, 2016) [48]	Problema de roteamento de veículos heterogêneo

A memória na Busca Tabu pode ser implementada como sendo de longo ou curto prazo, sendo que ambas as formas, permitem ao algoritmo controlar o número de soluções geradas e a redução da possibilidade de que uma mesma solução seja gerada várias vezes. Este procedimento permite que o algoritmo faça uma melhor exploração no espaço de soluções.

A memória a longo prazo salva todas as soluções e suas informações obtidas durante o processo de busca, porém, este tipo de memória apresenta como desvantagens o consumo da memória computacional e o elevado custo se aplicado a cada iteração. A memória de curto prazo armazena em uma lista tabu um conjunto de movimentos proibidos de serem realizados pelo algoritmo. Nesta lista, que funciona como uma fila de tamanho fixo, são armazenados os últimos movimentos realizados pelo algoritmo.

Apesar de ser o tipo de estrutura mais implementado na Busca Tabu, o problema relacionado a esta estrutura está na escolha do tamanho da lista que, se não for dimensionada adequadamente, pode fazer com que o algoritmo fique preso a um ótimo local ou

ainda possibilitar a revisitação de soluções de forma frequente. Pois, ainda que a lista tabu permita que o algoritmo encontre boas soluções por meio da utilização das informações a respeito dos espaços de busca já percorridos, não é possível salvar todas as informações ou soluções visitadas. Assim, na meta-heurística Busca Tabu é inevitável que uma reavaliação de função ou até mesmo uma revisitação de uma mesma região ocorram.

Remover as soluções idênticas geradas durante a execução de um algoritmo ajuda a manter a diversidade do conjunto de amostras de soluções, além disso, como em muitos problemas a avaliação de função é o processo mais caro do algoritmo, permitir reavaliações de soluções é um desperdício dos recursos computacionais [88].

Para evitar um gasto computacional nas avaliações de função, em [59] é proposto um operador de singularidade para remover as soluções idênticas do algoritmo. O operador só permite que uma nova solução seja inserida na população do algoritmo se a distância de *hamming* desta solução em relação aos demais indivíduos for superior a um limiar. Aplicado a um algoritmo genético o novo operador se mostrou eficiente ao realizar a exploração em regiões muito grandes do espaço de busca, por evitar a convergência prematura do algoritmo para um mínimo local. Porém, apesar da proposta melhorar a eficiência do algoritmo em relação a exploração da região factível há um aumento do custo computacional gerado pelo algoritmo para a remoção das soluções idênticas do algoritmo. Este aumento se deve ao número de comparações necessárias para verificar se o novo indivíduo está inserido ou não na população atual. Em termos de custo computacional, está quantidade de comparações não é viável.

Novas abordagens para evitar replicações de soluções nas meta-heurísticas são apresentadas tratadas no estado da arte. Parte das metodologias são baseadas na utilização de uma estrutura de dados que funcione como um operador de memória nos algoritmos, que é usado para armazenar as soluções encontradas e, em conjunto com outros operadores, evitar a reavaliação de uma mesma solução ou apenas para manter a diversidade do conjunto de soluções. As estruturas de dados mais usadas para armazenar o arquivo com as soluções das meta-heurísticas são: lista, fila, árvore e tabela *hash* (mais detalhes sobre estas estruturas podem ser obtidos em [14]).

A maior parte dos trabalhos que abordam este tema, por lidarem com uma grande quantidade de dados, fazem uso das estruturas de dados tabela *hash* e árvore, por ambas permitirem que a inserção e a busca sejam feitas de forma eficiente e com menor quantidade de comparações. Com exceção dos trabalhos que estudam a aplicação da meta-heurística busca tabu, onde a maior parte das implementações utilizam uma estrutura do tipo fila.

Um algoritmo genético binário sem revisitação, chamado de *Non-Revisiting Genetic Algorithm* (NrGA) é proposto em [88], utilizando como memória a estrutura de dados árvore *binary space partitioning* (BSP). Esta estrutura baseia-se nas informações da computação gráfica e geometria computacional para fazer subdivisões recursivas do

espaço [72]. A memória no NrGA é usada para armazenar o histórico de todas as regiões que já foram visitadas pelo algoritmo com o intuito de eliminar totalmente uma reavaliação.

A comunicação entre a árvore BSP e o AG acontece todas as vezes em que o AG gera uma nova solução e a memória é acessada para verificar se esta solução está ou não armazenada na árvore. A solução só é avaliada e inserida na memória se não possuir registro na BSP. Quando o registro existir, o algoritmo realiza uma busca em uma região que ainda não foi explorada. A nova solução é então avaliada e inserida tanto memória quanto na população do AG. Os resultados obtidos, após a aplicação deste procedimento, mostram que o desempenho do NrGA é superior ao AG padrão, visto que ao eliminar as revisitações evita-se também uma convergência prematura do algoritmo ao impedir que o mesmo fique preso a uma bacia de atração, além da manutenção da diversidade do conjunto de soluções.

Algumas melhorias são propostas ao NrGA desenvolvido por [88]. Em [92] é proposto um operador de mutação adaptativo para o NrGA que ajusta automaticamente o tamanho do passo do operador conforme a distribuição das soluções armazenadas na BSP. De acordo com o valor deste parâmetro é possível identificar se uma região foi pouco visitada ou não. Em [91], é proposta uma nova extensão deste algoritmo para tratar problemas em que o espaço de busca é composto por variáveis contínuas. Neste último trabalho o tamanho do passo da mutação é calculado considerando apenas um nó da arvore e não mais uma sub-região da BSP, como era feito em [92]. Em [12], os autores utilizam o NrGA contínuo e propõem que, a cada iteração do algoritmo, todas as soluções avaliadas sejam embaralhadas e reinseridas novamente na árvore, removendo a dependência da ordem em que as soluções são geradas. Nesta nova abordagem, o tamanho do passo no operador mutação é calculado considerando a nova distribuição dos indivíduos na árvore BSP.

Estes trabalhos mostram que a estrutura BSP pode ser aplicada a algoritmos estocásticos para eliminar todas as revisitações de uma mesma solução e evitar a convergência prematura desses algoritmos. Além do AG a estrutura BSP proposta em [88] também foi incorporada a outras meta-heurísticas, mostrando que esta estratégia realmente é capaz de reduz o custo computacional das avaliações de função em qualquer algoritmo que for utilizado. Em [90], os autores propõem o algoritmo *Non-Revisiting Simulated Annealing* (NrSA) que é uma integração do algoritmo padrão *Simulated Annealing* com a ideia de não revisitação de uma solução. Em [11], é proposto o algoritmo *Non-Revisiting Particle Swarm Optimization* (NrPSO). Ambos os trabalhos reforçam a ideia de que a redução de avaliações de função utilizando essa estrutura é relevante, por diminuir o desperdício de recursos computacionais e por ser facilmente aplicada a outros algoritmos, podendo ser estendido para a resolução de problemas reais.

Aplicações do algoritmo NrGA proposto em [88] na resolução de problemas de

otimização são encontradas em [89], que utiliza a metodologia para resolver o problema do caixeiro viajante e em [83] que aplica o algoritmo no problema de otimização de redes de distribuição para redução de perdas. Em [28], a estratégia de não-revisitação (Nr) é incorporada a um AG simples e ao algoritmo *Particle Swarm Optimization* para resolver um problema real aplicado à engenharia, que consiste na otimização de sistemas típicos de ar condicionado central. Em ambos os trabalhos, a abordagem de não revisitação proporcionou desempenho superior ao desempenho obtido pelas versões convencionais das meta-heurísticas usadas, por encontrar soluções melhores e de forma mais rápida. No último trabalho, que aplica o método a um problema real, a vantagem da metodologia é ainda mais evidente, uma vez que o tempo de processamento de uma avaliação de função é substancial, devido à natureza complexa da modelagem do problema.

A estrutura de memória BSP incorporada a um algoritmo evolucionário binário é apresentado em [13] e denominado pelos autores por *History Driven Evolutionary Algorithm* (HdEA). O algoritmo proposto utiliza a BSP para armazenar as operações realizadas, as posições e os valores de aptidão de todas as soluções avaliadas. A estrutura de arquivo proposta é usada para auxiliar na orientação da direção de busca, indicando quais as regiões que possuem soluções promissoras para o problema. Essa formulação também permite concluir quando o algoritmo está preso em um ótimo local, a medida que uma mesma solução é revisitada inúmeras vezes. Os autores do trabalho descrevem que o uso da árvore BSP como estrutura de memória supera a estrutura de dados tabela *hash* por facilitar a inserção e a busca de um indivíduo na árvore, principalmente a medida que se aumenta o tamanho do espaço de busca.

Embora os algoritmos HdEA e NrGA utilizem a mesma estrutura de memória (árvore BSP) para armazenar todas as soluções avaliadas e apliquem uma mutação adaptativa, eles apresentam as seguintes diferenças:

(i) o NrGA realiza a busca no espaço discreto enquanto o HdEA faz a busca no espaço contínuo;

(ii) no NrGA a árvore BSP é utilizada para determinar o tamanho do passo da mutação adaptativa, enquanto no HdEA a memória é usada para encontrar uma aproximação para o valor de função e uma solução;

(iii) no HdEA o número de nós na árvore é exatamente o mesmo que o número de soluções avaliadas, enquanto que no NrGA isso não acontece, devido à poda da árvore que acontece a cada iteração.

O operador de mutação adaptativo proposto em [13] a ajusta o valor do passo da mutação de forma aleatória, porém a direção é determinada de acordo com o histórico de soluções armazenadas na BSP.

Em [51] a estrutura de memória BSP é usada para estimar boas configurações (escolha do melhor operador de cruzamento e os valores dos seus parâmetros) ao longo

das iterações do algoritmo. A metodologia proposta é testada no algoritmo HdEA e em outros três algoritmos clássicos: AG, o algoritmo de Otimização por Enxame de Partículas (PSO) e o algoritmo de Evolução Diferencial (DE), e se mostrou eficiente por permitir o ajuste automático dos parâmetros (operador de cruzamento, taxa de cruzamento, mutação, fator e peso de aprendizagem) desses algoritmos.

Em [56], os autores propõem um algoritmo genético não-revisitado contínuo (cNrGA) que utiliza uma árvore BSP para armazenar todo o histórico de soluções e um operador de mutação adaptativo que é aplicado as regiões do espaço particionado. No espaço contínuo, a geração de soluções replicadas pelo AG é bem menos frequente que no espaço discreto, porém, no operador de cruzamento, a probabilidade de replicações de soluções é alta. Assim, mesmo que o algoritmo não tenha muitas soluções repetidas, armazenar as soluções em um histórico permite que melhores decisões sejam tomadas durante a aplicação dos operadores. Assim, todas as soluções geradas pelo cNrGA são armazenadas na memória e, esta estrutura é usada para evitar reavaliações das soluções e para a aplicação da mutação, na busca por novas soluções.

O uso de uma estrutura de dados do tipo árvore como estratégia de memória não é vantajoso quando o número de avaliações de função é muito maior que o espaço disponível na memória. Embora seja razoável armazenar todo o histórico, quando o problema envolve muitas avaliações da função, a pesquisa na árvore pode ser mais lenta que a aplicação de outros operadores. Uma tática que pode ser usada nesses casos é manter o uso constante da memória, fazendo podas na árvore toda vez que ela atinge um certo limite. Este procedimento é usado em [56], onde são propostas duas estratégias de podas: na primeira é podado da árvore o arquivo menos usado (nós que menos visitados) e na segunda é realizadas podas aleatórias (um nó qualquer da árvore). Comparada com a versão original do cNrGA, a metodologia desenvolvida por [56] não interferiu nos resultados obtidos e para até 90% de poda não apresentou diferença significativa. A metodologia se mostra vantajosa apenas pelos mecanismos de poda permitirem uma melhor exploração do histórico de memória e a aplicação em problemas com necessitam de um maior número de avaliações da função.

As investigações a respeito da replicação de soluções no operador de cruzamento iniciou-se com o trabalho [17]. Indícios mostram que com a remoção das soluções duplicadas pelo operador de cruzamento, a diversidade do conjunto de soluções do AG aumenta [69]. Seguindo esta ideia, no trabalho [69] a tabela *hash* é utilizada em um AG como estrutura de memória para reduzir a quantidade de comparações realizadas pelo algoritmo. Esta técnica contribui para a diversidade da população e melhoria da eficiência dos operadores de mutação e cruzamento.

A estrutura de dados tabela *hash*, que também é utilizada para armazenar o histórico dos algoritmos, permite que o armazenamento e a recuperação dos dados sejam feitos de forma mais eficiente e com um custo computacional menor do que as estruturas

de dados do tipo árvore. Neste intuito, é possível ver em pesquisas recentes, o uso da tabela *hash* nos algoritmos evolucionários para armazenar os indivíduos que já foram avaliados [8, 44, 67].

Em [44] é usada uma estrutura de dados do tipo *hash*-fila de tamanho fixo vinculada a um algoritmo genético. Neste método, quando todas as posições da tabela estiverem preenchidas, ao inserir um novo indivíduo, a solução mais antiga é descartada e a nova solução é inserida no lugar. Em [67] a tabela *hash* armazena apenas as melhores soluções avaliadas recentemente pelo AG. O que diferencia esta abordagem da anterior é o tratamento aplicado para determinar o tamanho ideal da tabela *hash*, onde é proposto, uma vez que a tabela esteja com todas as suas posições preenchidas, a criação de uma nova tabela maior e que todos os indivíduos da tabela antiga sejam realocados nesta nova tabela.

Uma investigação teórica a respeito do uso de um arquivo de memória em um Algoritmo Evolucionário EA(1+1) com busca local aleatória é apresentada em [75]. O algoritmo proposto, que utiliza como arquivo de memória a estrutura de dados tabela *hash* é denominado por EA-m. Na estrutura de memória utilizada, são armazenadas todas as soluções visitadas até que se encontre uma melhora do resultado em relação ao valor da função. Quando essa melhora é obtida, toda informação salva na memória é descartada.

A implementação binária do algoritmo genético multiobjetivo NSGA-II utilizando uma estrutura de memória do tipo tabela *hash*, para armazenar todas as soluções visitadas durante a execução do algoritmo multiobjetivo, é desenvolvido em [8]. Neste trabalho foi adotada a função *hash* de Pearson [66] para evitar a degradação do desempenho da estrutura de memória ao longo das iterações do algoritmo. A memória, utilizada no operador de avaliação, avalia e armazena todas as soluções encontradas pelo algoritmo, e no caso de replicações uma nova solução não-visitada é criada. Esta estratégia usada para evitar reavaliações de uma mesma solução e permite que a recuperação e armazenamento dos dados sejam feitos com um custo computacional baixo.

Em [10], é proposto um operador de memória completo para melhorar a eficiência dos AGs. O operador, que ele nomeia de REGISTRO, é uma sub-rotina da função de avaliação que funciona substituindo os indivíduos levando em consideração o histórico de soluções. Ao contrário dos trabalhos que calculam o valor de aptidão por aproximação, como em [13], nesta abordagem as soluções são realmente avaliadas. A memória não impede a revisitação das soluções, pelo contrário, ela aceita essas soluções e evita apenas que seja realizada a avaliação da função. Esse procedimento permite por si só aumentar o desempenho do AG, a medida que encontra as mesmas soluções que versões clássicas mas com apenas 5% das avaliações.

Para evitar que várias buscas locais sejam realizadas em uma mesma solução, no trabalho [15] foi proposto o uso de uma tabela *hash* no algoritmo VNS para resolver um problema discreto de sequenciamento de tarefas em sistemas flexíveis de manufatura.

Como o algoritmo VNS é determinístico, ao realizar uma busca local utilizando a mesma estrutura de vizinhança em uma mesma solução, o resultado final é sempre igual. Assim, permitindo que o algoritmo use uma estrutura de memória para armazenar todas as soluções geradas e o resultados obtidos da busca local nesta solução, as informações da busca local podem ser recuperadas pelo algoritmo toda vez que uma solução idêntica for gerada. O algoritmo proposto, denotado por CSO-VNS-*Hash*, se difere da metodologia CSO-VNS, também proposta no trabalho, apenas em relação ao número de avaliações de função. A estratégia de memória permite reduzir o número médio de avaliações em 75%. Em relação aos valores mínimos, médios e desvios das soluções encontradas, o desempenho dos algoritmos foi semelhante.

De uma forma geral, os trabalhos referenciados reforçam a importância da estratégia de memória incorporada às meta-heurísticas para impedir reavaliações de soluções duplicadas e como uma nova ferramenta para orientar o algoritmo na exploração do espaço de busca. A memória é usada para armazenar as soluções encontradas e todas as informações relevantes a respeito dessas soluções (valores de aptidão, localização no espaço de busca, resultado da busca local e etc.), que também podem ser utilizadas na proposição de operadores adaptativos que fazem o auto-ajuste dos parâmetros. Assim, os mecanismos de memória podem ser considerados fortes aliados para o aumento da diversidade do conjunto de soluções geradas e como uma boa estratégia para evitar a convergência prematura dos algoritmos de otimização.

Diferente da maioria dos trabalhos que empregam a estratégia de memória em um algoritmo genético binário ou contínuo, neste trabalho a estratégia de memória é aplicada a uma meta-heurística de busca local para problemas combinatórios e a estrutura de dados usada é uma tabela *hash*.

Visto que algumas das metodologias citadas enfrentam dificuldades quando é elevado a quantidade de dados que precisam ser armazenados na memória (o custo com a pesquisa na árvore pode ficar caro, necessidade de podas em todas as iterações, tratamento de colisões na tabela *hash* e determinação do tamanho desta tabela), uma das contribuições deste trabalho é a proposta de uma nova função *hash*, para problemas permutacionais e inteiros, que permite que as soluções armazenadas tenham um bom espalhamento e um baixo índice de colisão na tabela, reforçando a vantagem desta estrutura de dados em relação a estrutura do tipo árvore no que diz respeito a pesquisa e armazenamento das soluções replicadas, mesmo quando o problema tratado envolve muitas avaliações de solução. A estrutura que tem como finalidade principal armazenar as soluções geradas para evitar reavaliações, também é utilizada na proposição de um novo operador de vizinhança adaptativo para a busca por novas soluções em regiões ainda não exploradas do espaço de busca.

2.3 Operadores Adaptativos em Meta-heurísticas

Muitos problemas reais são inerentemente problemas de otimização multiobjetivo, uma vez que dependem da otimização de dois ou mais critérios. Alguns desses problemas podem ser resolvidos utilizando métodos de programação matemática, porém, como a maioria deles apresenta características que impedem o uso desses métodos, como a quantidade de ótimos locais, o uso das meta-heurísticas multiobjetivo tem se tornado cada vez mais necessário.

Os Algoritmos Evolucionários Multiobjetivo (MOEAs) vêm ganhando destaque nas últimas décadas devido a sua flexibilidade, simplicidade de implementação e eficácia na busca pelo melhor conjunto de soluções. Os MOEAs são caracterizados por aplicarem sucessivas operações de mutação, cruzamento e seleção na população, com o intuito de obter o melhor conjunto de soluções para o problema tratado. Dentre os algoritmos disponíveis na literatura, podem ser citados: NSGA II [21], NSGA-III [20], SPEA 2 [99], PICEA-g [84], GrEA [87], KnEA [95] e o Algoritmo Evolucionário Multiobjetivo Baseado em Decomposição (MOEA/D) [94].

No geral, os MOEAs apresentam uma convergência mais lenta quando comparados às demais meta-heurísticas, por lidarem com objetivos conflitantes e trabalharem com um conjunto de soluções candidatas em paralelo. Neste intuito, muitos trabalhos concentram esforços em desenvolver mecanismos para diversificar o conjunto de soluções e melhorar a eficiência desses algoritmos minimizando a distância entre a fronteira de soluções não-dominadas e a fronteira de Pareto-Ótimo. Algumas das técnicas envolvem a criação de novos operadores e/ou adaptações aos operadores usuais de mutação, cruzamento e seleção.

O termo adaptativo é definido na biologia como a capacidade dos seres vivos de se ajustarem ao ambiente. Na computação evolutiva, este termo parte do mesmo princípio e a aplicação deste conceito está relacionada a proposição de técnicas de ajustes ou controle de parâmetros, como por exemplo, o ajuste do tamanho do passo do operador de mutação.

O desempenho dos algoritmo pode ser fortemente afetado pela variedade de parâmetros que precisam ser ajustados. No trabalho de [23] é apresentada uma revisão completa sobre o controle de parâmetros no AG e demonstrado que os valores dos parâmetros ideais variam durante a evolução do algoritmo. Porém, boa parte dos trabalhos se dedica apenas a fazer adaptações no operador de mutação. Em [54, 79], para controlar a taxa de probabilidade na mutação, é proposto um operador de mutação adaptativo que usa o valor de aptidão do pai. Assim, os indivíduos são submetidos a baixas probabilidades de mutação se possuírem um valor alto de aptidão. Caso contrário, se o valor de aptidão for baixo, a probabilidade de mutação é aumentada.

As estruturas de memória acopladas aos AG, para armazenar as soluções obtidas

ao longo das gerações, também é utilizada por alguns autores para auxiliar o algoritmo a fazer o auto-ajuste do tamanho do passo do operador de mutação. Em [92, 91], por exemplo, é proposto um operador de mutação adaptativo para o NrGA, desenvolvido [88], que ajusta o tamanho do passo do operador conforme a distribuição das soluções que estão armazenadas na BSP. A mutação adaptativa é usada para encontrar um vizinho que esteja em uma sub-região que ainda não foi explorada e, ao mesmo tempo, próximo da solução atual. Nesta sub-região, a direção da mutação é aplicada de forma aleatória e a adaptação do operador é usada apenas para ajustar o tamanho do passo, que é alterado conforme o tamanho desta sub-região. Assim, quanto mais um ramo da árvore for visitado, menor é o seu sub-espaço e consequentemente menor deve ser o tamanho do passo. O valor deste parâmetro também indica se uma região foi pouco visitada e, caso isso ocorra, o valor da taxa de mutação é aumentado automaticamente. A técnica *one-gene-flip* é usada por [12] para ajustar o tamanho do passo no operador de mutação, no algoritmo NrGA, a partir das informações contidas na BSP. Nesta abordagem o valor deste parâmetro é obtido considerando as novas distribuições das informações armazenadas na árvore a cada iteração. A mutação *one-gene-flip* transforma apenas um gene, escolhido aleatoriamente, no indivíduo dentro de uma partição. Em relação a mutação adaptativa proposta em [92], que aplica a mutação de forma aleatória, esta última abordagem é menor prejudicial para o algoritmo do que os efeitos gerados pelo operador de cruzamento. Além disso, comparando com as versões antigas do NrGA, o operador de mutação adaptativo acelera a convergência do algoritmo.

Um operador adaptativo de mutação guiado e sem parâmetros, o *Guided Anisotropic Search* (GAS), é proposto nos trabalhos [13, 51] para o HdEA. Neste operador, o tamanho de passo de mutação é atribuído aleatoriamente e a direção de busca é baseada no histórico das soluções armazenas na BSP. A diferença entre os operadores de mutação do HdEA e NrGA está na direção do passo: enquanto no HdEA a direção é dada pela aproximação dos valores de aptidão, no NrGA a direção é escolhida de forma aleatória. Em [51], ainda é desenvolvido um sistema adicional de controle adaptativo para os valores dos parâmetros e para a escolha de quais os operadores que devem ser usados no HdEA. Os valores dos parâmetros e os operadores de cruzamento são alterados conforme as informações do conjunto de soluções armazenadas na memória.

Um novo operador de mutação para os MOEAs é apresentado em [58]. O operador *Pareto Rank Based Mutation Operator* (PRBMO) utiliza a ideia de amplitude máxima do conjunto de Pareto-Ótimo para determinar o tamanho máximo do passo da mutação. A novidade desta metodologia é a utilização de um *ranking* do Pareto para controlar a mutação, visto que quanto melhor o *ranking* (melhor a solução), menor deve ser o tamanho do passo. A vantagem do uso deste operador de mutação é que se evita a exploração de regiões que estejam longe das soluções ótimas do algoritmo. Uma vez detectada as regiões que possuem soluções nos piores *rankings*, o tamanho do passo é

alterado e aumentado para que as novas soluções obtidas estejam em regiões melhores, ou, quando as regiões possuem soluções nos melhores *rankings*, o tamanho do passo é diminuído para que haja uma melhor exploração nesta região. O estudo realizado nesta pesquisa mostra que o NSGA-II utilizando o operador PRBMO obtém desempenho melhor que sua versão original em todos os experimentos realizados.

O conceito de taxa de diversidade da população é utilizado para adaptação da probabilidade de mutação de um AG em [96]. A mutação é usada para gerar a diversidade de uma população convergente. Assim, o trabalho desenvolve um AG adaptativo e utiliza o método para resolver um problema de roteamento de veículos com janela de tempo. Nota-se, pelos resultados obtidos, que uma adaptação automática das probabilidades de mutação e cruzamento mantém a diversidade do conjunto de soluções e evita uma convergência prematura do algoritmo. Na mesma linha, em [93] também é utilizado o conceito de parâmetros adaptativos para manter a diversidade do conjunto de soluções. Neste trabalho, os parâmetros de cruzamento e mutação são alterados de acordo com os parâmetros extraídos do algoritmo *K-means clustering*, que faz uso da lógica *fuzzy* para ajustar de forma adaptativa esses parâmetros. Os autores mostram que a técnica é eficiente, uma vez que aumenta a velocidade de convergência e impede que o algoritmo fique preso em um ótimo local.

Em [74], as taxas de mutação são calculadas através de uma média de dois fatores: o valor de aptidão do pai e uma medida de diversidade da população. Comparado com o AG padrão, na resolução de funções multimodais com diferentes graus de complexidade, o AG com adaptação das probabilidades dos parâmetros atingiu o seu objetivo de manter a diversidade do conjunto de soluções e aumentar a velocidade de convergência do algoritmo. Em [54], também é proposto um algoritmo genético baseado nos ajustes de probabilidade dos operadores considerando o valor de aptidão. Porém, diferente do trabalho anterior, este considera apenas o valor de aptidão individual, e as probabilidades são alteradas com o intuito de evitar repetições no cálculo de aptidão de um mesmo indivíduo.

Contrário aos trabalhos anteriores que propõem adaptações nos operadores de mutação de cruzamento, para determinar as taxas de probabilidade ou tamanho do passo da mutação, os trabalhos recentes de [8, 52] concentram esforços na proposição de novos operadores para os MOEAs.

Em [8] é proposto um operador para construir uma vizinhança adaptativa para o AG binário, usando a ideia de codificação de comprimento de variável. No trabalho, cada variável *i* da solução é representada por uma sequência binária e é fornecido um parâmetro adicional para representar a profundidade desta variável. O parâmetro adicional indica que somente os *bits* que estão nesta profundidade são mais significativos e podem ser mutáveis pelos operadores genéticos. Esta metodologia, aplicada ao AG quando é encontrada uma solução que já foi avaliada, permite uma exploração mais intensiva nas regiões de busca para obter novas soluções. Os resultados mostram que o algoritmo genético

multiobjetivo NSGA-II em conjunto com o operador de vizinhança adaptativo mostra-se mais eficiente, em relação à qualidade da soluções, que outras versões do NSGA-II disponíveis na literatura. No trabalho [52], é proposto um operador para selecionar quais operadores devem ser usados em uma determinada iteração do AG. O método possui duas tarefas: a primeira é decidir o quanto um operador deve ser aplicado no processo de busca, a segunda é selecionar qual operador será usado. A métrica usada para avaliar o operador é a taxa de melhoria do valor de função da solução, quando aplicado o operador selecionado. Essa taxa mede a qualidade do operador e indica se ele deve ou não ser aplicado na iteração atual. Os testes realizados no trabalho mostram que o operador se mostrou robusto e permitiu melhorar significativamente o desempenho dos algoritmos. Em ambos os trabalhos, os novos operadores propostos se mostraram eficientes quando comparados com as versões clássicas dos algoritmos utilizados.

2.3.1 Operadores Adaptativos Baseados em Controle

Visto que estudos a respeito da adaptação dos parâmetros pelo próprio algoritmo vêm recebendo destaque na literatura, trabalhos recentes utilizam a ideia de inspiração por controle realimentado como uma nova ferramenta para o ajuste automático dos valores dos parâmetros nos operadores adaptativos. Exemplos de estudos inspirados neste conceito, para permitir procedimentos de adaptação dos parâmetros dos operadores, são obtidos nos trabalhos [41, 53, 60, 76, 80].

Em [73, 76] são propostas novas técnicas para melhorar a eficiência dos MOEAs, aumentando a diversidade e qualidade das soluções do conjunto de Pareto, baseadas em operações por “esferas”. A ideia principal destes trabalhos é que uma esfera representa aproximadamente um domínio, no qual as informações obtidas por um ponto no centro dessa esfera representa todo o conjunto de soluções que estejam contidas no seu interior. Um métrica baseada em contagem de esferas, *Integrated Sphere Counting* (ISC), é proposta em [73] para avaliar a qualidade dos conjuntos de estimativas do conjunto de Pareto. Além dos operadores baseados em esferas, o princípio de controle realimentado é usado em [76] para reduzir o tamanho do arquivo do conjunto de soluções a fim de alcançar um conjunto de Pareto com soluções bem distribuídas no espaço de busca. O efeito dinâmico do esquema de controle aplicado ao MOEA permitiu ao algoritmo obter um alto desempenho em relação à qualidade do conjunto de Pareto.

Uma adaptação dinâmica das probabilidades dos operadores de cruzamento e mutação, com o objetivo de manter a diversidade genética da população e evitar uma convergência prematura para mínimos locais é proposta em [80]. Os autores destacam a importância de fazer alterações nessas probabilidades para manter um conjunto de soluções diversificado, com a introdução de novas soluções com características diferentes e evitando a perda de soluções já existentes. Essa metodologia, junto com as demais

apresentadas no trabalho, permitiu melhorar o desempenho dos algoritmos genéticos testados.

Um AG, com parâmetros adaptativos dos operadores de cruzamento, mutação e seleção é proposto em [60]. A metodologia tem por objetivo manter a diversidade do conjunto de soluções por meio de um operador que adapta os parâmetros dos operadores de mutação e cruzamento ao longo da execução do algoritmo. Além da adaptação dos parâmetros dos operadores, como também é feito em [80], neste trabalho é utilizado um controlador para coordenar a pressão do operador de seleção do algoritmo. O mecanismo de seleção controlado permite manter a diversidade e os melhores valores de *fitness* no conjunto de soluções. Com os mesmos objetivos dos trabalhos anteriores, em [41], os operadores propostos buscam manter a diversidade do conjunto de soluções fazendo buscas nas regiões mais promissões do espaço dos objetivos. Um ponto importante deste trabalho é que, assim como em [60], o operador de seleção adaptativo utiliza o controle realimentado para determinar a pressão de seleção do algoritmo.

Um algoritmo de otimização multiobjetivo que adota mutação adaptativa e um operador de seleção modificado para atualização do arquivo de soluções é proposto em [53]. Um arquivo externo é usado para armazenar as soluções não-dominadas, e o operador de mutação é usado conforme o valor de aptidão de cada indivíduo. A mutação é aplicada de acordo com a localização da solução no conjunto de soluções não-dominadas, ou seja, os indivíduos que estão na fronteira do conjunto recebem valores altos para o passo da mutação. Esse processo permite melhorar a convergência e a diversidade do conjunto.

O estado da arte mostra que utilizar técnicas para controle e auto-ajuste nos parâmetros dos operadores são consideradas boas estratégias na otimização multiobjetivo, principalmente quando o intuito é garantir a diversidade no conjunto de amostras de Pareto e aumento da velocidade de convergência dos algoritmos. Além disso, esses operadores adaptativos permitem classificar o conjunto de soluções de acordo com as regiões mais promissoras do espaço de busca, garantindo que o algoritmo retorne um conjunto de soluções de boa qualidade. Porém, a maioria dos trabalhos utilizam esses conceitos para a proposição de operadores adaptativos de mutação e cruzamento apenas em algoritmos evolucionários multiobjetivo.

Neste sentido, os novos operadores adaptativos desenvolvidos nesta tese para um MOEAs são baseados em controle realimentado e têm por finalidade melhorar a descrição do conjunto de estimativas de Pareto. O operador baseado em controle realimentado proposto é usado para manter o equilíbrio da quantidade de soluções do arquivo externo do algoritmo em todas as iterações. Além do controle do tamanho do arquivo, um operador de cruzamento é utilizado para melhorar o preenchimento dessas soluções no arquivo. O diferencial deste trabalho é que os operadores propostos utilizam os conceitos de adaptação e controle para melhorar a distribuição das soluções por todo o espaço de

soluções e além disso, o controlador foi implementado de forma que todos os seus parâmetros sejam ajustados dentro do próprio algoritmo, conforme a eficiência do mesmo. Assim, tendo visto que a metodologia permite obter um conjunto de Pareto final mais representativo e sem acrescentar muito esforço computacional ao algoritmo, a técnica pode ser aplicada na resolução de problemas reais e de forte relevância industrial.

Ainda no âmbito dos operadores adaptativos, diferente da maioria dos trabalhos que aplicam este conceito em algoritmos evolucionários, este trabalho tem por objetivo ampliar esta ideia para diferentes classes de meta-heurísticas. Neste propósito, o operador adaptativo para meta-heurísticas de busca local é desenvolvido para que, em conjunto com uma estratégia de memória, seja usado para aumentar a exploração do espaço de objetivos.

Capítulo 3

Operadores Adaptativos em Algoritmos Multiobjetivo

Este capítulo apresenta uma nova classe de operadores adaptativos desenvolvidos para os algoritmos evolucionários multiobjetivo, que têm por finalidade melhorar a descrição do conjunto de estimativas do conjunto de Pareto. As técnicas aqui propostas são inspiradas na ideia de controle realimentado. Assim, com a finalidade de apresentar os operadores propostos, este capítulo descreve, de forma sucinta, as principais definições que envolvem a otimização multiobjetivo e a metodologia proposta.

Ainda neste capítulo, a metodologia apresentada é aplicada a um problema de otimização dos parâmetros operacionais e geométricos de um sistema de extrusão de polímero. Este problema prático e de relevância industrial, trata de uma aplicação de otimização multiobjetivo em um modelo de simulação, cujo objetivo é adquirir conhecimento sobre possíveis modos de operações eficientes a serem implementadas em uma máquina real. O problema consiste em obter diferentes soluções ótimas para auxiliar os engenheiros de polímeros ao selecionar as melhores configurações de um sistema de extrusão de polímeros. O problema tratado é referenciado em [16, 31].

3.1 Introdução

Uma das principais preocupações relacionadas ao desenvolvimento dos MOEAs tem sido a de encontrar um conjunto diversificado de soluções. Neste intuito, esta tese introduz novos operadores adaptativos com o objetivo de aumentar a capacidade do MOEA em representar o conjunto de estimativas do Pareto. Os conceitos de arquivo externo e operadores adaptativos foram adotados.

Os operadores propostos são fundamentados nas ideias de controle realimentado, inspirado em [65], e controle por esferas, usados para estabelecer e manter o equilíbrio dinâmico das soluções do conjunto de Pareto ao longo das iterações do algoritmo utilizando o conceito de distância entre duas soluções.

A Figura 3.1 ilustra como a ideia do controle realimentado é utilizada neste trabalho. Na ilustração a grandeza de medição que alimenta o laço do controlador Proporcional-Integral (PI) é o erro que é calculado pela equação $e = q - a$. O controlador PI controla o valor do parâmetro ρ, que representa o raio de domínio (raio da esfera) de uma solução do conjunto. A variável a indica a quantidade de elementos do arquivo externo do algoritmo. As variáveis ρ e a representam, respectivamente, a variável de entrada e a variável controlada do controlador PI. A quantidade de possíveis elementos do arquivo externo é pré-estabelecida e está indicada na figura pela variável q.

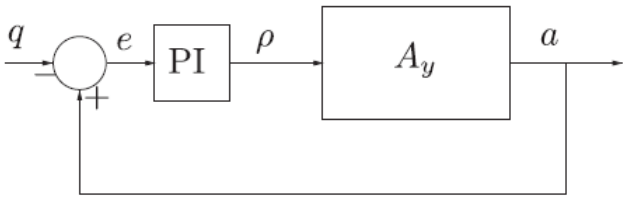


Figura 3.1: Representação do laço do controle por realimentação PI que controla o número de pontos armazenados no conjunto A_y.

O controlador PI atinge o equilíbrio quando ($e = 0$), este valor significa que o resultado desejado foi alcançado, ou seja, ($a = q$). Assim, variáveis relacionadas ao erro q e a, são definidas de forma que o laço do controlador atinja o equilíbrio apenas quando uma boa descrição do conjunto de Pareto for obtida. Esta técnica de controle é aplicada para induzir um comportamento global do sistema que apresenta baixa sensibilidade a variações das condições iniciais e dos parâmetros do algoritmo. Em [65], por exemplo, a técnica de controle é aplicada para obtenção de resultados estáveis, repetindo o alcance de boas soluções no conjunto.

Os operadores de controle por esfera introduzidos neste capítulo são: o operador *archive-set reduction*, que controla a quantidade de soluções não-dominadas que são armazenadas no arquivo externo e o operador de cruzamento *surface-filling*, que é aplicado ao conjunto de soluções não-dominadas para preencher os espaços vazios deste conjunto. O nome, operadores de controle por esferas ou *sphere-control*, foi inspirado no fato dos operadores melhorarem a representatividade das soluções na superfície de Pareto usando a ideia de que cada solução é o centro de uma esfera de raio ρ.

É importante ressaltar que os operadores *archive-set reduction* e *surface-filling* devem ser usados em conjunto, uma vez que os seus efeitos são complementares e, para atingir o comportamento desejado do conjunto de Pareto (conjunto com boa distribuição e espalhamento das soluções), é necessário que haja uma interação dinâmica entre esses operadores. O equilíbrio entre as ações dos operadores é alcançado pelo controle realimentado. Assim, o conceito de controle realimentado torna-se uma ferramenta importante nos estudos relacionados a adaptação de parâmetros na computação evolutiva.

Na sequência são apresentados alguns conceitos básicos a respeito de otimização multiobjetivo.

3.2 Algoritmos Evolucionários Multiobjetivo

Muitos problemas reais consistem em resolver simultaneamente problemas com dois ou mais objetivos que precisam ser otimizados. Esses objetivos são, na maioria das vezes, conflitantes, o que significa que a melhoria de um objetivo provoca a deterioração do outro. Problemas com esta característica são chamados de Problemas de Otimização Multiobjetivo (MOP).

A formulação geral de um problema de otimização multiobjetivo de minimização é dada por [18]:

$$
\begin{aligned}
X^* = \quad & \arg\min_{x} \quad \mathbf{f}(\mathbf{x}) && \text{(3-1)}\\
& \text{sujeito a :} \quad h_i(\mathbf{x}) \leq 0 \quad \forall i = 1,...,r && \text{(3-2)}\\
& \qquad\qquad\quad g_j(\mathbf{x}) = 0 \quad \forall j = 1,...,p && \text{(3-3)}\\
& \qquad\qquad\quad \mathbf{x}_i^{(L)} \leq \mathbf{x}_i \leq \mathbf{x}_i^{(U)} \quad \forall i = 1,\cdots,n && \text{(3-4)}\\
& && \text{(3-5)}
\end{aligned}
$$

em que $f(\cdot) : \mathbb{R}^n \mapsto \mathbb{R}^k$ é o vetor de funções objetivo do problema , $\mathbf{x} \in \mathbb{R}^n$ é um vetor de n variáveis de decisão $\mathbf{x} = (x_1, x_2, \cdots, x_n)^T$ que formam o espaço de variáveis de otimização, ou o espaço de decisão. X^* é um conjunto de soluções eficientes, ou Pareto-Ótimas.

As equações (3-2) e (3-3) representam, respectivamente, as restrições de igualdade e desigualdade. Uma solução X do problema é dita factível se satisfizer ao mesmo tempo todas as restrições. Os valores $\mathbf{x}_i^{(L)}$ e $\mathbf{x}_i^{(U)}$ representam, respectivamente, o valor mínimo e máximo para cada variável $\mathbf{x}_i$ (restrições de caixa).

Resolver o MOP é encontrar um conjunto de soluções X^* que satisfaça as restrições e que minimize (maximize) ao mesmo tempo todas as funções objetivo ($\mathbf{f}(\mathbf{x})$). A Figura 3.2 (a) ilustra o conjunto F_x, ou seja, a região factível do problema.

As soluções nos problemas de otimização multiobjetivo são classificadas em soluções dominadas e não dominadas. Para melhor entendimento, considere $\mathbf{x}_1$ e $\mathbf{x}_2$ dois pontos quaisquer do espaço de variáveis. A relação $\mathbf{x}_1 \prec \mathbf{x}_2$, que indica que $\mathbf{x}_1$ domina $\mathbf{x}_2$, é válida se:

$$f(\mathbf{x}_1) \leq f(\mathbf{x}_2) \quad \text{e} \quad f(\mathbf{x}_1) \neq f(\mathbf{x}_2)$$

Essa relação indica que $\mathbf{x}_1$ é melhor que $\mathbf{x}_2$ em todos os objetivos. De forma equivalente, diz-se que $f(\mathbf{x}_1)$ domina $f(\mathbf{x}_2)$. Não existindo essa relação, dizemos que a solução é não

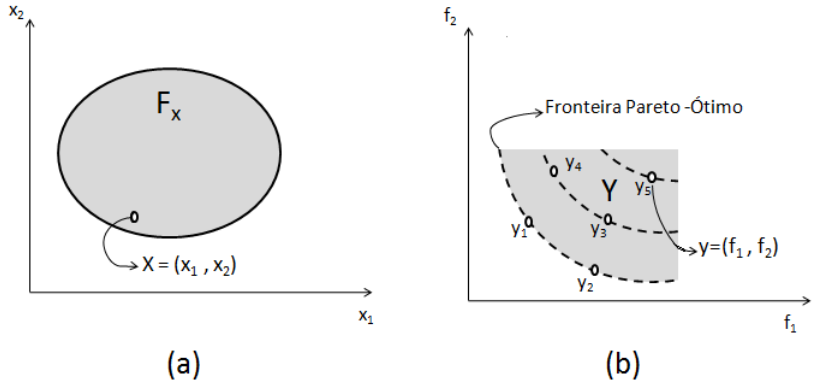


Figura 3.2: (a) Conjunto F_x de soluções factíveis do Problema Multiobjetivo e (b) espaço de objetivos e a fronteira Pareto-Ótimo.

dominada. A relação de dominância está ilustrada na Figura 3.2 (b), para os pontos y_1 e y_3 pertencentes ao espaço de objetivos Y. O ponto y_1 apresenta um valor menor que y_3 para as funções objetivos f_1 e f_2. Assim, podemos dizer que y_1 domina y_3, ou seja, a relação $y_1 \prec y_3$ é válida.

O conjunto de Pareto $\mathcal{P}$ é definido por $\mathcal{P} \triangleq \{x^* \in \Omega \mid \nexists\, x \in \Omega \quad \text{tal que} \quad x \prec x^*\}$, onde $\Omega \subseteq \mathbb{R}^n$ é o conjunto de soluções factíveis. A solução $x^* \in F_x$ de um MOP é chamada Pareto-Ótima se não existe qualquer outra solução x do problema que a domine:

$$f(x) \leq f(x^*) \;\text{ e }\; f(x) \neq f(x^*)$$

Na Figura 3.2 (b), os pontos y_1 e y_2 são exemplos de soluções eficientes. Note que ambos os pontos não são dominados por nenhum outro ponto do conjunto Y, e pertencem ao que chamamos de Fronteira de Pareto.

Um dos problemas centrais da otimização multiobjetivo é encontrar esse conjunto de soluções de Pareto $\mathcal{P}$. Esse conjunto é caracterizado pela relação conflitante existente entre as funções objetivo, onde não é possível encontrar uma solução que seja melhor em um objetivo, sem que ela se torne pior em outro.

3.2.1 MOEAs com Operadores de Controle por Esfera

Neste trabalho é proposto um MOEA utilizando os operadores de controle por esferas. Estes operadores podem ser aplicados a qualquer algoritmo evolucionário multiobjetivo. Neste trabalho, a escolha pelo algoritmo NSGA-II se deu pela difusão desta técnica na literatura. Assim, o Algoritmo Genético Multiobjetivo com Controle por Esferas, ou *Sphere-Control Multiobjective Genetic Algorithm* (SCMGA), proposto neste trabalho consiste no algoritmo NSGA-II utilizando os operadores propostos. Os algoritmos NSGA-II e a versão SCMGA são descritos a seguir.

NSGA-II

O NSGA-II, desenvolvido por [21], é um algoritmo multiobjetivo que utiliza o conceito de dominância para classificar a população em fronteiras conforme o grau de dominância. Neste algoritmo, as soluções que estão localizadas na primeira fronteira do conjunto de Pareto são consideradas as melhores soluções de cada geração, enquanto as que estão localizadas na última fronteira são consideradas as piores soluções (soluções mais dominadas). No NSGA-II a população é classificada por categorias de qualidades (conforme sua localização nas fronteiras), o que permite ao algoritmo priorizar aquelas que são melhores classificadas. O elitismo, neste algoritmo, é dado seguindo o critério de dominância.

O NSGA-II se diferencia de um algoritmo genético simples pelo processo de seleção, que é separado em dois mecanismos: classificação rápida por não-dominância (*Fast Non-Dominated Sorting*) e a distância de agrupamento (*Crowding Distance*).

No processo de *fast non-dominated sorting*, todos os indivíduos (população de pais e população de filhos) são classificados em níveis (fronteira) de acordo com sua dominância. Assim, os indivíduos que possuírem dominância zero (não dominados) são inseridos na primeira fronteira e representam a estimativa do conjunto Pareto-Ótimo [21]. Os demais indivíduos são inseridos nas próximas fronteiras, sendo os piores indivíduos inseridos na última fronteira.

O processo para obter as soluções de estimativas Pareto-Ótimas tendem a convergir para uma mesma região do espaço de busca. Uma característica esperada é que essas soluções estejam espalhadas neste espaço. O operador *crowding distance* é, então, aplicado para esta finalidade. O *crowding distance* ordena as soluções de uma mesma fronteira de acordo com a sua distância em relação aos demais vizinhos. Este processo de ordenação acontece de acordo com uma estimativa de densidade, como mostra a Figura 3.3. A estimativa de densidade de soluções, em torno de uma solução qualquer da fronteira, é dada pelo cálculo da distância Euclidiana de k pontos de cada lado deste ponto, onde k representa a quantidade de objetivos. O valor i é uma estimativa do perímetro do cuboide formado usando os vizinhos mais próximos como vértices.

Finalizado o processo de classificação, o operador de seleção é então aplicado para selecionar os indivíduos para compor a próxima geração a partir dos indivíduos da primeira fronteira. O processo de seleção do algoritmo permite que haja um melhor espalhamento dos resultados, evitando aglomerações de soluções sobre um mesmo ponto. Em outras palavras, as soluções que estão localizadas em uma região de menor aglomeração têm maior probabilidade de serem selecionadas. A Figura 3.4 apresenta de forma simplificada todo o procedimento do NSGA-II.

O NSGA-II utilizado neste trabalho está disponível no PISA [97], que é uma plataforma para a utilização de algoritmos de busca. O algoritmo é semelhante a versão original proposta por [19].

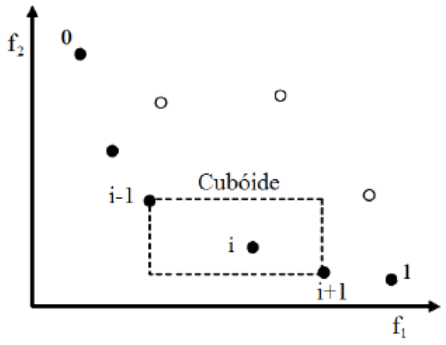


Figura 3.3: Cálculo do operador *crowding cistance*. Os pontos representados pelos círculos preenchidos são as soluções da primeira fronteira (conjunto de estimativas de Pareto-Ótimo). Fonte [21].

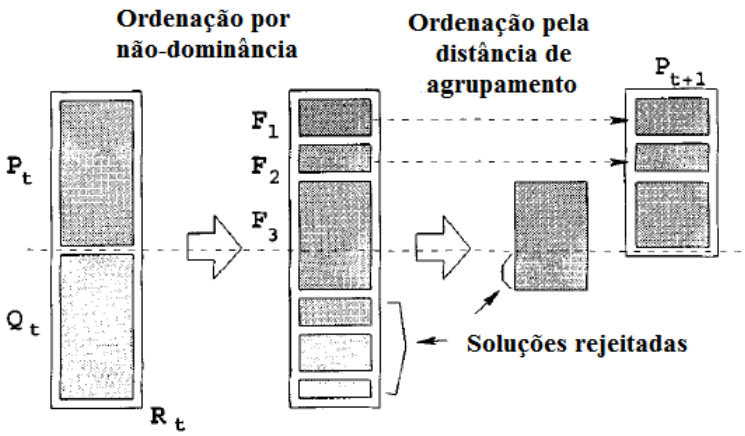


Figura 3.4: Ilustração do procedimento NSGA-II. Fonte [21].

SCMGA

O Algoritmo Multiobjetivo com Controle por Esferas (SCMGA), proposto nesta seção, é semelhante ao NSGA-II original. A diferença entre os dois algoritmos está ilustrado na Figura 3.5. O algoritmo SCMGA realiza todos os procedimentos do NSGA-II e no final são aplicados os operadores de controle por esferas no conjunto de soluções não-dominadas e realizada a auto-adaptação dos parâmetros dos operadores, este operadores estão em destaque na figura.

De forma mais específica, as diferenças entre essas duas abordagens são:

1. **A forma de controle do tamanho do conjunto de soluções não-dominadas:** no NSGA-II, a seleção elitista permite que a população atual sirva como um arquivo, que é construído usando o operador *crowding distance*. No entanto, apesar deste controle do tamanho do arquivo, baseado no *crowding distance*, ser eficiente para duas funções objetivo, para um número maior de objetivos o operador não é tão eficiente [45]. No SCMGA, o conceito de um arquivo externo é utilizado. O controle da dimensão deste arquivo é feito por meio de um controlador PI, que controla o valor da variável ρ. Esta variável corresponde ao raio das esferas que possui uma

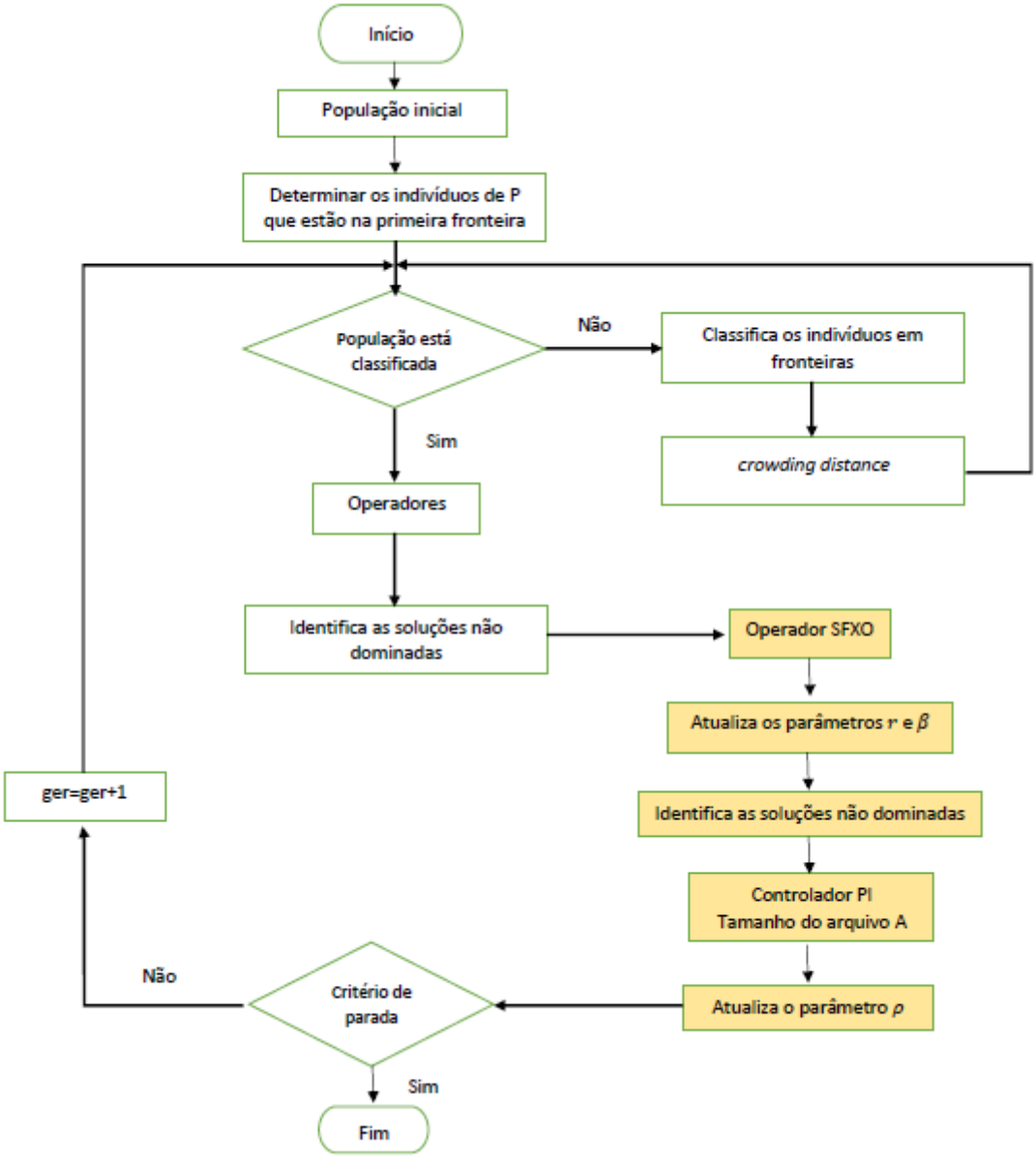


Figura 3.5: Procedimento do SCMGA: o algoritmo realiza todos os procedimentos do NSGA-II e no final são aplicados os operadores de controle por esferas no conjunto de soluções não-dominadas a cada iteração do algoritmo.

solução como centro, a fim de manter um arquivo de soluções representativo e do tamanho desejado. Como este método utiliza a ideia de esferas para controlar o tamanho do arquivo, ele pode ser generalizado para tratar problemas em espaços de qualquer dimensão, uma vez que as esferas podem ser totalmente parametrizadas apenas pelo tamanho do raio. O arquivo interage com a população final de cada iteração do SCMGA .

2. **Inserção do operador de cruzamento** *surface-filling***:** este operador é executado no final de cada iteração do algoritmo. O objetivo é preencher todos os espaços vazios do conjunto de soluções não-dominadas. O número de operações realizadas pelo cruzamento depende da eficiência medida por este operador nas iterações

anteriores.

3. **Auto-adaptação dos parâmetros do algoritmo:** O controlador PI possui três parâmetros: o parâmetro referente ao raio das esferas (ρ), o parâmetro que representa a proporção de elementos submetidos ao operador de cruzamento (r) e o parâmetro relacionado a distribuição Beta (β). Esses parâmetros são ajustados dentro do próprio algoritmo SCMGA. O parâmetro ρ é ajustado conforme o número atual de soluções que se deseja obter no arquivo. O parâmetro β é ajustado conforme a distância esperada da solução em relação o indivíduo pai no cruzamento *surface-filling*. E, por fim, o parâmetro r é ajustado de acordo com a eficiência do operador de cruzamento *surface-filling*, ou seja, se o número de operações bem-sucedidas é maior do que as operações não-sucedidas, o valor de r diminui. Caso contrário, este valor é aumentado.

O Algoritmo 3.1 apresenta o pseudocódigo do procedimento SCMGA. Os valores de entrada do algoritmo N e q, indicam, respectivamente, o tamanho da população inicial e o tamanho esperado do conjunto de soluções. A saída do algoritmo é o conjunto A com as soluções de estimativas de Pareto.

Na função (**start**_ρ_β_r) da linha 3 do Algoritmo 3.1 é definido o valor dos parâmetros iniciais do controlador (ρ, β e r do controlador). O parâmetro ρ, raio das esferas, é definido de tal forma que as q esferas do arquivo de soluções de dimensão $(m-1)$ ocupem um volume unitário equivalente a esta dimensão, onde m é a dimensão de espaço de busca. Os parâmetros β e r são calculados, respectivamente, conforme o valor médio dos intervalos dos limites inferior e superior.

A população inicial, de tamanho N, é gerada de forma aleatória na linha 4. Em seguida, na linha 6, a função **non_dominated**(P) determina quais os indivíduos da população P estão na primeira fronteira (não-dominadas). Até que se atinja o critério de parada, o algoritmo realiza os seguintes passos: a função **fast_non_dominated_sorting**(R_t) (linha 9) classifica os indivíduos em fronteiras; nas linhas 12-16, as soluções de cada fronteira são ordenadas conforme as distância obtidas pela função **crowding_distance_assignement**$(\mathcal{F}_i)$. Em seguida, (linhas 19-21), são aplicados os operadores de seleção, cruzamento e mutação, nesta ordem. Na linha 23 determina-se quais são os indivíduos, dos conjuntos de soluções da iteração atual e anterior, que são soluções não-dominadas. Até este passo, o procedimento do SCMGA é equivalente ao procedimento do NSGA-II original. As diferenças do NSGA-II para a proposta deste trabalho, são os novos operadores aplicados nas linhas 24-29.

O operador de cruzamento proposto *surface-filling*, indicado pela função **Surf_Fill_XO**, é aplicado ao final da iteração para preencher os espaços vazios do conjunto de soluções não-dominadas (A). Em seguida, a função **Updatade**_r_β faz a atualização dos valores dos parâmetros do controlador (r e β) e um novo conjunto de soluções não-dominadas é obtido. O operador *archive-set reduction* e o controlador PI, indicados

pela função (ρ_**Arquive_Control**), é aplicado para controlar o tamanho do arquivo A. Por fim, o valor do raio das esferas ρ do conjunto A é atualizado, função (**Update**_ρ(A,q)).

Algoritmo 3.1: *SCMGA*

Entrada: N, q
Saída: A

início
  $t \leftarrow 0$;
  ρ, β, $r \leftarrow$ **start**_ρ_β_r;
  $P_t \leftarrow$**new_population**(N); /* gera um conjunto aleatório de tamanho N */
  $Q_t \leftarrow \emptyset$;
  $A \leftarrow$ **non_dominated**(P_t);
  enquanto *não atingir o critério de parada* **faça**
    $R_t \leftarrow P_t \cup Q_t$;
    $\mathcal{F} \leftarrow$**fast_non_dominated_sorting**(R_t);
    $P_{t+1} \leftarrow \emptyset$;
    $i \leftarrow 1$;
    enquanto $|P_{t+1}| + |\mathcal{F}_i| \leq N$ **faça**
      $C_i \leftarrow$ **crowding_distance_assignement** ($\mathcal{F}_i$);
      $P_{t+1} \leftarrow P_t \cup \mathcal{F}_i$;
      $i \leftarrow i+1$;
    fim
    $\mathcal{F}_i \leftarrow$**sort**($\mathcal{F}_i, C_i$, descending); /* Classificação das *front* C_i */
    $P_{t+1} \leftarrow P_{t+1} \cup \mathcal{F}_i\,[1:(N-|P_{t+1}|)]$;
    $Q_{t+1} \leftarrow$**selection**(P_{t+1}, N);
    $Q_{t+1} \leftarrow$**crossover**(Q_{t+1});
    $Q_{t+1} \leftarrow$**mutation**(Q_{t+1});
    $t \leftarrow t+1$;
    $A \leftarrow$**non_dominated**($A \cup Q_t$);
    $(T,k_1,k_2,k_3,k_4,k_5,k_6) \leftarrow$ **Surf_Fill_XO** (A, ρ, r, 4, β);
    $(r,\beta) \leftarrow$ **Update_r_β** (r, β, k_1,k_2,k_3,k_4,k_5,k_6);
    $Q_t \leftarrow Q_t \cup T$;
    $A \leftarrow$ **non_dominated**($A \cup T$);
    $A \leftarrow$ ρ_**Arquive_Control**(A, ρ);
    ρ $\leftarrow$ **Update**_ρ(A,q);
  fim
fim

3.3 Operador de Controle por Esferas

O operador de controle por esferas, ou *sphere-control*, utiliza a ideia de esferas para melhorar a distribuição e o espalhamento das soluções ao longo da superfície de Pareto. A distribuição das soluções se refere a posição (cobertura) das soluções no espaço dos objetivos enquanto o espalhamento se refere as distâncias entre elas. O operador utiliza como premissa o fato de que as soluções que estão próximas no espaço de parâmetros também podem estar próximas no espaço dos objetivos.

Para explicar o funcionamento dos operadores, considere os conjuntos A e P, que representam, respectivamente, o arquivo externo com todas as soluções não dominadas e a população atual do algoritmo.

$$\begin{aligned} A &\triangleq \{\widetilde{x}_1, \cdots, \widetilde{x}_a\} \\ P &\triangleq \{x_1, \cdots, x_p\} \end{aligned} \tag{3-6}$$

com $|A| = a$ e $|P| = p$. As imagens desses conjuntos no espaço de objetivos são dados por:

$$\begin{aligned} A_y = \{\widetilde{y}_1, \cdots, \widetilde{y}_a\} &\triangleq \{f(\widetilde{x}_1), \cdots, f(\widetilde{x}_a)\} \\ P_y = \{y_1, \cdots, y_p\} &\triangleq f(x_1), \cdots, f(x_p) \end{aligned} \tag{3-7}$$

Os conjuntos A_y e P_y são normalizados entre 0 e 1, de acordo com os valores extremos de A_y. Os valores 0 e 1 representam, respectivamente, os valores mínimo e máximo de cada dimensão. Cada vez que o conjunto A_y é atualizado no algoritmo, esses conjuntos são novamente normalizados. Os operadores por esfera são usados para aumentar a representatividade do conjunto A_y, para tanto eles utilizam a ideia de distância e avaliam a distribuição das soluções ao longo deste conjunto.

O parâmetro ρ, que representa o raio de domínio de uma solução, precisa ser definido de forma a garantir que as soluções cubram todo o conjunto $\mathcal{P}$ de forma aproximadamente uniforme (distâncias entre soluções aproximadamente iguais). Este parâmetro é usado para orientar o algoritmo de forma que sejam geradas soluções que representam a região dentro da esfera de raio centrado no mesmo ponto. Isso quer dizer que quaisquer dois vizinhos devem estar localizados por pelo menos 2ρ de distância um do outro no espaço de parâmetros.

Nas regiões do conjunto em que os pontos são densamente distribuídos, o operador de *archive-set reduction* é aplicado para reduzir a quantidade de pontos. Em caso contrário, se o conjunto tiver poucos pontos, o operador de cruzamento *surface-filling*é aplicado para incluir novas soluções no arquivo. Assim, os operadores propostos são aplicados em cada região utilizando a ideia de comparação das distâncias entre os pontos no espaço de parâmetros.

O valor do parâmetro ρ, se for pré-definido no algoritmo, pode fazer com que o tamanho do arquivo A seja demasiadamente grande ou pequeno demais, o que pode ser um problema, uma vez que, a priori, não se tenha conhecimento do tamanho do conjunto de Pareto. Assim, neste trabalho, o parâmetro ρ é ajustado dinamicamente durante a execução do algoritmo. Este ajuste dinâmico é realizado considerando como referência uma quantidade de pontos que se deseja ter neste arquivo (q). A adaptação dinâmica permite que o algoritmo alcance uma boa dispersão do conjunto de soluções ao longo das iterações.

A Figura 3.6 ilustra o efeito dos operadores de controle por esfera no arquivo

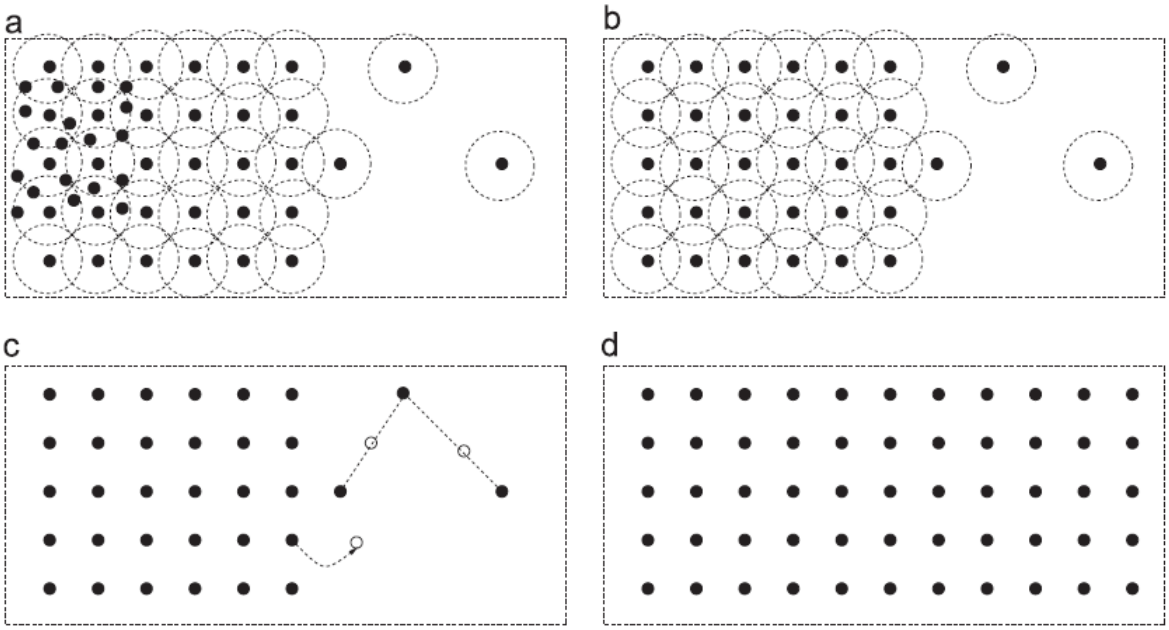


Figura 3.6: Operador de controle de esfera aplicado no espaço de parâmetros: (a) Representa um conjunto de soluções não-dominadas distribuídas de forma desproporcional. (b) O controle do tamanho de arquivo, realizado pelo operador *archive-set reduction*, elimina os pontos das regiões mais populosas. (c) O operador de cruzamento *surface-filling* é utilizado para gerar novos pontos que estão localizados nas regiões mais vazias. (d) Após um certo número de iterações, as soluções do conjunto de soluções não-dominadas tornam-se uniformemente distribuídas.

externo de soluções não-dominadas. As soluções que, no início do algoritmo, estavam concentradas em apenas uma região do espaço, recebem a aplicação dos operadores *archive-set reduction* e *surface-filling*. Tal aplicação é realizada utilizando operações por esferas, ao longo das iterações do algoritmo, gerando um novo conjunto de soluções não-dominadas, com soluções uniformemente distribuídas.

3.3.1 Controle do Tamanho do Arquivo

Considere o conjunto de soluções não-dominadas:

$$A = \{x_1, x_2, \cdots, x_m, x_{m+1}, \cdots, x_a\}$$

ordenado de forma que os m primeiros elementos possuem valor mínimo de função objetivo e os demais pontos foram preenchidos de forma aleatória. Considere também que o conjunto A_y (imagem do conjunto A) foi ordenado da mesma maneira. O tamanho destes conjuntos é controlado pelo Algoritmo 3.2. No algoritmo, as linhas 4 e 5 controlam, respectivamente, os conjuntos A e A_y. Neste algoritmo, são retirados dos conjuntos todos os elementos em que a distância entre eles é menor que ρ.

O número de soluções eficientes que são mantidas no arquivo é uma variável que

pode ser medida facilmente e também usada para ajustar o parâmetro ρ de modo a atingir uma cobertura das possíveis soluções do conjunto $\mathcal{P}$. O ajuste do parâmetro ρ também pode ser realizado levando em consideração a quantidade de soluções que se deseja no conjunto A. No entanto, o sistema de controle realimentado permite que este valor chegue ao equilíbrio, por si só, sem precisar do conhecimento do valor do parâmetro a.

Algoritmo 3.2: ρ_Archive_Control

Entrada: A, A_y, ρ
Saída: A, A_y
1 **início**
2 $i \leftarrow 1$;
3 **enquanto** $i \leq |A|$ **faça**
4 $A \leftarrow A - \{x_j \mid \|y_i - y_j\| < \rho,\ i \neq j\}$;
5 $A_y \leftarrow A_y - \{y_j \mid \|y_i - y_j\| < \rho,\ i \neq j\}$;
6 $i \leftarrow i + 1$;
7 **fim**
8 **fim**

O procedimento de ajuste do parâmetro ρ é apresentado no Algoritmo 3.3. O valor inicial de ρ é dado de tal forma que q esferas de dimensão $[m-1]$ ocupem um volume unitário no espaço de dimensão $[m-1]$. Os parâmetros do controle são iniciados com os seguintes valores: $K_p = 0,6$; $s = 0,1$ e $K_n = 0,9$. O pseudocódigo da função **Update_ρ** é apresentado no Algoritmo 3.3.

Algoritmo 3.3: Update_ρ

Entrada: A, q, ρ
Saída: ρ
1 **início**
2 **Parâmetros:** s, K_p, K_n;
3 $a \leftarrow |A|$;
4 **se** $a < q$ **então**
5 $e \leftarrow (a-q)/q$;
6 **se** $e > s$ **então**
7 $\Delta \leftarrow K_p \times s$;
8 **senão**
9 $\Delta \leftarrow K_p \times e$;
10 **fim**
11 $\rho \leftarrow (1+\Delta)$;
12 **senão**
13 $\rho \leftarrow K_n \times \rho$;
14 **fim**
15 **fim**

Este algoritmo realiza o ajuste do parâmetro ρ da seguinte forma: se a quantidade de pontos desejadas no arquivo q for maior que a quantidade de pontos do arquivo atual a, ou seja, $(q > a)$, então o valor de ρ deve ser aumentado (linha 13). Caso contrário, o valor de ρ deve ser diminuído. O incremento no valor de ρ é realizado pelo ajuste percentual dado pela variável Δ (linhas 9 e 11). Se o erro relativo for maior que um limitante s, indica

que o valor de Δ está saturado, e então, com o intuito de evitar variações bruscas no valor de ρ, Δ é atualizado (linha 7). O decremento do valor de ρ, caso necessário, é realizado na linha 13 do algoritmo. Este procedimento é semelhante aos controladores Proporcional-Integral (PI) com saturação de sinal de controle, usados em sistemas de controle industrial. A ação do controle sobre a variável ρ é incremental e com efeito de controle integral. Este termo integral no controlador PI induz um comportamento estável, com menor erro.

Esta metodologia torna o MOEA robusto em relação ao parâmetro ρ. Mesmo nos casos em que o conjunto de estimativas de Pareto é definido para uma dimensão menor que $[m-1]$, o algoritmo obteve um conjunto de estimativas de Pareto uniformemente espaçadas no espaço.

3.3.2 *Surface-filling crossover*

O operador de cruzamento *surface-filling crossover* retorna um conjunto de novas soluções e é usado pelo SCMGA para preencher todas os espaços vazios do conjunto de soluções não-dominadas, gerando um conjunto com soluções uniformemente distribuídas.

Considere a função $v(\cdot)$:

$$v(y_i) = \left|\{y_i| \left\|y_i - y_j\right\| < 3\rho\}\right| \tag{3-8}$$

que representa a cardinalidade do conjunto de pontos, a partir do conjunto A_y que pertence a uma esfera de raio 3ρ em torno do argumento da função y_j . O conjunto A_y é ordenado de forma crescente por $v(y_i)$:

$$A_y = \{y_i | v(y_i) \leq v(y_{i+1})\} \tag{3-9}$$

O conjunto A recebe a ordenação de acordo com a Equação (3-9) e o tamanho do conjunto A é dado por $|A| = a$. Assim, o conjunto A fica organizado de tal forma que as primeiras soluções são as que possuem menos vizinhos no espaço de objetivos.

Este operador utiliza duas variáveis controladas: a variável r, que está relacionada a proporção de elementos do conjunto A que são submetidos ao cruzamento, e a variável β, que está relacionada ao ajuste na distribuição Beta. O parâmetro r recebe valores entre 0 e 1. Este parâmetro é uma variável de controle que é ajustada dentro do próprio algoritmo pelo controlador, este procedimento é apresentado no Algoritmo 3.5.

O operador SFXO é aplicado nas $r.|A|$ primeiras soluções do conjunto A. A ordem de escolha dos indivíduos para o cruzamento SFXO se dá por: o primeiro candidato a pai p_1 é escolhido de forma aleatória e os demais candidatos são escolhidos de acordo com a proximidade com o pai atual. No final do procedimento, um conjunto de pais $p_1; p_2; \cdots, p_{r|A|}$ recebe o cruzamento do operador SFXO, e $r|A| - 1$ operações de cruzamento são realizadas respeitando a seguinte ordem:

$$(p_1 \times p_2),(p_2 \times p_3),(p_3 \times p_4),...,(p_{(r|A|-1)} \times p_{(r.|A|)}).$$

O pseudocódigo do procedimento SFXO é apresentado no Algoritmo 3.4. Neste algoritmo, as entradas são o conjunto A e as variáveis controladas r e β. O operador retorna o conjunto de novas soluções geradas pelo cruzamento, sendo o conjunto T formado pelos elementos que não são dominados por nenhum elemento de A. Os contadores k_i, representam as relações de dominância de cada solução ($\widehat{x}_i$) em relação aos pais (p_i e p_{i+1}) e são usados para fazer a adaptação dos parâmetros β e r.

Algoritmo 3.4: Surf_Fill_XO

Entrada: A
Entrada: r, β
Saída: T, F
Saída: k_1,k_2,k_3,k_4,k_5,k_6
início
  $k_1,k_2,k_3,k_4,k_5,k_6 \leftarrow 0$;
  $T,F \leftarrow \emptyset$;
  $p \leftarrow$ **Sequence_SFXO**(A,r);
  para $i = 1$ **até** $(r.|A|-1)$ **faça**
    $b =\leftarrow Beta(\alpha,\beta)$;
    $\widehat{x}_i \leftarrow (1-b).p_i + b.p_{i+1}$;
    $\widehat{y}_i \leftarrow f(\widehat{x}_i)$;
    $vf \leftarrow FALSE$;
    se $\|\widehat{y}_i - f(p_i)\| < \rho$ **então**
      $k_5 \leftarrow k_5 + 1$;
      $vf \leftarrow TRUE$;
    fim
    se $\|\widehat{y}_i - f(p_{i+1})\| < \rho$ **então**
      $k_6 \leftarrow k_6 + 1$;
      $vf \leftarrow TRUE$;
    fim
    se $\widehat{y}_i \prec f(p_i)$ *ou* $\widehat{y}_i \prec f(p_{i+1})$ **então**
      $k_1 \leftarrow k_1 + 1$;
      $T \leftarrow T \cup \widehat{x}_i$;
      $F \leftarrow F \cup \widehat{y}_i$;
    senão se $\widehat{y}_i \succ f(p_i)$ *ou* $\widehat{y}_i \succ f(p_{i+1})$ **então**
      $k_2 \leftarrow k_2 + 1$;
    senão
      se $vf = FALSE$ **então**
        se $\nexists\ j \setminus f(A^j) \prec \widehat{y}_i$ **então**
          $k_3 \leftarrow k_3 + 1$;
          $T \leftarrow T \cup \widehat{x}_i$;
          $F \leftarrow F \cup \widehat{y}_i$;
        senão
          $k_4 \leftarrow k_4 + 1$;
        fim
      fim
    fim
  fim
fim

O operador realiza o seguinte procedimento: a função **Sequence_SFXO**(A,r),

linha 4, cria uma sequência de pais $p_1; p_2; ...; p_{r.|A|}$ para aplicar o operador de cruzamento SFXO. A variável β, nas linhas 6 e 7, é usada para definir se a solução gerada está mais próxima de p_i ou p_{i+1}. As soluções não-dominadas são inseridas no conjunto T, linha 28, e este conjunto é inserido na população atual do AG.

O parâmetro β, usado na linha 6 do Algoritmo 3.4, foi implementado como uma variável controlada que varia dinamicamente ao longo das iterações do algoritmo, com o objetivo de fazer os ajustes na distribuição Beta. Os valores baixos de β levam a mediana da PDF a se aproximar de 1 (solução gerada pelo cruzamento estará próxima de p_{i+1}) enquanto valores mais altos a se aproximar 0 (solução gerada pelo cruzamento estará próxima de p_i). O valor do parâmetro α da distribuição Beta (linha 6) é fixo em $\alpha = 4$, para de gerar funções côncavas de distribuição de probabilidade (PDFs). As configurações da PDF para as distribuições Beta, considerando $\alpha = 4$ e diferentes valores de β, são apresentados na Figura 3.7.

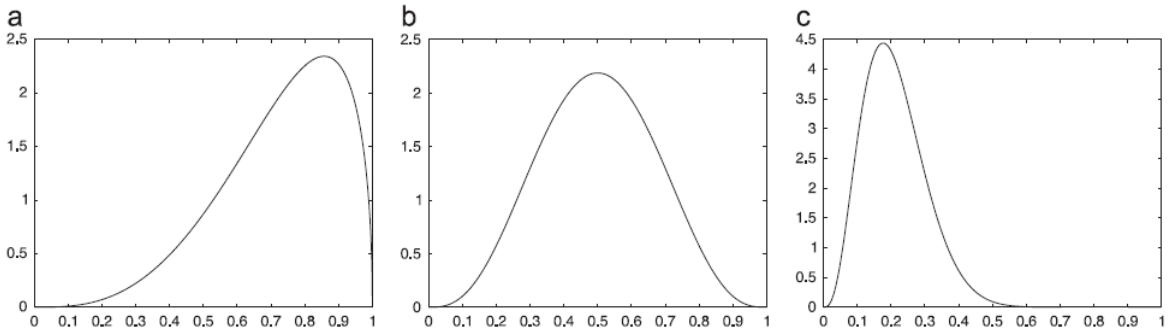


Figura 3.7: Diferentes formas de distribuição Beta, para $\alpha = 4$ e três valores distintos do parâmetro β. (a) $\beta = 1,5$, (b) $\beta = 4$ e (c) $\beta = 15$

O operador de cruzamento *surface-filling* tem por objetivo intercalar as soluções que tem menos vizinhos dentro no espaço de soluções. Assim, os indivíduos do conjunto de soluções não-dominadas são submetidas ao cruzamento SFXO, para tentar preencher os espaços vazios deste conjunto e gerar uma melhor descrição do conjunto. Cada solução obtida pelo operador é analisada e conduzida de acordo com os seguintes contadores k_i:

Contador k_1: $\widehat{x}_i$ domina p_i ou p_{i+1};
Contador k_2: $\widehat{x}_i$ é dominado por p_i ou p_{i+1};
Contador k_3: $\widehat{x}_i$ não é dominado por p_i e nem por p_{i+1} ou por nenhuma outra solução que esteja em A;
Contador k_4: $\widehat{x}_i$ não é dominado por p_i e nem por p_{i+1}, mas são dominados por alguma outra solução em A;
Contador k_5: A distância entre $\widehat{x}_i$ e p_i é inferior a ρ;
Contador k_6: A distância entre $\widehat{x}_i$ e p_{i+1} é inferior a ρ.

Os contadores k_i são incrementados cada vez que uma solução produz a sua respectiva condição como verdadeira. Além disso, estes seis contadores são usados

para estimar a eficiência do operador de cruzamento SFXO e para atualizar os valores dos parâmetros r e ρ. O pseudo-código do procedimento que faz as atualizações dos parâmetros é apresentado no Algoritmo 3.5.

Neste algoritmo, as entradas são as variáveis controladas r e β e os contadores (k_i), usados para realizar a atualização destas variáveis controladas. As variáveis n_{SFXO}, S_{SFXO} e U_{SFXO} representam, respectivamente, o número de operações realizadas, bem sucedidas e sem sucesso do operador de cruzamento SFXO. Essas operações são medidas utilizando os contadores k_i (linhas 4-6).

Os cálculos para determinar se uma nova solução estará mais próxima de p_i ou p_{i+1} são apresentados, respectivamente, nas linhas 9 e 11 do algoritmo. O controle da variável r é feito nas linhas (15-20), onde podem ocorrer um aumento do número de operações do SFXO (linha 16) ou a diminuição dessa quantidade de operações (linha 18). O controle de saturação da variável β é calculado para um limite inferior e superior, respectivamente, nas linhas 22 e 24. Da mesma forma, o controle de saturação da variável r é calculado para o limite inferior e superior, nas linhas 28 e 30.

Assim, o ajuste dos parâmetros β e r são realizados conforme apresentado no Algoritmo 3.5 e de acordo com os resultados observados nas operações realizadas pelo operador de cruzamento SFXO:

- **Ajuste da variável β:** se mais de 25% das soluções geradas pelo cruzamento estiverem mais próximas do pai p_1, então a distribuição Beta é ajustada de tal forma que, na próxima geração, o operador terá maior probabilidade de gerar soluções mais distantes dessa região. O mesmo acontece considerando o pai p_2. Isso garante que o operador de cruzamento possa gerar soluções que também estejam localizadas em regiões mais distantes das regiões que estão os pais escolhidos.
- **Ajuste da variável r:** esta variável interfere no aumento ou diminuição da quantidade de operações do SFXO. Se a quantidade de operações bem sucedidas for maior que a quantidade de operações sem sucesso, então o número de operações de cruzamento esperadas na próxima geração é maior. Por outro lado, quando as quantidade de operações ruins é maior, este número é reduzido.

Para evitar variações muito bruscas no algoritmo, é realizado um controle de saturação das variáveis β e r. O ajuste dessas variáveis é sempre executado em pequenos passos, ou seja, os valores serão aumentados ou diminuídos considerando uma variação de 10%.

No final do procedimento de cruzamento realizado pelo operador SFXO, são armazenas no conjunto T, todas as soluções não-dominadas por nenhum elemento de A e que possuem distâncias maiores que ρ dos pais p_i e p_{i+1}. Em seguida, no SCMGA, as soluções armazenadas em um conjunto T são inseridas na população atual do algoritmo (conjunto Q), com o intuito de melhorar a qualidade deste conjunto.

Algoritmo 3.5: Update_ρ_β

Entrada: ρ, β
Entrada: $k_1, k_2, k_3, k_4, k_5, k_6$
Saída: r, β

```
início
    b_exp ← log10(β);
    r_inv ← 1/r;
    n_SFXO ← k1 + k2 + k3 + k4;
    S_SFXO ← k1 + k3;
    U_SFXO ← k2 + k4;
    se (k5 ≥ 0,25.n_SFXO) ou (k6 ≥ 0,25.n_SFXO) então
        se k5 > k6 então
            b_exp ← 0,90.b_exp;
        senão
            b_exp ← 1,10.b_exp;
        fim
    fim
    se S_SFXO ≥ U_SFXO então
        r_inv ← 0.90.r_inv;
    senão
        r_inv ← 1.10.r_inv;
    fim
    se b_exp < 0.25 então
        b_exp ← 0.25;
    senão
        se b_exp > 1.25 então
            b_exp ← 1.25;
        fim
    fim
    se r_inv > 10 então
        r_inv ← 10;
    senão
        se r_inv < 2 então
            r_inv ← 2;
        fim
    fim
    r ← 1/r_inv;
    β ← 10^b_exp;
fim
```

Como os operadores de controle por esfera partem do princípio que as soluções que estão próximas no espaço de parâmetros também estão próximas no espaço de objetivos, podemos admitir que a operação do cruzamento, que é realizada no espaço de parâmetros, pode ser usado para preencher de forma adequada os espaços vazios do conjunto de Pareto, que está no espaço de objetivos. O operador *surface-filling crossover* consegue realizar este preenchimento, gerando soluções que estão na mesma fronteira e em pontos próximos dos pais. Assim, o operador se mostra uma ferramenta eficiente para encontrar novas soluções do conjunto Pareto-Ótimo aproximado quando a quantidade de soluções distribuídas por este conjunto não for representativo.

3.3.3 Complexidade do Operadores

O operador de controle do tamanho do arquivo (apresentado na seção 3.3.1) possui três operações consideradas caras computacionalmente:

1. Identificação dos pontos extremos do conjunto de soluções: esta operação é realizada antes de fazer o controle do tamanho do arquivo. Determinar esses pontos extremos do conjunto de estimativas de Pareto é importante para garantir que esses pontos não sejam deslocados. Essa operação possui custo computacional $O(|\mathcal{A}|)$, onde $|\mathcal{A}|$ representa o tamanho do arquivo que possui o conjunto de soluções do conjunto de estimativas de Pareto.

2. Cálculo das distâncias entre as soluções: esta operação é realizada para informar se as soluções estão todas aglomeradas em uma mesma região. O cálculo realizado possui um custo computacional de $O(|\mathcal{A}|^2)$.

3. Redução do conjunto de Pareto: no pior caso o laço *enquanto*, no Algoritmo 3.2, precisa realizar todas as $|\mathcal{A}| - 1$ comparações. Assim, no pior caso, a complexidade deste laço é dada por $O(|\mathcal{A}|^2)$.

Logo, a complexidade computacional do operador de controle do tamanho do arquivo é de $O(|\mathcal{A}|^2)$. É importante ressaltar que, o número de soluções no conjunto de estimativas de Pareto $\mathcal{A}$ é um parâmetro definido pelo utilizador e, portanto, não é afetado pelo número de funções objetivo e nem pela dimensão do problema.

O operador SFXO (apresentado na seção 3.3.2) executa para $r.|\mathcal{A}|$ vezes um operador de cruzamento, que é muito semelhante ao operador original. Assim, a complexidade deste algoritmo é dada por $r.|\mathcal{A}|$ vezes a complexidade do operador de cruzamento. Uma vez que o tamanho de $|\mathcal{A}|$ é controlado e o valor de r é delimitado, a complexidade deste procedimento é controlada pelo usuário.

Os algoritmos de controle que fazem os ajustes dos parâmetros ρ e β possuem a complexidade de ordem $O(1)$.

É importante destacar que, os cálculos realizados pelos operadores propostos geralmente não têm grande impacto no algoritmo, visto que na maior parte dos casos, o que pode gerar um maior custo computacional no algoritmo é a avaliação de função.

3.4 Resultados Preliminares: Comparação do SCMGA com o NSGA-II

A fim de avaliar a adequação da metodologia proposta, o algoritmo SCMGA (*Sphere-Control Multiobjetive Genetic Algorithm*) usando operadores baseados em controle por esfera, uma série de comparações quantitativas foi realizada utilizando 18 fun-

ções de referência da literatura. O algoritmo SCMGA desenvolvido foi comparado com a versão original do NSGA-II (implementação obtida na plataforma PISA [97]). O NSGA-II utiliza os mesmos operadores e a mesma estrutura que a versão original [19] e a função de probabilidade da distribuição dos operadores é ajustada para gerar valores reais pertencentes ao intervalo $[0,\ 1]$.

3.4.1 Descrição dos Problemas Teste

Os problemas testes de otimização multiobjetivo usados para avaliar o desempenho do algoritmo SCMGA em comparação com o NSGA-II, são descritos nesta seção.

Problema de Kursawe [46]

$$x^* = \arg\min_x \begin{cases} f_1 = \sum_{i=1}^{n-1} \left[-10.exp\left(-0.2\sqrt{x_i^2 + x_{i+1}^2} \right) \right] \\ f_2 = \sum_{i=1}^{n} \left[|x_i^{0.8}| + 5.sin(x_i^3) \right] \end{cases} \tag{3-10}$$

$$\text{sujeito a}: \quad -5 \leq x_i \leq 5, \quad \forall\ i = 1, \cdots, n \tag{3-11}$$

em que n representa o número de variáveis do problema. Três instâncias do problema de Kursawe são consideradas:

- **KUR10:** n=10;
- **KUR20:** n=20;
- **KUR30:** n=30;

Problema quadrático:

$$x^* = \arg\min_x \begin{cases} f_1 = (X - C^1)'.I.(X - C^1) \\ f_2 = (X - C^2)'.I.(X - C^2) \\ \vdots \\ f_{16} = (X - C^{16})'.I.(X - C^{16}) \end{cases} \tag{3-12}$$

$$\text{sujeito a}: \quad -10 \leq x_i \leq 10, \quad \forall i = 1, \cdots, n \tag{3-13}$$

em que n é o número de variáveis do problema; I é a matriz identidade de tamanho $n \times n$; e C^i é o mínimo da função quadrática i:

$$C^i = \begin{cases} c_i^j = 1, & se\ \ j = 1 \\ c_i^j = 0, & \text{caso contrário} \end{cases}$$

Geralmente, os problemas quadráticos são mais fáceis de serem resolvidos. No entanto, são úteis para avaliar o desempenho do algoritmo proposto, principalmente

quando aumenta o número de funções objetivo. Quatro casos deste problema são considerados e em todos os casos foram assumidos $n = 20$.

- **QUAD2:** duas funções objetivo, considerando apenas as funções f_1 e f_2;
- **QUAD4:** quatro funções objetivo, considerando as funções f_1 até f_4;
- **QUAD8:** oito funções objetivo, considerando as funções f_1 até f_8;
- **QUAD16:** dezesseis funções objetivo, considerando todas as funções.

Problema de Fonseca e Fleming [29]

$$x^* = \arg\min_x \left\{ \begin{array}{l} f_1 = 1 - exp\left[-\sum_{i=1}^{n}\left(x_i - \frac{1}{\sqrt{n}}\right)^2\right] \\ f_2 1 - exp\left[-\sum_{i=1}^{n}\left(x_i + \frac{1}{\sqrt{n}}\right)^2\right] \end{array} \right\} \quad (3\text{-}14)$$

$$\text{sujeito a : } \quad -4 \leq x_i \leq 4, \quad \forall i = 1, \cdots, n \quad (3\text{-}15)$$

onde n indica o número de variáveis do problema. Considerou-se apenas uma instância deste problema:

- **FONSECA:** n=15.

Problema de Viennet [82]

$$x^* = \arg\min_x \left\{ \begin{array}{ll} f_1 = & 0.5(x_1^2 + x_2^2) + sin(x_1^2 + x_2^2) \\ f_2 = & \frac{(3x_1 - 2x_2 + 4)^2}{8} + \frac{(x_1 - x_2 + 1)^2}{27} + 15 \\ f_3 = & \frac{1}{x_1^2 + x_2^2 + 1} - 1.1exp(-(x_1 + x_2)^2) \end{array} \right. \quad (3\text{-}16)$$

$$\text{sujeito a : } \quad -3 \leq x_i \leq 3 \quad \forall i = 1, 2 \quad (3\text{-}17)$$

Apenas uma instância foi considerada para este problema:

- **VIENNET**.

Problemas DTLZ [22]

Todos os sete casos sem restrições deste conjunto de problemas foram testados neste trabalho, com a seguinte configuração:

- **DTLZ1:** $n = 7,\ k = 5, M = 3$;

$$\min \left\{ \begin{array}{ll} f_1(x) = & 0.5[1 + g_1(x)]x_1x_2 \\ f_2(x) = & 0.5[1 + g_1(x)]x_1(1 - x_2) \\ f_3(x) = & 0.5[1 + g_1(x)](1 - x_1) \end{array} \right. \quad (3\text{-}18)$$

- **DTLZ2:** $n = 12,\ k = 10,\ M = 3$;

$$\min \begin{cases} f_1(x) = & [1+g_2(x)]\cos(x_1\pi/2)\cos(x_2\pi/2) \\ f_2(x) = & [1+g_2(x)]\cos(x_1\pi/2)\sin(x_2\pi/2) \\ f_3(x) = & [1+g_2(x)]\sin(x_2\pi/2) \end{cases} \tag{3-19}$$

- **DTLZ3:** $n = 12,\ k = 10,\ M = 3$;

$$\min \begin{cases} f_1(x) = & [1+g_1(x)]\cos(x_1\pi/2)\cos(x_2\pi/2) \\ f_2(x) = & [1+g_1(x)]\cos(x_1\pi/2)\sin(x_2\pi/2) \\ f_3(x) = & [1+g_1(x)]\sin(x_1\pi/2) \end{cases} \tag{3-20}$$

- **DTLZ4:** $n = 12,\ k = 10,\ M = 3,\ \alpha = 100$;

$$\min \begin{cases} f_1(x) = & [1+g_2(x)]\cos(x_1^a\pi/2)\cos(x_2^a\pi/2) \\ f_2(x) = & [1+g_2(x)]\cos(x_1^a\pi/2)\sin(x_2^a\pi/2) \\ f_3(x) = & [1+g_2(x)]\sin(x_1^a\pi/2) \end{cases} \tag{3-21}$$

- **DTLZ5:** $n = 12,\ k = 10,\ M = 3$;

$$\min \begin{cases} f_1(x) = & [1+g_2(x)]\cos(\theta_1(x)\pi/2)\cos(\theta_2(x)\pi/2) \\ f_2(x) = & [1+g_2(x)]\cos(\theta_1(x)\pi/2)\sin(\theta_2(x)\pi/2) \\ f_3(x) = & [1+g_2(x)]\sin(\theta_1(x)\pi/2) \end{cases} \tag{3-22}$$

- **DTLZ6:** $n = 12,\ k = 10,\ M = 3$;

$$\min \begin{cases} f_1(x) = & [1+g_3(x)]\cos(\theta_1(x)\pi/2)\cos(\theta_2(x)\pi/2) \\ f_2(x) = & [1+g_3(x)]\cos(\theta_1(x)\pi/2)\sin(\theta_2(x)\pi/2) \\ f_3(x) = & [1+g_3(x)]\sin(\theta_1(x)\pi/2) \end{cases} \tag{3-23}$$

- **DTLZ7:** $n = 22,\ k = 20,\ M = 3$;

$$\min \begin{cases} f_1(x) = & x_1 \\ f_2(x) = & x_2 \\ f_3(x) = & [1+g_4(x)]h(x) \end{cases} \tag{3-24}$$

Para estes problemas:

$$g_1(x) = 100\left\{|x| + \sum_{i=1}^{n}(x_i - 0.5)^2 - cos[20\pi(x_i - 0.5)]\right\} \quad (3\text{-}25)$$

$$g_2(x) = \sum_{i=1}^{n}(x_i - 0.5)^2 \quad (3\text{-}26)$$

$$g_3(x) = \sum_{i=1}^{n} x_i^{0.1} \quad (3\text{-}27)$$

$$g_4(x) = 1 + \frac{9}{|x|}\sum_{i=1}^{n} x_i \quad (3\text{-}28)$$

$$\theta_1(x) = \frac{\pi}{4[1+g_2(x)]}[1 + 2g_2(x)x_i] \quad (3\text{-}29)$$

$$\theta_2(x) = \frac{\pi}{4[1+g_3(x)]}[1 + 2g_3(x)x_i] \quad (3\text{-}30)$$

$$h(x) = 3 - \sum_{i=1}^{2}\left\{\frac{f_i(x)}{1+g_i(x)}\{1 + sin[3\pi f_i(c)]\}\right\} \quad (3\text{-}31)$$

$$\text{sujeito a:} \quad 0 \leq x_i \leq 1, \ \ \forall i \in 1, \cdots, n. \quad (3\text{-}32)$$

Os problemas DTLZ podem ser usados considerando qualquer quantidade de funções objetivo. O número de objetivos é definido por dois parâmetros: n que representa a quantidade de variáveis de decisão e k que representa um parâmetro escolhido a priori. Os valores de k são dados de acordo com o artigo referência e os valores de n são ajustados para problemas com três funções objetivo.

Problema do Caixeiro Viajante [70]

$$x^* = \arg\min_x \begin{cases} f_1(x) = x_{ij}C_{ij}^1 \\ f_2(x) = x_{ij}C_{ij}^2 \\ f_3(x) = x_{ij}C_{ij}^3 \end{cases} \quad (3\text{-}33)$$

$$\text{sujeito a :} \begin{cases} \sum_{i=1}^{n} x_{ij} = 1 & \forall \ i = 1, \cdots, n \\ \sum_{j=1}^{n} x_{ij} = 1 & \forall \ j = 1, \cdots, n \\ x_{ij} \in 0, 1 & \forall \ i, j(1, \cdots) \times (1, \cdots, n) \end{cases} \quad (3\text{-}34)$$

onde n é a quantidade de cidades; x_{ij} é uma variável binária de decisão (o valor 1 indica que as cidades i e j estão ligadas e 0 caso contrário); C^k é uma matriz de custo $n \times n$, onde cada elemento c_{ij} define o custo envolvido de sair da cidade i para a cidade j. A diferença entres as funções objetivo ($f_1(\cdot)$, $f_2(\cdot)$ e $f_3(\cdot)$) está relacionada à matriz de custo C^k. Numa situação prática, cada matriz representa um tipo de custos (tempo, a distância ou consumo de combustível). Duas instâncias do problema do caixeiro viajante (TSP), assumindo $n = 50$ (50 cidades), são consideradas neste trabalho:

- **TSP2:** Considera apenas duas funções objetivo: f_1 e f_2;
- **TSP3:** Considera as três funções objetivo do problemas.

Codificação e decodificação de uma solução: Neste trabalho, todas as soluções dos problemas descritos foram codificadas como um vetor de números reais $V = [1 \times n]$. Cada elemento deste vetor assume um valor pertencente ao intervalo $v_i \in [0,\ 1]$.

Nos problemas de natureza contínua (Kuarsawe, quadrático, Fonseca e Fleming, Viannet e DTLZs) as soluções foram decodificadas, para o domínio tratado, por meio da equação $x_i = L_i + v_i.(U_i - Li)$. Em que x_i é o valor da i-ésima variável no domínio do problema e as variáveis U_i e L_i representam, respectivamente, os limites inferior e superior das variáveis.

Nos problemas TSP, as soluções foram decodificadas usando um esquema de codificação de chaves aleatórias [3]. O procedimento transforma um vetor de números reais $1 \times n$ em uma permutação de n elementos. Essa permutação representa uma solução do problema, ou seja, a sequência de cidades que serão visitadas.

3.4.2 Implementação das Métricas de Comparação

O método que controla o tamanho do arquivo foi incluído para manter este arquivo com um tamanho fixo. Esta ideia de controle é baseada na função **crowding_distance_assignement**, apresentada na seção 3.2.1. No algoritmo SCMGA a cada iteração os seguintes procedimentos são realizados: controle do tamanho do arquivo externo pela função ρ*_Archive_Control*, aplicação do operador de cruzamento *surface-filling* e os ajustes dos parâmetros α, β e r, como mostra o Algoritmo 3.1.

O desempenho dos algoritmos NSGA-II e SCMGA para resolver as 18 funções de referência da literatura, são comparados com o propósito de estimar o efeito dos operadores de controle de esfera sobre a convergência do algoritmo. A comparação para cada problema teste, respeita os seguintes passos:

1. Cada algoritmo é executado k vezes considerando o mesmo conjunto de parâmetros e critério de parada.
2. Baseado nos valores de máximo e mínimo de cada função objetivo e considerando todas as execuções dos algoritmos, as fronteiras são normalizadas entre 0 e 1 (os valores 0 e 1 não estão, necessariamente, presentes nos conjuntos de estimativas Pareto-Ótimo).
3. Para cada uma das i execuções de k são:
   - **(a)** acrescentados os conjuntos obtidos pelos algoritmos NSGA-II ($A_i^{NSGA-II}$) e SCMGA (A_i^{SCMGA}) em um único conjunto A;
   - **(b)** verifica-se a dominância do conjunto A e cria-se um conjunto A' com as soluções não-dominadas deste conjunto;
   - **(c)** é feito $A_i^{NSGA-II} = A_i^{NSGA-II} \cap A'$ e $A_i^{SCMGA} = A_i^{SCMGA} \cap A'$;
   - **(d)** emprega-se algumas métricas de qualidade de Pareto (PQM) para avaliar o desempenho da convergência dos algoritmos NSGA-II e SCMGA :

- $q_i^{NSGA-II} = PQM(A_i^{NSGA-II}, A_i^{SCMGA})$;
- $q_i^{SCMGA} = PQM(A_i^{SCMGA}, A_i^{NSGA-II})$.

4. Ordenação dos vetores, $q^{NSGA-II}$ e q^{SCMGA}, em ordem crescente.

No final deste processo, uma quantidade de k quantis, com a relação das convergências dos algoritmos analisados é retornada. Esses quantis podem ser utilizados para fazer uma análise da dominância estocástica dos algoritmos. Para maiores detalhes, consultar [9]. A partir do princípio de dominância estocástica, é possível dizer que um algoritmo multiobjetivo A_1 domina estocasticamente outro algoritmo $A_2 (A_1 \succ A_2)$ se, e somente se:

$$A_1 \succ A_2 : \begin{cases} q_i^{A_1} \geq q_i^{A_2} & \forall\, i \in 1, \cdots, k \\ \exists i | q_i^{A_1} > q_i^{A_2} \end{cases} \tag{3-35}$$

Portanto, dizemos que o algoritmo A_1 domina o algoritmo A_2, se todos os k quantis de A_1 são superiores ou iguais aos quantis de A_2 e, se pelo menos, um quantil de A_1 é maior do que o mesmo quantil de A_2. Este tipo de análise fornece uma comparação precisa dos resultados gerados pelo algoritmo.

Apesar deste processo ser melhor do que uma comparação feita por uma única execução do algoritmo ou por uma análise visual, ele ainda apresenta o seguinte problema: para um número baixo de execuções dos algoritmos, a ordem em que os arquivos são comparados pode afetar os vetores q. Isto acontece pelo fato do resultado da análise de dominância depender da ordem em que os conjuntos são comparados. Esta dependência de ordem pode gerar resultados imprecisos, visto que duas execuções diferentes podem gerar soluções diferentes. Para contornar este problema, procedimentos com testes de permutação são utilizados para realizar a análise dos algoritmos. Em cada teste, permutações aleatórias dos índices dos arquivos obtidos pelos algoritmos são geradas e os arquivos são comparados na sequência destes índices. O processo é repetido P vezes e o valor de P é pre-determinado.

Cada teste de permutação gera um conjunto de quantis q para cada algoritmo. O valor final do quantil q_i é obtido através da determinação da média de todos os quantis (q_i) obtidos nos P testes. Este procedimento reduz a variabilidade das soluções fornecidas pelo método, quando se compara os mesmos algoritmos.

Métricas para medir a qualidade do conjunto de estimativas de Pareto

Três métricas de qualidade do conjunto de Pareto (PQM) são usadas para estimar a qualidade do conjunto de estimativas de Pareto obtidas pelos algoritmos: a métrica C, para avaliar a fronteira do conjunto; a métrica Δ, para avaliar o espaçamento e, a métrica integral da contagem de esferas, que tenta avaliar os dois aspectos ao mesmo tempo.

Métrica C (CM): proposta em [100], esta métrica estima a cobertura de duas estimativas do conjunto de Pareto utilizando a ideia de dominância. Considere o conjunto $\mathcal{X}$, vetores candidatos a solução do problema, e sejam $\mathcal{A} \subseteq \mathcal{X}$ e $\mathcal{B} \subseteq \mathcal{X}$ dois conjuntos de vetores de decisão. A cobertura de $\mathcal{A}$ em relação a $\mathcal{B}$, $(C(\mathcal{A},\mathcal{B}))$, pode ser calculada pela equação:

$$C(\mathcal{A},\mathcal{B}) = \frac{|\{a \in \mathcal{A} \mid \exists\, b \in \mathcal{B} : b \preceq a\}|}{|\mathcal{A}|} \tag{3-36}$$

Nesta expressão, a fração indica qual a proporção de soluções do conjunto $\mathcal{A}$ que são dominados por soluções do conjunto $\mathcal{B}$. Se $C(\mathcal{A},\mathcal{B}) = 0$, então $\mathcal{A}$ não tem solução dominada por nenhum elemento de $\mathcal{B}$. Por outro lado, se $C(\mathcal{A},\mathcal{B}) = 1$, então, todas as soluções de $\mathcal{A}$ são dominadas por pelo menos uma solução do conjunto $\mathcal{B}$. Deste modo, valores mais baixos de C sugerem uma aproximação mais adequada da primeira fronteira de Pareto.

Métrica Δ (ΔM): esta métrica é usada para estimar se as soluções do conjunto aproximado de Pareto estão espaçadas no domínio das funções objetivo [21]. Seja $\mathcal{X}$ um conjunto de vetores candidatos a solução do problema e seja $\mathcal{A} \subseteq \mathcal{X}$ um conjunto de vetores candidatos. A métrica Δ para o conjunto $\mathcal{A}$, $\Delta(\mathcal{A})$, é calculada pela seguinte equação:

$$\Delta(\mathcal{A}) = \frac{\sum_{i=1}^{k} d_i^e + \sum_{i=1}^{|\mathcal{A}|} |d_i - \overline{d}|}{\sum_{i=1}^{k} d_i^e + |\mathcal{A}|\overline{d}} \tag{3-37}$$

onde k representa a quantidade de funções objetivo; d_i^e a distância Euclidiana entre os pontos extremos em $\mathcal{A}$ e em $\mathcal{X}^*$; d_i é a distância Euclidiana entre o ponto $i \in \mathcal{A}$ e o seu vizinho mais próximo; $\overline{d}$ é a média de todas as distâncias d_i; e, $\mathcal{X}^*$ uma amostragem do conjunto real de Pareto do problema. O resultado obtido na Equação (3-37) diz, que valores baixos de Δ estão associados a um conjunto de Pareto melhor espaçado.

Integral da Contagem de Esferas (ISC): esta métrica é usada para estimar a qualidade dos conjuntos de estimativas do conjunto de Pareto, [73]. O procedimento é apresentado no Algoritmo 3.6. No algoritmo a variável n_r é o número de raios em que a contagem de esferas é realizada e r_d o vetor com os valores dos raios a serem analisados. Para todas as execuções são adotados valores $n_r = 11$ e $r_d = [0.001, \cdots, 0.1]$. Observe que, na comparação do conjunto de estimativas de Pareto, o ISC é aplicado somente nos pontos do algoritmo que não são dominados pelos demais pontos. Assim, essa métrica permite identificar qual a fronteira dos algoritmos apresentam maior espalhamento, pela contagem do número de esferas.

A métrica ISC mede a eficácia do conjunto de estimativas de Pareto para diferentes amostragens. Quanto mais regular é a distribuição destas amostras, maior será

o valor da métrica ISC. Neste sentido, o ISC atribui valores mais altos para as melhores aproximações do conjunto de Pareto (fronteira que possui melhor distribuição).

Algoritmo 3.6: *ISC*

Entrada: A, n_r, r_d
início
  $ISC \leftarrow 0$;
  para $i = 1$ **até** n_r **faça**
    $\mathcal{C} \leftarrow A$;
    $c_i \leftarrow 1$;
    $r \leftarrow r_d(i)$;
    Coloque uma esfera de raio r centrada em um ponto do conjunto $\mathcal{C}$;
    enquanto $|\mathcal{C}| > 0$ **faça**
      Inicializa um contador de esferas;
      $\mathcal{C} \leftarrow \mathcal{C} - \mathcal{R}$;
      Dentre os pontos restantes, centralize outra esfera no ponto mais próximo do ponto central anterior;
      $c_i \leftarrow c_i + 1$;
    fim
    $ISC \leftarrow ISC + c_i$
  fim
fim

Essas métricas foram escolhidas por se mostrarem eficientes na comparação da qualidade dos conjuntos de Pareto obtidas pelos algoritmos e com baixo custo computacional.

Nota-se ainda que os passos do processo de comparação descritos no inicio desta seção podem ser usados para as três métricas. Apenas no caso específico de ΔM, os passos $3a - 3c$ não devem ser executados, uma vez que esta métrica requer que os conjuntos de entrada tenham o mesmo número de elementos. Além disso, os testes de permutação não são necessários nesta métrica, uma vez que ela não faz comparações por pares.

A comparação dos algoritmos, utilizando dominância estocástica, é apresentada na Equação (3-35). A informação visual desta comparação é fornecida utilizando *boxplot*, onde as linhas horizontais representam (de baixo para cima): os quantis $0,025$, quantis $0,25$, quantis $0,50$, quantis $0,75$ e quantis $0,975$.

3.4.3 Resultados Numéricos

As simulações foram realizadas em um iMac 27 (modelo 2011), com processador Core i7 de 3,4 GHz e 16 GB de RAM, usando Matlab 2010. Os algoritmos, NSGA-II e SCMGA, foram implementados usando os mesmos parâmetros:

- Tamanho da população: 100 indivíduos.
- Critério de parada: número máximo de avaliações de função:
  - **QUAD2:** 10.000 avaliações.
  - **KUR10** e **QUAD4:** 20.000 avaliações.

  - **FONSECA, VIENNET, DTLZ1, DTLZ2, DTLZ3, DTLZ4, DTLZ5, DTLZ6** e **DTLZ7:** 30.000 avaliações.
  - **KUR20:** 35.000 avaliações.
  - **QUAD8:** 40.000 avaliações.
  - **KUR30:** 50.000 avaliações.
  - **TSP2:** 75.000 avaliações.
  - **QUAD16:** 80.000 avaliações.
  - **TSP3:** 100.000 avaliações.
- Probabilidade de cruzamento (por par): $0,90$.
- Cruzamento $\eta : 20$.
- Probabilidade mutação (individual): $1/n$.
- Mutação $\eta : 20$.
- Tamanho máximo do arquivo:
  - **QUAD2** e **TSP2**: 50 indivíduos.
  - **KUR10, KUR20, KUR30, QUAD4, FONSECA, VIENNET, TSP3, DTLZ1, DTLZ2, DTLZ3, DTLZ4, DTLZ5, DTLZ6** e **DTLZ7:** 100 indivíduos.
  - **QUAD8:** 200 indivíduos.
  - **QUAD16:** 400 indivíduos.
- Número de execuções dos algoritmos: 100 por instância.
- Número de permutações testes: 1000 por instância.

A métrica ΔM depende dos pontos extremos da fronteira de Pareto verdadeira. Os pontos exatos das funções QUAD, KUR, FONSECA, VIENNET e DTLZ, são conhecidos na literatura. Para os problemas TSP esses pontos não são conhecidos, para superar essa limitação, a melhor solução encontrada para cada objetivo em todas as execuções do algoritmo são usadas como os pontos extremos.

As Figuras 3.8 até 3.25 apresentam as comparações entre os algoritmos NSGA-II e SCMGA utilizando as métricas CM, ΔM e ISC. As Figuras 3.8, 3.9, e 3.10 apresentam os resultados para as instâncias KUR10, KUR20 e KUR30, respectivamente. As Figuras 3.11, 3.12, 3.13 e 3.14 os resultados observados para os problemas QUAD2, QUAD4, QUAD8 e QUAD16. A Figura 3.15 o resultado observado para o problema FONSECA e, a Figura 3.16 para o problema VIENNET. As Figuras 3.17 até 3.23 apresentam os resultados obtidos para as sete instâncias DTLZ. Finalmente, as Figuras 3.24 e 3.25 os resultados encontrados para as instâncias TSP2 e TSP3.

A informação visual obtida pela comparação feita a partir das métricas mostram que, valores mais baixos das métricas CM e ΔM indicam melhor do desempenho do algoritmo em relação a cobertura e espalhamento, respectivamente, das soluções pelo

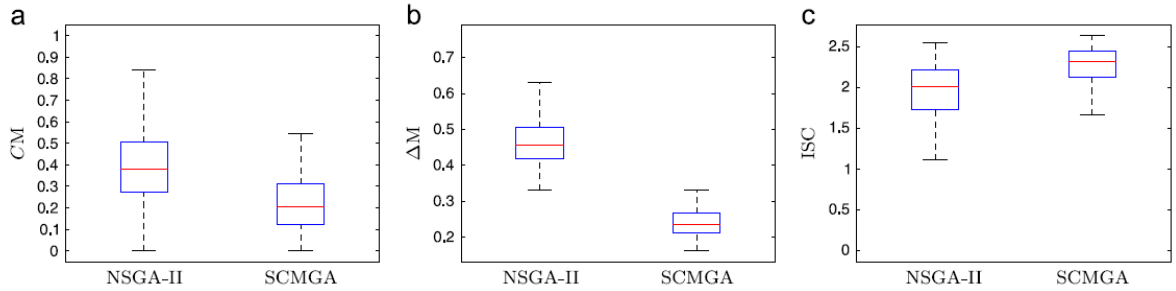


Figura 3.8: Resultados da função **KUR10**. (a) CM, (b) ΔM, (c) ISC,

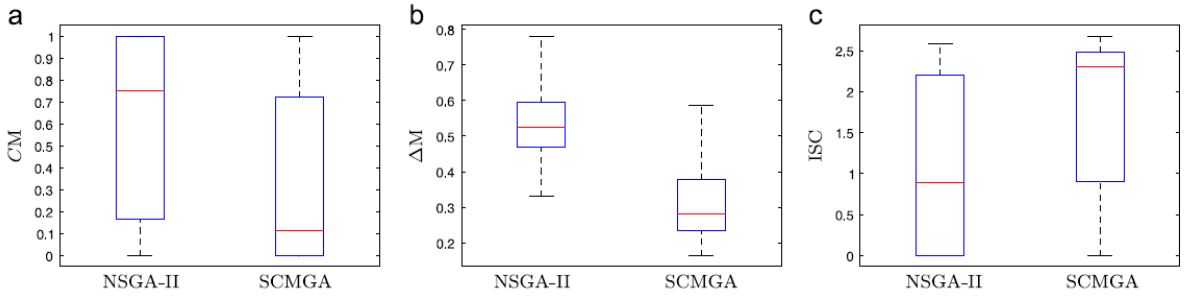


Figura 3.9: Resultados da função **KUR20**. (a) CM, (b) ΔM, (c) ISC,

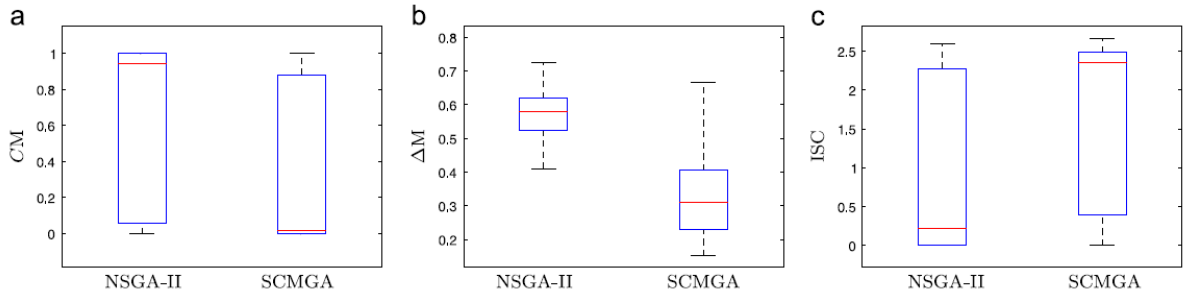


Figura 3.10: Resultados da função **KUR30**. (a) CM, (b) ΔM, (c) ISC,

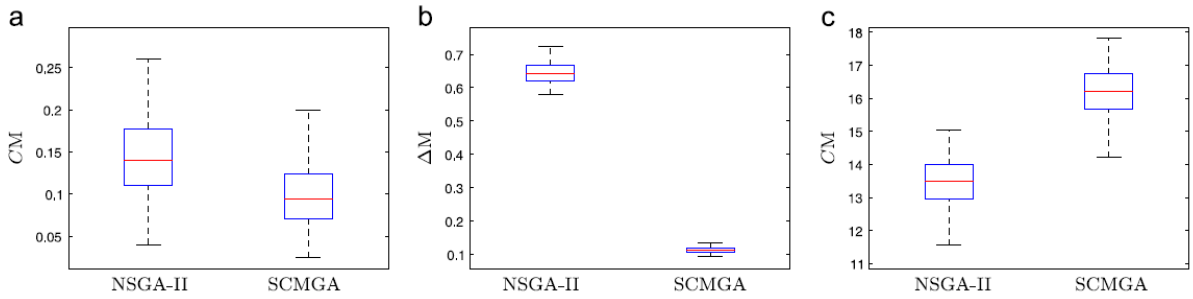


Figura 3.11: Resultados da função **QUAD2**. (a) CM, (b) ΔM, (c) ISC,

conjunto de Pareto. Enquanto valores mais altos da métrica ISC indicam que o conjunto de Pareto obtido pelo algoritmo possui melhor distribuição.

A partir da análise visual dos *boxplots* é possível notar que, o SCMGA estocasticamente domina o algoritmo NSGA-II na maioria dos problemas e para todas as métricas de desempenho (os quantis da SCMGA são melhores do que os quantis correspondentes do NSGA-II). A única exceção é a métrica (CM) aplicada a instância DTLZ7, em que

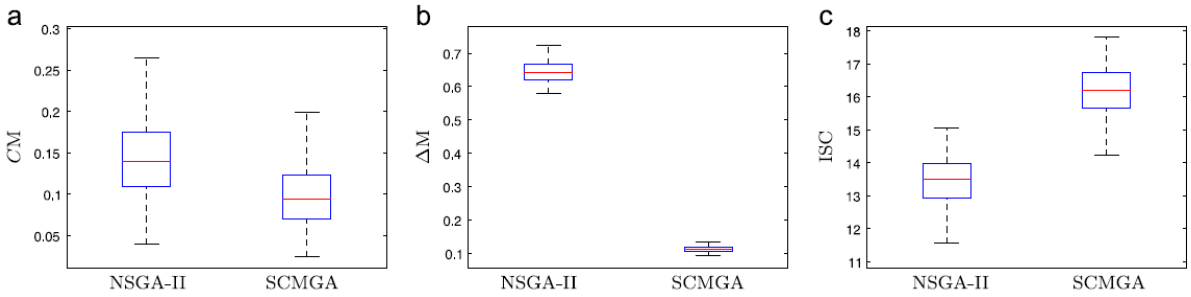


Figura 3.12: Resultados da função **QUAD4**. (a) CM, (b) ΔM, (c) ISC,

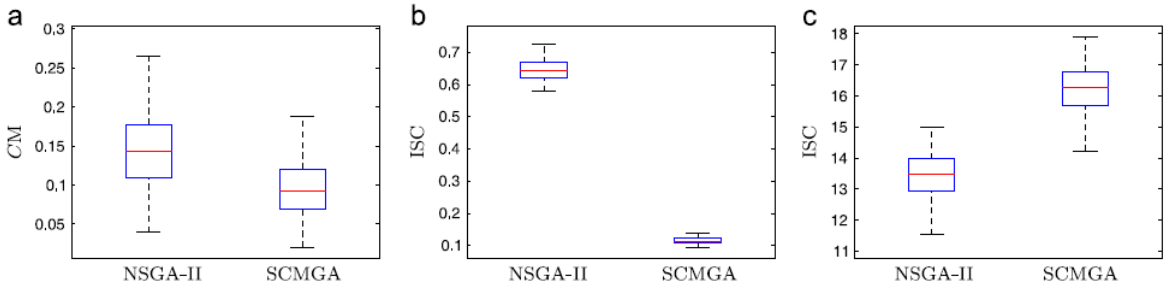


Figura 3.13: Resultados da função **QUAD8**. (a) CM, (b) ΔM, (c) ISC,

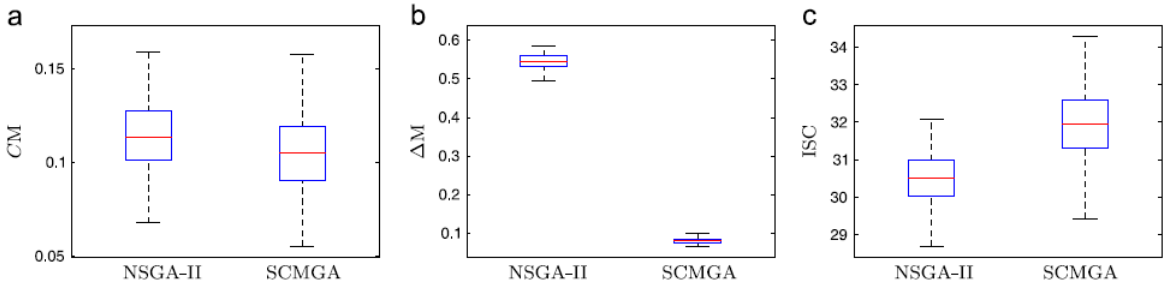


Figura 3.14: Resultados da função **QUAD16**. (a) CM, (b) ΔM, (c) ISC,

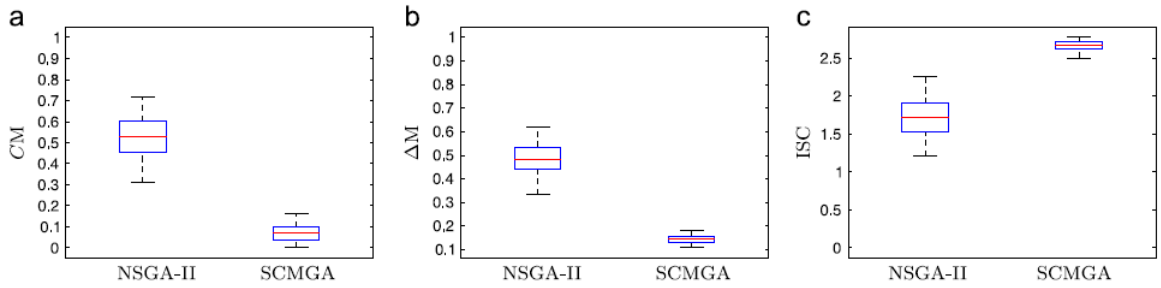


Figura 3.15: Resultados da função **FONSECA**. (a) CM, (b) ΔM, (c) ISC,

o NSGA-II obteve resultados melhores. No entanto, nota-se que, o algoritmo proposto superou o original NSGA-II para as métricas ΔM e ISC neste problema.

As análises dos problema KUR sugerem que o SCMGA é menos afetado pelo aumento na dimensão do problema do que o algoritmo NSGA- II, uma vez que, a diferença entre esses algoritmos aumenta na medida em que aumenta a dimensão do problema. Em todos os problemas QUAD, onde o espaço das variáveis de decisão é fixo

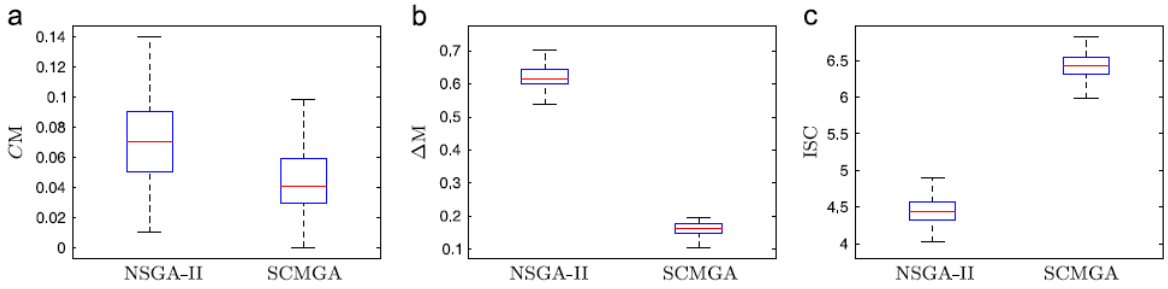


Figura 3.16: Resultados da função **VIENNET**. (a) CM, (b) ΔM, (c) ISC,

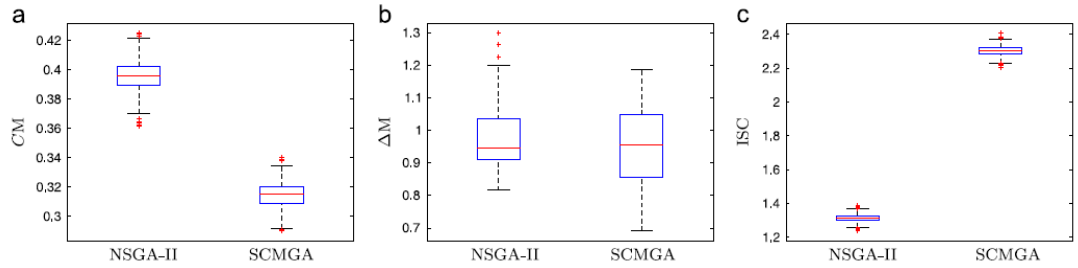


Figura 3.17: Resultados da função **DTLZ1**. (a) CM, (b) ΔM, (c) ISC,

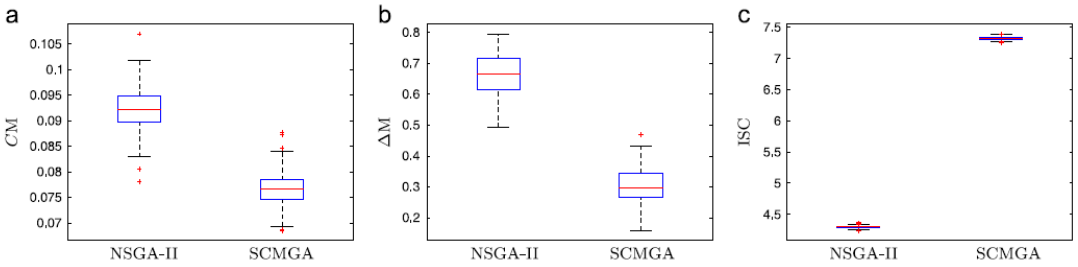


Figura 3.18: Resultados da função **DTLZ2**. (a) CM, (b) ΔM, (c) ISC,

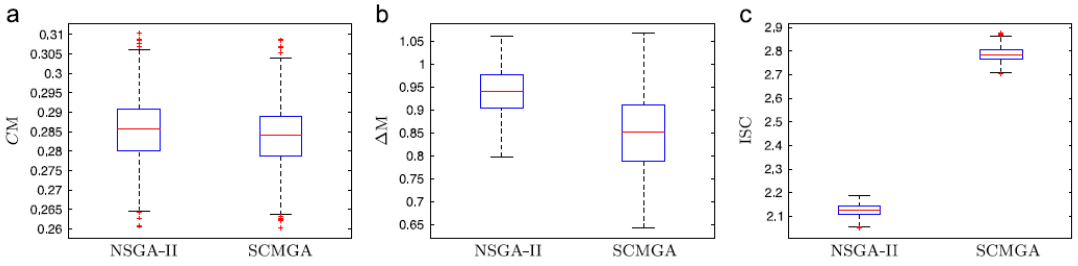


Figura 3.19: Resultados da função **DTLZ3**. (a) CM, (b) ΔM, (c) ISC,

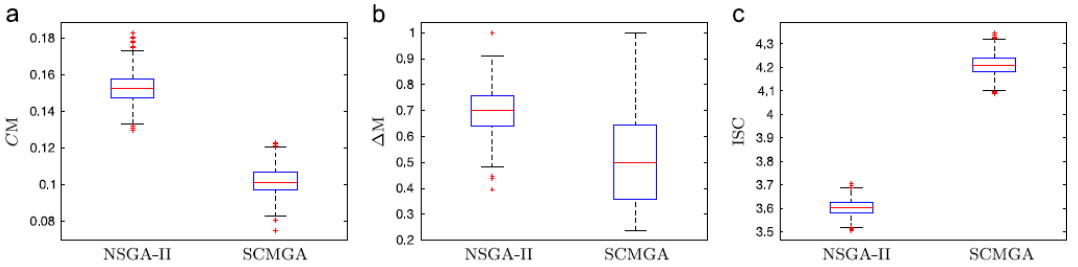


Figura 3.20: Resultados da função **DTLZ4**. (a) CM, (b) ΔM, (c) ISC,

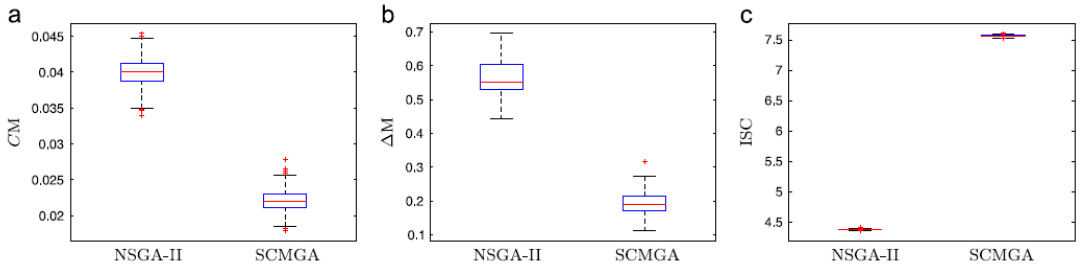


Figura 3.21: Resultados da função **DTLZ5**. (a) CM, (b) ΔM, (c) ISC,

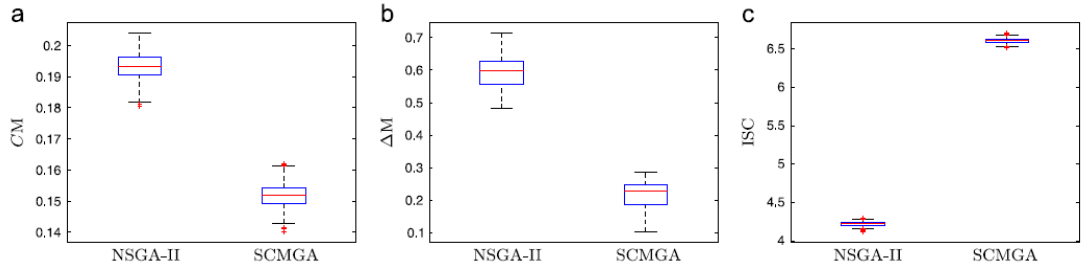


Figura 3.22: Resultados da função **DTLZ6**. (a) CM, (b) ΔM, (c) ISC,

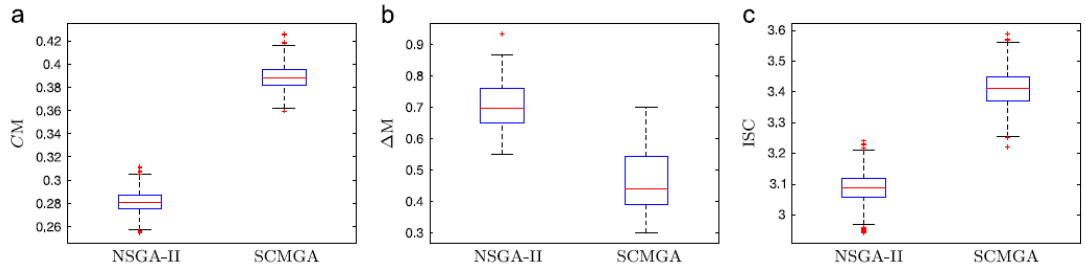


Figura 3.23: Resultados da função **DTLZ7**. (a) CM, (b) ΔM, (c) ISC,

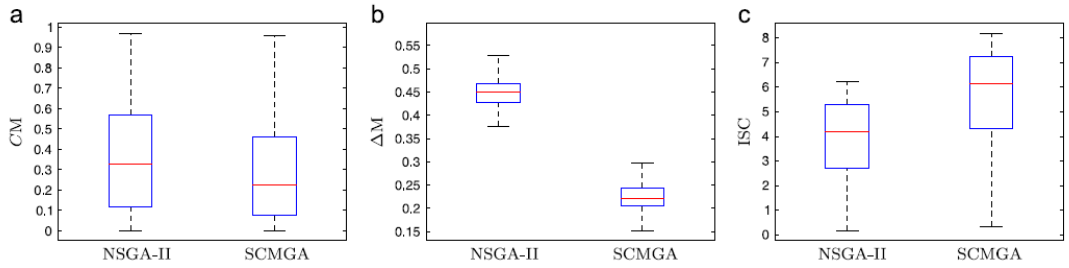


Figura 3.24: Resultados da função **TSP2**. (a) CM, (b) ΔM, (c) ISC,

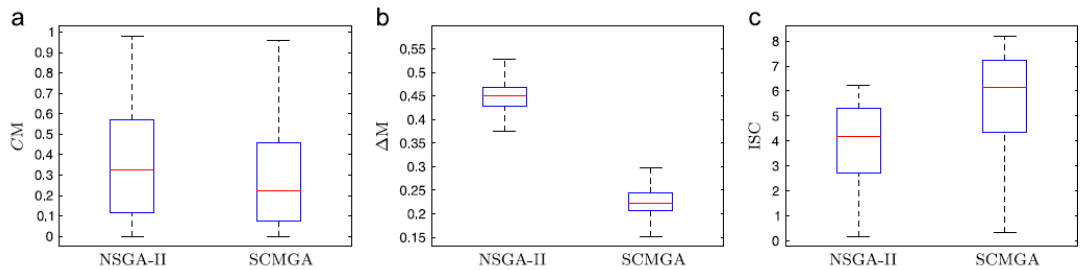


Figura 3.25: Resultados da função **TSP3**. (a) CM, (b) ΔM, (c) ISC,

em 20 e, apenas a dimensão do espaço de objetivos varia, os resultados são semelhantes

em todos os casos considerados. Sugerindo que, o número de funções objetivo não afetam a eficiência relativa observada entre os algoritmos.

Em uma análise ampla, nota-se que o algoritmo proposto pode obter uma quantidade maior de soluções eficientes no conjunto de Pareto do que o NSGA-II. Além disso, as soluções obtidas pelo SCMGA são, na maior parte das vezes, melhor espaçadas no domínio dos objetivos, do que as soluções do NSGA-II. Esta característica é altamente desejável.

Os tempos computacionais em segundos despendidos pelos algoritmos estão apresentados na Tabela 3.1. Nesta tabela, mostra-se a média e o desvio-padrão observado entre as execuções, em valores por execução. É possível notar que, a diferença entre os tempos de processamento dos algoritmos não é significante, em qualquer um dos casos considerados. No pior dos casos, o algoritmo proposto é mais lento que o NSGA-II em 10% e mais rápido em outros casos. Isso significa que, a sobrecarga causada pelo emprego dos operadores de auto-controle não é significante quando comparado com as outras operações realizadas pelo algoritmo. Além disso, é importante destacar que essa diferença entre os tempos, seria mais evidente se considerados problemas onde o tempo necessário para avaliar a função é alto.

Tabela 3.1: Os tempos computacionais por execução. $\bar{x}$ indica o tempo computacional médio e s o desvio padrão dos tempos computacionais em 100 execuções do algoritmo.

	NSGA-II		SCMGA		
Instância	$x = base$	$s = base$	$x = base$	$s = base$	*Base*
KUR10	1,00	0,01	1,06	0,02	11,90
KUR20	1,00	0,01	1,06	0,01	19,95
KUR30	1,00	0,01	1,06	0,00	28,06
QUAD2	1,06	0,08	1,00	0,05	43,51
QUAD4	1,00	0,05	1,01	0,08	54,44
QUAD8	1,00	0,05	1,00	0,04	91,56
QUAD16	1,11	0,01	1,00	0,01	287,65
FONSECA	1,00	0,12	1,07	0,16	36,56
VIENNET	1,00	0,00	1,00	0,00	17,72
DTLZ1	1,00	0,02	1,06	0,01	18,83
DTLZ2	1,00	0,01	1,03	0,00	18,45
DTLZ3	1,00	0,01	1,07	0,01	18,99
DTLZ4	1,00	0,00	1,03	0,00	18,42
DTLZ5	1,00	0,01	1,03	0,00	18,55
DTLZ6	1,00	0,00	1,03	0,00	18,41
DTLZ7	1,00	0,00	1,04	0,00	18,16
TSP2	1,01	0,01	1,00	0,03	56,53
TSP3	1,00	0,01	1,03	0,02	68,37

Com base nos resultados obtidos, temos evidências que o SCMGA supera o algoritmo NSGA-II por ser capaz de obter um conjunto mais representativo de Pareto,

e melhor espaçadas no domínio das funções objetivo, para a maioria dos problemas testados, com um acréscimo de custo computacional inferior a 10%. Maiores detalhes sobre está análise podem ser obtidos em [7], artigo originado deste trabalho.

3.5 SCMGA Aplicado ao Problema de Extrusão de Polímeros

O processo de extrusão de polímeros é uma tecnologia de processamento que permite a produção contínua de artefatos plásticos, como tubos, canos, chapas, fibras, entre outros. Neste processo, a matéria-prima (mistura de resinas e aditivos, por exemplo) é amolecida e expulsa, através de uma matriz instalada no equipamento, que dá a forma final ao produto a ser obtido. O resultado é um produto que conserva a sua forma ao longo de sua extensão. O processo de extrusão é realizado em um equipamento conhecido como extrusora.

Na Figura 3.26 mostra a vista do corte de uma extrusora de parafuso (rosca) único, usada no processamento de materiais poliméricos. Este tipo de extrusora é o mais utilizado pela indústria, por permitir gerar produtos homogêneos, com baixo custo e de boa qualidade. Os componentes básicos de uma extrusora convencional de um parafuso único são:

(i) um funil de alimentação que recebe os polímeros;
(ii) um tambor aquecido;
(iii) um parafuso de Arquimedes que gira no interior do cilindro com uma velocidade (N);
(iv) um barril de aquecimento, com a temperatura definida pelo operador;
(v) uma matriz extrusora.

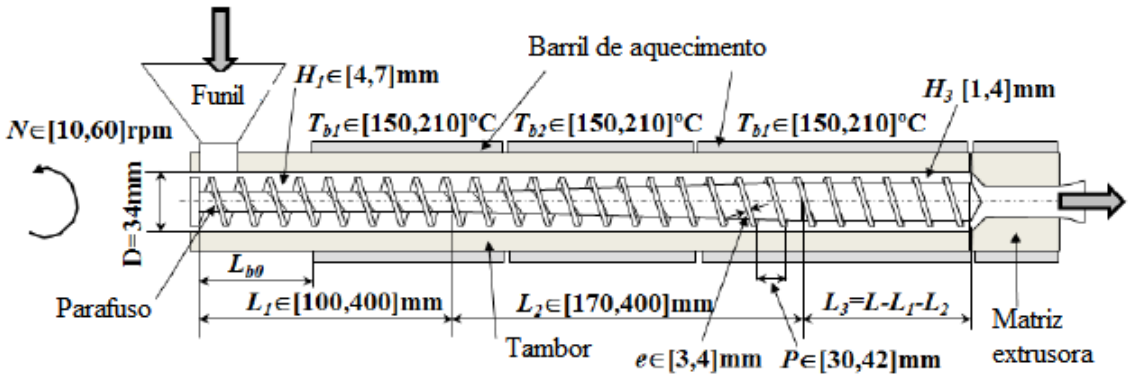


Figura 3.26: Vista do corte de uma extrusora de parafuso único e seus componentes básicos.

O desempenho do processo de extrusão depende de três parâmetros diferentes: as propriedades do polímero, a geometria do processo de extrusão e as condições de operação. As diferentes propriedades dos polímeros envolvem a energia térmica (como, por exemplo, o coeficiente do calor de condução e a temperatura de fusão); as características físicas (como, por exemplo, coeficientes de atrito e densidade) e reologia (medida da resistência do fluxo do polímero). Como normalmente o processo envolve um único polímero, essas propriedades são constantes. Neste caso, uma extrusora mais simples, como a apresentada na Figura 3.26, é usada.

Na Figura 3.26, a extrusora possui um único parafuso e três zonas geométricas: seção de alimentação, seção de compressão e seção de dosagem. A seção de alimentação tem a função de pré-aquecer o polímero e transportá-lo para as próximas seções. A altura deste canal é constante, com valor igual a H_1 e o comprimento é adequado de forma a garantir uma taxa de alimentação para as seções seguintes. A seção de compressão apresenta altura do canal que decresce uniformemente, com valor variando entre H_1 e H_2. Esta seção é responsável por variar a massa específica do material de acordo com a mudança de fase que ele sofre. Por fim, a seção de dosagem, que tem por finalidade homogeneizar o material plastificado, possui a altura do canal constante, porém menor que a do canal de alimentação (H_3). O parafuso também é caracterizado, na figura, pelo passo (P) e a largura do espaço (e). As condições operacionais são as variáveis que são controladas pelo operador da máquina, que neste exemplo, são as variáveis: velocidade de rotação do parafuso (N) e temperaturas dos barris (T_{b1}, T_{b2} e T_{b3}).

Assim, temos que as principais funções da extrusora estão relacionadas ao transporte do material sólido do funil até a zona do barril aquecido para a fundição deste polímero e a homogeneização e mistura dos novos aditivos, para criar a pressão necessária que permite ao polímero passar pela matriz. A matriz é a etapa final do processo.

O desempenho do dispositivo de extrusão, como apresentado anteriormente, depende das propriedades do polímero, da geometria do sistema e das condições operacionais. Assim, um ambiente termo-mecânico é desenvolvido de forma que o polímero passe por diferentes estados térmicos e físicos, como pode ser visto na Figura 3.27. Na figura: (i) neste momento, os sólidos são transportados para dentro do funil, onde, por ação da gravidade, eles são levados para o interior do cilindro; (ii) pela ação da rotação do parafuso e, devido ao atrito entre o parafuso e as paredes do cilindro, o polímero sólido é pressurizado e formada uma massa plástica (esta ação estende-se do funil de alimentação até a posição no parafuso onde as partículas sólidas se plastificam); (iii) o calor gerado pelo atrito e o calor conduzido forma um filme fluido; (iv) o mecanismo de fusão derrete o material, e, finalmente o polímero é pressurizado e transportado para a matriz.

A modelagem deste processo envolve a ligação de todas as zonas funcionais, tendo em conta as restrições impostas. A resolução do modelo é baseada em diferenças finitas para as derivadas de primeira ordem e no método de Crank-Nicolson (método

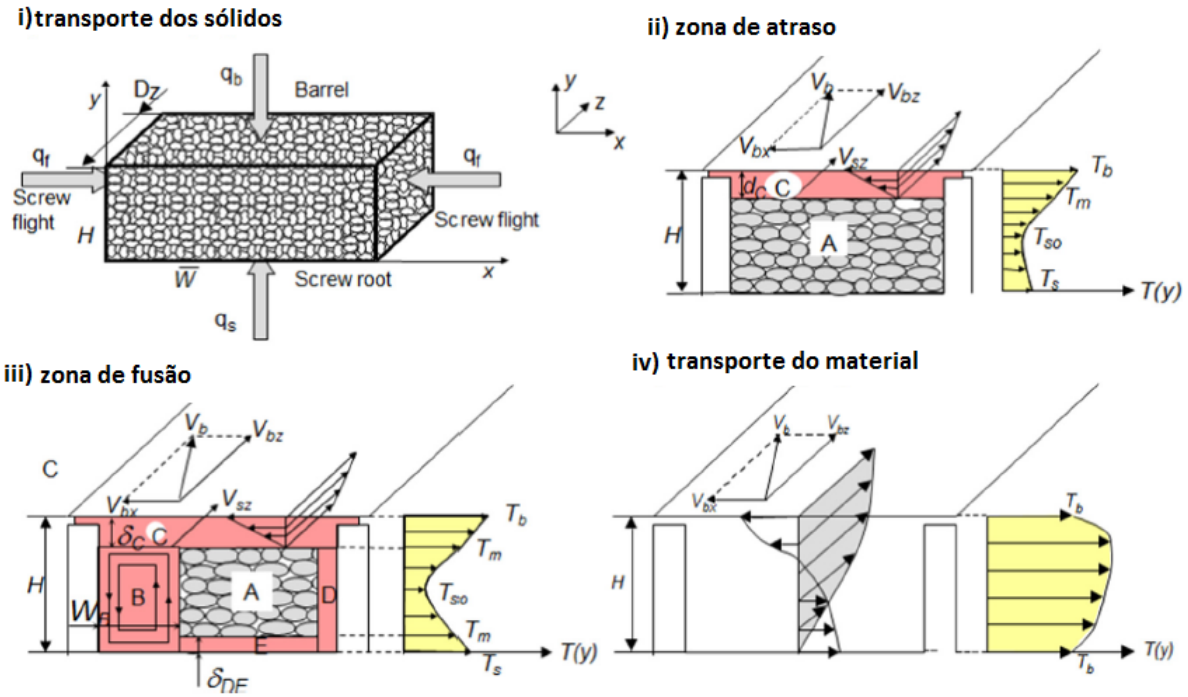


Figura 3.27: Estados térmicos e físicos que envolve os passos do processo funcional termo-mecânico. (i) transporte dos sólidos para dentro do funil e interior do cilindro; (ii) pressurização do polímero sólido para formar uma massa plástica; (iii) formação de um filme fluido e (iv) pressurização da massa plástica e transporte para a matriz da extrusora.

trapezoidal) para as derivadas de segunda ordem. O sistema de equações resultante, considerando as equações de análise de movimento e de energia, é resolvidos pelo método de Eliminação de Gauss com pivotação. A estabilidade das soluções foi assegurada levando em conta a convergência do campo de velocidade desenvolvido, simultaneamente, a área de atuação com a convergência da temperatura. Mais detalhes da implementação e modelagem podem ser encontradas no trabalho [16].

A complexidade do processo em cada zona funcional, Figura 3.27, e a existência de várias zonas são consideradas e resolvidas ao mesmo tempo. Neste sentido, a convergência do algoritmo deve ser assegurada para as diferentes zonas tratadas.

O planejamento desenvolvido para resolver o problema de encontrar as melhores configurações de um sistema de extrusão de polímeros deve ser capaz de calcular algumas características de desempenho importantes no processo de extrusão, como: as propriedades do polímero, a geometria do sistema e as condições operacionais. É importante destacar que, neste processo, a solução pode não ser única, uma vez que diferentes combinações da geometria do parafuso e/ou das condições de operação podem produzir o mesmo desempenho.

Uma estratégia é utilizar um algoritmo de otimização multiobjetivo considerando cada uma das características relevantes como uma função objetivo. Neste trabalho, os objetivos considerados na resolução do problema de otimização multiobjetivo aplicado

ao problema de extrusão de polímeros são:

Q**:** desempenho do sistema, que é caracterizado pela saída de massa na máquina (kg/h).
L_{melt}**:** o comprimento do parafuso necessário para fundir o polímero (m).
T_{melt}**:** a temperatura média de fusão do polímero na saída da matriz (C).
$Power$**:** o consumo de energia necessário para girar o parafuso (W).
$WATS$**:** a medida da capacidade de mensurar a média de deformação ($WATS$).

Uma descrição detalhada acerca do processo de extrusão de polímeros, envolvendo equipamentos com várias configurações diferentes, pode ser encontrada em [68].

3.5.1 Resultados

O algoritmo SCMGA foi utilizado para identificar configurações viáveis para um sistema de extrusão de polímeros. Para analisar os resultados obtidos pelo algoritmo, o conjunto de soluções obtidas são comparadas com o conjunto de soluções geradas pelos algoritmos NSGA-II e o RPSGA - *Reduced Pareto Set Genetic Algorithm* [31, 16]. O algoritmo RPSGA foi escolhido para comparação por ter sido usado em estudos anteriores para resolver o mesmo problema multiobjetivo. As soluções obtidas por ele são usadas como referência para a avaliação dos algoritmos NSGA-II e SCMGA.

As simulações foram realizadas em um único núcleo de um iMac 27 (modelo 2011), com processador Core i7 3.4GHz e 16GB de RAM, usando Matlab 2010. O critério de parada utilizado foi o número fixo de 20.000 avaliações de função para cada algoritmo.

Foram realizados três estudos com otimizações diferentes, como é apresentado na Tabela 3.2. No primeiro estudo, apenas as condições operacionais são consideradas como variáveis de decisão, ou seja, as variáveis N, T_{b1}, T_{b2} e T_{b3} (casos 1-4). No segundo conjunto, são considerados apenas a geometria de parafuso, ou seja, as variáveis L_1, L_2, H_1, H_3, P e e (casos 5-7). Finalmente, no terceiro estudo (casos 8-10) onde todas as variáveis de decisão foram consideradas.

Tabela 3.2: Estudos com as otimizações realizadas.

Casos	Tipo de otimização	Variáveis de Decisão	Objetivos
1	Condições Operacionais	N, T_{b1}, T_{b2}, T_{b3}	Q, L_{melt}
2	Condições Operacionais	N, T_{b1}, T_{b2}, T_{b3}	Q, T_{melt}
3	Condições Operacionais	N, T_{b1}, T_{b2}, T_{b3}	Q, $WATS$
4	Condições Operacionais	N, T_{b1}, T_{b2}, T_{b3}	Todos
5	Geometria do Parafuso	L_1, L_2, H_1, H_3, P, e	Q, $Power$
6	Geometria do Parafuso	L_1, L_2, H_1, H_3, P, e	Q, $WATS$
7	Geometria do Parafuso	L_1, L_2, H_1, H_3, P, e	Todos
8	Ambos	Todos	Q, $Power$
9	Ambos	Todos	Q, $WATS$
10	Ambos	Todos	Todos

Devido à complexidade do processo e a fim de verificar melhor as interações existentes entre as variáveis no procedimento de otimização, estudos preliminares foram realizados considerando apenas dois objetivos em cada execução. Neste primeiro estudo, o desempenho do sistema caracterizado pela saída de massa (Q), foi considerado o objetivo mais importante e que deve ser utilizado em todas as execuções, a fim de aumentar os lucros. A Tabela 3.3 enumera os objetivos utilizados, a finalidade de otimização e os intervalos de variações.

Tabela 3.3: Objetivos, propósito da otimização e os intervalos das variações

Objetivo	**Tipo de Otimização**	**Intervalos das Variações**
Saída (kg/h)	Maximização	$[1, \quad 20]$
Comprimento do Parafuso (m)	Minimização	$[0,2 \quad 0,9]$
Temperatura de Fusão (C)	Minimização	$[150, \quad 210]$
Consumo de Energia (W)	Minimização	$[0, \quad 9200]$
WATS	Maximização	$[0, \quad 1300]$

As populações finais obtidas em todas as execuções dos algoritmos SCMGA, NSGA-II e RPSGA são comparadas utilizando uma métrica baseado na noção de atingir um objetivo [30]. Esta métrica atribui a cada vetor de objetivos z uma probabilidade de que este ponto seja encontrado em uma única execução. Não é possível saber qual é a solução ótima, mas este valor pode ser estimado baseado em aproximações do conjunto de amostras. Assim, diferentes execuções são realizadas, a fim de se obter aproximações para o conjunto de amostras. De acordo com [55], as diferenças entre dois algoritmos podem ser observadas por meio da representação gráfica do conjunto final de soluções no espaço dos objetivos, onde a diferença entre os resultados empíricos das funções entre os algoritmos são significativas.

A comparação do desempenho dos algoritmos no caso 1 é apresentada na Figura 3.28. Para esta análise foram realizados dez execuções diferentes de cada algoritmo. Na figura, (a) ilustra a comparação dos algoritmos RPSGA e NSGA-II e (b) a comparação dos algoritmos NSGA-II e SCMGA. Neste caso particular, é possível notar em (a) que o NSGA-II é ligeiramente melhor que o RPSGA por encontrar um conjunto de soluções mais representativo. Em (b) quando a comparação é feita entre os algoritmos NSGA-II e SCMGA é possível notar que ambos os algoritmos são capazes de gerar um conjunto representativo de soluções, porém, o SCMGA se destaca em relação ao NSGA-II pelo fato do conjunto de soluções apresentar uma melhor distribuição no espaço dos objetivos e pela probabilidade dessas soluções serem obtidas em menos 80% das execuções do algoritmo (indicado na figura pelos pontos mais escuros).

As Figuras 3.29, 3.30, 3.31 apresentam os resultados obtidos pelos algoritmos RPSGA, NSGA-II e SCMGA para os casos restantes (casos 2-8).

Os resultados da otimização das condições operacionais, casos 1, 2 e 3 são apresentados na Figura 3.29. Nestes testes, visualmente não é possível ver muita diferença

entre a fronteira de Pareto gerada pelos algoritmos RPSGA e NSGA-II, apresentadas nos quadros (a), (c) e (e) para cada caso. Nos quadros (b), (d) e (f) que comparam a fronteira de Pareto dos algoritmos RPSGA e SCMGA, para cada caso, é possível notar que o conjunto resultante do SCMGA é mais completo que o conjunto resultante do RPSGA. No quadro (b), que apresenta a otimização do caso 1, nos extremos das fronteiras o SCMGA consegue obter soluções que dominam as soluções obtidas pelo RPSGA.

Os resultados para o estudo de caso 4, que otimiza as condições operacionais e considera todos os cinco objetivos são considerados, são apresentados na Figura 3.30. Nesta figura os resultados obtidos pelos algoritmos RPSGA e SCMGA são apresentados por projeções considerando dois a dois dos objetivos. O objetivo em comum em todos os quadros é o desempenho do sistema. Para cada par de objetivo, é possível notar que o SCMGA encontra um conjunto de soluções mais representativo (maior quantidade e espalhamento) no espaço dos objetivos que o algoritmo RPSGA. Os resultados para os casos 7 e 10, que também consideram cinco objetivos, são semelhantes aos resultados encontrados no caso 4.

Os casos 6 (que otimiza as condições operacionais) e 9 (otimiza as condições operacionais e a geometria do parafuso) são problemas cujos objetivos devem ser maximizados. Os resultados para estes casos são apresentados na Figura 3.31, onde o quadro (a) apresenta o resultado do caso 6 e o quadro (b) do caso 9. Em ambos quadros, não é possível notar muita diferença entre as soluções obtidas pelos algoritmos RPSGA e SCMGA. Isto se deve ao fato de que agora, no processo de extrusão, é permitida uma variação da geometria do sistema. Este efeito da variação da geometria é mais evidente, quando estes resultados são comparados com os resultados do caso 3 representados na Figura 3.29 pelos quadros (e) e (f).

Na otimização multiobjetivo, o objetivo principal é encontrar um conjunto de boas soluções (estimativa do conjunto Pareto-Ótimo) que otimize ao mesmo tempo todas as funções objetivo do problema. Esse conjunto de soluções são usadas por um tomador de decisões para auxiliar na escolha das melhores configurações dos parâmetros que devem ser empregados na fabricação de um produto específico. Este conjunto de soluções são abastecidos de informações a respeito dos limites do desempenho das tarefas na fábrica, incluindo as informações sobre os *trade-off* envolvidos na escolha entre as diferentes soluções do conjunto de estimativas de Pareto-Ótimo.

Neste trabalho, as funções conflitantes estão relacionadas ao consumo de energia, capacidade de produção e qualidade do produto. Uma boa descrição do conjunto de Pareto (uma boa amostragem que cobre de forma uniforme toda a extensão do conjunto) é importante a fim de fornecer um conjunto completo com várias alternativas possíveis para o tomador de decisão.

Neste contexto, a metodologia proposta (algoritmo SCMGA) mostrou ser uma ferramenta eficaz para encontrar boas descrições do conjunto de estimativas do conjunto

de Pareto (maior cobertura e espalhamentos), que os outros algoritmos, principalmente quando mais de dois objetivos são considerados. Levando a evidência de que a metodologia proposta é adequada para tratar problemas de aplicações similares ao problema estudado neste capítulo.

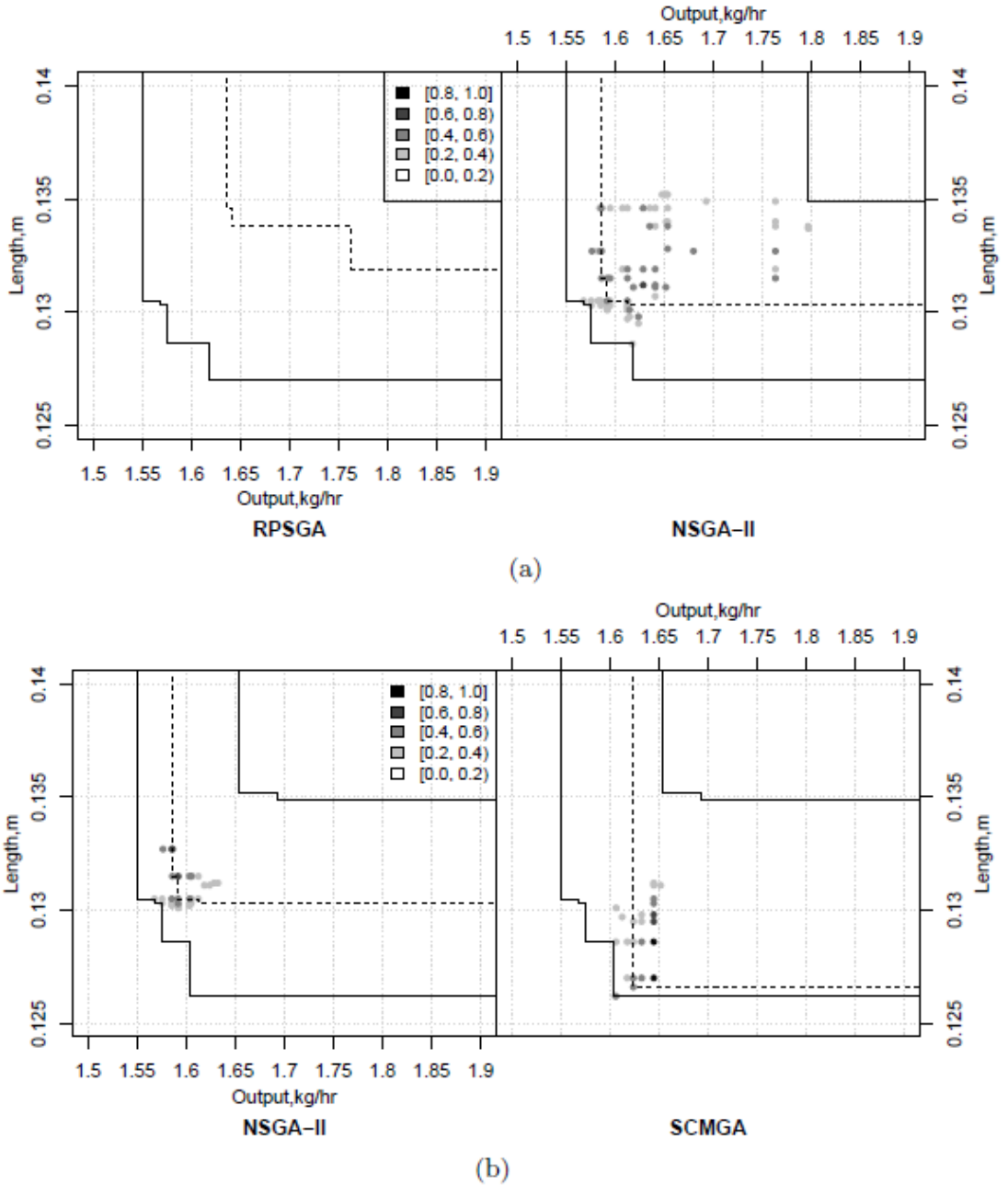


Figura 3.28: Resultados empíricos dos algoritmos para a otimização do Caso 1: (a) comparação dos algoritmos RPSGA e NSGA-II e (b) comparação dos algoritmos NSGA-II e SCMGA.

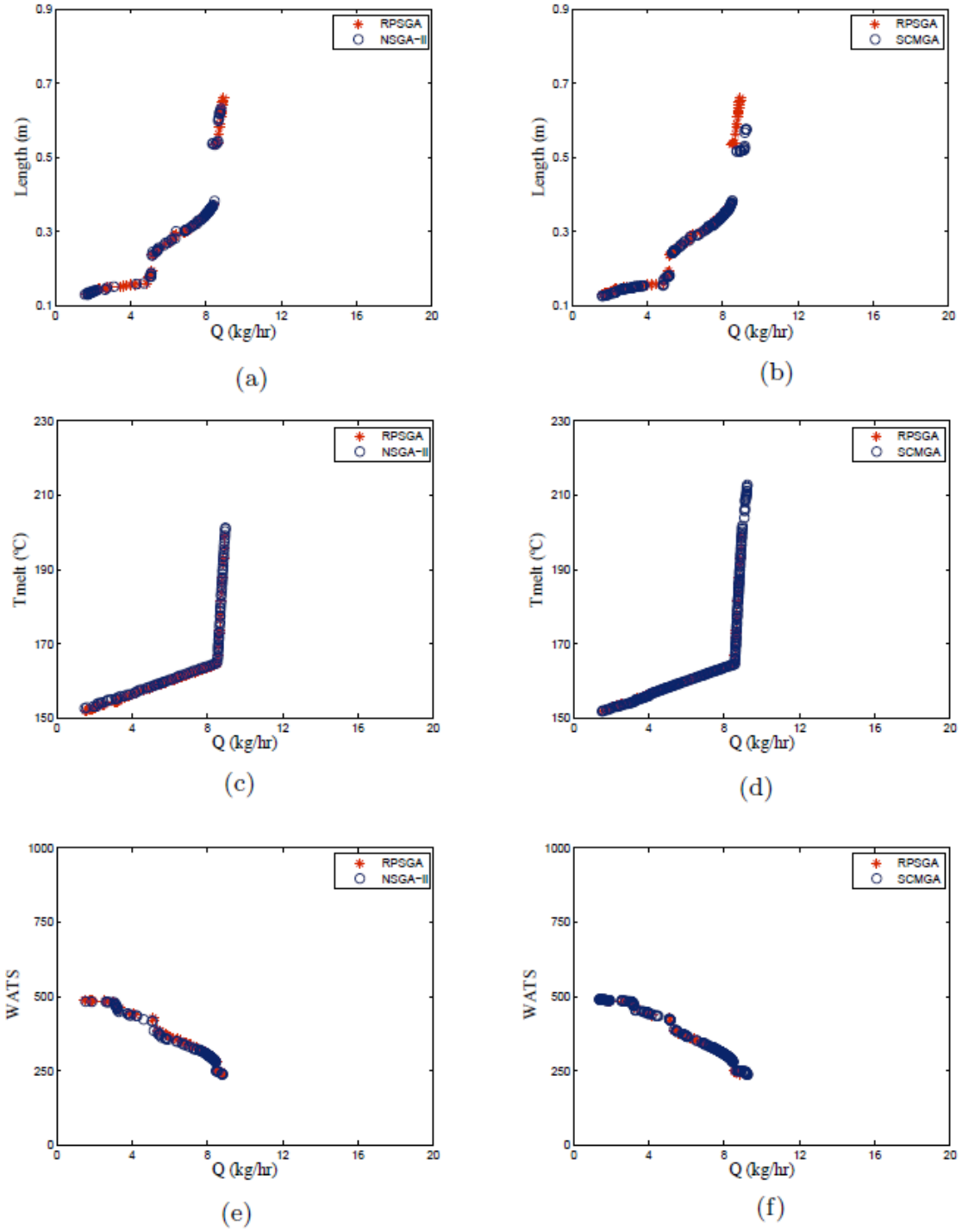


Figura 3.29: Os resultados obtidos durante a otimização das condições operacionais, casos 1, 2 e 3. (a) Caso 1: NSGA-II × RPSGA. (b) Caso 1: RPSGA × SCMGA. (c) Caso 2: NSGA-I × RPSGA. (d) Caso 2: RPSGA × SCMGA. (e) Caso 3: NSGA-II × RPSGA. (f) Caso 3: RPSGA × SCMGA.

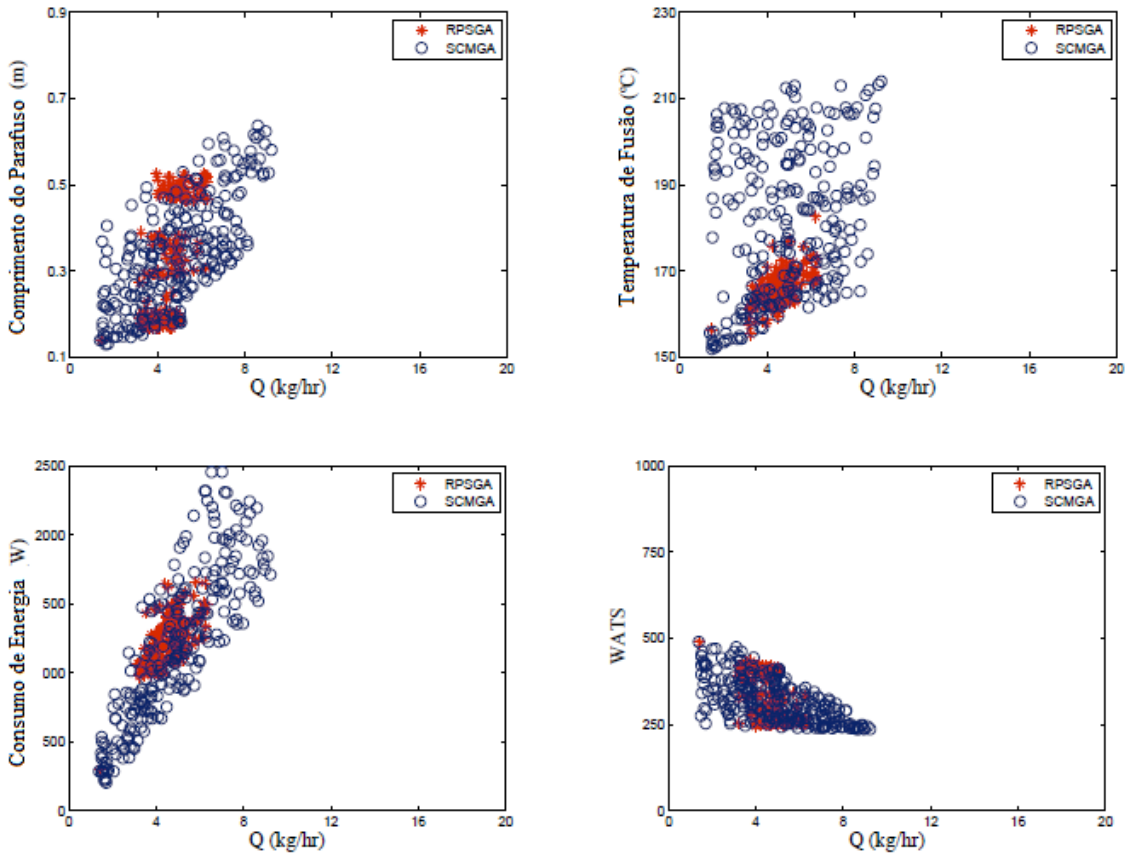


Figura 3.30: Os resultados obtidos pelos algoritmos RPSGA e SCMGA durante a otimização das condições operacionais no caso 4. A figura apresenta o conjunto de soluções considerando dois a dois dos objetivos.

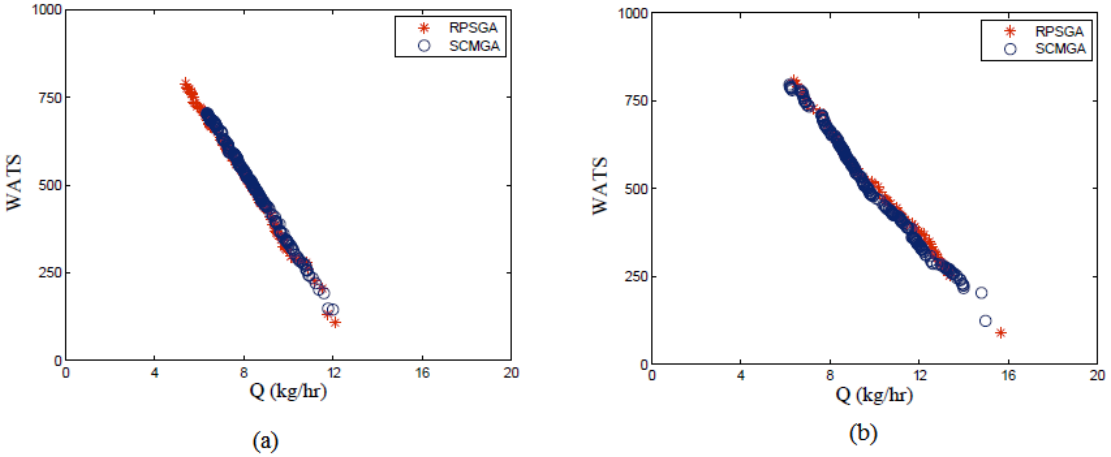


Figura 3.31: Resultados obtidos durante a otimização: (a) Caso 6: geometria do parafuso. (b) Caso 9: geometria do parafuso e condições operacionais.

Capítulo 4

Operadores Adaptativos em Meta-heurísticas de Busca Local

Este capítulo é proposto um novo operador baseado em uma estratégia de memória para meta-heurísticas de busca local. Este operador tem por finalidade melhorar a exploração do espaço de busca de soluções e evitar reavaliações de soluções replicadas pelo algoritmo. Assim, de forma sucinta, são apresentadas a meta-heurística VNS e suas versões adaptativas, que fazem uso de uma estratégia de memória e um operador de vizinhança adaptativo.

Uma análise da estrutura de memória é realizada e por fim, a metodologia proposta é aplicada ao problema *common due-date scheduling*. Este problema trata de um caso específico de programação de tarefas, em uma única máquina, com data de entrega comum.

4.1 Introdução

Os problemas de otimização combinatória em geral são caracterizados por estudarem arranjos, ordenações e permutações de um conjunto de elementos discretos, de forma a encontrar, entre todos os possíveis subconjuntos, aquele que retorna o menor custo.

Em problemas com estrutura permutacional, uma solução é representada por uma permutação de n elementos, onde n é o número de variáveis do problema. No caso de problemas NP-difíceis, encontrar a melhor solução consiste em examinar todas as $n!$ possíveis soluções, ou parte significativa delas, o que implica em complexidade computacional fatorial.

Por serem facilmente adaptáveis para diferentes problemas, cada vez mais as meta-heurísticas têm se tornado importantes ferramentas de otimização. Autores tem proposto novas estratégias e adaptações destes algoritmos, a fim de aprimorar o desempenho e expandir as possibilidades de uso dessas ferramentas.

Os métodos propostos baseados na estratégia de memória são embasados no conceito de utilizar uma estrutura de dados, como uma memória de longo prazo, no algoritmo. A ideia de memória, fundamentada nos trabalhos [13, 92, 8], é usada para armazenar todas as soluções encontradas pelo algoritmo e evitar reavaliações de uma mesma solução. O objetivo desta técnica é reduzir o esforço computacional, evitando várias reavaliações desnecessárias de uma mesma solução.

Neste capítulo é introduzido um novo operador de vizinhança adaptativo aplicado a uma meta-heurística de busca local. Este operador, utilizado em conjunto com uma estrutura de memória, permite realizar uma nova busca local em vizinhanças que ainda não foram exploradas pelo algoritmo, além de evitar reavaliações de soluções.

O operador de vizinhança adaptativo faz uso de estruturas de vizinhanças adaptativas para realizar buscas por novas regiões do espaço de soluções. A vizinhança é chamada de adaptativa por apresentar os mesmos movimentos das vizinhanças de busca local, porém com a diferença dos movimentos serem aplicados por blocos ao invés de ser aplicada por elemento. Este operador é utilizado pela meta-heurística toda vez que uma solução replicada é encontrada.

Neste trabalho, como a meta-heurística será aplicada a um problema de otimização combinatória, os operadores propostos são construídos para o espaço de variáveis discretas, mais especificamente permutações.

4.2 Meta-heurísticas de Busca Local com Operadores Adaptativos

Neste trabalho é proposta uma meta-heurística de busca local com estratégia de memória e um operador de vizinhança adaptativo. Tanto o operador quanto a estratégia de memória podem ser aplicados em outras meta-heurísticas de busca local. Neste trabalho, os operadores propostos são empregados à meta-heurística de Busca em Vizinhança Variável - *Variable Neighborhood Search* (VNS).

São desenvolvidas duas novas versões para o VNS clássico: o VNS-CM, que consiste no algoritmo VNS com uma estrutura de memória, e o VNS-CMVA, que consiste no VNS com estrutura de memória e vizinhança adaptativa. Os algoritmos VNS-CM e VNS-CMVA e a versão original do VNS são descritos a seguir.

4.2.1 Busca em Vizinhança Variável - *Variable Neighborhood Search* (VNS)

A Busca em Vizinhança Variável, proposta por [62], é uma meta-heurística que explora o espaço de soluções realizando uma sequência de buscas locais, por meio de trocas sistemáticas entre as estruturas de vizinhança pré-ordenadas. O objetivo é

examinar diferentes vizinhanças para encontrar diferentes mínimos locais, para, só então, determinar qual é o mínimo global estimado. Quando o processo não encontra uma solução melhor que a solução atual, supõe-se que o algoritmo esteja preso a um mínimo local.

Diferente de outras meta-heurísticas, o VNS não segue uma mesma trajetória e realiza gradativamente a exploração por vizinhanças que estão localizadas distantes da região da solução atual.

O pseudocódigo do VNS é apresentado no Algoritmo 4.1. Os parâmetros de entrada do algoritmo são a solução inicial do problema é indicado (s_0) e a quantidade de estruturas de vizinhanças (r). A variável k indica o tipo de estrutura de vizinhança que esta sendo aplicada. Como essas estruturas já foram ordenadas, o algoritmo inicia pela primeira delas (linha 4).

Algoritmo 4.1: *VNS*

```
   Entrada: s_0, r
   Saída: s e fo_s
1  início
2      s ← s_0;
3      enquanto (Critério de parada não for satisfeito) faça
4          k ← 1;
5          enquanto (k ≤ r) faça
6              Gere um vizinho qualquer s' ∈ N^(k)(s) ;
7              s'' ← BuscaLocal(s');
8              se (f(s'') < f(s)) então
9                  s ← s'';
10                 k ← 1;
11             senão
12                 k ← k + 1;
13             fim
14         fim
15     fim
16     Retorne s;
17 fim
```

O VNS parte da solução inicial $s = s_0$, que será a base inicial da busca no espaço de soluções e, enquanto não atingir o critério de parada, segue os seguintes passos: iniciando pela estrutura de vizinhança $k = 1$, seleciona-se aleatoriamente um vizinho s' de s pertencente a esta vizinhança $N^{(k)}(s)$ (linha 6); na linha 7 uma busca local é aplicada neste vizinho e realiza-se o critério de aceitação para verificar se a nova solução é melhor que a solução corrente (linha 8). Se a solução resultante da busca local, s'', for melhor a solução atual s, atualiza-se a solução corrente (linhas 9 e 10). Quando o processo de busca local fica preso em um mínimo local, ou seja, não produz uma solução melhor, o algoritmo altera a estrutura de vizinhança por outra que ainda não tenha sido usada, linha

12. Então, o processo de realizar uma nova busca local na solução atual, considerando a nova vizinhança, é repetido até que o critério de parada seja satisfeito.

Estruturas de Vizinhança

Em um problema de otimização combinatória, uma solução é representada por uma permutação de n números inteiros. Para $n = 5$, uma solução é dada pela sequência $s = \{1,\ 2,\ 3,\ 4,\ 5\}$.

A vizinhança de uma solução s é dada pelo conjunto de soluções $N(s) \in S$, onde S é todo o espaço de busca. Cada solução vizinha de s, representada por $s' \in N(s)$, é obtida a partir de um movimento realizado nesta solução. Neste trabalho, são utilizados quatro movimentos para determinar as estruturas de vizinhança de s:

1. Troca de ordem entre duas posições consecutivas na solução.
2. Troca de ordem entre duas posições quaisquer da solução.
3. Realocação de um elemento em outra posição.
4. Troca de ordem entre três posições quaisquer da solução.

Esses movimentos definem, respectivamente, as vizinhanças $N^{(1)}$, $N^{(2)}$, $N^{(3)}$, e $N^{(4)}$, que são ordenadas no VNS por ordem crescente de cardinalidade. A Figura 4.1 ilustra os quatro tipos de movimento usados neste trabalho e um exemplo de cada vizinho da solução s gerado pela aplicada de cada um deles.

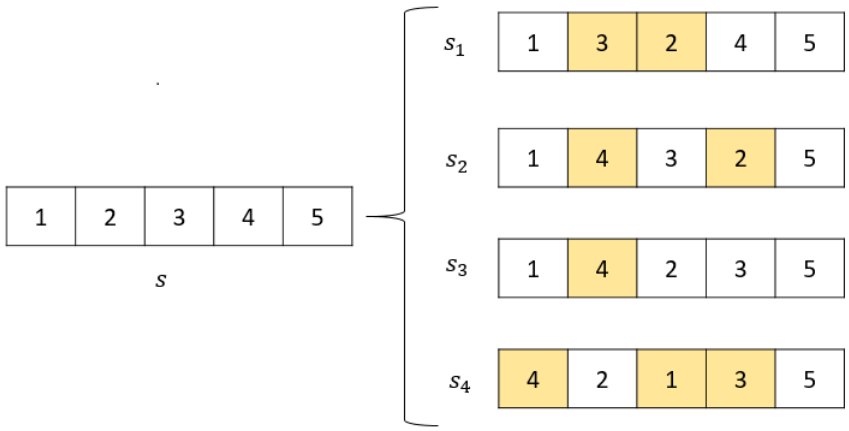


Figura 4.1: Exemplos dos movimentos realizados nas estruturas de vizinhanças $N^{(1)}$, $N^{(2)}$, $N^{(3)}$ e $N^{(4)}$

Na vizinhança $N^{(1)}$, o movimento realizado é a troca de ordem entre duas posições adjacentes. Esta estrutura é bem simples e, para uma solução de tamanho n, são gerados apenas $(n-1)$ vizinhos. No exemplo ilustrado pela Figura 4.1 a solução $s_1 \in N^{(1)}$ é obtida fazendo a troca entre os elementos consecutivos 2 e 3. A vizinhança $N^{(2)}$ possui cardinalidade igual a $n(n-1)/2$, que representa a quantidade de movimentos possíveis ou

vizinhos gerados por esta vizinhança. No exemplo, a solução $s_2 \in N^{(2)}$ é obtida fazendo a troca entre os elementos 4 e 2.

A estrutura de vizinhança $N^{(3)}$ possui cardinalidade igual a $(n-1)^2$. Na Figura 4.1, a solução $s_3 \in N^{(3)}$ é obtida realocando o elemento 4 na posição seguinte ao elemento 1. E, por fim, a estrutura de vizinhança $N^{(4)}$, com cardinalidade $(n^2-n)(n-2)/6$, que é usada para gerar o vizinho s_4. A solução $s_4 \in N^{(4)}$ é obtida realizando a troca de posições entre os elementos 1, 3 e 4.

Busca Local

A busca local no VNS foi implementada utilizando uma heurística de refinamento. Essas heurísticas, muito comuns na resolução de problemas de otimização, são técnicas baseadas na noção de vizinhança. No geral, partem de uma solução inicial qualquer e, dada uma vizinhança, percorrem o espaço de busca a cada iteração do algoritmo, de vizinho em vizinho. Neste trabalho, é usada a heurística Descida em Vizinhança Variável ou *Variable Neighborhhod Descent* (VND), proposto por [62].

Na heurística VND, o espaço de soluções é explorado por meio de trocas de estruturas de vizinhanças, sendo aceitas somente as soluções que apresentam melhoria em relação à solução atual. O método sempre retorna para a primeira vizinhança quando uma solução melhor é encontrada. O pseudocódigo desta heurística é apresentado no Algoritmo 4.2.

Algoritmo 4.2: *VND*

```
   Entrada: f(·), N(·), r, s
   Saída: s
1  início
2      k ← 1; /* Tipo de estrutura de vizinhança atual */
3      ;
4      enquanto (k ≤ r) faça
5          Encontre o melhor vizinho s' ∈ N^(k)(s);
6          se (f(s') < f(s)) então
7              s ← s';
8              k ← 1;
9          senão
10             k ← k + 1;
11         fim
12     fim
13     Retorne s;
14 fim
```

As entradas do Algoritmo 4.2 são a quantidade de estruturas de vizinhanças (r) e as estruturas vizinhanças ($N^{(k)}$). A exploração de cada vizinhança para encontrar o melhor vizinho (s') da solução atual (s), acontece na linha 4. A busca se inicia seguindo a ordem pré-estabelecida das vizinhanças $N = \{N^{(1)}, N^{(2)}, \cdots, N^{(k)}, \cdots, N^{(r)}\}$.

Na linha 5 do Algoritmo 4.2, é feita uma comparação para verificar se o vizinho s' encontrado é melhor que a solução atual s. Se s' for melhor que s, então a solução corrente é atualizada (linhas 6 e 7), caso contrário, troca-se a estrutura de vizinhança (linha 9).

A busca pelo melhor vizinho (linha 4 do Algoritmo 4.2) pode ser feita utilizando outras heurísticas de refinamento. As heurísticas mais utilizadas são o método *best improvement*, *first improvement* ou o método de descida/subida randômica. Como esta busca, dependendo do problema e do tamanho de cada vizinhança, pode ser cara computacionalmente, é comum que boa parte dos trabalhos utilizem o *first improvement* como heurística de refinamento. A justificativa desta escolha é que, neste método, a busca é interrompida assim que é encontrada a primeira solução de melhora. Neste trabalho, será usada a heurística *first improvement*.

O *First Improvement*, ou método de primeira melhora, ao invés de explorar toda a vizinhança pára de realizar a busca assim que o primeiro vizinho que melhora que a solução atual é encontrado. Apenas no pior caso (ótimo local), toda a vizinhança é explorada.

4.2.2 Variantes do VNS

As versões das meta-heurísticas de busca local com estrutura de memória e operadores de vizinhança adaptativos propostos nesta seção são semelhantes ao VNS clássico. O que distingue essas variantes do VNS da versão padrão é o operador proposto baseado em uma estrutura de vizinhança adaptativa e estratégia de memória.

As versões da meta-heurística VNS geradas neste trabalho são:

VNS: versão clássica do VNS com o conjunto de estruturas de vizinhanças.
VNS-CM: versão do VNS com o acréscimo de uma estratégia de memória.
VNS-CMVA: versão do VNS-CM utilizando o operador de vizinhança adaptativo.

VNS com memória (VNS-CM)

O algoritmo VNS com memória, que vamos denotar por VNS-CM, é uma versão do algoritmo VNS padrão com o acréscimo da estrutura de memória. Nesta proposta, a memória é acrescentada ao algoritmo apenas com a finalidade de armazenar todas as soluções encontradas ao longo das iterações do algoritmo. Verifica-se, mais adiante, que esta estratégia permite que o número total de avaliações de função seja reduzido, o que se torna vantagem quando os problemas considerados possuem função de custo que requer um grande esforço computacional para avaliação.

Na versão VNS-CM, todas as informações a respeito das soluções encontradas ficam armazenadas em uma tabela *hash* para que possam ser consultadas e recuperadas sempre que preciso pelo algoritmo. A diferença entre o VNS e o VNS-CM está apenas na maneira como a avaliação de função é realizada. O procedimento da avaliação de

função da meta-heurística VNS-CM é apresentada no Algoritmo 4.3. Neste procedimento, o operador de memória é chamado na linha 2 para consultar se a solução já foi ou não visitada e avaliada, esta consulta é feita por meio de uma pesquisa na tabela *hash*. Em caso afirmativo, recupera-se o valor de função desta solução armazenado na memória e retorna esta informação para o algoritmo (linha 7). Caso contrário, a solução é avaliada pela função de custo e as informações desta solução, junto com seu valor de função, são armazenadas na memória do algoritmo (linhas 4 e 5).

Algoritmo 4.3: *Avaliação_Função_VNSCM*

```
   Entrada: s_o
   Saída: fvl
1  início
2      [RFlag,fvlaux] = anrecover(Tind,memory);
3      se RFlag = False então
4          [fvl] = feval(so);
5          [memory] = anstoreM(so,fvl,memory);
6      senão
7          [fvl] = fvlaux;
8      fim
9  fim
```

VNS com memória e vizinhança adaptativa (VNS-CMVA)

O algoritmo VNS com memória e vizinhança adaptativa, chamado por VNS-CMVA, é uma versão do VNS-CM com o acréscimo do operador de vizinhança adaptativo.

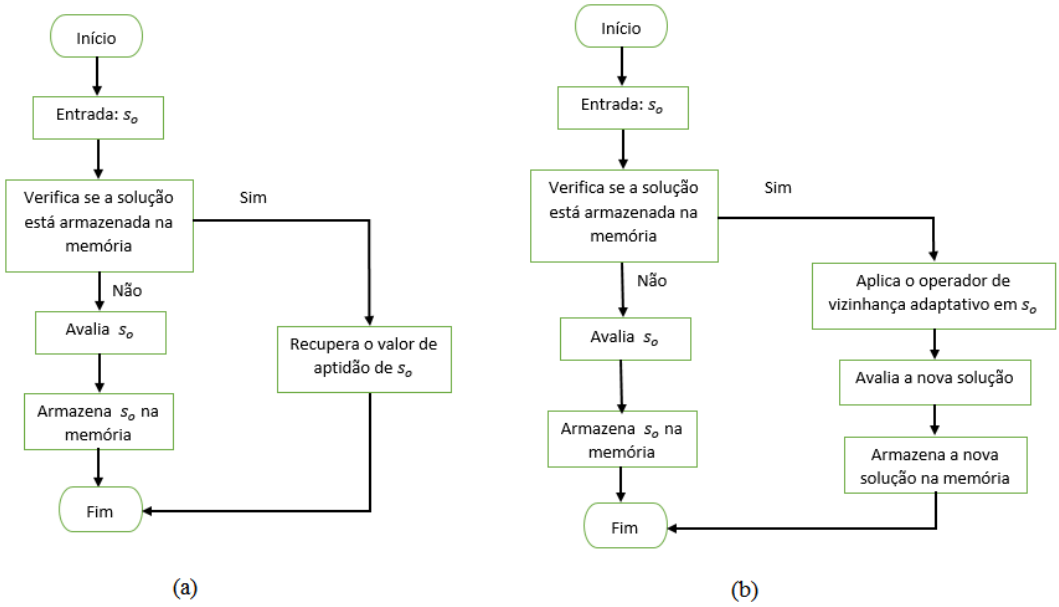


Figura 4.2: Procedimento de avaliação de uma função: (a) avaliação realizada pelo VNS-CM e (b) avaliação realizada pelo VNS-CMVA.

A diferença entre as versões VNS-CM e VNS-CMVA está apenas no uso do operador de vizinhança adaptativo presente na função de avaliação. A Figura 4.2 ilustra esta diferença. No VNS-CM, quando a pesquisa na memória informa que a solução atual já foi avaliada, o algoritmo apenas retorna o valor de aptidão desta solução. No VNS-CMVA, ao verificar que a solução s foi avaliada, o operador de vizinhança adaptativo é chamado para gerar uma nova solução s' que ainda não foi visitada, e consequentemente uma nova avaliação de função é realizada. O operador de avaliação do VNS-CMVA não reduz o número de avaliações de funções, mas permite que novos espaços de buscas sejam explorados.

Detalhes deste procedimento são apresentados no Algoritmo 4.4. Na linha 3, a memória do algoritmo é consultada para verificar se a solução s_o já foi avaliada pelo algoritmo e se está armazenada na tabela *hash*. Diante da resposta da função $anrecover(\cdot)$, que pode ser *TRUE* (verdadeiro) quando a solução está armazenada na memória do algoritmo ou *FALSE* (falso) quando não foi avaliada, é executada uma das seguintes ações:

1. se a resposta for *FALSE*, o operador avalia a solução por meio da função de custo do problema e essas informações são armazenadas na memória do algoritmo (linhas 5 e 6).
2. se a resposta for *TRUE*, o operador gera um indivíduo ainda não avaliado, vizinho da solução atual, e então esta nova solução é avaliada e armazenada na memória do algoritmo (linhas 8–11).

Algoritmo 4.4: *Avaliação_de_Função_VNSCMVA*

```
   Entrada: s_o
   Saída: fvl
1  início
2      enquanto (s_o estiver em memoria) faça
3          RFlag = anrecover(Tind,memory);
4          se RFlag = False então
5              [fvl] = feval(so);
6              [memory] = anstoreM(so,fvl,memory);
7          senão
8              [viz] = adpNeighborhood(so,memory);
9              so = viz;
10             [fvl] = feval(so);
11             [memory] = anstoreM(so,fvl,memory);
12         fim
13     fim
14 fim
```

4.3 Operadores

4.3.1 Estrutura de Memória

As tabelas *hash*, ou tabelas de dispersão, são estruturas de dados que armazenam as informações desejadas levando em consideração somente o valor absoluto de uma chave. O objetivo é que a busca por um registro, nesta estrutura, seja realizada de forma mais rápida [14].

Cada registro que se deseja armazenar na tabela *hash*, que neste estudo são todas as informações de uma solução do algoritmo, é identificado por uma chave única, usada para determinar em qual posição da tabela *hash* este registro será gravado. A posição da tabela, correspondente a uma chave, é determinada por:

$$pos = FuncaoHash(key) \tag{4-1}$$

onde *pos* indica a posição que o registro será armazenado, $FuncaoHash(\cdots)$ é a função *hash* adotada e *key* é a chave única que identifica cada registro.

A inserção de um elemento na base de dados da tabela *hash* é realizada respeitando duas etapas:

Etapa 1: através de uma função ($FuncaoHash$) escolhida a priori, a chave do registro *i* é transformada em um endereço (*pos*) da tabela.

Etapa 2: o registro é armazenado na posição (*pos*) da tabela *hash*. Caso duas ou mais chaves distintas sejam transformadas em um mesmo endereço, algum mecanismo deve ser usado para lidar ou contornar o problema de armazenar vários registros em uma mesma posição.

Os elementos podem ser armazenados na tabela *hash* utilizando o acesso direto, como é ilustrado na Figura 4.3(a). Neste exemplo, cada registro possui uma chave (*key*) única que corresponde a um índice da tabela. Os registros são armazenados no seu índice correspondente que, na figura, estão indicados por setas.

Outra forma de acesso às posições da tabela é por meio da aplicação de uma função *hash* (Equação 4-1). Nesta função, para cada valor de chave (*key*), é retornado um valor (*pos*) denominado endereço base de (*key*). O registro de chave (*key*) é armazenado na tabela *hash* na posição (*pos*).

Um problema relacionado à função *hash* é o fato dela não garantir injetividade, ou seja, podem existir chaves $x \neq y$, tal que $FuncaoHash(x) = FuncaoHash(y)$. Em outras palavras, podem existir diferentes chaves que apontam para a mesma posição da tabela. Este fenômeno recebe o nome de colisão. Na prática, uma colisão acontece quando existe a tentativa de armazenar na mesma posição da tabela *hash* registros diferentes.

A Figura 4.3 (b) ilustra a ocorrência de colisão na tabela *hash*. Considere como exemplo a função *hash*, $FuncaoHash(x) = mod(x,7)$, e as chaves 17, 35, 45, 53 e 64. A

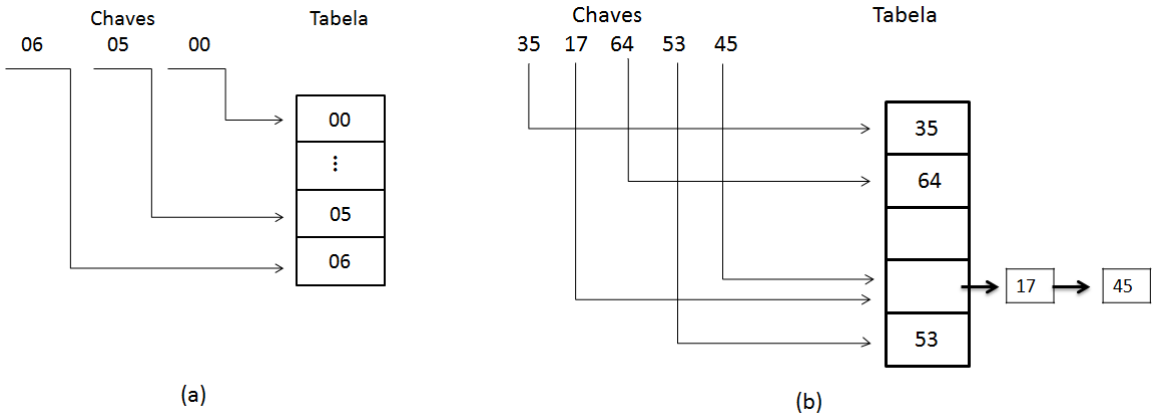


Figura 4.3: Exemplo de uma tabela *hash*: (a) com acesso direto e (b) com colisão em um mesmo endereço.

posição de um registro é calculada pelo resto da divisão da chave (*key*) pelo número primo 7. No exemplo, a função *hash* gera o mesmo valor (*pos*) para as chaves 17 e 45, pois a divisão de ambas por 7 produz o resto 3. O registro 17 é armazenado primeiro na posição 3 e ao tentar salvar o registro 45, ocorrerá uma colisão. Para corrigir o problema gerado pela colisão uma tática que pose ser usada é a implementação de uma lista encadeada em conjunto com a tabela *hash*. A lista encadeada permite armazenar todos os registros na ordem em que aparecem, este procedimento é ilustrado na Figura 4.3 (b).

É comum que muitas colisões aconteçam, uma vez que, na maioria dos casos, a quantidade de registros que se pretende armazenar na estrutura de dados é muito maior que o tamanho da própria tabela. Considerando m o tamanho da tabela *hash*, caso existam colisões, para recuperar um registro nesta estrutura é necessário que se faça uma busca sequencial na lista encadeada desta posição. No geral, o custo de recuperação de um registro dentro da tabela *hash* varia de $O(1)$ até $O(n)$, o pior caso, que acontece quando todos os registros estão armazenados na mesma posição.

Dessa forma, a complexidade da busca por um registro na tabela *hash* vai depender da qualidade da função *hash* implementada. Uma função ruim pode elevar o custo de pesquisa na estrutura, sendo que, quanto mais próximo de $O(1)$ for a complexidade da busca, melhor.

A tabela *hash* é usada como estratégia de memória da meta-heurística VNS, com o propósito de atuar como uma memória de longo prazo no algoritmo. O objetivo desta estrutura é apenas armazenar todas as soluções geradas no decorrer do algoritmo. A maneira como vai ser usada, depende dos demais operadores da meta-heurística. A tabela *hash* foi implementada utilizando uma lista encadeada para o tratamento dos casos de colisões, e cada registro da *hash* é uma *struct* com as seguintes características:

registro.in = uma solução do algoritmo;
registro.fvl = o valor de aptidão da solução armazenada;
registro.adp = parâmetro usado pelo operador de vizinhança adaptativo;

registro.BN = parâmetro usado pelo operador de vizinhança adaptativo.

O tamanho da tabela *hash* é definido conforme o trabalho [8] pelo número de bits (b_{ind}) utilizados para a indexação da *hash*. É usada a Equação (4-2) e $b_{ind} = 16$ para determinar este tamanho. É importante destacar que este valor pode ser escolhido conforme a quantidade de dados que se pretende armazenar, neste trabalho, optou-se por seguir os passos do trabalho de referência.

$$M = 2^{b_{ind}} - 1 \tag{4-2}$$

Nos problemas de otimização combinatória, uma quantidade enorme de soluções podem ser geradas pelas meta-heurísticas, o que demanda uma escolha bem cuidadosa da função *hash*, para que as soluções sejam armazenadas de forma uniforme por toda a extensão da tabela, ou seja, tenha poucas colisões. Uma boa escolha para a função garante que a recuperação de uma informação na memória tenha um custo computacional próximo a $O(1)$, o que é particularmente interessante.

O algoritmo adotado como função *hash* é inspirado na função proposta em [8]. Algumas adaptações foram necessárias na função de referência, visto que a chave utilizada no trabalho é diferente da tratada nesta tese. Os autores utilizam como chave um vetor solução onde cada variável i é representada por uma sequência binária de comprimento n_{bits}. Além disso, na função de referência é considerado um parâmetro extra (dep_i), que determina uma profundidade em relação aos bits que representam cada variável i. Nesta tese, a chave usada é representada por um vetor n-dimensional, composto por uma permutação de n elementos inteiros, ou seja, $x = [x_1, x_2, \ldots, x_n]$. Este vetor é uma solução do problema tratado. O pseudocódigo da função *hash* é definido no algoritmo 4.5.

Algoritmo 4.5: *Função Hash*

```
   Entrada: in
   Saída: pos
1  início
2      key = in;
3      n = length(key);
4      nHash = 2^b_ind;
5      H = 0;
6      para i=1 até n faça
7          C = key(i)*prime(i);
8          t = mod(H+C,nHash);
9          H = T(t+1);
10     fim
11     pos = H+1;
12 fim
```

Cada solução da meta-heurística *in* é utilizada como chave para encontrar a posição onde ela será armazenada na memória, linha 2. No algoritmo, *prime* é um vetor

que contém os 10.000 primeiros números primos; n é a quantidade de variáveis de decisão (ou tamanho do vetor in) e T é um vetor que contém uma permutação aleatória da sequência $\{0, \cdots, M-1\}$.

Diferente da função proposta em [8], onde os autores calculam o valor de C usando uma função que transforma cada variável i (que é uma sequência binária) em seu correspondente em $\mathbb{R}$, linha 7 do algoritmo, cada variável (que é um número inteiro) será multiplicada por um elemento do vetor de números primos para o cálculo da variável C. O vetor de números primos também é usado como uma estratégia para diminuir o número de colisões.

Nas linhas 8 e 9, é calculado um fator H que é usado no cálculo final da posição pos. Assim, a função $hash$ proposta, transforma a chave (key) em um endereço base da tabela $hash$. O endereço é denotado por (pos) e equivale a um número inteiro pertencente ao intervalo $[0,\ (M-1)]$.

4.3.2 Operador Vizinhança Adaptativo

Soluções replicadas podem ser criadas durante a resolução de problemas de otimização combinatória via meta-heurísticas. Evitar que estas soluções sejam avaliadas várias vezes é interessante nos casos em que uma avaliação de função é considerada custosa. Porém, além de evitar essa reavaliação, é pertinente que também se faça uso de alguma estratégia para contornar essas replicações, uma vez que, criando sempre a mesma solução, o algoritmo pode ficar preso em uma região do espaço de busca.

A essência do operador de vizinhança adaptativo é promover uma busca por uma vizinhança adaptativa da solução replicada, para gerar uma nova solução que ainda não foi visitada pelo algoritmo. Quem fornece a este operador a informação de que uma solução foi ou não visitada é a estrutura de memória.

Este operador é chamado pelo algoritmo durante a avaliação de uma solução para gerar um novo indivíduo que ainda não esteja armazenado na memória. O pseudocódigo do operador de vizinhança adaptativo é apresentado no Algoritmo 4.6.

A entrada deste algoritmo é a solução atual s_o e o arquivo de memória. Considere que a solução s_o já tenha sido avaliada pelo algoritmo, assim, os passos deste procedimento são: pesquisa-se na memória do algoritmo por informações desta solução, linhas 2 e 3, utilizando as funções $RecuperaMemoria(\cdot)$ e $NvizAdap$, respectivamente. Essas funções retornam a posição que a solução foi armazenada na tabela $hash$ e se ela possui algum vizinho adaptativo. A partir destas informações, o operador executa a melhor estratégia para obter uma nova solução, que ainda não foi avaliada e que seja vizinha de s_o.

O valor de $NvizAdap = 0$, indica que a solução s_o não possui vizinhos adaptativos, ou seja, é a primeira vez que esta solução entra no operador. Neste caso, uma nova solução é gerada, utilizando as estruturas de vizinhanças adaptativas implementadas, e,

Algoritmo 4.6: *Operador_Vizinhança_Adaptativo*

```
   Entrada: so, memory
   Saída: viz
1  início
2      [pos] = RecuperaMemoria(so,memory);
3      NvizAdap = length(memorykey.bn);
4      se NvizAdap ==0 então
5          [viz] = EstruturaVizAdaptativa(so,0);
6      senão
7          NumEB = memorykey.BN.ad;
8          se NunEB >= (n/2) então
9              [viz]= ConstrucaodaSolucao(·);
10         senão
11             [viz] = EstruturaVizAdaptativa(so,NumEB);
12         fim
13     fim
14 fim
```

considerando a quantidade inicial de elementos por bloco igual a $adp = 2$. As vizinhanças utilizadas por este operador são as mesmas da busca local do algoritmo e usadas na mesma ordem. A diferença consiste que, neste operador, os movimentos são realizados por blocos e não por posições. Por essa diferença e por permitir a exploração de novas regiões do espaço de busca, vamos chamá-las de estruturas de vizinhança adaptativas.

Ainda no algoritmo 4.6, caso a solução s_o possua vizinhos adaptativos ($NvizAdap > 0$), o operador verifica para quais quantidades de elementos por blocos ($NumEB$) já foram feitas as explorações no espaço de busca. Se esse valor for maior ou igual a $n/2$, significa que todo o espaço de busca adaptativo foi explorado. Neste caso, uma nova solução é gerada aleatoriamente utilizando a função $ConstrucaodaSolucao(\cdot)$. Caso contrário, um vizinho adaptativo é gerado utilizando uma das estruturas de vizinhança adaptativa, porém, a partir do valor atualizado de $NumEB$. Usando esse valor atualizado, evita-se que o algoritmo faça novamente buscas por regiões que já foram exploradas pelo operador.

As estruturas de vizinhança do operador vizinhança adaptativo são as mesmas estruturas da busca local do VNS (seção 17) e são usadas na mesma ordem. A diferença entre os movimentos aplicados por este operador em relação ao operador de busca local, é que os movimentos são realizadas por blocos de variáveis. Em outras palavras, o movimento ao invés de ser aplicado nos elementos da solução será aplicado em blocos de tamanho k formados por elementos desta solução.

As vizinhanças adaptativas são aqui denotadas por $Na^{(1)}$, $Na^{(2)}$, $Na^{(3)}$, e $Na^{(4)}$. Os movimentos realizados por essas vizinhanças, utilizando a ideia de blocos, são definidos respectivamente, por:

1. Troca de ordem entre dois blocos na sequência adjacente.

2. Troca de ordem entre dois blocos quaisquer da solução.
3. Realocação de um bloco em outra posição.
4. Troca de ordem entre três blocos quaisquer da solução.

A Figura 4.4 ilustra os movimentos realizados considerando uma solução de tamanho $n = 15$. No exemplo, a solução é dividida em cinco blocos com três elementos em cada. Os vizinhos s_1, s_2, s_3 e s_4 são obtidos utilizando as estruturas $Na^{(1)}$, $Na^{(2)}$, $Na^{(3)}$, e $Na^{(4)}$, respectivamente.

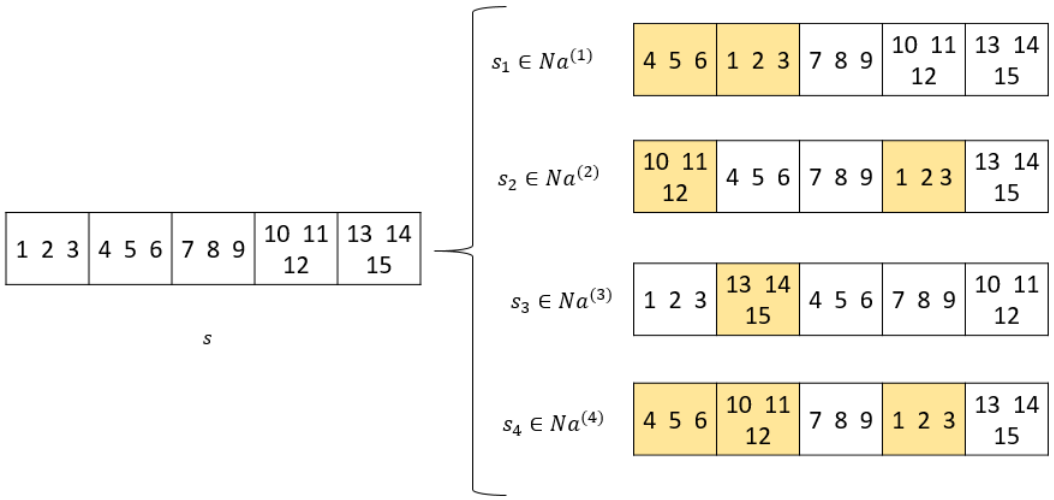


Figura 4.4: Movimentos realizados pelas das vizinhanças adaptativas $Na^{(1)}$, $Na^{(2)}$, $Na^{(3)}$, e $Na^{(4)}$.

Toda solução é gerada com os mesmos parâmetros iniciais obtidos durante a construção da solução inicial, onde o valor de $adp = 2$, refere-se à quantidade inicial de elementos que devem conter os blocos durante a primeira entrada pelo operador e $BN = [\,]$ informando que a solução não possui vizinho adaptativo. Na medida que buscas são realizadas em uma mesma solução, o valor de adp é atualizado até que atinja o máximo de $n/2$. Neste caso, toda vez que a solução replicada entrar no operador, entende-se que já foi realizada buscas em toda sua vizinhança adaptativa e novas soluções são geradas pela heurística de construção de uma solução inicial.

4.4 Análise da função *hash*

A estrutura de dados tabela *hash* é utilizada como memória nos algoritmos VNS-CM e VNS-CMVA e a função *hash* calcula em qual posição um registro será armazenado nesta tabela. Uma boa função é considerada aquela que possui o seu cálculo rápido e que gere poucas colisões. Além disso, é desejável que o preenchimento da tabela *hash* seja uniforme para qualquer conjunto de chaves.

Neste trabalho, o valor de referência para a quantidade de posições da tabela *hash* é o valor de indexação usado em [8]. O valor de referência é $b_{ind} = 16$, que resulta em uma tabela de tamanho $M = 2^{b_{ind}} - 1 = 65.535$.

Para verificar a qualidade e eficiência da função *hash* será analisada a ocupação da tabela para diferentes quantidades de dados. Para tanto, é feito um histograma com as frequências das colisões presentes na tabela e um mapa de calor para analisar a distribuição das soluções por toda a tabela *hash*. Para a análise, três conjuntos de soluções distintas e gerados de forma aleatória, são inseridas na tabela. O primeiro conjunto possui 50.259 elementos e, o segundo e terceiro com 62.485 e 89.225 elementos, respectivamente.

O mapa de calor, ou *HeatMap*, é usado para verificar por meio das cores a quantidade de elementos que estão armazenados por posição na estrutura de memória. Como a tabela *hash* é implementada na forma de um vetor, e sua dimensão elevada não permite a visualização das cores, foi feita uma adaptação desta tabela para um formato de matriz. A Figura 4.5 apresenta os mapas de calor gerados para os conjuntos de chaves 1, 2 e 3, quando inseridos na tabela *hash*.

Na escala de cor da legenda do mapa de calor, as cores vermelho e azul indicam uma quantidade elevada de elementos inseridos nesta posição, e as cores amarela e verde indicam uma quantidade baixa de elementos. Assim, quanto mais escuro for o tom de vermelho ou azul, maior é o número de colisões ocorrido. Em contrapartida, quanto mais claro for o tom de amarelo e verde, menor é o número de colisões. O desejável é que as cores verde e amarela sejam predominantes no quadro.

Na Figura 4.5, para os mapas gerados dos três conjuntos, é possível notar claramente a predominância da verde e amarela em relação as demais. Para o conjunto 3 a cor amarela é um pouco mais evidente que para o conjunto 1 e 2. Esta característica do mapa de calor, mostra o número de colisões gerados pela função *hash* proposta é baixo, visto que a maioria das posições parecem estar ocupadas com no máximo dois elementos.

A frequência com a quantidade de colisões geradas durante o armazenamento dos registros pertencentes aos conjuntos 1, 2 e 3 na tabela, são apresentados na Figura 4.6. Nota-se a maioria das colisões existentes para ambos os conjuntos foi de ordens dois e três. O número máximo de colisões ocorridas em uma posição foi de ordem oito, no conjunto 3, e ordem seis nos conjuntos 1 e 2.

Neste contexto, podemos dizer que a função *hash* apresenta um bom desempenho em relação à distribuição dos registros por toda a estrutura de dados, visto que a maioria das colisões que ocorreram foi de ordem 2. O número máximo de colisões , que aconteceu em apenas um dos conjuntos, foi de ordem 8, porém em poucas posições da tabela. Como na grande maioria das posição não ocorreu colisão, ou quanto obteve este valor foi mínimo, temos que a função *hash* proposta alcança os objetivos esperados por distribuir os dados de forma uniforme por toda a extensão da estrutura de memória.

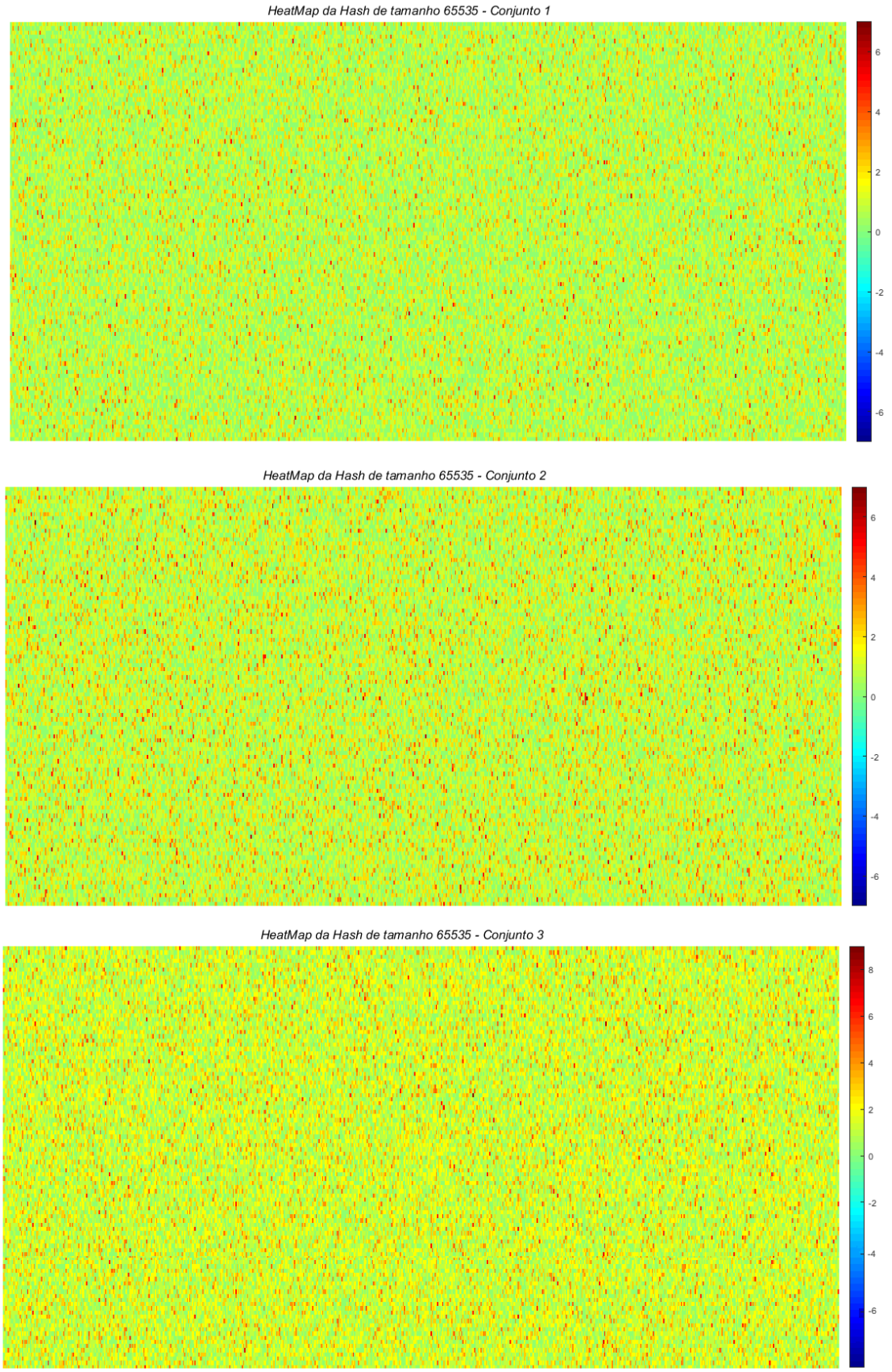


Figura 4.5: Mapa de calor da tabela *hash* de com valor de indexação $b_{ind} = 16$, para os conjuntos 1, 2 e 3

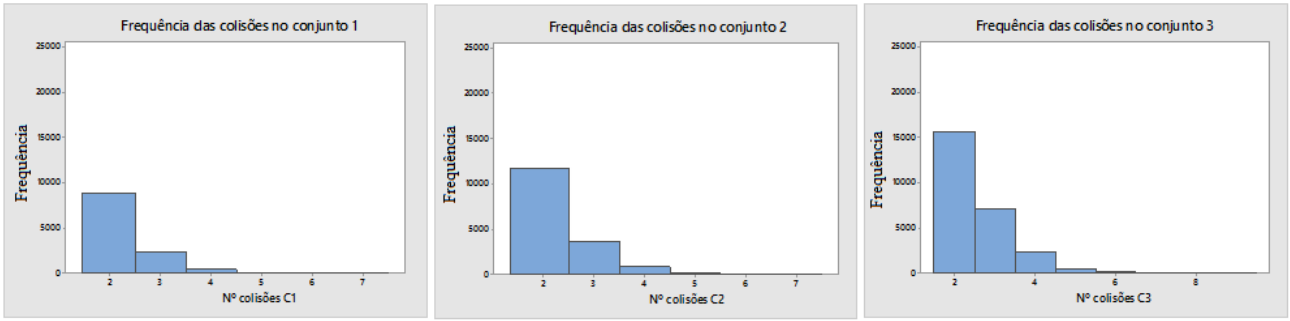


Figura 4.6: Frequência relativa do número colisões ocorridas pela inserção dos conjuntos de dados 1, 2 e 3 na tabela *hash*.

4.5 Operador de Vizinhança Adaptativo no VNS aplicado a um Problema de *Scheduling*

4.5.1 Descrição do Problema

O problema *common due-date scheduling* é um problema de programação de tarefas em uma única máquina com a data de entrega comum. Um exemplo prático de uma aplicação deste tipo de problema corresponde, por exemplo, a um sistema de montagem em que todos os componentes de um mesmo produto devem estar prontos ao mesmo tempo. O objetivo é encontrar uma data ideal para a entrega comum de todos os produtos e o cronograma com a sequência de produção (ordem em que cada tarefa será executada), a fim de otimizar um determinado critério que esteja relacionado à data de entrega do trabalho [35]. A descrição detalhada do problema, assim como o conjunto de instâncias utilizadas nesta tese, são apresentados em [5].

O Problema é caracterizado por ter uma data de entrega comum, d, e um conjunto de n tarefas que devem ser processadas em uma única máquina a partir de um tempo inicial zero. Cada tarefa $i = 1, \cdots, n$ possui um tempo de processamento $p(i)$ conhecido a priori. Se o tempo de conclusão C_i da tarefa i for menor que a data de entrega comum d, isso indica que esta tarefa foi concluída com antecipação e é medida por $E = d - C_i$. De forma análoga, se a tarefa for concluída em um tempo maior que d, indica que a tarefa foi realizada com atraso $T = C_i - d$.

Como não é possível saber se a tarefa será concluída antes ou depois da data comum, as expressões $E_i = \max\{d - C_i, 0\}$ e $T_i = \max\{0, C_i - d\}$, determinam respectivamente, cada tarefa i foi entregue antes ou após o prazo. Além disso, à cada tarefa é atribuída uma penalidade *earliness* $\alpha(i)$ e *tardiness* $\beta(i)$, que correspondem a penalidades por antecipação e atraso.

O objetivo do problema é encontrar a configuração (sequência) de tarefas que minimiza a soma dos custos das penalidades por antecipação (*earliness*) ou por atraso (*tardiness*) da conclusão das tarefas. A função objetivo do problema é representada pela

Equação (4-3).

$$f(S) = \sum_{i=1}^{n} \alpha_i E_i + \sum_{i=1}^{n} \beta_i T_i \tag{4-3}$$

em que S representa uma sequência com a ordem em que as tarefas são realizadas.

O valor da data de entrega comum (d) influencia na complexidade do problema a ser resolvido. Quanto menor for o valor de d, maior será a quantidade de tarefas entregues com atraso. Para $d = 0$ o problema passa a ser trivial, pois basta ordenar a sequência de tarefas por ordem decrescente das penalidades de atraso. Para valores de $d \geq \sum_{i=1}^{n} p_i$ dizemos que o problema é irrestrito, caso contrário, o problema é dito restrito. Neste segundo caso, para diferentes valores de d os problemas a serem resolvidos podem se tornar mais fáceis ou mais difíceis de serem solucionados. Em [5] é feito um estudo detalhado destes valores e apresentados alguns trabalhos relacionados.

A data de entrega comum é calculada, assim como em [5], pela Equação (4-4):

$$d = round(h \sum_{i=1}^{n} p(i)) \tag{4-4}$$

onde a variável n representa a quantidade de tarefas. A variação do valor de h utilizado na equação torna o problema mais ou menos restritivo. O autor utiliza valores para h iguais a $(0,2)$, $(0,4)$, $(0,6)$ e $(0,8)$.

As premissas envolvidas na resolução do problema são:

1. os tempos de processamento de cada tarefa são determinísticos;
2. a máquina processa uma única tarefa por vez;
3. a máquina sempre está disponível, ou seja, não existe tempo ocioso entre a realização das tarefas;
4. as tarefas realizadas, assim como as penalidades por antecipação e atraso, são conhecidas antecipadamente;
5. as tarefas são independentes;
6. todas as tarefas devem ser entregues em uma mesma data (data comum).

Na literatura alguns trabalho concentram esforços para resolver o problema em questão, que é considerado um problema NP-Difícil [36]. Em [35], encontra-se um estudo aprofundado sobre os principais trabalhos desenvolvidos na área. Alguns autores fazem uso das meta-heurísticas para resolver o problema, como em [40] que utiliza a busca tabu, [50] algoritmos genéticos e em [25] as meta-heurísticas *Simulated Annealing* e Estratégias Evolutivas.

Uma pesquisa mais recente sobre o problema é encontrado em [64]. O autor utiliza o problema e as instâncias apresentados em [5]e propõe o uso do algoritmo *Particle Swarm Otimization* (PSO) para resolvê-lo. Os experimentos mostraram um

bom desempenho do PSO aplicado ao problema, por ser encontrado novos limitantes superiores para 82% dos valores de referência da literatura.

4.5.2 Construção da Solução

A construção da solução inicial é realizada de acordo com o procedimento apresentado no algoritmo 4.7. Para este problema, a solução é construída elemento a elemento utilizando uma heurística inspirada na construção inicial da meta-heurística GRASP.

Algoritmo 4.7: *Constroi_Solução*

Entrada: n, p, α, β, d
Saída: in, adp, BN

```
1  início
2      [ord] = sort(p);
3      in = [ ];
4      c = 0;
5      para i = 1 até n faça
6          Tamlista = ceil(1 + rand(1)*(n-i-1));
7          JobsCand = ord(1:TamLista);
8          c = c + p(JobsCand);
9          Fun = α_i max(d − c, 0) + β_i max(c − d, 0),   i = 1, ··· , TamList;
10         [custoMenor, jobs] = min(Fun);
11         in(i) = jobs;
12         c = c+ p(in(i));
13         atualiza valores de p, α, β;
14     fim
15     adp = 2;
16     BN = [ ];
17 fim
```

Os parâmetros de entrada deste procedimento são: a quantidade de tarefas n; o tempo de processamento de cada tarefa; as penalidades α e β (que correspondem, respectivamente, às penalidades por atraso e antecipação da entrega das tarefas) e d a data de entrega comum dos trabalhos.

Na linha 2, as tarefas são ordenadas de acordo com o tempo de processamento de cada uma delas. Nas linhas 5 até 14, as tarefas são inseridas uma a uma na solução inicial in. Esta inserção ocorre da seguinte maneira: é calculada de forma aleatória uma quantidade de possíveis tarefas para serem inseridas na solução (linha 6); a lista com as possíveis tarefas são criadas (linha 7); são calculados o tempo de processamento final e o valor das penalidades, para cada uma das tarefas da lista *JobsCand*, caso sejam inseridas na solução (linhas 8 e 9); insere em in a tarefa que gerar o menor valor de penalidade (linha 10). Os parâmetros do problema são atualizados e o procedimento de inserção de cada tarefa é repetido até que todas as tarefas sejam inseridas da solução in.

As linhas 15 e 16 criam dois parâmetros adicionais que são usados pelo operador de vizinhança adaptativo. A variável $adp = 2$ indica a quantidade inicial de tarefas por bloco e a variável BN a quantidade de vizinhos encontrados pela busca local. Essas variáveis dizem respeito ao operador vizinhança adaptativo, e todas as soluções quando criadas, por padrão, são iniciadas com estes valores.

4.5.3 Problemas Testes

Para avaliar os métodos propostos para resolver o problema de *scheduling* com data de entrega comum, é usado um conjunto de instâncias da literatura. Este conjunto de instâncias foi desenvolvido por [5]. Os problemas-teste permite medir a eficiência da metodologia proposta e a qualidade das soluções obtidas por cada versão do VNS.

São sete conjuntos de problemas que, possuem por sua vez, uma quantidade de 10 problemas diferentes, gerando um total de 70 problemas testes. Cada conjunto é diferenciado por possuir uma quantidade específica de tarefas, que varia em 10, 20, 50, 100, 200, 500 e 1000 tarefas.

Considerando os 70 problemas propostos e que a variável h na equação (4-4) pode variar em quatros valores, a quantidade total de problemas-testes são 280. Devido ao número elevado de problemas, foram escolhidas apenas as instâncias de tamanhos 50, 100, 200 e 500, e valores de $h = 0,2$ e $h = 0,4$, para trabalhar com os problemas mais restritos. Ao todo foram testadas 80 instâncias. A escolha por valores menores de h se justifica pelo fato do atraso ser geralmente muito mais crítico que o adiantamento.

4.5.4 Resultados

Nesta seção são apresentados os resultados computacionais obtidos pela aplicação da metodologia proposta, variações do VNS, na solução do problema de *scheduling* em uma única máquina com data de entrega comum para 80 problemas-teste da literatura. Os resultados obtidos pelos algoritmos são comparados com as soluções de referência da literatura, [64], que são usadas como limitantes superiores.

É feita uma análise preliminar dos comportamento dos algoritmos aplicados na resolução de alguns problemas-testes, para determinar os valores dos parâmetros e o melhor conjunto de estruturas de vizinhanças, que devem ser usados nos algoritmos para resolver todos os problemas propostos. Em seguida, as versões do VNS com os parâmetros calibrados são usados para resolver todos os 40 testes da literatura, considerando dois valores de h. Os resultados obtidos pelas variações do VNS comparados. Por fim, as soluções encontradas são comparadas com o limitante superior [64]. O intuito é verificar a contribuição do operador adaptativo baseado em memória, em uma meta-heurística, para resolver um problema discreto de otimização.

Os algoritmos foram implementados utilizando o *software* MATLAB 2015[1], e os testes foram feitos em um PC Intel Core i7, com 8 GB de memória RAM, em ambiente Windows 10. As análises estatísticas foram realizadas com o auxílio do pacote computacional Minitab, em sua versão 17.

Algumas análises preliminares foram realizadas com o intuito de configurar os algoritmos para o problema testado. O objetivo é determinar os valores para os parâmetros do algoritmo e a quantidade e quais estruturas de vizinhanças devem ser usadas. Considera-se este estudo importante, visto que a melhor configuração para estes parâmetros pode variar de acordo com o problema a ser resolvido, e uma má escolha para estes valores pode interferir no desempenho do algoritmo. Assim, as análises envolvem:

- uma investigação a respeito da influência de cada estrutura de vizinhança na qualidade das soluções encontradas, visto que, tanto a meta-heurística escolhida como o operador proposto, trabalham baseados nestas estruturas. O objetivo é determinar quais e quantas interferem na qualidade de solução encontrada;
- uma investigação considerando diferentes valores de um conjunto de parâmetros, para se obter a melhor configuração desse conjunto.

4.5.5 Análise das Estruturas de Vizinhança

Nesta seção, é investigada a influência das quatro estruturas de vizinhanças implementadas em relação a qualidade da solução final gerada pelo algoritmo. O objetivo é avaliar a contribuição de cada uma delas no resultado final.

Considera-se relevante este levantamento gerado, visto que a busca local do VNS e o operador de vizinhança adaptativo, fazem uso destas estruturas. Um mau uso ou um uso desnecessário de uma estrutura pode aumentar o esforço computacional do algoritmo sem ganho de desempenho.

As estruturas de vizinhança analisadas partem de quatro movimentos diferentes:

movimento 1: Troca de ordem entre duas posições (ou blocos) na sequência adjacente. Este movimento é responsável por gerar a vizinhança $N^{(1)}$.

movimento 2: Troca de ordem entre duas posições (ou blocos) quaisquer da solução. Este movimento é responsável por gerar a vizinhança $N^{(2)}$.

movimento 3: Realocação de uma posição (ou bloco) em outra posição. Este movimento é responsável por gerar a vizinhança $N^{(3)}$.

movimento 4: Troca de ordem entre três posições (ou blocos) quaisquer da solução. Este movimento é responsável por gerar a vizinhança $N^{(4)}$.

[1] Matlab é uma marca registrada da Mathworks.

Essas estruturas são ordenadas no VNS por ordem crescente de cardinalidade, ou seja, a estrutura com menor cardinalidade é usada primeiro no algoritmo. Os conjuntos de estruturas de vizinhança, que podem ser usados nas versões do VNS são:

Conjunto A: Contém as quatro estruturas de vizinhança.

Conjunto B: Contém três estruturas de vizinhança, a vizinhança $N^{(1)}$ é retirada deste fator.

Conjunto C: Contém três estruturas de vizinhança, a vizinhança $N^{(2)}$ é retirada deste fator.

Conjunto D: Contém três estruturas de vizinhança, a vizinhança $N^{(3)}$ é retirada deste fator.

Conjunto E: Contém três estruturas de vizinhança, a vizinhança $N^{(4)}$ é retirada deste fator.

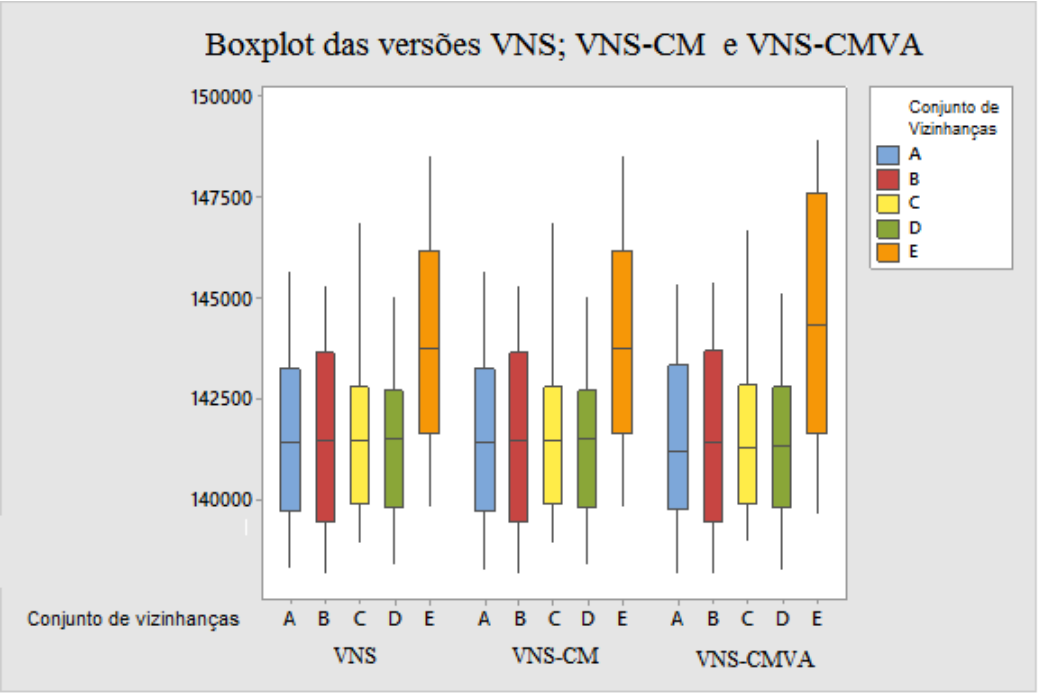
O objetivo é verificar se existe diferença entre as versões do VNS, em relação ao resultado final obtido pelo algoritmo, quando uma a uma das estruturas de vizinhanças são retiradas. Assim, foram realizadas 30 execuções de cada problema considerando cada versão do VNS com cada um dos conjuntos de estruturas. Foram usados três problemas de tamanhos diferentes, ($n = 100, 200$ e $n = 500$) e valor de $h = 0,2$ (que considera o caso mais restrito). A escolha dos problemas testes se deu de forma aleatória dentre todas as instâncias disponíveis. A escolha dos parâmetros dos algoritmos também aconteceu de forma aleatória, uma vez que neste momento o objetivo é apenas verificar o desempenho das estruturas. Assim, os parâmetros usados são: $Card = 0,10$, $Npert = 1$ e $b_{ind} = 16$ e como critério de parada o valor máximo de iterações igual a 100.

Para testar a igualdade das médias das soluções obtidas em várias execuções dos algoritmos contra a desigualdade das mesmas, optou-se pelo uso do teste de hipótese ANOVA (*Analysis of Variance*) [63].

São apresentados, de forma detalhada, os resultados obtidos para o problema-teste *sch*100. Para os demais problemas, *sch*200 e *sch*500, as análises dos resultados são equivalentes e não serão descritas.

Uma análise descritiva dos dados foi feita a priori para facilitar a interpretação das informações geradas na execução do ANOVA. A Figura 4.7 apresenta os resultados do *Boxplot* considerando a função objetivo como resposta, para cada um dos algoritmos (VNS, VNS-CM e VNS-CMVA) combinado a um conjunto de estruturas de vizinhanças (A, B, C, D e E). Fazendo apenas uma análise visual dos dados, observa-se no *boxplot* que as soluções geradas pelos algoritmos apresentaram pequena variabilidade e comportamento bem parecidos nos dados. Porém, os algoritmos utilizando o conjunto E, aparentemente, apresentaram médias superiores e diferentes das demais. Como estamos tratando de um problema de minimização, o conjunto E aparenta apresentar os piores resultados.

Com a finalidade de confirmar a hipótese nula, que indica que não existem diferenças entre as médias geradas das soluções encontradas pelas versões do VNS usando



,

Figura 4.7: *Boxplot* com as soluções geradas pelos algoritmos VNS, VNS-CM e VNS-CMVA utilizando os conjuntos de estruturas de vizinhança, para o problema *sch*100.

cada um dos conjunto de estruturas de vizinhanças, foi aplicado o teste de hipótese ANOVA. Como os resultados obtidos para os três algoritmos são os mesmos, vamos apresentar apenas o resultado obtido para a versão VNS-CMVA. O resultado do ANOVA indica que para P-valor= 0, rejeita-se a hipótese nula e podemos concluir que as médias são diferentes.

Confirmada as diferenças entre as médias, com a ajuda do teste de Tukey, Figura 4.9, é possível afirmar que, para os intervalos criados com 5% de significância, os conjuntos de estruturas AE, BE, CE e DE, com aplicação no VNS-CMVA, possuem médias diferentes. Quando esses conjuntos são agrupados, é possível observar que o conjunto E é diferente dos demais, já que, estatisticamente, os conjuntos A, B , C e D não são diferentes.

Os gráficos em relação ao resíduo não são apresentados, mas é interessante destacar que as suposições de normalidade e homocedasticidades dos resíduos foram satisfeitas. A nível de curiosidade, a Figura 4.10 mostra o único conjunto de vizinhanças que, estatisticamente, é diferente dos restantes nos algoritmos VNS e VSN-CM. Trata-se do conjunto E. Para o algoritmo VNS-CMVA o resultado foi o mesmo.

O conjunto E é caracterizado pela retirada da estrutura de vizinhança 4. Nos testes, este conjunto de vizinhanças aplicado nas versões do VNS se mostrou diferente dos demais conjuntos e com a pior média (Figura 4.7). Embasados nas análises, podemos concluir que, com a retirada da estrutura de vizinhança 4 (que realiza o movimento de troca entre três posições), a qualidade das soluções geradas pelos algoritmos piora em

One-way ANOVA: VNS-CMVA versus Conjunto de Vizinhanças

Method

Null hypothesis	All means are equal
Alternative hypothesis	At least one mean is different
Significance level	α = 0,05

Equal variances were assumed for the analysis.

Factor Information

Factor	Levels	Values
Conjunto de Vizinhanças	5	A; B; C; D; E

Analysis of Variance

Source	DF	Adj SS	Adj MS	F-Value	P-Value
Conjunto de Viz.	4	191889209	47972302	8,79	0,000
Error	145	791124917	5456034		
Total	149	983014127			

Model Summary

S	R-sq	R-sq(adj)	R-sq(pred)
2335,82	19,52%	17,30%	13,87%

Figura 4.8: Resultado do ANOVA para analisar as diferenças entre as médias das amostras geradas pelo algoritmo VNS-CMVA com cada conjunto de vizinhanças.

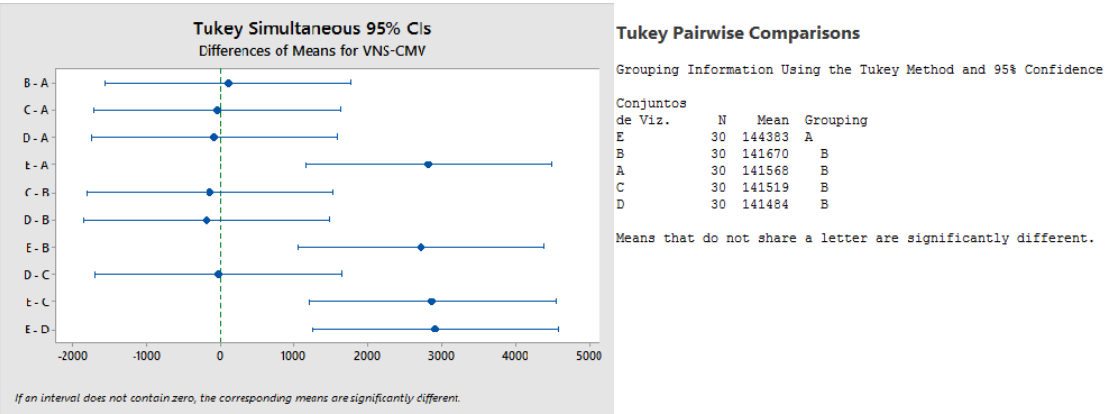


Figura 4.9: Agrupamento realizado pelo teste de Tukey considerando cada conjunto de vizinhança no algoritmo VNS-CMVA.

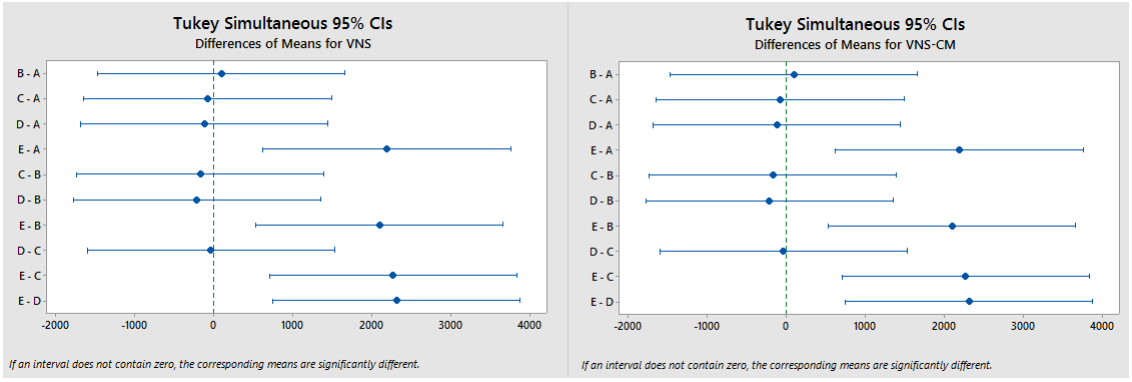


Figura 4.10: Teste de Tukey considerando cada conjunto de vizinhança nos algoritmos VNS e VNS-CM.

relação ao valor de função objetivo. Em relação aos demais conjuntos de vizinhanças, não foi possível chegar a alguma conclusão. Com isso, temos que a estrutura de vizinhança 4 interfere na qualidade do conjunto de soluções geradas pelo algoritmo.

Para os problemas de tamanho 200 e 500, o comportamento dos algoritmos e os resultados estatísticos obtidos para cada conjunto de vizinhança foi o mesmo. A Figura 4.11 apresenta os resultados do boxplot, considerando a função objetivo para cada uma das meta-heurísticas e conjuntos de estruturas de vizinhança. Fazendo apenas uma análise visual, todos conjuntos (A, B, C, D e E) apresentam pequena diferença na variabilidade dos dados e comprovado estatisticamente que o comportamentos dos algoritmos são parecidos. O conjunto E, aplicado aos algoritmos VNS e VNS-CM também apresentou a pior média.

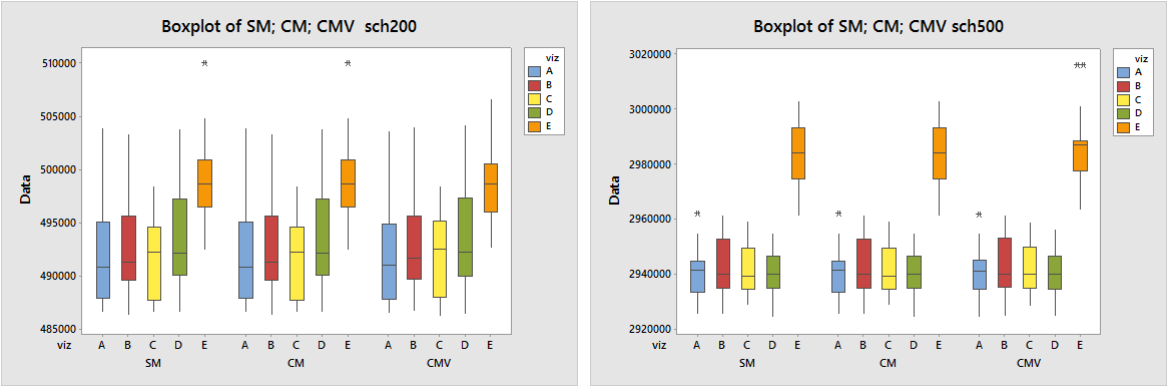


Figura 4.11: Boxplot dos resultados dos problemas de tamanho 200 e 500 para cada um dos fatores e os algoritmos VNS, VNS-CM e VNS-CMVA.

Em relação aos resultados de cada algoritmo para os conjuntos A, B, C e D, não é possível observar diferença entre as médias das soluções. Ou seja, ambos os conjuntos não se diferem estatisticamente e possuem comportamentos parecidos se aplicados a cada meta-heurística. Porém, como pretende-se obter um algoritmo eficaz e que, ao mesmo tempo, possua um baixo custo computacional, observado que não existe diferença entre as médias dos conjuntos de vizinhanças, o custo computacional gerado pela exploração no espaço de busca será o fator de decisão. As estruturas 1, 2 e 3 possuem cardinalidade iguais a $(n-1)$, $n(n-1)/2$ e $n(n-1)^2$ para uma solução com n variáveis, o que nos mostra que uma busca pela vizinhança 3 geraria mais custo ao algoritmo. Com base nestes resultados, optou-se por utilizar o conjunto D nos algoritmos VNS, VNS-CM e VNS-CMVA, pelo fato das estruturas que compõem este comjunto possuir menor cardinalidade.

4.5.6 Ajustes dos Parâmetros do VNS

As versões implementadas do VNS possuem parâmetros relacionados ao VNS padrão (nível de perturbação de um vizinho da solução atual) e a metodologia desenvolvida (cardinalidade da busca local e indexação da tabela *hash*). Nesta seção, um teste é

realizado para definir qual é a melhor configuração dos valores desses parâmetros, para que a meta-heurística VNS seja aplicada de forma eficiente na resolução do problema de *scheduling*.

Os parâmetros que precisam ser ajustados no algoritmo são:

$Npert$: indica o nível de perturbação do VNS. Este parâmetro, que está relacionado ao vizinho gerado pelo VNS para fazer a busca local, representa a quantidade de vezes que um movimento é aplicado na solução atual s para gerar o vizinho s'.

b_{ind}**:** indica o tamanho da indexação da tabela *hash*. Este parâmetro define o tamanho da tabela *hash* que é calculado usando a seguinte equação: $M = 2^{b_{ind}} - 1$.

*Card***:** indica o tamanho da busca realizada nas vizinhanças de uma solução atual. A exploração das soluções em uma estrutura de vizinhança na busca local é interrompida quando é encontrada a primeira solução de melhora ou quando é realizada a busca por x vizinhos nesta vizinhança. O tamanho deste espaço é determinado pelo parâmetro de cardinalidade *Card*. No trabalho, para que a busca não seja exaustiva, o tamanho do espaço de busca é calculado considerando uma porcentagem do tamanho da estrutura de menor cardinalidade, ou seja:

Vizinhança 1: $Card \cdot (n-1)$; sendo $n-1$ a quantidade de vizinhos gerados nesta estrutura.

Vizinhança 2: $3\times$(tamanho da busca na vizinhança 1);

Vizinhança 3: $3^2\times$(tamanho da busca na vizinhança 1);

Vizinhança 4: $3^3\times$(tamanho da busca na vizinhança 1);

Um planejamento de experimentos foi realizado com o objetivo de encontrar a melhor combinação entre esses três parâmetros. O Planejamento de Experimentos (DOE) ajuda a investigar os efeitos das variáveis de entrada (fatores) na variável de saída (resposta). Uma série de testes, onde são feitas alterações nas variáveis de entradas, é realizada e são coletadas e analisadas as informações obtidas na resposta. No presente trabalho, o planejamento realizado é classificado como planejamento fatorial. Este tipo de experimento permite estimar a significância de efeitos principais e efeitos de interação entre os fatores analisados.

No experimento, a variável de resposta é o valor de função objetivo, informação obtida nas execuções dos algoritmos VNS, VNS-CM e VNS-CMVA considerando o conjunto de estruturas de vizinhanças obtido na seção anterior. Os fatores analisados e seus respectivos valores, são: Nível de perturbação $Npert = 1,\ 2$ e 3 ; indexação da tabela *hash* $B_{ind} = 14,\ 15$ e 16 e parâmetro da cardinalidade $Card = 0,10,\ 0,3$ e $0,50$. Foram feitas 30 replicações, para um problema de cada tamanho. São apresentados as análises dos resultados obtidos para um problema de tamanho 100.

A análise fatorial do experimento mostrou que, nos três algoritmos, nenhum dos efeitos são estatisticamente significativos para α padrão igual a 5%. A Figura 4.12

apresenta o valor absoluto dos efeitos no diagrama de Pareto obtido pelo experimento, nos três algoritmos. Este diagrama é outra evidência de que nenhum dos fatores se mostraram estatisticamente significativos, visto que nenhum deles atinge a linha de referência. No entanto é preciso observar que os fatores podem ser pouco significativos e não acusar no teste.

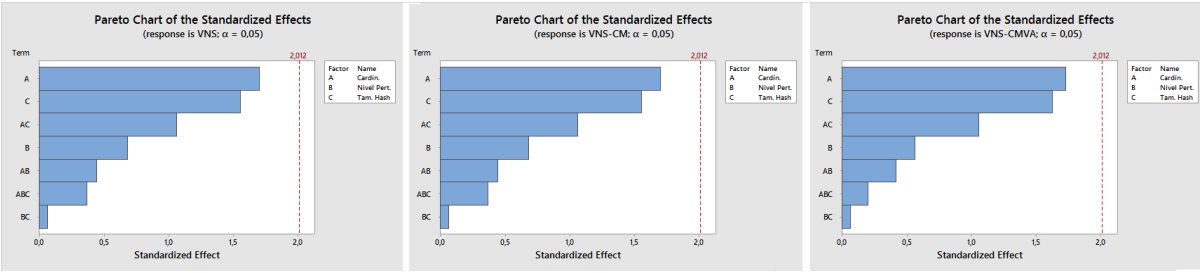


Figura 4.12: Diagrama de Pareto obtido pelo DOE

A Figura 4.13 mostra o impacto dos três fatores, (a cardinalidade, o nível de perturbação e o tamanho da indexação da tabela *hash*) na resposta final dos algoritmos VNS, VNS-CM e VNS-CMVA. A linha central horizontal mostra o resultado médio de todos os ensaios. Apesar da não acusação de efeitos pelos fatores estudados, é possível analisar o efeito de cada fator individualmente e obter uma combinação que gere o menor valor de função nos três algoritmos, sendo eles $Card = 0,5$, $Npert = 1$ e $B_{ind} = 16$.

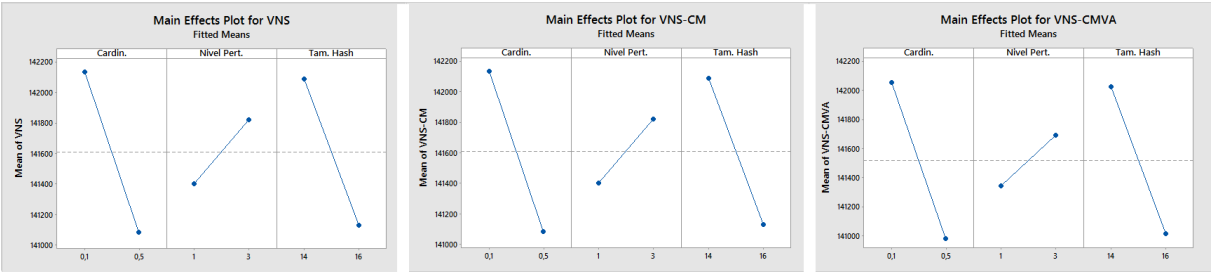


Figura 4.13: Efeito dos fatores principais nos algoritmos VNS, VNS-CM e VNS-CMVA

Uma interação significativa entre os fatores poderia afetar a interpretação dos efeitos principais. No entanto, como a interação não é significativa, não é necessário examinar o gráfico de interação entre os fatores estudados.

Os resultados obtidos para os problemas de tamanhos 50, 200 e 500 foi o mesmo. Assim, os parâmetros obtidos são usados nas seções seguintes para resolver todos os problemas-teste.

4.5.7 Resultados Finais

Uma vez determinados o conjunto de estruturas de vizinhança (seção 4.5.5) e os valores para os parâmetros (seção 4.5.6), as versões VNS-CM e VNS-CMVA e a versão padrão VNS, são usadas para resolver o problema de *scheduling* com data de entrega comum. Nesta seção são apresentados os resultados obtidos pelos três algoritmos ao

resolver 80 instâncias-teste com 50, 100, 200 e 500 tarefas e considerando valores de $h = 0,2$ e $h = 0,4$. Nos testes computacionais foram realizadas 30 execuções para cada problema e cada algoritmo, e o critério de parada foram iterações.

Em cada execução usou-se a função $rng(see)$ do MATLAB, que tem por finalidade controlar a geração de números aleatórios. Essa função usa um valor *see*, inteiro não negativo, que equivale a uma semente do gerador de números aleatórios para que funções como *rand*, *randn* e *randi* produzam uma sequência de números previsível. Assim, partindo do mesmo valor de semente *see* em cada uma das 30 execuções, garantimos que as versões do VNS tenham comportamentos parecidos.

Comparação entre as versões do VNS

As Tabelas 4.1 e 4.2 apresentam os resultados médios das soluções obtidas para todos os problemas após a aplicação dos algoritmos VNS, VNS-CM e VNS-CMVA. As colunas estão dispostas da seguinte maneira: na primeira coluna, indicada por problema teste, está o nome da instância; as colunas 2 e 5 indicam os resultados obtidos pelo VNS padrão, e as demais colunas os resultados obtidos pelas versões VNS-CM e VNS-CMVA. Os resultados apresentados nas colunas 2-4 são obtidos considerando o valor $h = 0,2$ e as colunas 5-7 para valor de $h = 0,4$.

Na Tabela 4.1 são apresentados os resultados para os problemas com 50 e 100 tarefas. Observa-se que, em todos os problemas, a média das soluções encontradas pelo VNS-CM é sempre a mesma média das soluções obtidas pelo VNS padrão. Este resultado é esperado, visto que a diferença entre os dois algoritmos está apenas na estratégia de memória usada para evitar reavaliações. Dos 40 problemas apresentados nesta tabela, o VNS-CMVA obteve média menor que as versões VNS e VNS-CM, em 28 deles. As menores médias encontradas estão destacadas em negrito. Os resultados obtidos para os problemas com 200 e 500 tarefas são apresentados na Tabela 4.2. Nota-se que o comportamento dos algoritmos se manteve para os problemas com maior quantidade de tarefas: o VNS-CM possui as mesmas médias que o VNS e o VNS-CMVA obteve média menor, que os demais algoritmos, para 29 dos 40 problemas apresentados.

O número médio de avaliações de função realizadas pelos algoritmos VNS, VNS-CM e VNS-CMVA em 30 execuções, para cada um dos problemas, é apresentado na Tabela 4.3. Nesta tabela, as colunas estão dispostas da seguinte maneira: na primeira coluna, indicada por problema teste, está o nome da instância; as colunas 2 e 5 indicam a quantidade de avaliações realizadas pelo VNS padrão, e nas demais colunas, as quantidades de avaliações realizadas pelas demais versões do VNS. As colunas 2-4 apresentam as médias considerando $h = 0,2$ e as colunas 5-7 considerando $h = 0,4$.

Em relação à quantidade de avaliações de função realizadas por cada versão do VNS para encontrar a solução final, é possível notar que os resultados obtidos pela metodologia proposta se mostraram satisfatórios, à medida em que o VNS-CM obtém os

Tabela 4.1: Solução média das 30 execuções de cada versão do VNS para os problemas com 50 e 100 tarefas.

Problema	h=0,2			h=0,4		
Teste	VNS	VNS-CM	VNS-CMVA	VNS	VNS-CM	VNS-CMVA
sch50k1	38.781	38.781	**38.765**	22298	22298	**22292**
sch50k2	29.184	29.184	**29.161**	**16976**	**16976**	16993
sch50k3	32.321	32.321	**32.307**	19059	19059	**19034**
sch50k4	26.674	26.674	**26.650**	**15752**	**15752**	15780
sch50k5	**30.496**	**30.496**	30.507	17171	17171	**17156**
sch50k6	33.122	33.122	**33.119**	19376	19376	**19343**
sch50k7	41.289	41.289	**41.287**	**21655**	**21655**	21655
sch50k8	42.154	42.154	**42.134**	24060	24060	**24042**
sch50k9	**32.019**	**32.019**	32.023	18703	18703	**18682**
sch50k10	**31.468**	**31.468**	31.471	18232	18232	**18207**
sch100k1	142.040	142.040	**142.014**	83.905	83.905	**83.873**
sch100k2	**122.498**	**122.498**	122.517	71.285	71.285	**71.283**
sch100k3	126.614	126.614	**126.546**	**77.435**	**77.435**	77.516
sch100k4	126.141	126.141	**126.107**	77.337	77.337	**77.330**
sch100k5	**120.557**	**120.557**	120.587	68.554	68.554	**68.500**
sch100k6	135.651	135.651	**135.641**	76.341	76.341	**76.227**
sch100k7	**132.950**	**132.950**	132.994	75.956	75.956	**75.920**
sch100k8	156.224	156.224	**156.204**	**92.248**	**92.248**	92.298
sch100k9	114.541	114.541	**114.378**	**67.623**	**67.623**	67.699
sch100k10	116.133	116.133	**116.120**	70.038	70.038	**69.930**

Tabela 4.2: Solução média das 30 execuções de cada versão do VNS para os problemas com 200 e 500 tarefas.

Problema	h=0,2			h=0,4		
Teste	VNS	VNS-CM	VNS-CMVA	VNS	VNS-CM	VNS-CMVA
sch200k1	**490.826**	**490.826**	**490.826**	**291.946**	**291.946**	292.216
sch200k2	533.254	533.254	**533.230**	315.091	315.091	**315.058**
sch200k3	481.932	481.932	**481.862**	289.989	289.989	**289.930**
sch200k4	577.703	577.703	**577.702**	**347.874**	**347.874**	348.023
sch200k5	504.905	504.905	**504.888**	301.001	301.001	**300.934**
sch200k6	**474.063**	**474.063**	474.085	277.447	277.447	**277.255**
sch200k7	449.802	449.802	**449.624**	271.263	271.263	**271.213**
sch200k8	485.966	485.966	**485.881**	**275.218**	**275.218**	275.364
sch200k9	520.999	520.999	**520.932**	**306.704**	**306.704**	306.881
sch200k10	**531.245**	**531.245**	531.329	319.202	319.202	**319.176**
sch500k1	2.942.033	2.942.033	**2.942.029**	1.780.363	1.780.363	**1.780.220**
sch500k2	3.346.220	3.346.220	**3.345.848**	1.986.427	1.986.427	**1.986.264**
sch500k3	**3.094.061**	**3.094.061**	3.094.191	1.858.539	1.858.539	**1.858.482**
sch500k4	3.209.232	3.209.232	**3.209.080**	**1.879.918**	**1.879.918**	1.880.735
sch500k5	3.098.982	3.098.982	**3.098.900**	1.797.754	1.797.754	**1.797.522**
sch500k6	2.777.437	2.777.437	**2.777.305**	1.796.803	1.796.803	**1.796.694**
sch500k7	3.155.511	3.155.511	**3.155.050**	**1.894.570**	**1.894.570**	1.894.827
sch500k8	**3.105.995**	**3.105.995**	3.106.417	1.799.341	1.799.341	**1.799.335**
sch500k9	3.345.850	3.345.850	**3.345.632**	1.965.577	1.965.577	**1.964.995**
sch500k10	3.105.946	3.105.946	**3.105.622**	1.831.335	1.831.335	**1.831.004**

mesmos resultados que o VNS padrão para todos os problemas testados, porém com uma quantidade de avaliações menor. A estratégia de memória foi incorporada ao algoritmo para diminuir o número de avaliações realizadas e reduzir o esforço computacional em relação às avaliações de função. Estes resultados mostram que o objetivo da estratégia foi alcançado, visto que para os 40 problemas com valores de $h = 0,2$ o VNS-CM realizou em média $19,42\%$ a menos de avaliações que o VNS. Para os mesmos problemas com valores de $h = 0,4$ a redução do número de avaliações foi em média de $19,26\%$.

Sobre o VNS-CMVA, é possível notar que em todos os testes este algoritmo obteve um número médio de avaliações maior que o VNS-CM, porém, como mostram as Tabelas 4.1 e 4.2, com uma média de soluções menor para a maioria dos problemas. O número maior de avaliações de função, em relação ao VNS-CM, é justificado pelo operador de vizinhança adaptativo, que realiza novas buscas pelo espaço de soluções toda vez que encontra uma solução já visitada. Durante essa busca, uma nova solução é encontrada e avaliada pelo algoritmo. Ao comparar o número de avaliações realizadas pelos algoritmos VNS e VNS-CMVA, podemos notar que para a maioria dos problemas o VNS-CMVA encontra soluções melhores que o VNS, com um número menor de avaliações. Esta última evidência está indicada na tabela pela cor vermelha.

Em comparação ao VNS padrão, o algoritmo VNS-CMVA obteve média menor em 30 dos 40 problemas considerando $h = 0,2$, com uma redução em torno de $1,53\%$ do número de avaliações. Para $h = 0,4$ essa redução foi de $1,85\%$ em 26 dos 40 problemas analisados.

Assim, de acordo com os resultados apresentados, temos evidências de que as novas estratégias incorporadas a meta-heurística VNS contribuem para a melhoria do desempenho do algoritmo, na medida em que: com o acréscimo de uma estrutura de memória o algoritmo (VNS-CM) consegue encontrar as mesmas soluções que a versão clássica com uma redução de quase 20% do número de avaliações do algoritmo, e que com o acréscimo de uma estrutura de memória e vizinhança adaptativa o algoritmo (VNS-CMVA) consegue encontrar soluções melhores e com uma redução de quase 2% do número de avaliações que o VNS.

A Figura 4.14 apresenta o gráfico das curvas médias de cada versão do VNS, em 30 execuções, para um problema de tamanho 50. A curva média é dada em relação a solução média encontrada versus o número médio de avaliações de função nas 100 iterações. A análise da curva média reforça os resultados apresentados nas Tabelas 4.1, 4.2 e 4.3, onde o VNS-CM sempre encontra as mesmas soluções que o VNS clássico com um número menor de avaliações de funções, enquanto o VNS-CMVA encontra soluções melhores que o VNS e VNS-CM, com um número de avaliações menor que VNS e maior que o VNS-CM.

Tabela 4.3: Número médio de avaliações de função utilizadas pelos algoritmos para resolver os problemas-teste.

	h=0,2			h=0,4		
Problema	**VNS**	**VNS-CM**	**VNS-CMVA**	**VNS**	**VNS-CM**	**VNS-CMVA**
sch50k1	210.219	**149.167**	209.454	210.579	**149.841**	210.871
sch50k2	209.251	**150.449**	210.601	211.884	**151.603**	210.174
sch50k3	209.926	**148.219**	211.276	209.971	**148.884**	209.836
sch50k4	211.096	**150.644**	211.119	209.904	**148.547**	210.556
sch50k5	208.689	**147.234**	208.959	210.961	**150.053**	210.039
sch50k6	211.647	**150.295**	210.871	212.154	**151.507**	210.567
sch50k7	209.184	**147.938**	208.914	211.884	**150.161**	210.297
sch50k8	126.483	**91.820**	127.455	128.204	**93.769**	127.171
sch50k9	127.637	**92.664**	127.070	128.913	**94.633**	129.095
sch50k10	127.090	**92.038**	127.313	128.427	**93.964**	127.212
sch100k1	139.699	**113.444**	138.160	137.809	**111.912**	136.918
sch100k2	137.782	**111.877**	138.754	137.431	**111.602**	137.971
sch100k3	137.350	**111.241**	139.051	138.997	**112.912**	140.563
sch100k4	137.404	**111.413**	137.458	137.755	**111.774**	137.512
sch100k5	137.269	**111.300**	136.756	140.536	**114.147**	140.023
sch100k6	136.135	**110.308**	135.136	140.374	**114.162**	139.024
sch100k7	136.459	**110.538**	134.839	137.458	**111.402**	139.483
sch100k8	137.431	**111.603**	136.081	137.890	**111.977**	136.270
sch100k9	137.323	**111.352**	136.054	138.970	**112.981**	137.431
sch100k10	137.674	**111.824**	136.783	138.916	**112.900**	139.186
sch200k1	192.133	**160.714**	190.934	194.077	**162.355**	188.375
sch200k2	189.897	**158.893**	188.666	191.485	**160.172**	196.215
sch200k3	192.619	**161.093**	191.582	196.572	**164.473**	194.369
sch200k4	190.610	**159.460**	187.435	194.369	**162.599**	192.133
sch200k5	192.878	**161.282**	186.819	195.567	**163.672**	192.457
sch200k6	195.081	**163.267**	189.671	193.591	**161.921**	194.693
sch200k7	190.902	**159.687**	190.092	197.382	**165.113**	193.818
sch200k8	189.476	**158.375**	186.528	194.984	**163.070**	191.777
sch200k9	190.319	**159.109**	189.865	192.263	**160.844**	192.003
sch200k10	193.559	**161.828**	194.401	194.369	**162.544**	190.869
sch500k1	582.067	**489.184**	576.316	229.069	**197.646**	225.419
sch500k2	220.688	**190.445**	214.856	236.175	**203.853**	219.867
sch500k3	223.064	**192.518**	219.111	235.700	**203.429**	227.967
sch500k4	224.360	**193.627**	214.770	236.391	**204.040**	210.774
sch500k5	228.183	**196.960**	220.969	232.158	**200.386**	231.229
sch500k6	223.647	**193.069**	221.120	235.247	**203.064**	231.467
sch500k7	229.479	**198.059**	226.110	223.993	**193.275**	224.036
sch500k8	226.823	**195.788**	212.631	235.592	**203.390**	229.976
sch500k9	228.378	**197.096**	224.641	231.423	**199.773**	223.453
sch500k10	226.671	**195.646**	218.226	236.975	**204.532**	227.687

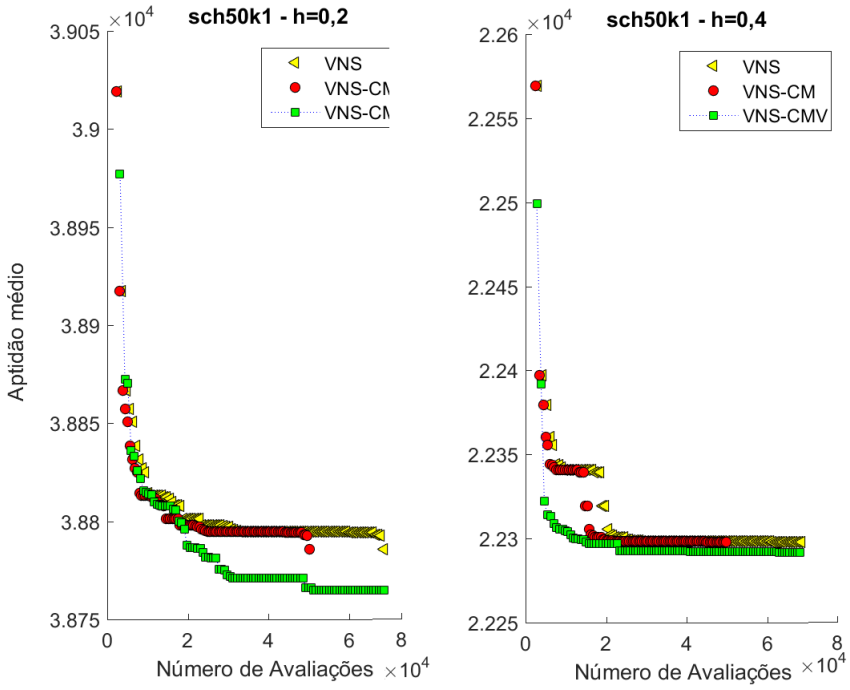


Figura 4.14: Gráfico das curvas médias de cada versão do VNS para 30 execuções.

Comparação com as Soluções da Literatura

Os resultados obtidos pelo VNS-CM e VNS-CMVA foram comparados com os limitantes superiores relatados na literatura [64]. O desvio médio percentual ($dev\%$) com relação ao limitante superior foi utilizado como medida de desempenho:

$$dev = 100 \times \frac{SOL_{VNS} - LS}{LS} \tag{4-5}$$

em que SOL_{VNS} é a melhor solução encontrada nas 30 execuções do VNS e LS é a solução de referência da literatura. De acordo com a fórmula utilizada, valores iguais a zero indicam que não houve variação de solução e valores negativos indicam que a solução encontrada foi menor que o limitante superior, ou seja, indica a superioridade da meta-heurística proposta.

A Tabela 4.4 apresenta a melhoria das soluções obtidas em cada problema teste, para as versões VNS-CM e VNS-CMVA, em relação ao limitante superior obtido em [64]. Os resultados mostram que houve melhoria nos resultados das soluções encontradas pelas versões VNS-CM e VNS-CMVA em quase todos os problemas testados, com exceção do problema $sch500k6$, indicando a superioridade do método em relação a referência da literatura. O desvio relativo das versões VNS-CM e VNS-CMVA apresentaram variações de melhoria nas soluções de $1,06\%$ à $10,48\%$ para os problemas resolvidos com $h = 0,2$ e $1,51\%$ à $11,91\%$ para os problemas resolvidos com $h = 0,4$. Dessa forma, podemos dizer que o VNS-CM e VNS-CMVA obtiveram resultados superiores aos limitantes da literatura.

Tabela 4.4: Melhoria das soluções obtidas pelo VNS-CM e VNS-CMVA em relação ao limitante superior

Problemas	**h=0,2**		**h=0,4**	
Teste	**VNS-CM**	**VNS-CMVA**	**VNS-CM**	**VNS-CMVA**
sch50k1	-8,50	-8,46	-8,91	-9,41
sch50k2	-8,79	-8,88	-10,83	-10,90
sch50k3	-9,54	-9,61	-11,91	-11,91
sch50k4	-10,25	-10,25	-10,32	-10,17
sch50k5	-9,61	-9,53	-10,91	-10,91
sch50k6	-10,30	-10,30	-10,32	-10,48
sch50k7	-9,34	-9,35	-11,66	-11,66
sch50k8	-7,22	-7,22	-6,96	-6,99
sch50k9	-10,10	-10,07	-11,02	-11,47
sch50k10	-10,48	-10,48	-5,37	-5,42
sch100k1	-5,36	-5,36	-4,93	-4,78
sch100k2	-4,73	-4,73	-5,64	-5,76
sch100k3	-5,46	-5,46	-4,80	-5,72
sch100k4	-6,08	-6,08	-5,85	-5,91
sch100k5	-5,54	-5,54	-6,26	-5,95
sch100k6	-5,25	-5,25	-2,63	-3,41
sch100k7	-6,29	-6,29	-5,48	-5,25
sch100k8	-5,42	-5,42	-5,33	-5,30
sch100k9	-6,00	-6,00	-5,78	-5,79
sch100k10	-5,20	-5,20	-4,46	-4,64
sch200k1	-3,22	-3,19	-3,36	-3,36
sch200k2	-5,23	-5,22	-3,46	-3,46
sch200k3	-4,28	-4,27	-3,17	-3,17
sch200k4	-4,53	-4,57	-3,29	-3,29
sch200k5	-4,66	-4,64	-3,05	-3,05
sch200k6	-4,65	-4,68	-4,79	-4,79
sch200k7	-5,90	-5,86	-2,92	-2,92
sch200k8	-4,99	-4,97	-2,70	-2,70
sch200k9	-5,30	-5,32	-3,77	-3,77
sch200k10	-3,32	-3,33	-3,39	-3,39
sch500k1	-2,20	-2,20	-1,97	-2,02
sch500k2	-1,81	-1,81	-2,09	-2,14
sch500k3	-1,24	-1,24	-1,53	-1,51
sch500k4	-1,50	-1,50	-2,76	-2,69
sch500k5	-1,35	-1,35	-1,95	-1,89
sch500k6	-1,45	-1,45	9,38	9,40
sch500k7	-1,52	-1,52	-2,50	-2,42
sch500k8	-1,06	-1,06	-2,15	-2,28
sch500k9	-1,38	-1,38	-1,98	-2,07
sch500k10	-1,51	-1,51	-2,45	-2,39

A Tabela 4.5 apresenta as melhores soluções obtidas pelos algoritmos VNS-CM e VNS-CMVA e os resultados apresentados em [64]. A primeira coluna indica os problemas-teste, as colunas 2 e 5, os limitantes superiores encontrados em [64] para os problemas resolvidos com $h = 0,2$ e $h = 0,4$, respectivamente. As colunas (3 e 4) e (5 e 7) apresentam os resultados obtidos pelas versões do VNS, para os valores de $h = 0,2$ e $h = 0,4$, nesta ordem.

O que se observa é que o VNS-CMVA encontra uma quantidade maior de limitantes que o VNS-CM. Porém, vários dos melhores resultados foram obtidos pelas três versões do VNS. Os novos limitantes superiores estão indicados na tabela em negrito.

Tabela 4.5: Resultados com os novos limitantes superiores obtidos pelas algoritmos propostos.

Problemas	**h=0,2**			**h=0,4**		
Teste	**LS**	**VNS-CM**	**VNS-CMVA**	**LS**	**VNS-CM**	**VNS-CMVA**
sch50k1	40.734	**37.271**	37.289	23.792	21671	**21553**
sch50k2	30.739	28.037	**28.008**	18.042	16.088	**16.076**
sch50k3	34.505	31.214	**31.190**	20.700	**18235**	**18235**
sch50k4	27.803	**24.953**	**24.953**	16.693	**14.970**	14.996
sch50k5	32.332	**29.226**	29.251	18.167	**16.185**	**16.185**
sch50k6	35.102	31.511	**31.486**	20.402	18.296	**18.263**
sch50k7	43.229	39.190	**39.188**	23.228	**20.520**	**20.520**
sch50k8	43.969	**40.796**	**40.796**	24.947	23.211	**23.203**
sch50k9	34.326	**30.858**	30.870	20.008	17803	**17713**
sch50k10	32.999	**29.542**	**29.542**	18.238	17259	**17250**
sch100k1	146.132	**138.293**	**138.293**	86.280	**82.025**	82.154
sch100k2	125.331	**119.398**	**119.398**	73.459	69.315	**69.225**
sch100k3	130.414	**123.296**	**123.296**	80.207	**76.359**	75.617
sch100k4	131.132	**123.160**	**123.160**	79.947	75.271	**75.222**
sch100k5	124.882	**117.967**	**117.967**	71.609	67.349	**67.129**
sch100k6	139.961	**132.619**	**132.619**	77.087	75.058	**74.462**
sch100k7	137.407	**128.764**	**128.764**	78.410	**74.116**	74.290
sch100k8	161.424	**152.678**	**152.678**	95.612	90.542	**90.516**
sch100k9	118.859	**111.727**	**111.727**	69.812	65.778	**65.771**
sch100k10	119.795	**113.567**	**113.567**	72.389	69.160	**69.027**
sch200k1	502.920	**486.725**	486.891	298.080	**288.056**	**288.056**
sch200k2	556.476	527.409	**527.355**	322.705	**311.543**	**311.543**
sch200k3	497.396	476.153	**476.131**	296.023	**286.634**	**286.634**
sch200k4	599.074	571.944	**571.699**	356.197	**344.478**	**344.478**
sch200k5	524.601	**500.148**	500.240	306.819	**297.465**	**297.465**
sch200k6	490.165	467.380	**467.225**	287.009	**273.269**	**273.269**
sch200k7	470.485	**442.724**	442.894	275.805	**267.746**	**267.746**
sch200k8	505.432	**480.189**	480.319	280.383	**272.807**	**272.807**
sch200k9	545.248	516.354	**516.265**	315.441	**303.538**	**303.538**
sch200k10	544.871	526.762	**526.704**	327.299	**316.210**	**316.210**
sch500k1	2.995.837	**2.929.791**	**2.929.791**	1.809.150	1.773.475	**1.772.627**
sch500k2	3.396.297	**3.334.810**	**3.334.810**	2.019.880	1.977.671	**1.976.598**
sch500k3	3.114.316	**3.075.574**	**3.075.574**	1.877.994	**1.849.351**	1.849.619
sch500k4	3.243.457	**3.194.819**	**3.194.819**	1.923.249	1.871.531	**1.870.191**
sch500k5	3.127.020	**3.084.895**	**3.084.895**	1.824.148	1.789.733	**1.788.572**
sch500k6	2.807.257	**2.766.588**	**2.766.588**	**1.637.076**	1.791.023	**1.790.621**
sch500k7	3.193.413	**3.144.882**	**3.144.882**	1.931.460	**1.883.114**	1.884.763
sch500k8	3.127.525	**3.094.527**	**3.094.527**	1.829.059	1.789.802	**1.787.333**
sch500k9	3.376.250	**3.329.603**	**3.329.603**	1.995.866	1.956.437	**1.954.551**
sch500k10	3.137.808	**3.090.386**	**3.090.386**	1.867.701	1.823.046	**1.821.930**

Capítulo 5

Considerações Finais

5.1 Conclusão

Esta tese apresentou novos operadores adaptativos aplicados em meta-heurísticas para a resolução de problemas de otimização. Foram feitos dois estudos de caso que seguem linhas distintas. No primeiro estudo, são propostos operadores baseados em controle realimentado para solução de problemas de otimização multiobjetivo e desenvolvido um algoritmo SCMGA que faz uso de operadores baseados em controle do raio de esferas para garantir uma melhor distribuição do conjunto de estimativas de Pareto. No segundo estudo, é proposto um operador de vizinhança adaptativo baseado em estratégia de memória para problemas de otimização combinatória.

Os operadores propostos na primeira parte do trabalho são baseados na ideia de controle realimentado, a fim de estabelecer e manter um equilíbrio dinâmico das soluções do conjunto de Pareto. Nesta abordagem, foram introduzidos dois novos operadores para os algoritmo genéticos multiobjetivo: um operador de redução de arquivo (*archive-set reduction*) e um operador de cruzamento (*surface-filling*). O primeiro operador é usado para controlar a quantidade de soluções não dominadas que serão armazenadas em um arquivo externo do algoritmo. O segundo operador é usado para preencher os espaços vazios do conjunto de soluções não-dominadas. A metodologia proposta ainda aplica uma técnica de controle para que sejam realizados auto-ajustes dos parâmetros dentro do próprio algoritmo. Os dois operadores propostos foram incluídos em uma versão padrão do NSGA-II, aqui denominada de Algoritmo Genético Multiobjetivo com Controle por Esferas (SCMGA).

O SCMGA foi utilizado, neste trabalho, para resolver um problema de otimização multiobjetivo real, que consiste em obter um conjunto de configurações para um sistema de extrusão de polímero. As soluções encontradas são comparadas com a versão padrão do NSGA-II e o algoritmo RPSGA. De acordo com os resultados obtidos, podemos concluir que o SCMGA foi capaz de gerar uma descrição mais representativa por toda a extensão do conjunto de Pareto-Ótimo em relação aos demais algoritmos, o que proporciona uma amostragem de configurações para o sistema mais uniforme. Verificou-

se também uma diferença de desempenho do SCMGA em relação aos demais métodos quando usados em problemas com mais de duas funções objetivo. Esses resultados indicam que o algoritmo SCMGA proposto pode ser uma boa escolha como uma nova ferramenta para fornecer as informações detalhadas sobre o conjunto de estimativas do Pareto-Ótimo, a fim de auxiliar o decisor na escolha da melhor configuração em uma máquina de extrusão de polímeros. Além disso, é interessante destacar que os operadores desenvolvidos foram aplicados ao NSGA-II, mas podem ser usados em qualquer algoritmo genético multiobjetivo, que possua um arquivo externo.

No segundo estudo, um operador de vizinhança adaptativo, baseado em uma estrutura de memória, foi desenvolvido a fim de melhorar o desempenho das meta-heurísticas convencionais de busca local. A estratégia de memória utilizada neste trabalho faz uso de uma tabela *hash*. Esta estrutura de dados tem por finalidade armazenar todas as soluções geradas pelo algoritmo, para que estas informações sejam usadas, durante todo o procedimento, a fim de impedir avaliações de uma mesma solução diversas vezes pelo algoritmo. O operador de vizinhança adaptativo faz uso da ideia de vizinhanças adaptativas, para realizar uma nova busca local por regiões que ainda não foram exploradas pelo algoritmo, e utiliza a estrutura de memória para decidir quando fazer a busca por novas regiões.

A metodologia proposta neste segundo estudo consiste em duas novas versões do VNS, os algoritmos VNS-CM e VNS-CMVA, que são versões adaptativas do VNS caracterizados por apresentarem uma memória e o novo operador de vizinhança adaptativo. As novas versões do VNS foram aplicadas para resolver um problema de *scheduling* com data de entrega comum em uma única máquina. Para este problema, após fazer um ajuste dos parâmetros por meio de testes estatísticos, os resultados encontrados pelo VNS clássico e as duas versões adaptativas foram comparados. Os resultados permitem concluir que as versões adaptativas se mostram eficientes por encontrarem soluções de boa qualidade e com uma redução considerável no número de avaliações de função. O VNS-CM reduziu o número de avaliações em quase 20% e o VNS-CMVA em torno de 2%, quando comparados a versão padrão do VNS. No entanto, o VNS-CMVA obteve médias menores para a maioria dos problemas. Uma comparação com os resultados da literatura, mostra que ambas as versões foram eficientes. As soluções encontradas mostram melhoria em torno de até 12% em 39 dos 40 problemas-teste. As novas versões do VNS encontram novos limitantes superiores para quase todos os problemas.

Esses resultados indicam que o uso de estratégias de memória e vizinhança adaptativa, aplicadas em meta-heurística para resolver o problema de *scheduling*, se mostraram satisfatórios. Além disso, recomenda-se o uso destas estratégias para qualquer meta-heurística de busca local. Nos casos em que a função objetivo é considerada custosa, a diferença entre as versões do VNS deve ser mais evidente, visto que as técnicas empregadas diminuem o número de avaliações realizadas pelo algoritmo. Neste trabalho,

foi escolhido o VNS, mas as técnicas aqui desenvolvidas podem ser facilmente aplicadas à qualquer outra meta-heurística.

5.2 Trabalhos Futuros

Para o estudo de caso 1, que envolve operadores para MOEAS, as sugestões são:

- Testar os novos operadores e as técnicas de auto ajuste dos parâmetros em outros algoritmos multiobjetivo.
- Aplicar a ideia de memória nos operadores para auxiliar o ajuste dos parâmetros e escolha dos indivíduos que receberão as operações de cruzamento e mutação.
- Aplicar a metodologia em novos problemas reais.

Para o estudo de caso 2, que envolve operadores para meta-heurísticas de busca local, as sugestões são:

- Implementar técnicas de controle para o auto-ajuste dos parâmetros.
- Usar mecanismos para melhorar a eficiência do operador de vizinhança adaptativo na exploração das regiões do espaço de busca.
- Testar novas estruturas de vizinhanças.
- Melhorar o uso da estrutura de memória para fazer avaliações por aproximação e escolher de forma adaptativa qual região para se fazer a busca por novas soluções.
- Testar os operadores em outras meta-heurísticas.
- Aplicar em um problema prático com a função de avaliação custosa.

5.3 Produção Bibliográfica

CARRANO, Eduardo.G., Coelho, D.G., Gaspar-Cunha, A., Wanner, E.F., Takahashi, R.H.C. *Feedback-control operators for improved Pareto-set description: Application to a polymer extrusion process*. Engineering Applications of Artificial Intelligence, v. 38, p. 147-167, 2015.

Referências Bibliográficas

[1] ALVAREZ-VALDÉS, R.; PARAJÓN, A.; TAMARIT, J. M. **A tabu search algorithm for large-scale guillotine (un)constrained two-dimensional cutting problems**. *Computers & Operations Research*, 29(7):925 – 947, 2002.

[2] ATKINSON, R.; SHIFFRIN, R. **Human memory: A proposed system and its control processes1**. volume 2 de **Psychology of Learning and Motivation**, p. 89 – 195. Academic Press, 1968.

[3] BEAN, J. C. **Genetic algorithms and random keys for sequencing and optimization**. *ORSA journal on computing*, 6(2):154–160, 1994.

[4] BEYER, H.-G.; SENDHOFF, B. **Covariance matrix adaptation revisited - the CMSA evolution strategy**. In: *International Conference on Parallel Problem Solving from Nature*, p. 123 – 132. Springer, 2008.

[5] BISKUP, D.; FELDMANN, M. **Benchmarks for scheduling on a single machine against restrictive and unrestrictive common due dates**. *Computers & Operations Research*, 28:787–801, 2001.

[6] BUI, L. T.; ABBASS, H. A.; ESSAM, D. **Local models - an approach to distributed multi-objective optimization**. *Computational Optimization and Applications*, 42(1):105 – 139, 2009.

[7] CARRANO, E. G.; COELHO, D. G.; GASPAR-CUNHA, A.; WANNER, E. F.; TAKAHASHI, R. H. **Feedback-control operators for improved Pareto-set description: Application to a polymer extrusion process**. *Engineering Applications of Artificial Intelligence*, 38:147 – 167, 2015.

[8] CARRANO, E. G.; MOREIRA, L. A.; TAKAHASHI, R. **A new memory based variable-length encoding genetic algorithm for multiobjective optimization**. In: *Evolutionary Multi-Criterion Optimization*, volume 6576 de **Lecture Notes in Computer Science**, p. 328–342. Springer Berlin Heidelberg, 2011.

[9] CARRANO, E. G.; TAKAHASHI, R. H.; FONSECA, C. M.; NETO, O. M. **Nonlinear network optimization-an embedding vector space approach**. *IEEE Transactions on Evolutionary Computation*, 14(2):206–226, 2010.

[10] CHARALAMPAKIS, A. E. **Registrar: a complete-memory operator to enhance performance of genetic algorithms**. *Journal of Global Optimization*, 54(3):449–483, 2012.

[11] CHOW, C. K.; YUEN, S. Y. **A non-revisiting particle swarm optimization**. In: *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, p. 1879–1885. IEEE, 2008.

[12] CHOW, C. K.; YUEN, S. Y. **Continuous non-revisiting genetic algorithm with random search space re-partitioning and one-gene-flip mutation**. In: *Evolutionary Computation (CEC), 2010 IEEE Congress on*, p. 1 –8, july 2010.

[13] CHOW, C. K.; YUEN, S. Y. **An evolutionary algorithm that makes decision based on the entire previous search history**. *IEEE Transctions on Evolutionary Computation*, 15:741–769, 2011.

[14] CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. **Introduction to algorithms**. The MIT Press, second edition edition, 2001.

[15] COSTA, T. A. **Método para Sequenciamento de Tarefas em Sistemas Flexíveis de Manufatura Baseado em Metaheurísticas e Controle Supervisório**. PhD thesis, Programa de Pós-Graduação em Engenharia Elétrica, Universidade Federal de Minas Gerais, 2015.

[16] DA CUNHA, A. G. L. **Modelling and optimisation of single screw extrusion**. PhD thesis, Universidade do Minho, 1999.

[17] DAVIS, L. **Adapting operator probabilities in genetic algorithms**. In: *proc. 3rd International conference on genetic algorithms*, p. 61–69, 1989.

[18] DEB, K. **Multi-objective optimization using evolutionary algorithms**, volume 16. John Wiley & Sons, 2001.

[19] DEB, K.; AGRAWAL, R. B. **Simulated binary crossover for continuous search space**. *Complex Systems*, 9(3):1–15, 1994.

[20] DEB, K.; JAIN, H. **An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints**. *IEEE Transactions on Evolutionary Computation*, 18(4):577–601, 2014.

[21] DEB, K.; PRATAP, A.; AGARWAL, S.; MEYARIVAN, T. **A fast and elitist multiobjective genetic algorithm: NSGA-II**. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[22] DEB, K.; THIELE, L.; LAUMANNS, M.; ZITZLER, E. **Scalable test problems for evolutionary multiobjective optimization**. Springer, 2005.

[23] EIBEN, Á. E.; HINTERDING, R.; MICHALEWICZ, Z. **Parameter control in evolutionary algorithms**. *IEEE Transactions on evolutionary computation*, 3(2):124–141, 1999.

[24] EIBEN, A. E.; MICHALEWICZ, Z.; SCHOENAUER, M.; SMITH, J. E. **Parameter control in evolutionary algorithms**. In: *Parameter setting in evolutionary algorithms*, p. 19–46. Springer, 2007.

[25] FELDMANN, M.; BISKUP, D. **Single-machine scheduling for minimizing earliness and tardiness penalties by meta-heuristic approaches**. *Computers & Industrial Engineering*, 44(2):307–323, 2003.

[26] FEO, T. A.; RESENDE, M. G. **Greedy randomized adaptive search procedures**. *Journal of global optimization*, 6(2):109–133, 1995.

[27] FIALHO, Á. R. S. **Adaptive operator selection for optimization**. PhD thesis, Citeseer, 2011.

[28] FONG, K.; YUEN, S.; CHOW, C. K.; LEUNG, S. W. **Energy management and design of centralized air-conditioning systems through the non-revisiting strategy for heuristic optimization methods**. *Applied Energy*, 87(11):3494–3506, 2010.

[29] FONSECA, C. M.; FLEMING, P. J. **An overview of evolutionary algorithms in multiobjective optimization**. *Evolutionary computation*, 3(1):1–16, 1995.

[30] FONSECA, C. M.; FLEMING, P. J. **On the performance assessment and comparison of stochastic multiobjective optimizers**. In: *International Conference on Parallel Problem Solving from Nature*, p. 584–593. Springer, 1996.

[31] GASPAR-CUNHA, A.; COVAS, J. A. **RPSGe – educed pareto set genetic algorithm: application to polymer extrusion**. In: *Metaheuristics for Multiobjective Optimisation*, p. 221 – 249. Springer, 2004.

[32] GILOVICH, T.; GRIFFIN, D.; KAHNEMAN, D. **Heuristics and biases: The psychology of intuitive judgment**. Cambridge university press, 2002.

[33] GLOVER, F. **Future paths for integer programming and links to artificial intelligence**. *Computers & operations research*, 13(5):533–549, 1986.

[34] GOLDBARG, M. C.; GOLDBARG, E. G.; LUNA, H. P. L. **Otimização Combinatória e Meta-Heurísticas: Algoritmos e Aplicações**. 1 edition, 2016.

[35] GORDON, V.; PROTH, J.-M.; CHU, C. **A survey of the state-of-the-art of common due date assignment and scheduling research**. *European Journal of Operational Research*, 139(1):1–25, 2002.

[36] HALL, N. G.; KUBIAK, W.; SETHI, S. P. **Earliness–tardiness scheduling problems, ii: Deviation of completion times about a restrictive common due date**. *Operations Research*, 39(5):847–856, 1991.

[37] HANSEN, N.; OSTERMEIER, A. **Completely derandomized self-adaptation in evolution strategies**. *Evolutionary computation*, 9(2):159–195, 2001.

[38] HANSEN, P. **The steepest ascent mildest descent heuristic for combinatorial programming**. In: *Congress on numerical methods in combinatorial optimization, Capri, Italy*, p. 70–145, 1986.

[39] HUNING, A.; RECHENBERG, I.; EIGEN, M. **Evolutionsstrategie. optimierung technischer systeme nach prinzipien der biologischen evolution**, 1976.

[40] JAMES, R. **Using tabu search to solve the common due date early/tardy machine scheduling problem**. *Computers & Operations Research*, 24(3):199–208, 1997.

[41] JEBARI, K.; BOUROUMI, A.; ETTOUHAMI, A. **Parameters control in GAs for dynamic optimization**. *International Journal of Computational Intelligence Systems*, 6(1):47–63, 2013.

[42] KIRKPATRICK, S.; GELATT, C. D.; VECCHI, M. P.; OTHERS. **Optimization by simmulated annealing**. *science*, 220(4598):671–680, 1983.

[43] KRAMER, O. **Evolutionary self-adaptation: a survey of operators and strategy parameters**. *Evolutionary Intelligence*, 3(2):51–65, 2010.

[44] KRATICA, J. **Improving performances of the genetic algorithm by caching**. *Computers and Artificial Intelligence*, 18:271–283, 1999.

[45] KUKKONEN, S.; DEB, K. **Improved pruning of non-dominated solutions based on crowding distance for bi-objective optimization problems**. In: *2006 IEEE International Conference on Evolutionary Computation*, p. 1179–1186. IEEE, 2006.

[46] KURSAWE, F. **A variant of evolution strategies for vector optimization**. In: *International Conference on Parallel Problem Solving from Nature*, p. 193–197. Springer, 1990.

[47] LÜ, Z.; HAO, J.-K. **Adaptive tabu search for course timetabling**. *European Journal of Operational Research*, 200(1):235 – 244, 2010.

[48] LAI, D. S.; DEMIRAG, O. C.; LEUNG, J. M. **A tabu search heuristic for the heterogeneous vehicle routing problem on a multigraph**. *Transportation Research Part E: Logistics and Transportation Review*, 86:32–52, 2016.

[49] LARRANAGA, P.; LOZANO, J. A. **Estimation of distribution algorithms: A new tool for evolutionary computation**, volume 2. Springer Science & Business Media, 2002.

[50] LEE, C. Y.; KIM, S. J. **Parallel genetic algorithms for the earliness-tardiness job scheduling problem with general penalty weights**. *Computers & industrial engineering*, 28(2):231–243, 1995.

[51] LEUNG, S. W.; YUEN, S. Y.; CHOW, C. K. **Parameter control system of evolutionary algorithm that is aided by the entire search history**. *Applied Soft Computing*, 12(9):3063 – 3078, 2012.

[52] LI, K.; FIALHO, A.; KWONG, S.; ZHANG, Q. **Adaptive operator selection with bandits for a multiobjective evolutionary algorithm based on decomposition**. *IEEE Transactions on Evolutionary Computation*, 18(1):114–130, 2014.

[53] LIN, Q.; CHEN, J. **A novel micro-population immune multiobjective optimization algorithm**. *Computers & Operations Research*, 40(6):1590–1601, 2013.

[54] LIU, D.-P.; FENG, S.-T. **A novel adaptive genetic algorithms**. In: *Machine Learning and Cybernetics, 2004. Proceedings of 2004 International Conference on*, volume 1, p. 414–416. IEEE, 2004.

[55] LÓPEZ-IBÁNEZ, M.; PAQUETE, L.; STÜTZLE, T. **Exploratory analysis of stochastic local search algorithms in biobjective optimization**. In: *Experimental methods for the analysis of optimization algorithms*, p. 209–222. Springer, 2010.

[56] LOU, Y.; YUEN, S. Y. **Non-revisiting genetic algorithm with adaptive mutation using constant memory**. *Memetic Computing*, p. 1–22, 2016.

[57] LOURENÇO, H. R.; MARTIN, O. C.; STÜTZLE, T. **Iterated local search**. In: *Handbook of metaheuristics*, p. 320–353. Springer, 2003.

[58] MATEO, P. M.; ALBERTO, I. **A mutation operator based on a Pareto ranking for multi-objective evolutionary algorithms**. *Journal of Heuristics*, 18(1):53–89, 2012.

[59] MAULDIN, M. L. **Maintaining diversity in genetic search**. *Proc. National Conf. Artif. Intell.*, p. 247–250, 1984.

[60] MC GINLEY, B.; MAHER, J.; O'RIORDAN, C.; MORGAN, F. **Maintaining healthy population diversity using adaptive crossover, mutation, and selection**. *IEEE Transactions on Evolutionary Computation*, 15(5):692–714, 2011.

[61] MISEVIČIUS, A. **Using iterated tabu search for the traveling salesman problem**. *Information technology and control*, 32(3), 2015.

[62] MLADENOVIĆ, N.; HANSEN, P. **Variable neighborhood search**. *Computers & Operations Research*, 24(11):1097–1100, 1997.

[63] MONTGOMERY, D. C. **Design and analysis of experiments**. John Wiley & Sons, 2008.

[64] NEARCHOU, A. C.; OMIROU, S. L. **A particle swarm optimization algorithm for scheduling against restrictive common due dates**. *International Journal of Computational Intelligence Systems*, 6(4):684–699, 2013.

[65] OGATA, K. **Modern Control Engineering**. Prentice Hall PTR, 4 edition, 2001.

[66] PEARSON, P. K. **Fast hashing of variable-length text strings**. *Communications of the ACM*, 33(6):677–680, 1990.

[67] POVINELLI, R. J.; FENG, X. **Improving genetic algorithms performance by hashing fitness values**. *Artificial Neural Networks on Engineering*, 1999.

[68] RAUWENDAAL, C.; OSSWALD, T.; TELLEZ, G.; GRAMANN, P. **Flow analysis in screw extruders-effect of kinematic conditions**. *International Polymer Processing*, 13(4):327–333, 1998.

[69] RONALD, S. **Duplicate genotypes in a genetic algorithm**. In: *Proceedings Computação Evolutiva, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE Conferência Internacional sobre*, p. 793–798, 1998.

[70] SCHRIJVER, A. **On the history of combinatorial optimization (till 1960)**. *Handbooks in operations research and management science*, 12:1–68, 2005.

[71] SCHWEFEL, H.-P. **Adaptive mechanismen in der biologischen evolution und ihr einfluss auf die evolutionsgeschwindigkeit**. *Interner Bericht der Arbeitsgruppe Bionik und Evolutionstechnik am Institut für Mess-und Regelungstechnik Re*, 215(3), 1974.

[72] SHIRLEY, P.; ASHIKHMIN, M.; MARSCHNER, S. **Fundamentals of computer graphics**. CRC Press, 2015.

[73] SILVA, V. L.; WANNER, E. F.; CERQUEIRA, S. A.; TAKAHASHI, R. H. **A new performance metric for multiobjective optimization: The integrated sphere counting**. In: *2007 IEEE Congress on Evolutionary Computation*, p. 3625–3630. IEEE, 2007.

[74] SRINIVAS, M.; PATNAIK, L. M. **Adaptive probabilities of crossover and mutation in genetic algorithms**. *IEEE Transactions on Systems, Man, and Cybernetics*, 24(4):656–667, 1994.

[75] SUNG, C. W.; YUEN, S. Y. **Analysis of (1+1) evolutionary algorithm and randomized local search with memory**. *Evol. Comput.*, 19(2):287–323, June 2011.

[76] TAKAHASHI, R. H.; GUIMARÃES, F. G.; WANNER, E. F.; CARRANO, E. G. **Feedback-control operators for evolutionary multiobjective optimization**. In: *International Conference on Evolutionary Multi-Criterion Optimization*, p. 66–80. Springer, 2009.

[77] TAKAHASHI, R. H.; PALHARES, R. M.; DUTRA, D. A.; GONÇALVES, L. P. **Estimation of Pareto sets in the mixed control problem**. *International Journal of Systems Science*, 35(1):55–67, 2004.

[78] TALBI, N.; BELARBI, K. **Design of optimal fuzzy controllers for stabilization of a helicopter simulator using hybrid elite genetic algorithm and tabu search**. In: *2015 4th International Conference on Electrical Engineering (ICEE)*, p. 1–5. IEEE, 2015.

[79] THIERENS, D. **Adaptive mutation rate control schemes in genetic algorithms**. In: *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*, volume 1, p. 980–985. IEEE, 2002.

[80] VASCONCELOS, J.; RAMIREZ, J. A.; TAKAHASHI, R.; SALDANHA, R. **Improvements in genetic algorithms**. *IEEE Transactions on magnetics*, 37(5):3414–3417, 2001.

[81] VENDITTI, L.; PACCIARELLI, D.; MELONI, C. **A tabu search algorithm for scheduling pharmaceutical packaging operations**. *European Journal of Operational Research*, 202(2):538–546, 2010.

[82] VLENNET, R.; FONTEIX, C.; MARC, I. **Multicriteria optimization using a genetic algorithm for determining a pareto set**. *International Journal of Systems Science*, 27(2):255–260, 1996.

[83] WANG, C.; GAO, Y. **Determination of power distribution network configuration using non-revisiting genetic algorithm**. *IEEE Transactions on Power Systems*, 28(4):3638–3648, 2013.

[84] WANG, R.; PURSHOUSE, R. C.; FLEMING, P. J. **Preference-inspired coevolutionary algorithms for many-objective optimization**. *IEEE Transactions on Evolutionary Computation*, 17(4):474–494, 2013.

[85] WANNER, E. F.; GUIMARÃES, F. G.; TAKAHASHI, R. H.; FLEMING, P. J. **Local search with quadratic approximations into memetic algorithms for optimization with multiple criteria**. *Evolutionary computation*, 16(2):185–224, 2008.

[86] YAN, L.; CHANGRUI, Y. **A new hybrid algorithm for feature selection and its application to customer recognition**. In: *International Conference on Combinatorial Optimization and Applications*, p. 102–111. Springer, 2007.

[87] YANG, S.; LI, M.; LIU, X.; ZHENG, J. **A grid-based evolutionary algorithm for many-objective optimization**. *IEEE Transactions on Evolutionary Computation*, 17(5):721–736, 2013.

[88] YUEN, S. Y.; CHOW, C. K. **A non-revisiting genetic algorithm**. In: *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, p. 4583 –4590, sept. 2007.

[89] YUEN, S. Y.; CHOW, C. K. **Applying non-revisiting genetic algorithm to traveling salesman problem**. In: *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, p. 2217–2224. IEEE, 2008.

[90] YUEN, S. Y.; CHOW, C. K. **A non-revisiting simulated annealing algorithm**. In: *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, p. 1886–1892. IEEE, 2008.

[91] YUEN, S. Y.; CHOW, C. K. **Continuous non-revisiting genetic algorithm**. In: *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*, p. 1896 –1903, may 2009.

[92] YUEN, S. Y.; CHOW, C. K. **A genetic algorithm that adaptively mutates and never revisits**. *IEEE Transactions on Evolutionary Computation*, 13:454–472, 2009.

[93] ZHANG, J.; CHUNG, H. S.-H.; LO, W.-L. **Clustering-based adaptive crossover and mutation probabilities for genetic algorithms**. *IEEE Transactions on Evolutionary Computation*, 11(3):326–335, 2007.

[94] ZHANG, Q.; LI, H. **MOEA/D: A multiobjective evolutionary algorithm based on decomposition**. *IEEE Transactions on evolutionary computation*, 11(6):712–731, 2007.

[95] ZHANG, X.; TIAN, Y.; JIN, Y. **A knee point-driven evolutionary algorithm for many-objective optimization**. *IEEE Transactions on Evolutionary Computation*, 19(6):761–776, 2015.

[96] ZHU, K. Q. **A diversity-controlling adaptive genetic algorithm for the vehicle routing problem with time windows**. In: *Tools with Artificial Intelligence, 2003. Proceedings. 15th IEEE International Conference on*, p. 176–183. IEEE, 2003.

[97] ZITZLER, E. **PISA - a platform and programming language independent interface for search algorithms.**, 2010. último acesso 14.02.2010.

[98] ZITZLER, E.; LAUMANNS, M.; THIELE, L.; OTHERS. **SPEA2: Improving the strength pareto evolutionary algorithm**. In: *Eurogen*, volume 3242, p. 95–100, 2001.

[99] ZITZLER, E.; LAUMANNS, M.; THIELE, L.; OTHERS. **SPEA2: Improving the strength pareto evolutionary algorithm**. In: *Eurogen*, volume 3242, p. 95–100, 2001.

[100] ZITZLER, E.; THIELE, L. **Multiobjective optimization using evolutionary algorithms - a comparative case study**. In: *International Conference on Parallel Problem Solving from Nature*, p. 292–301. Springer, 1998.

[101] ZITZLER, E.; THIELE, L. **Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach**. *IEEE transactions on Evolutionary Computation*, 3(4):257–271, 1999.