Marcos Daniel V. Baroni

Indexação Multidimensional para Problemas da Mochila Multiobjetivo com Paretos de Alta Cardinalidade

Vitória - Espírito Santo - Brasil
26 de julho de 2018

Marcos Daniel V. Baroni

Indexação Multidimensional para Problemas da Mochila Multiobjetivo com Paretos de Alta Cardinalidade

Tese de Doutorado apresentada de acordo com o regimento do Programa de Pós-graduação em Informática da Universidade Federal do Espírito Santo, como requisito para obtenção do grau de Doutor em Ciência da Computação.

Universidade Federal do Espírito Santo – UFES
Departamento de Informática
Programa de Pós-Graduação em Informática

Orientador: Dr. Flávio Miguel Varejão

Vitória - Espírito Santo - Brasil
26 de julho de 2018

Marcos Daniel V. Baroni

Indexação Multidimensional para Problemas da Mochila Multiobjetivo com Paretos de Alta Cardinalidade

Tese de Doutorado apresentada de acordo com o regimento do Programa de Pós-graduação em Informática da Universidade Federal do Espírito Santo, como requisito para obtenção do grau de Doutor em Ciência da Computação.

Trabalho aprovado.
Vitória, 26 de julho de 2018:

Dr. Flávio Miguel Varejão
Departamento de Informática - UFES
Orientador

Dr.ª Maria Claudia Silva Boeres
Departamento de Informática - UFES

Dr. Thomas Walter Rauber
Departamento de Informática - UFES

Dr. Alexandre Loureiros Rodrigues
Departamento de Estatística - UFES

Dr.ª Simone de Lima Martins
Instituto de Computação - UFF

Resumo

Diversos problemas reais envolvem a otimização simultânea de múltiplos critérios, os quais são, geralmente, conflitantes entre si. Estes problemas são denominados multiobjetivo e não possuem uma única solução, mas um conjunto de soluções de interesse, denominadas soluções eficientes ou não dominadas. Um dos grande desafios a serem enfrentados na resolução deste tipo de problema é o tamanho do conjunto solução, que tende a crescer rapidamente dado o tamanho da instância, degradando a performance dos algoritmos. Dentre os problemas multiobjetivos mais estudados está o problema da mochila multiobjetivo, pelo qual diversos problemas reais podem ser modelados. Este trabalho propõe a aceleração do processo de solução do problema da mochila multiobjetivo, através da utilizando da árvore k-d como estrutura de indexação multidimensional para auxiliar a manipulação das soluções. A performance da abordagem é analisada através de experimentos computacionais, realizados no contexto exato utilizando um algoritmo estado da arte. Testes também são realizados no contexto heurístico, utilizando a adaptação de uma meta-heurística para o problema em questão, sendo esta também uma contribuição do presente trabalho. Segundo os resultados, para o contexto exato a proposta foi eficaz, apresentam speedup de até 2.3 para casos bi-objetivo e 15.5 em casos 3-objetivo, não sendo porém eficaz no contexto heurístico, apresentando pouco impacto no tempo computacional. Em todos os casos, porém, houve considerável redução no número de avaliações de soluções.

Palavras Chave: Problema da Mochila Multiobjetivo, Indexação Multidimensional, Meta-heurística, Algoritmo Exato.

Abstract

Several real problems involve the simultaneous optimization of multiple criteria, which are generally conflicting with each other. These problems are called multiobjective and do not have a single solution, but a set of solutions of interest, called efficient solutions or non-dominated solutions. One of the great challenges to be faced in solving this type of problem is the size of the solution set, which tends to grow rapidly given the size of the instance, degrading algorithms performance. Among the most studied multiobjective problems is the multiobjective knapsack problem, by which several real problems can be modeled. This work proposes the acceleration of the resolution process of the multiobjective knapsack problem, through the use of a árvore k-d as a multidimensional index structure to assist the manipulation of solutions. The performance of the approach is analyzed through computational experiments, performed in the exact context using a state-of-the-art algorithm. Tests are also performed in the heuristic context, using the adaptation of a meta-heuristic for the problem in question, being also a contribution of the present work. According to the results, the proposal was effective for the exact context, presenting a speedup up to 2.3 for bi-objective cases and 15.5 for 3-objective cases, but not effective in the heuristic context, presenting little impact on computational time. In all cases, however, there was a considerable reduction in the number of solutions evaluations.

Keywords: Multiobjective Knapsack Problem, Multidimensional Indexing, Metaheuristic, Exact Algorithm.

Sumário

Lista de ilustrações

Lista de tabelas

1 Introdução

Diferentemente de problemas de otimização escalar, problemas multiobjetivo envolvem a otimização simultânea de múltiplas de funções objetivo geralmente conflitantes entre si. Estes problemas não possuem tipicamente uma única solução, mas um conjunto de soluções de interesse chamado *conjunto Pareto*. Este conjunto é formado pelas soluções ditas *não dominadas* ou *eficientes*, para as quais não é possível melhorar um dos valores de objetivo sem degradar um outro.

Dentre os problemas multiobjetivo bastante estudados está o problema da mochila multiobjetivo. Semelhantemente ao problema da mochila $0-1$ clássico, uma instância do problema da mochila multiobjetivo consiste de um conjunto de itens possuindo cada um deles um determinado peso, dos quais deve-se selecionar um subconjunto, cujos pesos somados não excedam uma capacidade definida. Porém, diferentemente do caso clássico, cada item selecionado contribui simultaneamente para múltiplos objetivos, os quais se deseja maximizar.

Diversos problemas reais podem ser modelados como uma instância do problema da mochila multiobjetivo. Um deles é o problema de seleção de projeto (TENG; TZENG, 1996). Nele deseja-se selecionar, dentre um portfolio definido de projetos, um subconjunto para ser executado considerando um orçamento limitado. A execução de cada um dos projetos beneficia múltiplos critérios como, por exemplo, múltiplas partes interessadas. Deseja-se então determinar o subconjunto que maximiza simultaneamente a contribuição total para cada critério estabelecido, o que caracteriza um problema multiobjetivo.

Outro problema que pode ser modelado como um problema da mochila multiobjetivo é o problema de orçamento de capital com risco (ROSENBLATT; SINUANY-STERN, 1989). Nesse problema, considera-se um conjunto de investimentos, do qual deseja-se selecionar um subconjunto para ser executado, dado um orçamento limitado. Cada investimento possui um valor de lucro e também um fator de risco (ou variância) associado, fazendo com que a seleção de um subconjunto de investimentos configure, não apenas um lucro total, mas também um valor de risco associado total. A escolha do melhor subconjunto de investimentos depende de uma soma ponderada do lucro total ao risco total acumulado. Porém, o fator de ponderação, que representa a vantagem do lucro sobre o risco, não é previamente conhecido, mas determinado em um momento posterior. Por esse motivo, a etapa de solução do problema deve considerar a maximização simultânea do lucro e do fator risco acumulado, o que caracteriza o problema multiobjetivo.

Em (JENKINS, 2002) o problema da mochila multiobjetivo é utilizado para modelar um problema de planejamento de ações para remediar poluição em porções terrestres. Nesse problema, deseja-se planejar a execução de ações de remoção de poluição para tratar diversos postos de guarda marítima. Essas ações não serão aplicadas em todos os

postos simultaneamente, mas separadas em duas fases, sendo que para a primeira existe um orçamento limitado. Após uma avaliação preliminar dos postos quanto ao nível de poluição, um quadro de ações é estabelecido para cada posto. A partir de então, deve-se decidir, quais postos serão tratados imediatamente (primeira fase) e quais serão tratados posteriormente (segunda fase). Para cada posto, a aplicação imediata das ações provê dois critérios de retribuição: (a) uma pontuação junto ao respectivo órgão ambiental e (b) uma métrica associada à estimativa de incompletude da avaliação preliminar do posto. O objetivo do problema é definir, para cada posto, a fase em que as ações serão aplicadas, buscando maximizar tanto a pontuação junto ao órgão ambiental quanto à pontuação de estimativa de incompletude de avaliação. Uma vez que no momento da solução não se sabe a relação de importância entre os dois critérios, o problema é caracterizado como multiobjetivo.

1.1 Motivação

Um dos grandes desafios em problemas multiobjetivos, inclusive do problema da mochila multiobjetivo, é a alta cardinalidade que o conjunto Pareto pode alcançar, especialmente em casos com mais de 2 objetivos (EHRGOTT, 2013). A Figura 1 apresenta o tamanho médio do conjunto Pareto, dado o tamanho da instância (número de itens) para uma das classes de instância do problema da mochila multiobjetivo com 3 objetivos. Pode-se observar o rápido crescimento do conjunto Pareto à medida que a quantidade de itens na instância aumenta, chegando próximo à média de 1000 soluções para instâncias de apenas 30 itens.

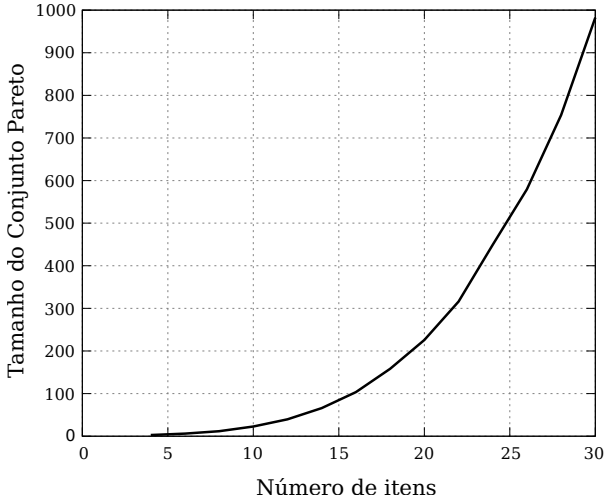


Figura 1 – Tamanho do conjunto Pareto dado o tamanho da instância de problemas da mochila 3-objetivo.

Esse rápido crescimento tende a degradar a eficiência dos algoritmos, uma vez que o processo de resolução do problema deve gerar e manter uma grande quantidade de

soluções, especialmente em contextos exatos, nos quais é necessário a enumeração completa das soluções que compõem o conjunto Pareto. Este motivo tem levado pesquisadores a desenvolverem métodos heurísticos, que buscam dar uma solução aproximada, demandando um menor esforço computacional (DEB et al., 2002; ZITZLER; THIELE, 1998; ZHANG; LI, 2007).

Outra alternativa viável, porém não trivial, de se reduzir o tempo computacional demandado por um algoritmo é através da otimização de seus procedimentos, como por exemplo, pela utilização de uma estrutura de dados mais adequada para a operação, que pode reduzir o esforço computacional demandado pelo algoritmo (CORMEN et al., 2009; AHO; HOPCROFT, 1974; KNUTH, 1973).

1.2 Objetivo

Este trabalho propõe a otimização dos procedimentos de manipulação do conjunto de soluções, mais especificamente da operação de verificação de dominância de uma solução, dado um conjunto de soluções, visto que esta operação é de grande demanda tanto em contextos exatos quanto em contextos heurísticos. Para isso, a operação de verificação de dominância é interpretada como o problema de decidir se existe algum ponto em uma determinada região do espaço multidimensional.

Esse tipo de problema, chamado *busca de faixa*, é bem conhecido em alguns ramos da computação, como na computação gráfica e desenvolvimento de jogos, onde é necessário verificar a colisão entre pontos e polígonos (SEDGEWICK, 1988). Visando a eficiência da operação, a busca de faixa é geralmente executada com o auxílio de uma estrutura de indexação multidimensional, das quais a árvore k-d é uma das mais indicadas para este fim (AGARWAL; ERICKSON et al., 1999; SEDGEWICK, 1988).

O objetivo do presente trabalho é demonstrar que a utilização de estruturas de indexação multidimensional, mais especificamente a árvore k-d, pode aumentar a performance de algoritmos para problemas da mochila multiobjetivo, mais especificamente em casos que possuam conjuntos Paretos com grande quantidade de soluções, ao reduzir o número total de avaliações de solução necessárias para a execução do algoritmo. A eficácia da proposta será avaliada através de testes computacionais, para os quais serão analisados o número de avaliações de solução e o tempo computacional demandado pelos algoritmos em instâncias consideradas pela literatura.

Esta avaliação será feita no contexto exato utilizando o algoritmo de Bazgan, considerado pela literatura como o método exato mais eficiente para o problema (BAZGAN; HUGOT; VANDERPOOTEN, 2009). No trabalho original do algoritmo, é proposta a utilização da árvore AVL como estrutura de dados auxiliar para indexação de soluções em casos bi-objetivo, sendo utilizada a lista encadeada para o caso 3-objetivo. As respectivas estruturas de dados foram originalmente utilizadas, provavelmente, diante do foco do

trabalho, o qual é dedicado à proposta do algoritmo e suas provas de corretude. Diante disso, perdura a carência de uma maior investigação sobre a utilização de estruturas de dados.

Para esta avaliação será utilizado um conjunto de 4 tipos de instâncias por ser o conjunto proposto pela literatura para testes em abordagem exata (BAZGAN; HUGOT; VANDERPOOTEN, 2009; FIGUEIRA et al., 2013; CORREIA; PAQUETE; FIGUEIRA, 2018), contemplando tanto casos fáceis como difíceis do problema através da exploração de diferentes características de instância.

A avaliação também será feita no contexto heurístico, por este ter uma demanda computacional diferente da abordagem exata, não necessitando de enumerar todo o conjunto Pareto, mas apenas uma aproximação deste. Para a abordagem heurística será utilizada uma implementação da metaheurística evolutiva chamada evolução estocástica por complexos (*shuffled complex evolution* em inglês). A evolução estocástica por complexos pretende ser uma metaheurística robusta, que apresentou êxito em diversos problemas (DUAN; SOROOSHIAN; GUPTA, 1992; ELBELTAGI; HEGAZY; GRIERSON, 2007; ZHAO et al., 2015; BHATTACHARJEE; SARMAH, 2014; BARONI; VAREJÃO, 2015; BARONI; VAREJÃO, 2016) porém, ainda não havia sido aplicada ao problema da mochila multiobjetivo, representando também uma contribuição do presente trabalho.

Para os testes computacionais no contexto heurístico, serão consideradas as mesmas instâncias utilizadas pela literatura para testar os principais algoritmos heurísticos (ZITZLER; THIELE, 1998; DEB et al., 2002; ZHANG; LI, 2007; ZOUACHE; MOUSSAOUI; ABDELAZIZ, 2018). Em ambos os contextos serão observados tanto o tempo computacional quanto o número de avaliações de solução necessários para a execução de cada algoritmo.

1.3 Histórico

A proposta do presente trabalho surgiu após a execução de um projeto de pesquisa abordando um problema real de otimização, levantado junto à EDP-Escelsa, empresa distribuidora de energia elétrica local. Um dos mecanismos da Agência Nacional de Energia Elétrica (ANEEL) para estimular a redução de perda de energia é a definição de metas, que estipulam a quantidade máxima de energia perdida pela distribuidora por causas não-técnicas. As distribuidoras, por sua vez, fazem seus planejamentos de forma a tentar atingir esta meta, executando diversas ações de redução de perda, respeitando orçamentos determinados, como orçamento de capital e orçamento de mão de obra.

Na ocasião, o plano de ações de redução de perda era selecionado manualmente por especialistas, buscando, por tentativa e erro, escolher dentre um portfolio de ações, a combinação que obtém o melhor resultado, respeitando os orçamentos determinados. Diante disso, a primeira etapa do projeto de pesquisa foi a modelagem do problema real

como um problema da mochila multidimensional (PUCHINGER; RAIDL; PFERSCHY, 2010). Apesar de também ser um problema de otimização escalar, o problema da mochila multidimensional difere do problema da mochila clássico por possuir múltiplas restrições de capacidade, que no problema real em questão, representa recursos como orçamento com capital, orçamento com mão de obra e quantidade máxima de execução das ações. Sabendo-se que o problema da mochila multidimensional é $\mathcal{NP}$-Hard, visou-se o desenvolvimento de uma heurística para a resolução do problema.

A forma ideal de se avaliar a qualidade da solução encontrada por uma heurística é comparando-a, quando possível, com a solução exata ou, se não, com uma solução que possua alguma garantia de qualidade. Diante disso, visando validar a qualidade das heurísticas desenvolvidas, foi utilizado um resolvedor exato que implementa o algoritmo *branch-and-cut* (PADBERG; RINALDI, 1991; MITCHELL, 2002) para obter as soluções ótimas para o problema.

O algoritmo *branch-and-cut* não só garante encontrar a solução ótima ao final da execução, mas também apresenta uma garantia de qualidade da melhor solução encontrada no decorrer do processo. Essa garantia é determinada através da estipulação de um limite superior para o valor de objetivo do problema, o qual é gradativamente reduzido à medida que as soluções do problema são enumeradas. O algoritmo é encerrado quando o valor de limite superior coincide com o valor objetivo da melhor solução encontrada.

Esse garantia de qualidade apresentada pelo algoritmo permite que o mesmo seja encerrado antes de sua execução completa, caso a qualidade da melhor solução encontrada seja satisfatória. De fato, em muitos casos, a solução exata do problema é encontrada em momentos iniciais do algoritmo, ficando a parte final da execução dedicada apenas à prova computacional da otimalidade da solução encontrada.

Durante a execução do resolvedor, observou-se que, apesar do encerramento total do algoritmo demandar horas, em poucos segundos alcançava-se uma solução com qualidade bem próxima à ótima. A Tabela 1 apresenta a progressão da qualidade da melhor solução encontrada pelo resolvedor para uma das instâncias do problema considerada pela literatura como uma das mais difíceis, possuindo 500 itens e 30 dimensões.

tempo (s)	**qualidade (%)**
0.02	50.00
0.05	95.85
0.10	99.20
0.80	99.63
2.40	99.66
7.30	99.71
20.20	99.76
51.10	99.77

Tabela 1 – Progressão da qualidade da melhor a solução encontrada pelo resolvedor exato.

Cada linha da Tabela 1 registra o momento em que uma nova melhor solução foi encontrada pelo resolvedor exato. A primeira coluna apresenta o tempo total de execução (em segundos). A segunda coluna apresenta a qualidade da melhor solução encontrada (em porcentagem) em relação ao limite superior computado. Pode-se observar que em menos de 1 segundo o algoritmo foi capaz de encontrar uma solução com qualidade mínima de 99.63% do valor ótimo. Fato semelhante pôde ser observado em todas as instâncias do problema, o que resultou por enfraquecer a justificativa de desenvolvimento de uma heurística. Diante disso, foi adotada como solução satisfatória a fornecida pelo resolvedor.

Contudo, durante a fase de pesquisa de algoritmo exatos, um dos algoritmos estudados foi o algoritmo de Nemhauser e Ulmann (NEMHAUSER; ULLMANN, 1969) adaptado para o problema da mochila multidimensional. O algoritmo de Nemhauser e Ulmann é um algoritmo de programação dinâmica que originalmente computa a solução exata para o problema da mochila clássico. Porém, com poucas adaptações, o algoritmo pode ser aplicado a algumas variações do problema clássivo, inclusive ao caso multidimensional.

Ao longo do processo de solução, o algoritmo de Nemhauser e Ulmann manipula uma crescente quantidade de estados, representando as possíveis soluções do problema. Essa grande quantidade de estados resulta na degradação da performance do algoritmo. Na tentativa de atenuar essa degradação, propôs-se a indexação multidimensional dos estados utilizando a árvore k-d, com o objetivo de acelerar sua manipulação.

A aplicação da proposta apresentou bons resultados, porém, diante do desempenho do algoritmo *branch-and-cut*, não pôde tornar o algoritmo de programação dinâmica competitivo no contexto multidimensional. Contudo, cogitou-se ser proveitosa a aplicação da proposta de indexação multidimensional em contextos onde a enumeração dos estados gerados é estritamente necessária, como o é no caso multiobjetivo.

1.4 Estrutura da Tese

A estrutura deste trabalho tem a seguinte organização. O Capítulo 2 aborda o problema da mochila multiobjetivo, apresentando as abordagens exata e heurística que foram utilizadas no trabalho. O Capítulo 3 apresenta a proposta de utilização da árvore k-d como estrutura auxiliar de aceleração para a resolução do problema. No Capítulo 4 são descritos os experimentos computacionais. No Capítulo 5 são apresentadas as conclusões e os trabalhos futuros.

2 O Problema da Mochila Multiobjetivo

Em problemas reais é comum a existência de situações nas quais se deseja otimizar mais de um objetivo, geralmente conflitantes. Estes problemas são chamados multiobjetivos e, diferentemente dos problemas de otimização escalar, não possuem uma solução melhor, mas várias soluções de interesse, chamadas *soluções eficientes*.

Um problema de otimização multiobjetivo com m objetivos pode ser descrito como uma função vetorial $f(x) = \Big(f_1(x), \ldots, f_m(x)\Big)$ para a qual se deseja encontrar vetores $x \in X$ que maximizem simultaneamente as m funções objetivo. Formalmente:

$$\begin{aligned} &\max\ f(x) = \Big(f_1(x), f_2(x), \ldots, f_m(x)\Big) \\ &\text{sujeito a} \quad x \in X \end{aligned}$$

A seguir são apresentadas algumas definições importantes, relacionadas a problemas multiobjetivo, necessárias para o desenvolvimento do presente trabalho.

Definição 1 (Dominância, Eficiência e conjunto Pareto)**.** *Considere um problema de otimização multiobjetivo. Diz-se que uma solução $x \in X$* domina *uma solução $y \in X$, denotado por $x\Delta y$ se, e somente se, x é ao menos tão boa quanto y em todos os objetivos e melhor que y em ao menos um dos objetivos. Formalmente:*

$$x\Delta y \iff \begin{cases} \forall i \in \{1, 2, \ldots, m\} : f_i(x) \geqslant f_i(y)\ e \\ \exists j \in \{1, 2, \ldots, m\} : f_j(x) > f_j(y) \end{cases} \tag{2.1}$$

Uma solução $x \in X$ é dita eficiente*, denotado por $\mathit{eff}(x)$, se, e somente se, x não é dominada por nenhuma outra solução pertencente a X. Formalmente:*

$$eff(x) \iff \nexists\Big(y \in X \wedge y\Delta x\Big)$$

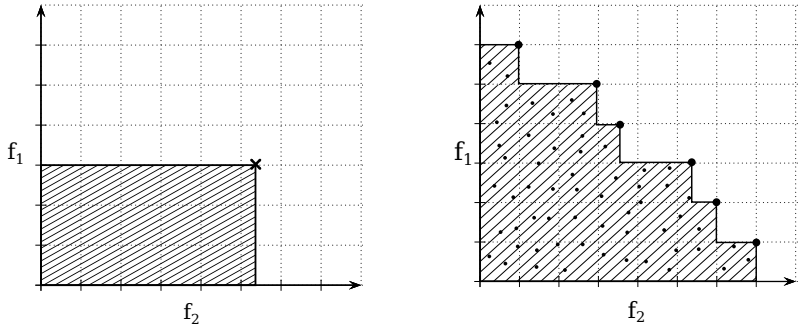
O conjunto de todas as soluções eficientes de um problema multiobjetivo, denotado por $Par(X)$, é chamado de conjunto Pareto *ou* conjunto Pareto-ótimo*. Formalmente:*

$$Par(X) = \{x \in X \mid \mathit{eff}(x)\}$$

Resolver um problema multiobjetivo consiste em determinar seu conjunto Pareto. Este conceito foi primeiramente elaborado por Vilfredo Pareto em 1896, que enunciou a relação Pareto-Ótima que diz: "não é possível melhorar uma característica do problema sem piorar outra", o que caracteriza a relação conflitante entre os objetivos na otimização multiobjetivo.

A Figura 2a ilustra o conceito de dominância de um problema multiobjetivo. A solução x em destaque domina todas as soluções sob a área hachurada por estas se identificarem

com a Equação 2.1. A Figura 2b ilustra o conceito de conjunto Pareto. As soluções sobre o traçado em destaque representam um conjunto Pareto por dominarem todas as outras soluções sob a área hachurada.



(a) Região de dominância de uma solução.

(b) Exemplo de conjunto Pareto.

Figura 2 – Exemplos de solução dominante e conjunto Pareto.

Convém ressaltar que para problemas multiobjetivo não existe, ou é desconhecida, qualquer função que agregue os valores de objetivo em um único valor, caso contrário, o problema poderia ser tratado como um problema de otimização escalar. Sendo assim, a decisão de qual solução será utilizada, ou quais soluções serão utilizadas, está fora do escopo de resolução.

Um dos problemas multiobjetivos mais estudados é o problema da mochila multiobjetivo, *multiobjective knapsack problem* em inglês (MOKP). Diversos problemas reais podem ser modelados como uma instância do MOKP, tais como seleção de projetos (TENG; TZENG, 1996), orçamento de capital (ROSENBLATT; SINUANY-STERN, 1989), carregamento de carga (TENG; TZENG, 1996) e planejamento de estoque (ISHIBUCHI; AKEDO; NOJIMA, 2015).

Uma instância de um problema da mochila multiobjetivo com m objetivos consiste em uma capacidade inteira $W > 0$ e n itens. Cada item i possui um peso inteiro positivo w_i e m lucros inteiros $p_i^1, p_i^2, \ldots, p_i^m$ não negativos, sendo p_i^k a contribuição do i-ésimo item para com o k-ésimo objetivo. Uma solução é representada por um vetor binário $x = (x_1, \ldots, x_n)$ tal que $x_k = 1$ representa a inclusão do k-ésimo item na mochila e $x_k = 0$ representa o caso contrário. Uma solução é viável se o peso total incluído na mochila não ultrapassa a

capacidade da mochila. Formalmente a definição do problema é a seguinte:

$$\begin{aligned} \max\ & f(x) = \Big(f_1(x), f_2(x), \ldots, f_m(x)\Big) \\ \text{sujeito a}\ & w(x) \leqslant W \\ & x \in \{0,1\}^n \end{aligned}$$

onde

$$f_j(x) = \sum_{i=1}^{n} p_i^j x_i \quad j = 1, \ldots, m$$
$$w(x) = \sum_{i=1}^{n} w_i x_i$$

O MOKP é considerado um problema $\mathcal{NP}$-Hard visto ser uma generalização do clássico problema da mochila $0-1$, para o qual $m = 1$ (GAREY; JOHNSON, 2002). É consideravelmente difícil determinar o conjunto Pareto para um MOKP, pois este tende a crescer rapidamente com o tamanho do problema, especialmente com o número de objetivos (ISHIBUCHI; TSUKAMOTO; NOJIMA, 2008). Até mesmo para casos bi-objetivos, problemas relativamente pequenos podem se apresentar inviáveis.

A Tabela 2 apresenta um exemplo de instância para o MOKP com 2 objetivos e 10 itens, com valores de peso e lucro entre 0 e 9 e capacidade de mochila $W = 28$. A Tabela 3 apresenta o conjunto Pareto da instância. Vale observar através da Tabela 3 a potencial complexidade do problema que, neste caso, possui 6 soluções eficientes, sendo uma instância de apenas 10 itens. Vale observar também o caráter único das soluções eficientes: se considerarmos a solução x_3, por exemplo, nenhuma solução possui simultaneamente f_1 maior que 32 e f_2 maior que 34, caso contrário, a solução x_3 poderia ser descartada.

	Itens									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
$\boldsymbol{p^1}$	4	9	3	1	8	7	2	5	6	7
$\boldsymbol{p^2}$	8	4	2	2	3	0	6	8	9	6
$\boldsymbol{w}$	7	8	5	8	3	5	6	2	4	9

$\boldsymbol{W}$	28

Tabela 2 – Exemplo de instância do problema da mochila bi-objetivo com 10 itens.

A seguir apresentaremos o conceito de solução ineficiente, a qual utilizaremos no desenvolvimento deste trabalho.

Definição 2 (Solução ineficiente)**.** *Considere um problema da mochila multiobjetivo. Uma solução $x \in X$ é considerada* ineficiente*, denotada por $ineff(c)$, se, e somente se, esta tem capacidade para conter ainda ao menos mais um item em sua solução. Formalmente:*

$$ineff(x) \iff \exists i \in \{1, \ldots, n\}\Big(w(x) + w_i \leqslant W \wedge x_i = 0\Big)$$

	1	2	3	4	5	6	7	8	9	10	f_1	f_2	w
x_1	x						x	x	x	x	24	37	28
x_2	x		x		x		x	x	x		28	36	27
x_3	x				x	x	x	x	x		32	34	27
x_4		x	x		x		x	x	x		33	32	28
x_5		x			x	x	x	x	x		37	30	28
x_6		x	x		x	x		x	x		38	26	27

Tabela 3 – Conjunto solução da instância exemplo.

Soluções ineficientes não são interessantes, uma vez que, com pouco custo computacional, podem ser *preenchidas* com mais alguns itens, gerando uma melhor solução.

A literatura contém várias propostas para resolver o MOKP de forma exata. Porém, nenhum método tem provado ser eficiente para grandes instâncias com mais de dois objetivos. Mesmo para problemas bi-objetivo, algumas instâncias de tamanho considerado médio têm apresentado dificuldades na determinação da solução exata, o que tem motivado o desenvolvimento de métodos heurísticos para determinar um conjunto Pareto aproximado em tempo computacional razoável.

A Seção 2.1 seguinte tratará da abordagem por solução exata para o MOKP, apresentando o algoritmo exato estado da arte, proposto por Bazgan, o qual será utilizado neste trabalho para analisar a eficiência da proposta de indexação multi-dimensional no contexto exato. A Seção 2.2 tratará da abordagem por solução heurística para o MOKP propondo uma baseada em um algoritmo populacional evolucionário, sendo também uma contribuição deste trabalho, o qual será utilizado para analisar a eficiência da proposta de indexação multi-dimensional no contexto heurístico.

2.1 Abordagem Exata

Várias abordagens exatas foram propostas pela literatura para encontrar o conjunto Pareto para o MOKP. Primeiramente pode-se mencionar um trabalho teórico (KLAMROTH; WIECEK, 2000), sem resultados experimentais, onde cinco modelagens diferentes de programação dinâmica são apresentadas e discutidas.

Um algoritmo de *duas fases* é proposto em (VISÉE et al., 1998) para a solução de casos bi-objetivos. A primeira fase do algoritmo utiliza uma versão escalar do problema, definida através da agregação das funções objetivo. Esta versão é então considerada para gerar um subconjunto do conjunto Pareto. A segunda fase do algoritmo utiliza um algoritmo de *branch and bound* para encontrar o restante do conjunto Pareto. Diferentes estratégias de buscas e cortes para o algoritmo branch and bound são discutidas, cujas performances são comparadas através de experimentos computacionais.

Um algoritmo baseado em um método de rotulação foi proposto em (CAPTIVO et

al., 2003) no qual o MOKP é primeiramente transformado em um problema de caminho mínimo bi-objetivo sem ciclos e então solucionado utilizando um método de rotulação inspirado pelo algoritmo proposto em (MARTINS; SANTOS, 1999), porém, explorando características específicas do problema transformado.

Atualmente o algoritmo exato que possui relatos de melhor performance é o algoritmo de programação dinâmica desenvolvido por Bazgan, Hugot e Vanderpooten em (BAZGAN; HUGOT; VANDERPOOTEN, 2009). O algoritmo é baseado em um clássico algoritmo de programação dinâmica para o problema da mochila proposto por Nemhauser e Ullmann em (NEMHAUSER; ULLMANN, 1969), cujo desenvolvimento foi inspirado nas observações de dominância entre soluções, apresentadas por Weignartner e Ness em (WEINGARTNER; NESS, 1967). O algoritmo de Bazgan, porém, aplica três dominâncias desenvolvidas especificamente para o caso multiobjetivo, permitindo o descarte de uma grande quantidade de estados ao longo das iterações do algoritmo.

Uma proposta de melhoria para o algoritmo de Bazgan foi apresentada por Figueira, Simões e Vanderpooten em (FIGUEIRA et al., 2013) para o caso bi-objetivo. O trabalho introduz novas definições de limites superiores e inferiores de valores de função objetivo, o que possibilita descartar uma quantidade ainda maior de estados intermediário, resultando em 20% de melhoria no tempo computacional médio para instâncias bi-objetivo, segundo experimentos computacionais apresentados.

Recentemente cinco técnicas algorítmicas de compressão de dados para o caso bi-objetivo foram apresentadas em (CORREIA; PAQUETE; FIGUEIRA, 2018) para o algoritmo de Bazgan como propostas de melhoria quanto a utilização de memória. Cada técnica propõe uma modelagem diferente na representação dos estados gerados pelo algoritmo. Testes computacionais são apresentados, comparando as cinco técnicas quanto a relação entre uso de memória e esforço computacional.

Para investigar a eficiência do método de indexação multi-dimensional para soluções do MOKP no contexto exato, o algoritmo de Bazgan será utilizado. A seguir será introduzido primeiramente o algoritmo de Nemhauser e Ullmann, que é a base para o desenvolvimento do algoritmo de Bazgan. Posteriormente serão descritas as três relações de dominância utilizadas no algoritmo de Bazgan, tal como introduzidas em (BAZGAN; HUGOT; VANDERPOOTEN, 2009), para então ser descrito o próprio algoritmo de Bazgan.

O algoritmo de Nemhauser e Ullmann foi primeiramente proposto para resolver o problema da mochila $0-1$. Porém, sua concepção é genérica e, com poucas adaptações, pode ser facilmente aplicada a diversas variações do problema da mochila. O algoritmo mantém durante a execução um conjunto de estados, cada um representando uma solução viável para o problema. A execução do algoritmo é definida por n iterações, sendo n o número total de itens da instância. A cada iteração um novo item é considerado como item candidato a fazer parte da solução. Novas soluções contendo este novo item são

então geradas e adicionadas ao conjunto de estados. Ao final dos n passos as soluções não eficientes são descartadas, restando o conjunto Pareto. O Algoritmo 1 descreve o algoritmo de Nemhauser e Ullmann para o MOKP.

Algoritmo 1: O algoritmo de Nemhauser e Ullmann para o MOKP.

```
   input: p, w, W
1  begin
2      S_0 = {(0, ..., 0)};
3      for k ← 1, n do
4          S_k ← S_{k-1} ∪ {(s^1 + p_k^1, ..., s^m + p_k^m, s^{m+1} + w_k) | s^{m+1} + w_k ⩽ W, s ∈ S_{k-1}};
5      end
6      P = {s ∈ S_n | ∄(a ∈ S_n | aΔs)};
7      return P;
```

O Algoritmo 1 inicia definindo o conjunto inicial de estados S_0 contendo apenas a solução vazia (linha 2). A cada k-ésima iteração é gerado o conjunto de estados S_k (linhas 3-5). Um estado $s_k = (s_k^1, \ldots, s_k^m, s_k^{m+1}) \in S_k$ representa uma solução que tem valor s_k^i como i-ésimo objetivo ($i = 1, \ldots, m$) e s_k^{m+1} de peso. Cada um dos estados $s_{k-1} \in S_{k-1}$ é incluído em S_k, além de uma cópia tendo o item k na solução, desde que esta seja viável (linha 4). Assim sendo, o conjunto S_k é formado por todas as soluções viáveis compostas de itens exclusivamente pertencentes aos k primeiros itens ($k = 1, \ldots, n$). Ao final dos n passos são descartadas as soluções não eficientes (linha 6) e retornado o conjunto Pareto (linha 7).

Pode-se observar que a linha 4 do algoritmo tem o potencial de dobrar o tamanho do conjunto S_k a cada iteração, uma vez que, para cada solução de S_{k-1} são adicionadas duas cópias em S_k, induzindo um crescimento exponencial do conjunto de estados. De fato o Algoritmo 1 pode vir a desnecessariamente manter algumas soluções não promissoras ao longo do algoritmo, ao descartar apenas as soluções inviáveis durante as iterações. Um exemplo são a própria solução vazia e outras soluções ineficientes, que são mantidas por todas as iterações, sendo descartadas apenas na linha 6.

O algoritmo Bazgan propõe reduzir este efeito através da utilização de algumas relações de dominância, capazes de descartar uma quantidade considerável de estados mantidos pelos conjuntos S_k. Este descarte resulta em ganho de performance, uma vez que o algoritmo trabalhará com conjuntos S_k menores.

A ordem na qual os itens são considerados é também uma questão crucial na implementação do algoritmo, pois a inserção de alguns itens com determinadas características pode fazer com que o conjunto de soluções mantidos pelo algoritmo cresça desde o início, o que não é o desejado. Sabe-se, por exemplo, que no caso do problema da mochila tradicional (escalar), para se obter uma boa solução, os itens devem ser geralmente considerados em ordem decrescente segundo a proporção p_i/w_i (EHRGOTT, 2013; MARTELLO; TOTH, 1990), porém, para o caso multiobjetivo não existe uma ordem natural semelhante.

Para tanto são introduzidas duas ordenações $\mathcal{O}^{sum}$ e $\mathcal{O}^{max}$ derivadas da agregação das ordens $\mathcal{O}^j$ inferidas pelas proporções p_i^j/w_i para cada objetivo $(j = 1, \ldots, m)$. Considere r_i^l o rank (ou seja, a posição) do item l na ordenação $\mathcal{O}^j$. $\mathcal{O}^{sum}$ denota uma ordenação segundo valores crescentes da soma dos ranks dos itens nas m ordenações $(i = 1, \ldots, m)$. $\mathcal{O}^{max}$ denota uma ordenação segundo valores crescentes de máximo ou piores ranks de itens nas m ordenações $\mathcal{O}^j(j = 1, \ldots, m)$, onde o pior rank do item l nas m ordenações $\mathcal{O}^j(j = 1, \ldots, m)$ é calculado como $max_{i=1}^m\{r_l^i\} + \frac{1}{mn}\sum_{i=1}^m r_l^i$ para distinguir itens com o mesmo rank máximo. Sempre que necessário será usada a notação $\mathcal{O}_{rev}$ para se referir à ordem reversa de uma ordenação $\mathcal{O}$.

Segundo os testes computacionais relatados em (BAZGAN; HUGOT; VANDERPOOTEN, 2009), a utilização da ordenação $\mathcal{O}^{max}$ reduz em, ao menos, 34% o tempo computacional total demandado pelo algoritmo, chegando a 99% em algumas instâncias, quando comparada a uma ordenação aleatória. Por este motivo, sempre que necessário, a ordenação $\mathcal{O}^{max}$ será utilizada neste trabalho para se definir a preferência de inserção dos itens na composição de uma solução.

A seguir, serão apresentadas algumas definições relativas a o contexto de programação dinâmica e conjunto de estados, importantes no desenvolvimento e aplicação das relações de dominâncias.

Definição 3 (Extensão, Restrição e Complemento)**.** *Considere o Algoritmo 1 e qualquer estado $s_k \in S_K(k < n)$. Um* complemento *de s_k é qualquer subconjunto $J \subseteq \{k+1, \ldots, n\}$ tal que $s_k^{m+1} + \sum_{j\in J} w_j \leqslant W$. Assume-se que qualquer estado $s_n \in S_n$ admite o conjunto vazio como único complemento. Um estado $s_n \in S_n$ é uma* extensão *de $s_k \in s_k(k \leqslant n)$ se, e somente se, existe um complemento I de s_k tal que $s_n^j = s_k^j + \sum_{i\in I} p_i^j$ para $j = 1, \ldots, m$ e $s_n^{m+1} = s_k^{m+1} + \sum_{i\in I} w_i$. O conjunto de extensões de s_k é denotado por $Ext(s_k)(k \leqslant n)$. Um estado $s_k \in S_k(k \leqslant n)$ é uma* restrição *do estado $s_n \in S_n$ se, e somente se, s_n é uma extensão de s_k.*

Definição 4 (Conjunto cobertura e Conjunto independente)**.** *Um conjunto $B \subseteq A$ é um* conjunto cobertura *(ou somente* cobertura*) de A com respeito à relação $\succsim$ se, e somente se, para todo $a \in A\backslash B$ existe $b \in B$ tal que $b \succsim a$. Um conjunto B é um* conjunto independente *com respeito a uma relação $\succsim$ se, e somente se, para todo $b, b' \in B, b \neq b', \neg(b \succsim b')$.*

Na seção seguinte serão apresentadas a definição de relação de dominância e a forma com que as relações de dominância são aplicadas ao Algoritmo 1.

2.1.1 Aplicação de Relações de Dominância

Definição 5 (Relação de dominância entre soluções)**.** *Considerando o Algoritmo 1, uma relação D_k sobre os elementos de $S_k, k = 1, \ldots, n$, é uma* relação de dominância *se, e*

somente se, para todo $s_k, s_{k'} \in S_k$ *vale*

$$s_k D_k s_{k'} \Rightarrow \forall s_{n'} \in Ext(s_{n'}), \exists s_n \in Ext(s_k), s_n \Delta s_{n'} \tag{2.2}$$

Considerando um passo k do algoritmo, uma relação de dominância D_k e duas soluções $s_k, s_{k'} \in S_k$, da Definição 5, conclui-se que se $s_k D_k s_{k'}$, então todas as soluções *geradas* por $s_{k'}$ nos passos seguintes serão dominadas por soluções geradas por s_k. Por este motivo, a solução $s_{k'}$ pode ser descartada do conjunto S_k, mesmo estas sendo viáveis. Vale notar que se $D_k^i,\ i = 1, \ldots, p$ $(p \geqslant 1)$ são relações de dominância então $D_k = \bigcup_{i=1}^{p} D_k^i$ é também uma relação de dominância.

Para se ter uma implementação eficiente do algoritmo de programação dinâmica, é recomendável utilizar múltiplas relações de dominância $D_k^1, \ldots, D_k^p$ $(p \geqslant 1)$ a cada execução da k-ésima iteração $k = 1, \ldots, n$, uma vez que cada relação D_k^i irá descartar um conjunto diferente de estados ao explorar características específicas do problema.

O Algoritmo 2 apresenta uma adaptação do Algoritmo 1 para utilizar múltiplas relações de dominância. O algoritmo inicia definindo o conjunto inicial de estados C_0 contendo apenas a solução vazia (linha 2). Em seguida são executadas n iterações onde, na k-ésima iteração, o k-ésimo item é considerado (linhas 3 a 7). Ao iniciar a k-ésima iteração, cada um dos estados $s_{k-1} \in C_{k-1}$ é incluído em C_k^0, além de uma cópia tendo o item k na solução, desde que esta seja viável (linha 4). Em seguida as p relações de dominância são aplicadas de forma sucessiva (linha 5 a 6). Para cada i-ésima relação de dominância, o conjunto C_k^i é definido pelos estados pertencentes à C_k^{i-1} que não são dominados por nenhum outro estado, segundo a relação de dominância D_k^i (linha 6). Após a aplicação das p relações de dominância, o conjunto C_k é definido como sendo o resultado da aplicação da última relação de dominância. Assim como o conjunto S_k do Algoritmo 1, o conjunto C_k é formado por todas as soluções viáveis compostas de exclusivamente dos primeiros k itens $(k = 1, \ldots, n)$, porém, o conjunto C_k é um conjunto reduzido, um a vez que cada i-ésima iteração teve por objetivo eliminar as soluções dominadas segundo a relação D_k^i. Concluídas as n iterações, o algoritmo retorna o conjunto C_n que caracteriza o conjunto Pareto, uma vez que todas as soluções dominadas foram descartadas.

Algoritmo 2: Programação dinâmica utilizando múltiplas relações de dominância.

```
1 begin
2     C_0 ← {(0, ..., 0)};
3     for k ← 1, n do
4         C_k^0 ← C_{k-1} ∪ {(s^1 + p_k^1, ..., s^m + p_k^m, s^{m+1} + w_k) | s^{m+1} + w_k ⩽ W, s ∈ C_{k-1}};
5         for i ← 1, p do
6             C_k^i ← {s ∈ C_k^{i-1} | ∄(s' ∈ C_k^{i-1})[s' D_k^i s]};
7         C_k ← C_k^p;
8     return C_n;
```

2.1.2 As Relações de Dominância Utilizadas

A seguir serão apresentadas as três relações de dominância utilizadas no algoritmo. Cada relação de dominância explora uma consideração específica. Por isto recomenda-se utilizar relações de dominância que sejam complementares. Além disso, para se escolher uma relação de dominância é necessário considerar sua potencial capacidade de descarte de estados diante do custo computacional requerido. As primeiras duas relações são razoavelmente triviais de se estabelecer, sendo a última ainda considerada mesmo um tanto mais complexa, devido a sua complementaridade diante das duas primeiras.

A primeira relação de dominância tem o objetivo de evitar a geração de soluções ineficientes através do descarte de soluções relativamente *vazias*, tipicamente geradas nas iterações iniciais do algoritmo. A segunda relação de dominância é uma generalização para o MOKP da clássica relação de dominância proposta por Weignartner e Ness e utilizada no Algoritmo 1. A terceira relação de dominância visa descartar soluções consideradas não promissoras, i.e., que não têm potencial de aumento de função objetivo suficiente para superar o de outras soluções já existentes.

2.1.2.1 A Relação D^r

A primeira relação de dominância se estabelece segundo a seguinte observação. Quando a capacidade residual de uma solução associada a um estado s_k da iteração k é maior ou igual à soma dos pesos dos itens restantes, i.e. itens $k+1, \ldots, n$, o único complemento de s_k que pode resultar em uma solução eficiente é o complemento máximo $I = \{k+1, \ldots, n\}$. Portanto, neste contexto, não se faz necessário gerar as extensões de s_k que não contenham todos os itens restantes. Define-se então a relação de dominância D_k^r sobre S^k para $k = 1, \ldots, n$ como:

$$\forall s_k, s_{k'} \in S_k, \;\; s_k D_k^r s_{k'} \Leftrightarrow \begin{cases} s_{k'} \in S_{k-1}, \\ s_k = (s_{k'}^1 + p_k^1, \ldots, s_{k'}^m + p_k^m, s_{k'}^{m+1} + w_k), \text{ e} \\ s_{k'}^{m+1} \leqslant W - \sum_{i=k}^{n} w_i \end{cases}$$

A seguinte proposição enuncia que a relação D_k^r é de fato uma relação de dominância.

Proposição 1 (Relação D_k^r)**.** *D_k^r é uma relação de dominância*

Demonstração. Considere dois estados s_k e $s_{k'}$ tal que $s_k D_k^r s_{k'}$. Uma vez que $s_k D_k^r s_{k'}$ temos que $s_k \Delta s_{k'}$ e consequentemente $s_k^{m+1} = s_{k'^{m+1}} + w_k \leqslant W - \sum_{i=k+1}^{n} w_i$. Sendo assim, qualquer subconjunto $I \subseteq \{k+1, \ldots, n\}$ é um complemento de $s_{k'}$ e s_k. Portanto, para todo estado $s_{n'} \in Ext(s_{k'})$, existe $s_n \in Ext(s_k)$, baseado no mesmo complemento que $s_{n'}$, tal que $s_n \Delta s_{n'}$. Isto estabelece que D_k^r satisfaz a Equação 2.2 da Definição 5. □

Esta relação de dominância é um tanto fraca, uma vez que em cada k-ésima iteração ela se aplica apenas entre um estado que não contém o k-ésimo item e sua extensão, que

contém o k-ésimo item. Apesar disso, é uma relação de dominância extremamente fácil de ser verificada, uma vez que, ao ser o valor de $W - \sum_{i=k}^{n}$ computado, a relação D_k^r requer apenas uma comparação para ser estabelecida entre dois estados.

2.1.2.2 A Relação D_k^{Δ}

A relação de dominância seguinte é a generalização para o caso multiobjetivo da relação de dominância utilizada no Algoritmo 1. Esta segunda relação de dominância é definida sobre s_k para $k = 1, \ldots, n$ por:

$$\forall s_k, s_{k'} \in S_k,\ s_k D_k^{\Delta} s_{k'} \Leftrightarrow \begin{cases} s_k \Delta s_{k'} & \text{e} \\ s_k^{m+1} \leqslant s_{k'}^{m+1} & \text{se } k < n \end{cases}$$

Observe que a condição sobre os pesos s_k^{m+1} e $s_{k'}^{m+1}$ garante que todo complemento de $s_{k'}$ é também um complemento de s_k. A seguinte proposição enuncia que D_k^{Δ} é de fato uma relação de dominância.

Proposição 2 (Relação D_k^{Δ})**.** *D_k^{Δ} é uma relação de dominância*

Demonstração. Considere estados s_k e $s_{k'}$ tal que $s_k D_k^{\Delta} s_{k'}$. Uma vez que $s_k D_k^{\Delta} s_{k'}$, tem-se que $s_k \Delta s_{k'}$, e consequentemente $s_k^{m+1} \leqslant s_{k'}^{m+1}$. Portanto, qualquer subconjunto $J \cup \{k+1, \ldots, n\}$ complemento de $s_{k'}$ é também complemento de s_k. Assim, para todo $e_{n'} \in Ext(s_{n'})$, existe $e_n \in Ext(s_n)$, baseado no mesmo complemento de $s_{n'}$. O que estabelece que D_k^{Δ} satisfaz a Equação 2.2 da Definição 5. □

A relação D_k^{Δ} é uma relação de dominância poderosa, uma vez que um estado pode possivelmente dominar todos os outros estados de maior peso. Esta relação requer no máximo $m + 1$ comparações para ser estabelecida entre dois estados.

2.1.2.3 A Relação D^b

A terceira relação de dominância é baseada na comparação entre extensões específicas de um estado e um limite superior de extensões de outros estados. Um limite superior de um estado é definido a seguir.

Definição 6 (Limite superior)**.** *Um vetor objetivo $u = (u^1, \ldots, u^m)$ é um* limite superior *de um estado $s_k \in S_k$ se, e somente se, $\forall s_n \in Ext(s_k)$ tem-se que $u^i \geqslant s_n^i, i = 1, \ldots, m$.*

Um tipo geral de relação de dominância pode ser derivada da maneira seguinte. Considere dois estados $s_k, s_{k'} \in S_k$. Se existe um complemento I de s_k e um limite superior $\tilde{u}$ de $s_{k'}$ tal que $s_k^j + \sum_{i \in I} p_i^j \geqslant \tilde{u^j},\ j = 1, \ldots, m$, então s_k domina $s_{k'}$.

Para se implementar este tipo de relação de dominância é necessário especificar os complementos e os limites superiores a serem utilizados. Na implementação do algoritmo em questão, são considerados dois complementos específicos I' e I'' obtidos pelo simples

algoritmo de preenchimento *guloso* definido a seguir. Após rerotular os itens $k+1,\ldots,m$ de acordo com a ordenação $\mathcal{O}^{sum}$ (e $\mathcal{O}^{max}$ respectivamente), o complemento I' (e I'' respectivamente) são obtidos através a inserção sequencial dos itens restantes na respectiva solução, de forma que a restrição de capacidade é respeitada.

Como definição do limite superior u é aplicado a cada valor de objetivo a definição de limite superior proposta por (MARTELLO; TOTH, 1990) para o caso escalar. Considere $\overline{W}(s_k) = W - s_k^{m+1}$ a capacidade residual associada ao estado $s_k \in S_k$. Denota-se por $c_j = min\Big\{l_j \in \{k_1,\ldots,n\} \;\Big|\; \sum_{i=k+1}^{l_j} w_i > \overline{W}(s_k)\Big\}$ a posição do primeiro item que não pode ser adicionado ao estado $s_k \in S_k$ quando os itens $k+1,\ldots,n$ são rerotulados de acordo com a ordenação $\mathcal{O}^j$. Desde modo, segundo (MARTELLO; TOTH, 1990), quando os itens $k+1,\ldots,n$ são rerotulados de acordo com a ordenação $\mathcal{O}^j$, um limite superior para o j-ésimo valor objetivo de $s_k \in S_k$ é para $j = 1,\ldots,m$:

$$u^j = s_k^j + \sum_{i=k+1}^{c_j-1} p_i^j + max\left\{\left\lfloor \overline{W}(s_k)\frac{p^j_{c_j+1}}{w_{c_j+1}}\right\rfloor, \left\lfloor p^j_{c_j} - \Big(w_{c_j} - \overline{W}(s_k)\Big).\frac{p^j_{c_j-1}}{w_{c_j-1}}\right\rfloor\right\} \tag{2.3}$$

Finalmente definimos D_k^b uma relação de dominância como caso particular do tipo geral para $k = 1,\ldots,n$ por:

$$\forall s_k, s_{k'} \in S_k,\ s_k D_k^b s_{k'} \Leftrightarrow \begin{cases} s_k^j + \sum_{i\in I'} p_i^j \geqslant \tilde{u^i}, & j = 1,\ldots,m \\ \text{ou} & \\ s_k^j + \sum_{i\in I''} p_i^j \geqslant \tilde{u^i}, & j = 1,\ldots,m \end{cases}$$

onde $\tilde{u} = (\tilde{u}_1,\ldots,\tilde{u}_m)$ é o limite superior para $s^{k'}$ computado de acordo com a Equação 2.3.

A seguinte proposição mostra que D_k^b é de fato uma relação de dominância.

Proposição 3 (Relação D_k^b)**.** *D_k^b é uma relação de dominância*

Demonstração. Considere os estados s_k e $s_{k'}$ tal que $s_k D_k^b s_{k'}$. Isto implica que existe $I \in I', I''$ capaz de gerar um estado $s_n \in Ext(s_{k'})$. Além disso, uma vez que $\tilde{u}$ é um limite superior de $s_{k'}$, temos $\tilde{u}\Delta s_{n'}, \forall s_{n'} \in Ext(s_{n'})$. Portanto, por transitividade de Δ, temos que $s_n \Delta s_{n'}$, o que estabelece que D_k^b satisfaz a condição da Equação 2.2 da Definição 5. □

A relação D_k^b é mais difícil de ser verificada do que as relações D_k^r e D_k^Δ uma vez que requer uma maior quantidade de comparações e informações sobre outros estados.

Obviamente, a relação D_k^b seria mais forte se utilizados complementos adicionais (de acordo com outras ordenações) para s_k e computado, ao invés de apenas um limite superior u, um conjunto de limites superiores, como, por exemplo, a proposta apresentada em (EHRGOTT; GANDIBLEUX, 2007). Porém, no contexto em questão, uma vez que se tem que verificar D_k^b para muitos estados, enriquecer D_k^b dessa forma demandaria um esforço computacional alto demais.

Para se ter uma implementação eficiente, são utilizadas as três relações de dominância apresentadas a cada iteração. Como dito anteriormente, uma relação de dominância

demanda maior esforço computacional do que outra para ser verificada. Além disso, mesmo que sejam parcialmente complementares, é frequente mais de uma relação de dominância ser válida para o mesmo par de estados. Por esse motivo é natural aplicar primeiramente relações de dominância que podem ser verificadas facilmente (como D_k^r e D_k^Δ) para depois verificar, em um conjunto reduzido de estados, relações mais custosas (como D_k^b).

Algoritmo 3: Algoritmo Bazgan.

```
   input: p, w, W
1  begin
2      S_0 ← {(0, ..., 0)};
3      o_1, ..., o_n = O^max;
4      for k ← 1, n do
5          S*_k ← {(s^1 + p^1_{o_k}, ..., s^m + p^m_{o_k}, s^{m+1} + w_{o_k}) | s ∈ S_{k-1}, s^{m+1} + w_{o_k} ⩽ W}
6                 ∪ {s ∈ S_{k-1} | s^{m+1} + w_{o_k} + ... + w_{o_n} > W};
7          S**_k ← {s ∈ S*_k | (∄u ∈ S*_k)[uΔs]};
8          S_k ← {s ∈ S**_k | (∄u ∈ S**_k)[lb(u)Δub(s)]};
9      return S_n;
```

O Algoritmo 3 apresenta o algoritmo Bazgan que aplica as três relações de dominâncias, definidas nesta Seção, ao algoritmo 2. Primeiramente o conjunto de estados inicial S_0 é definido contendo apenas o estado vazio (linha 2). Na linha 3 é definida a ordem com que os itens serão considerados nas iterações para serem introduzidos aos estados. A cada execução do laço das linhas 4 a 8 é gerado o conjunto S_k de estados, contendo apenas soluções viáveis, cujos itens é um subconjunto dos primeiros k itens sendo ainda S_k conjunto independente com respeito as relações de dominância D_k^r, D_k^Δ e D_k^b. Na linha 5 é definido o conjunto S_k^* que consiste de todos os estados contidos em S_{k-1} adicionados do item k, desde que sejam viáveis (linha 5) e também dos estados contidos no conjunto S_{k-1} desde que sua capacidade disponível não exceda a soma dos pesos dos itens restantes (linha 6), o que corresponde à aplicação da relação D_k^r. Na linha 7 são selecionados do conjunto S_k^* as soluções que não tem nenhuma outra que as dominem (aplicação da relação D_k^Δ). Na linha 8 são selecionados do conjunto S_k^{**} as soluções cujo limite superior não é dominado pelo limite inferior de nenhuma outra solução (aplicação da relação D_k^b). Ao final do algoritmo (linha 9) é retornado o conjunto S_n de estados não dominados representando o conjunto Pareto.

Como dito anteriormente o Algoritmo 3 será utilizado para verificar a eficácia da indexação multidimensional como proposta de aceleração da operação de verificação de dominância no contexto exato. Pode-se observar que a operação de verificação de dominância é utilizada nesse caso em dois pontos:

1. verificação da condição $u\Delta s$ (linha 7);
2. verificação da condição $lb(u)\Delta ub(s)$ (linha 8).

Nesses dois pontos do algoritmo será aplicada a proposta, permanecendo inalterado os demais pontos do algoritmo. Os impactos na performance serão examinados através de experimentos computacionais posteriormente apresentados e discutidos, no Capítulo 4.

A Seção seguinte abordará o contexto heurístico para o MOKP, no qual será proposta a implementação de uma heurística para o MOKP, baseada em um algoritmo evolucionário. Os principais algoritmos da literatura serão mencionados, e posteriormente o algoritmo evolucionário será apresentado para então ser aplicado ao MOKP.

2.2 Abordagem Heurística

Uma das principais dificuldades encontradas no problema da mochila multiobjetivo é a grande cardinalidade do conjunto Pareto. De fato, grande parte dos problemas multiobjetivos são considerados *intratáveis*, no sentido de possuir uma quantidade exponencial de soluções eficientes dado o tamanho da instância (EHRGOTT, 2013). Esta causa tem motivado pesquisadores a desenvolverem métodos heurísticos, que computam aproximações deste conjunto demandando menor esforço computacional quando comparado aos métodos exatos.

A maioria das heurísticas propostas para problemas multiobjetivo são adaptações de meta-heurísticas originalmente propostas para problemas de otimização escalar. Dentre esses métodos vale mencionar o algoritmo de recozimento simulado (CZYZŻAK; JASZKIEWICZ, 1998), o algoritmo de busca dispersa (SILVA; CLÍMACO; FIGUEIRA, 2006; SILVA; FIGUEIRA; CLÍMACO, 2007), algoritmo de busca tabu (GANDIBLEUX; FREVILLE, 2000), algoritmo imunológico artificial (GAO et al., 2014), algoritmo genético (ABDELAZIZ; KRICHEN; CHAOUACHI, 1999) e o algoritmo de estimação de distribuição (MARTINS et al., 2017).

Uma das heurísticas mais populares para o MOKP é o algoritmo genético de ordenação não dominada (NSGA-II) proposto em (DEB et al., 2002). O NSGA-II se diferencia das primeiras propostas de algoritmos evolucionários multiobjetivo por utilizar uma estratégia mais eficiente de ordenação de soluções dominantes e, consequentemente de definição de aptidão do indivíduo. O processo de ordenação de soluções dominantes agrupa as soluções em conjuntos ordenados, chamados *frontes não dominados*, nos quais as soluções dos primeiros conjuntos não são dominadas pelas soluções dos conjuntos seguintes. A estratégia consiste em realizar um pré-processamento sobre o conjunto de soluções, no qual são computados para cada solução x dois parâmetros: a quantidade de soluções que dominam x e o conjunto de soluções dominadas por x. Este pré-processamento permite a otimização do procedimento de ordenação. Sendo m o número de objetivos e N o tamanho da população, o procedimento de ordenação proposto requer apenas $\mathcal{O}(mN^2)$ comparações enquanto que os procedimentos anteriormente utilizados demandavam $\mathcal{O}(mN^3)$ comparações.

Com o objetivo de manter a diversidade das soluções, o NSGA-II também propõe a

utilização de uma métrica de distância entre os indivíduos da população baseada na média das distâncias entre uma solução e suas vizinhas imediatas pertencentes ao mesmo fronte não dominado. A média calculada caracteriza uma espécie de *medida de aglomeração*. Soluções rodeadas por outras soluções próximas possuem um alto valor de aglomeração, enquanto soluções que apresentam uma vizinhança distante possuem baixo valor de aglomeração. O algoritmo então tende a dar preferência às soluções com menor valor de aglomeração, uma vez que manter soluções com alto valor de aglomeração tende a diminuir a diversidade do conjunto de soluções. O NSGA-II então implementa um algoritmo genético com abordagem elitista, considerando quando necessário, a métrica proposta como critério de desempate.

Outra heurística popular para o MOKP é o algoritmo evolucionário de pareto robusto (SPEA-II), proposto em (ZITZLER; LAUMANNS; THIELE, 2001). O SPEA-II mantém durante o processo de otimização um conjunto de tamanho limitado, desassociado da população corrente, contendo soluções não dominadas encontradas durante o processo. Este conjunto, chamado de *arquivo externo* ou somente *arquivo* (*external archive* ou *archive* em inglês) é atualizado ao final de cada etapa de evolução, quando os melhores indivíduos da população corrente são tomados como candidatos para inserção nesse arquivo.

O SPEA-II também propõe utilizar um valor refinado de aptidão. Para as soluções não-dominadas, o valor de aptidão é associada ao número de soluções da população que são dominadas pela respectiva solução. Para os indivíduos dominados, o valor de aptidão é calculado com base nos valores de aptidão das respectivas soluções dominantes. Como critério de desempate é utilizada uma métrica de densidade, baseada na distância até os k vizinhos mais próximos. Assim que o processo evolutivo se encerra, o SPEA-II retorna o arquivo externo como sendo a aproximação do conjunto Pareto.

Outro algoritmo heurístico popular, para o qual se relata os melhores desempenhos para o MOKP, é o algoritmo multiobjetivo evolucionário baseado em decomposição (MOEA/D) (ZHANG; LI, 2007). O MOEA/D decompõe o problema original em N subproblemas escalares, utilizando um vetor de coeficientes de agregação. Esses N subproblemas são solucionados através da evolução simultânea de N populações, uma população para cada subproblema. A cada geração a população é composta pelo melhor indivíduo encontrado até o momento para aquele determinado subproblema, o que garante a convergência em direção ao conjunto Pareto. Além disso são definidas relações de vizinhança entre os subproblemas, baseadas nas distâncias entre seus vetores de agregação. Durante o processo de evolução, são utilizadas informações de subproblemas vizinhos, fazendo com que dois subproblemas vizinhos tenham soluções similares. O trabalho ainda discute diferentes estratégias para se definir os vetores de agregação.

Recentemente foi proposta uma heurística cooperativa para o MOKP baseado em inteligência de enxame (MOFPA) (ZOUACHE; MOUSSAOUI; ABDELAZIZ, 2018), apresentando resultados superiores às demais heurísticas. O MOFPA combina a estratégia

de movimento utilizada na otimização por enxame de partículas com a estratégia de movimento utilizada pelo algoritmo de otimização por colônia de vaga-lumes. O algoritmo mantém uma população de indivíduos cujas posições são atualizadas segundo a influência de outros indivíduos da população, em uma movimentação par-a-par. A estratégia de movimento a ser aplicada é definida segundo a relação de dominância entre os dois indivíduos: caso o indivíduo a ser movimentado domine o outro, o cálculo de movimento é feito segundo o método de otimização por enxame de partículas, caso contrário, o indivíduo é movimentado segundo o cálculo de atratividade entre indivíduos, definido pelo algoritmo de colônia de vaga-lumes. Visto que o movimento dos indivíduos ocorre em um espaço contínuo, uma função de transição é aplicada para discretização da solução. A estratégia de arquivo externo de soluções não dominadas é utilizada para garantir as melhores soluções encontradas durante o processo de evolução, assegurando assim a convergência do método em direção ao conjunto Pareto. Segundo os resultados computacionais apresentados, o MOFPA foi capaz de gerar soluções que superam diversas soluções estabelecidas pelos métodos anteriores.

Uma das propostas do presente trabalho é a implementação de um algoritmo evolutivo populacional chamado algoritmo de evolução estocástica por complexos, *shuffled complex evolution* (SCE) em inglês, para o problema da mochila multiobjetivo. O SCE é uma heurística evolutiva robusta que surgiu como proposta de solução para problemas hídricos de modelagem complexa (DUAN; SOROOSHIAN; GUPTA, 1992) e deste então tem sido utilizada em outros problemas de otimização escalar. Na seção seguinte o SCE será apresentado bem como as propostas de adaptação para ser aplicado ao caso multiobjetivo e de implementação para o MOKP.

Esta proposta de implementação do algoritmo SCE para o MOKP será utilizada para testar o desempenho da indexação multidimensional para o MOKP no contexto heurístico. Visando validar a heurística proposta, sua performance com relação à qualidade de solução também será comparada às principais heurísticas da literatura, cujos resultados serão apresentados no Capítulo 4.

2.2.1 Evolução Estocástica por Complexos

O Algoritmo de evolução estocástica por complexos (SCE) é um algoritmo evolutivo populacional originalmente proposto por Duan (DUAN; SOROOSHIAN; GUPTA, 1992) para auxiliar o planejamento de modelos de escoamento de águas pluviais. Esse algoritmo tem sido utilizado com sucesso em diversos problemas como seleção de projetos (ELBELTAGI; HEGAZY; GRIERSON, 2007), problemas de escalonamento (ZHAO et al., 2015), problema da mochila $0-1$ (BHATTACHARJEE; SARMAH, 2014) e problema da mochila multi-dimensional (BARONI; VAREJÃO, 2015; BARONI; VAREJÃO, 2016).

O SCE é inspirado na evolução natural que ocorre de forma simultânea em comunidades independentes. A Figura 3 apresenta um fluxograma resumido do SCE. O algoritmo

trabalha com uma população particionada em N comunidades, ou complexos, cada uma contendo M indivíduos. Inicialmente a população de $N * M$ indivíduos é tomada aleatoriamente do espaço de soluções viáveis. Após essa inicialização, a população é ordenada em ordem decrescente de aptidão e o melhor global é identificado. Toda a população é então particionada em N complexos, cada um contendo M indivíduos. Nesse processo de distribuição ocorre uma operação de *embaralhamento* (*shuffling* em inglês) dos indivíduos, no qual o primeiro indivíduo é incluído no primeiro complexo, o segundo indivíduo no segundo complexo, o M-ésimo indivíduo no M-ésimo complexo, o $M+1$-ésimo indivíduo no primeiro complexo, e assim por diante.

O próximo passo após a distribuição dos indivíduos em complexos é evoluir cada complexo K' vezes, sendo K' uma constante. Nesse processo, considerando que os indivíduos em cada complexo estão ordenados em ordem decrescente de aptidão, um subcomplexo de P indivíduos é selecionado dentre o respectivo complexo, utilizando uma distribuição triangular, em que o i-ésimo indivíduo tem a probabilidade $p_i = \frac{2(n+1-i)}{n(n+1)}$ de ser selecionado. A utilização da distribuição triangular tem por objetivo priorizar os indivíduos com melhor aptidão, favorecendo assim a convergência do algoritmo.

Após a seleção do subcomplexo, o seu pior indivíduo é identificado para ser substituído por um novo indivíduo. Esse novo indivíduo é gerado através do cruzamento do pior com um outro de melhor aptidão. Primeiramente o melhor indivíduo do subcomplexo é considerado. Caso o indivíduo gerado não seja melhor que o pior selecionado, o melhor indivíduo do complexo é então usado no cruzamento. Se esse último cruzamento não resultou em melhoria, o melhor indivíduo de toda a população é utilizado. Finalmente, se todos os cruzamentos não foram capazes de gerar um melhor indivíduo, esta pior solução selecionada é substituída por um novo indivíduo retirado de forma aleatória do espaço de soluções viáveis. Esse último procedimento é importante para impedir que o algoritmo fique *preso* num ótimo local. O procedimento de evolução descrito acima é apresentado no fluxograma da Figura 4.

Após evoluir todos os N complexos toda a população é novamente ordenada em ordem decrescente de aptidão e o processo continua até que a condição de parada seja satisfeita. Destaca-se que no fluxograma da Figura 3 a condição de parada é a execução de uma quantidade fixa de K evoluções.

Uma das adaptações necessárias para se aplicar o SCE a um problema multiobjetivo é a redefinição de métrica de qualidade de solução (aptidão), uma vez que esta não é explícita como no caso da otimização escalar. A métrica utilizada nesta proposta será a mesma adotada por heurísticas como NSGA-II e MOFPA, a qual se baseia na ordenação não dominada (*nondominated sorting*, em inglês) das soluções, por apresentarem os melhores resultados relatados na literatura.

A ordenação não dominada agrupa as soluções em conjunto chamados *frontes não dominados*, os quais possuem uma ordenação de dominância entre si. O primeiro fronte

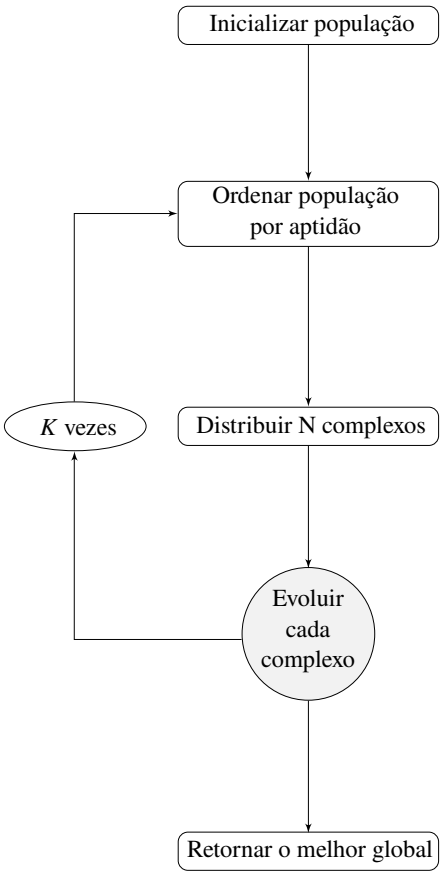


Figura 3 – Algoritmo de evolução estocástica por complexos.

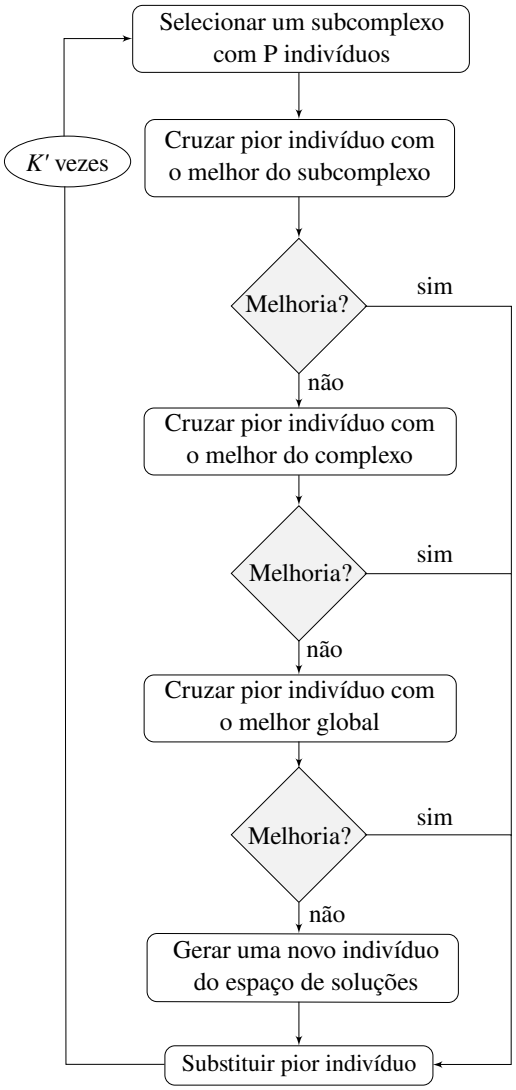


Figura 4 – Etapa de evolução executada para cada um dos complexos.

é formado pelas soluções não dominadas. Já o segundo fronte é formado pelas soluções dominadas apenas pelo primeiro fronte. O terceiro fronte, por sua vez, é formado pelas soluções dominadas pelo primeiro e segundo fronte, e assim sucessivamente. Dessa forma as soluções dos frontes anteriores são consideradas superiores às soluções dos frontes posteriores.

O Algoritmo 4 apresenta o algoritmo proposto por (DEB et al., 2002) e utilizado neste trabalho para ordenar as soluções em frontes não dominados. A primeira fase do algoritmo é apresentada nas linhas 2 a 8, onde algumas informações são pré-computadas. Nas linhas 9 a 17 executa-se a segunda fase do algoritmo, onde as soluções são finalmente agrupadas nos frontes. Primeiramente são inicializados para cada solução i os contadores de soluções dominadas n_i (linha 2) e o conjunto de soluções dominantes B_i (linha 3). Em seguida é feita uma comparação par-a-par através dos laços das linhas 4 e 5 para verificar

se uma solução domina a outra (linha 6). Caso isso aconteça, o contador de soluções dominantes e o conjunto de suas soluções dominadas é atualizado. Em seguida é definido o primeiro fronte de pareto, formado pelas soluções não dominadas por nenhuma outra, ou seja, soluções que tem $n_i = 0$ (linha 9). Enquanto houver alguma solução com ao menos uma solução dominante registrada (linha 11) o algoritmo define um novo fronte (linhas 12 a 16). Para cada solução que compõe o fronte anterior (linha 13), a lista de soluções dominadas é visitada (linha 14), decrementando-se seus contadores de dominantes (linha 15). Esse último passo faz com que a contagem de soluções dominantes n_i desconsidere as soluções recém agrupadas. Na linha 16 o novo fronte é definido. Quando todas as soluções estiverem agrupadas em frontes, o algoritmo é encerrado, retornando a lista de frontes.

Algoritmo 4: Algoritmo de ordenação de soluções em frontes não dominados.

```
   input: A = (a_1, ..., a_q) : conjunto de soluções
   result: (F_1, ..., F_k) : Lista de frontes de pareto
1  begin
2      n_1, ..., n_q ← 0;
3      B_1, ..., B_q ← {};
4      for i ← 1 : q do
5          for j ← 1 : q do
6              if a_i Δ a_j then
7                  n_j ← n_j + 1;        ▷ quantidade de soluções que dominam a_j
8                  B_i ← B_i ∪ {a_j};    ▷ conjunto das soluções dominadas por a_i
9      k ← 1;
10     F_1 ← {a_i ∈ A | n_i = 0};        ▷ soluções não-dominadas (1° fronte)
11     while ∃i ∈ {1, ..., q} | n_i > 0 do
12         k ← k + 1;
13         for a_i ∈ F_{k-1} do
14             for a_j ∈ B_i do
15                 n_j ← n_j − 1;        ▷ atualizando contagem
16         F_k ← {a_i ∈ A | n_i = 0};    ▷ formação do k° fronte
17     return (F_1, ..., F_k);
```

A Figura 5 apresenta um exemplo de população para um problema bi-objetivo sem estarem ordenadas. A Figura 6 apresenta esta mesma população agora ordenada, ou seja, agrupada em frontes não dominados, conforme computado pelo Algoritmo 4.

O desempate de aptidão entre indivíduos do mesmo fronte é feito pelo valor de *hiper-volume* de cada solução, como sugerido em (AUGER et al., 2012), por ser uma métrica que exige pouco esforço computacional e tende a prover um conjunto Pareto de melhor qualidade. O hiper-volume de uma solução x de um problema m-objetivo é dado por $\prod_{i=1}^{m} f_i(x)$ e representa o volume multidimensional dominado pela solução. As soluções de um mesmo fronte mas, porém, com maior hiper-volume, serão consideradas superiores as com menor hiper-volume.

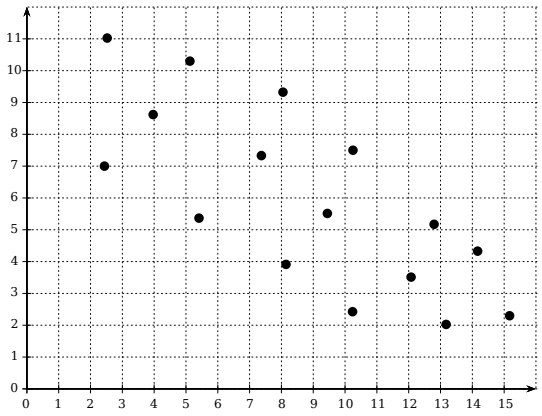


Figura 5 – População sem ordenação.

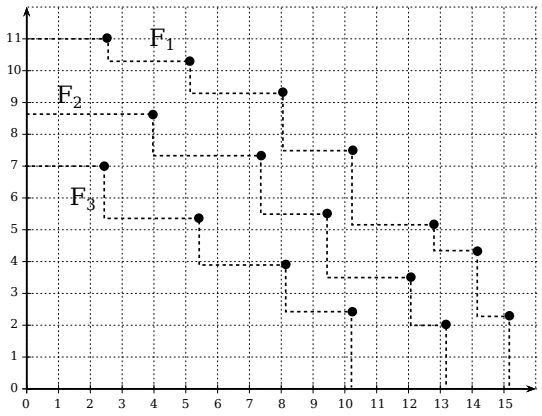


Figura 6 – População ordenada em frontes não dominados.

Uma última adaptação necessária para o SCE ser aplicado a problemas multiobjetivo é em relação ao retorno do algoritmo, que deve ser um conjunto de soluções não dominadas, representando uma aproximação do conjunto Pareto, ao invés de apenas uma solução apenas. Para que isso seja possível, será utilizada a estratégia de arquivo externo, o qual será atualizado a cada final de etapa de evolução. O Algoritmo 5 apresenta o procedimento de atualização do arquivo, dada uma nova solução candidata.

Algoritmo 5: Procedimento de atualização de arquivo, dada uma nova solução.

```
   input: A : arquivo, x : indivíduo
1  begin
2      if ∄(y ∈ A, yΔx) then
3          A ← A ∪ {x};                ▷ Inclusão de x no arquivo
4          A ← A\{z ∈ A | xΔz};        ▷ Remoção das soluções dominadas por x
5      return A;
```

O Algoritmo 5 recebe como entrada o conjunto arquivo A e uma solução candidata x. Primeiramente o algoritmo verifica se a solução x é não dominada pelo arquivo, ou seja, se não existe nenhuma solução $y \in A$ que domine x (linha 2). Em caso verdadeiro, dois passos são executados: x é adicionado ao arquivo (linha 3) e são retiradas do arquivo as soluções dominadas por x (linha 4). Finalmente o arquivo atualizado é retornado (linha 5).

Com a utilização do arquivo externo e a definição de aptidão baseada na ordenação não dominada, o SCE está adaptado para resolver problemas multiobjetivo. Entretanto, para se aplicar o SCE especificamente ao MOKP, ainda se faz necessário definir dois procedimentos: o procedimento de criação de uma nova solução aleatória viável e o procedimento de cruzamento entre duas soluções. Os procedimentos utilizados neste trabalho para a construção de uma solução aleatória e para o cruzamento de duas soluções são descritos nos Algoritmos 6 e 7 respectivamente.

Algoritmo 6: Construção de solução aleatória para o MOKP.

```
1 begin
2     v ← shuffle(1, 2, ..., n);
3     x ← 0⃗;                    ▷ solução vazia
4     for i ← 1 : n do
5         x[v_i] ← 1;            ▷ inserção de item
6         if w(x) > W then
7             x[v_i] ← 0;        ▷ verificação de viabilidade
8     return s;
```

O Algoritmo 6 primeiramente ordena os índices dos n itens numa ordem aleatória e armazena-os em uma lista v (linha 2). Após a definição de uma nova solução vazia (linha 3), o algoritmo tenta de forma iterativa preencher a solução com um item retirado da lista de índices (linhas 4-9). A viabilidade da solução é então verificada: se o item inserido tornar a solução inviável, ou seja, exceder a capacidade da mochila (linha 6), ele é então retirado da solução (linha 7). Após testar a inserção de todos os itens, a solução construída é retornada.

Algoritmo 7: Procedimento de cruzamento entre duas soluções do MOKP.

```
   input: x: pior indivíduo, y: melhor indivíduo, c: número de genes herdados
1  begin
2      v ← shuffle(1, 2, ..., n);
3      for i ← 1 : c do
4          x[v_i] ← y[v_i];          ▷ herança de genes
5      DROP-ADD(x);                  ▷ reparo e complementação de solução
6      Computar aptidão de x;
7      return x;
```

O procedimento de cruzamento (Figura 7) recebe como entrada o pior indivíduo x vindo do subcomplexo selecionado, um indivíduo y com maior aptidão que x e o parâmetro c sendo o número de genes a serem carregados de b. O parâmetro c controla o quão similar o novo indivíduo será do melhor indivíduo dado como entrada. Primeiramente os índices dos n itens são dispostos em ordem aleatória e armazenados numa lista (linha 2). Os c genes escolhidos são então carregados do melhor indivíduo para o pior indivíduo (linhas 3-4). Em seguida a solução é reparada utilizando o procedimento DROP-ADD (linha 5). O procedimento DROP-ADD, além de garantir a viabilidade da solução, busca melhorá-la, tentando preencher possíveis espaços vazios. Finalmente, a aptidão da solução gerada é atualizada (linha 6) e então retornada (linha 7). O procedimento de DROP-ADD é detalhado no Algoritmo 8.

O Algoritmo 8 é dividido em duas etapas principais: viabilização da solução (linhas 3 a 7) e complementação da solução (linhas 8 a 12). Inicialmente é atribuída a v a ordenação

Algoritmo 8: Procedimento DROP-ADD de reparo e complementação de solução.

```
   input: x: solução inviável
1  begin
2      v ← O^max(1, 2, ..., n);
3      i ← n;
4      while w(x) > W   ▷ tratamento de inviabilidade
5       do
6          x[v_i] ← 0;
7          i ← i − 1;
8      for i ← 1 : n        ▷ complementação da solução
9       do
10         if x[v_i] = 0 then
11             if w(x) + w[v_i] ⩽ W then
12                 x[v_i] ← 1;
13     return x;
```

$\mathcal{O}^{max}$ dos índices dos itens. Na linha 3 o último índice é atribuído a i. Enquanto a solução é inviável (linha 4) retira-se um item, dando prioridade na retirada dos últimos itens, segundo a ordenação em v (linha 6). Em seguida o algoritmo tenta inserir mais itens à solução, desde que a viabilidade seja mantida (linha 11), dando prioridade aos primeiros itens, segundo a ordenação em v (linha 12). Finalmente a solução é retornada na linha 13.

O Algoritmo 9 apresenta o pseudo-código do algoritmo SCE proposto para o MOKP com os procedimentos de adaptação ao caso multiobjetivo. Na linha 2 a população de $N * M$ indivíduos é inicializada utilizando o Algoritmo 6. Na linha 3 a população inicial é classificada em frontes não dominados, utilizando o Algoritmo 4. Na linha 4 o arquivo externo é inicializado a partir das soluções do 1º fronte. As linhas 5 a 14 executam as K iterações evolucionárias do algoritmo. Na linha 6 a população é ordenada em ordem decrescente de aptidão. Na linha 7 a população é distribuída em N utilizando o procedimento de embaralhamento descrito. O laço da linha 8 executa o procedimento de evolução sobre cada um dos N complexos, segundo o diagrama da Figura 4. Após a evolução dos M complexos, a população antiga (composta por $N * M$ indivíduos) juntamente com a nova população (composta por $N * K'$ indivíduos) são classificados em frontes não dominados, utilizando o Algoritmo 4 (linha 12). Na linha 13 é proposta a atualização do arquivo externo, utilizando o Algoritmo 5. Na linha 14 são selecionados dentre toda a população os melhores $N * M$ indivíduos para compor a população corrente. Ao final das K iterações o arquivo externo é retornado como aproximação do conjunto Pareto.

Ao analisar o Algoritmo 9, observa-se que a operação de verificação de dominância de solução faz-se necessária em três situações:

1. Classificar a população em frontes não dominados (linhas 3 e 12);

Algoritmo 9: Algoritmo SCE adaptado para o MOKP.

```
1  begin
2      Inicializar população de N * M indivíduos gerados aleatoriamente;
3      Classificar população em frontes não dominados;
4      Selecionar o 1º fronte para compor arquivo externo;
5      for k ← 1 : K do
6          Ordenar população por aptidão (desempate por hipervolume);
7          Distribuir população em M complexos;
8          for i ← 1 : N do
9              for k' ← 1 : K' do
10                 Selecionar subcomplexo com P indivíduos retirados do i-ésimo complexo;
11                 Evoluir pior indivíduo do subcomplexo gerando um novo indivíduo;
12         Classificar toda a população (nova e antiga) em frontes não dominados;
13         Propor atualização do arquivo utilizando as soluções do 1º fronte F_1;
14         Selecionar população;
15     return Arquivo externo;
```

2. Verificar se o indivíduo teve sua aptidão melhorada (linha 11);
3. Propor a atualização do arquivo, dada uma nova solução (linha 13).

Esses serão os pontos do algoritmo nos quais será utilizada a estratégia de indexação multidimensional, a fim de analisar a performance da proposta de aceleração da verificação de dominância no contexto heurístico. Os demais pontos do algoritmo permanecerão inalterados. Os impactos desta aplicação serão examinados através de experimentos computacionais apresentados e discutidos no Capítulo 4.

3 Indexação Multidimensional para Verificação de Dominância

Neste capítulo será apresentada a proposta de aceleração da operação de verificação de dominância através da utilização da árvore k-d como estrutura de indexação multidimensional de soluções. Para isso, as duas variações da operação de verificação de dominância e suas interpretações como problema de busca de faixa serão definidas. A utilização da árvore k-d será também comparada a da lista encadeada e a da árvore AVL.

3.1 A Verificação de Dominância e a Busca de Faixa

Como dito anteriormente, solucionar um problema multiobjetivo significa apresentar o seu conjunto Pareto, ou seja, o conjunto de soluções que não são dominadas por nenhuma outra. Geralmente o conjunto Pareto é construído pelos algoritmos de forma incremental, ou seja, através da adição progressiva de soluções candidatas. Por este motivo, um dos procedimentos mais executados durante o processo de solução é o de verificar se uma determinada solução é dominada por alguma outra pertencente a um conjunto Pareto candidato.

Um algoritmo também pode necessitar reduzir um conjunto de soluções a um conjunto independente, ou seja, conjunto no qual nenhuma das soluções é dominada por alguma outra do mesmo conjunto, o que demanda esforço quadrático sobre o número de soluções, se implementado como uma comparação par a par. Entretanto, se as soluções forem interpretadas como pontos em um espaço multi-dimensional, pode-se deduzir, da Equação 2.1, que este tipo de verificação corresponde ao problema de verificação de existência de pontos em uma determinada região retangular do espaço. Este problema é conhecido por alguns ramos da computação e é denominado *busca de faixa* (*range search* em inglês).

A Figura 7 ilustra o primeiro caso do problema de verificação de dominância: determinar se, dentre um conjunto de soluções, existe alguma que é dominada por uma solução x. O problema é equivalente ao de verificar se existe algum ponto na região sombreada, correspondente à região dominada pela solução x. No caso, existe uma solução que é dominada por x.

A Figura 8 ilustra o segundo caso do problema de verificação de dominância: determinar se, dentre um conjunto de soluções, existe alguma que domina a solução x. O problema é equivalente ao de verificar se existe algum ponto na região sombreada, correspondente à região que domina a solução x. No caso, não existe nenhuma solução.

Para que seja devidamente formalizado o mapeamento entre o problema de verificação de dominância e o problema de busca de faixa, é necessário primeiramente definir duas

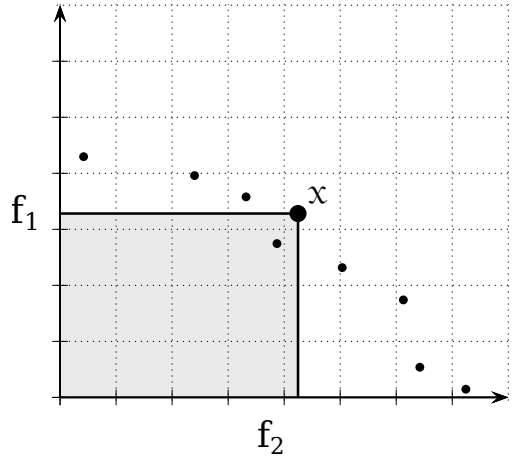


Figura 7 – Verificação da existência de solução dominada por x.

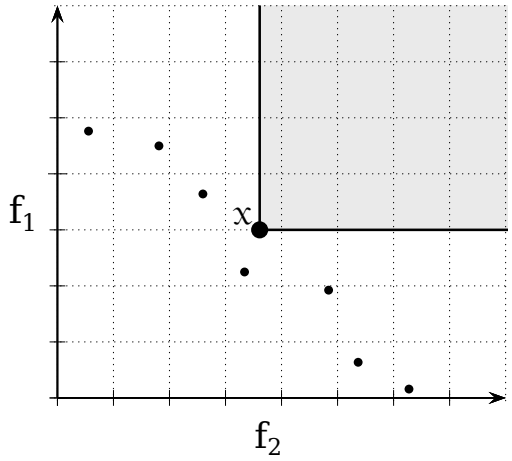


Figura 8 – Verificação da existência de solução que domine x.

regiões do espaço, referentes às áreas de interesse neste tipo de operação. Define-se a área $R_d(x)$ dominada pela solução x e a área $R_{d-}(x)$ a qual domina a solução x sendo respectivamente:

$$R_d(x) = \left\{ y \in \mathbb{R}^m \;\middle|\; y_i \leqslant f_i(x), i \in \{1, \ldots, m\} \right\}$$
$$R_{d-}(x) = \left\{ y \in \mathbb{R}^m \;\middle|\; y_i \geqslant f_i(x), i \in \{1, \ldots, m\} \right\}$$

As Figuras 9 e 10 ilustram as regiões definidas.

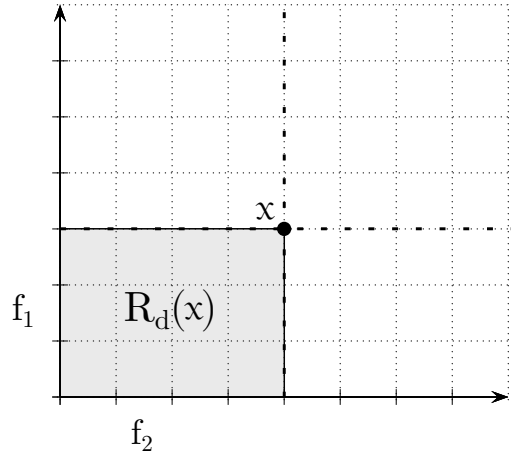


Figura 9 – Região $R_d(x)$ dominada pela solução x.

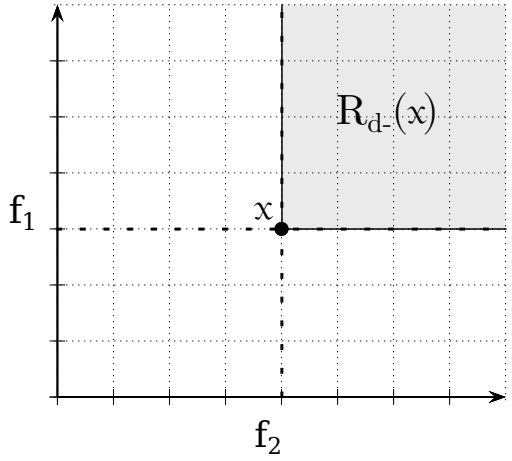


Figura 10 – Região $R_{d-}(x)$ pela qual a solução x é dominada.

Pode-se então definir, por exemplo, para um caso par a par, que o problema de verificar se uma solução x domina uma solução y é equivalente ao problema de verificar se o ponto $f(y)$ está contido na região $R_d(x)$, ou seja:

$$x \Delta y \;\Leftrightarrow\; f(y) \in R_d(x)$$

Para a primeira variação do problema, verificar se, dentre um conjunto Y de soluções, existe alguma solução $y \in Y$ dominada por x (Figura 7) equivale ao problema de determinar se algum ponto está contido na região $R_d(x)$. Formalmente:

$$\exists(y \in Y)[\ x \Delta y\ ] \ \Leftrightarrow\ \exists(y \in Y)[\ f(y) \in R_d(x)\ ]$$

Semelhantemente, o problema de verificar se, dentre um conjunto Y de soluções, existe alguma solução $y \in Y$ que domina uma solução x (Figura 8) equivale ao problema de verificar se existe algum ponto $f(y)$ contido na região $R_{d-}(x)$. Formalmente:

$$\exists(y \in Y)[\ y \Delta x\ ] \ \Leftrightarrow\ \exists(y \in Y)[\ f(y) \in R_{d-}(x)\ ]$$

O problema de se determinar a existência de um ponto numa determinada região do espaço é largamente aplicado, por exemplo, na área da computação gráfica e jogos, onde é necessário verificar a colisão entre pontos e polígonos. Para auxiliar na solução deste problema, uma das estruturas de dados recomendadas é a árvore k-d (AGARWAL; ERICKSON et al., 1999). A proposta do presente trabalho é utilizar a árvore k-d como estrutura auxiliar também nas operações de verificação de dominância, o que deve acelerar o procedimento, uma vez que as soluções estarão indexadas por uma estrutura própria para este tipo de operação.

A seguir são apresentadas três estruturas de dados que podem ser utilizadas para auxiliar à operação de busca de faixa. Duas das estruturas, a lista encadeada e a árvore AVL, já são utilizadas pela literatura no auxílio à verificação de dominância. A terceira estrutura, a árvore k-d, será proposta como estrutura ideal para ser utilizada neste tipo de operação.

3.2 Lista Encadeada

A lista encadeada é uma estrutura de dados na qual cada um dos elementos, dispostos em ordem linear, possuem um ponteiro que referencia um próximo elemento (CORMEN et al., 2009). A lista portanto não possui qualquer ordenação ou indexação baseada nos valores dos elementos. A sequência em que os elementos são dispostos é definida pela ordem de inserção dos elementos na lista. A Figura 11 apresenta uma lista encadeada contendo 7 números inteiros como elementos. O ponto ao final de cada estrutura representa o ponteiro para o próximo nó da lista.

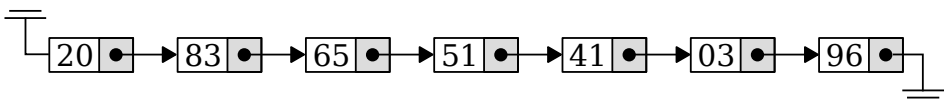


Figura 11 – Exemplo de lista encadeada contendo 7 elementos.

Uma vez que a lista encadeada é "cega" quanto aos valores dos elementos, a única forma de realizar uma operação de busca de faixa é partindo do primeiro da lista e avaliando cada elemento de forma sucessiva, até encontrar algum que esteja na faixa desejada. Caso não exista nenhum elemento na faixa desejada, todos os elementos serão avaliados. O Algoritmo 10 apresenta o algoritmo de busca de faixa utilizando uma lista encadeada.

O Algoritmo 10 inicia considerando o primeiro nó (head) como sendo o nó corrente (node). Enquanto o nó corrente não é *null* (linha 3), ou seja, enquanto não se alcançou o final da lista, o algoritmo recupera o elemento contido no nó (linha 4) e verifica se este é um ponto contido em $R_d(x)$ (linha 5). Em caso verdadeiro, é retornado o elemento (linha 6). Caso contrário, o algoritmo segue para o próximo nó da lista (linha 7) até alcançar o final da lista.

Algoritmo 10: Procedimento busca de faixa utilizando pontos mantidos por uma lista encadeada.

```
   input: x: ponto de interesse, L: lista de pontos
1  begin
2      node ← L.head;
3      while node ≠ null do
4          y ← node.info;
5          if y ∈ R_d(x) then
6              return y;
7          node ← node.next
8      return null;
```

A Figura 12 apresenta pontos dispostos em um plano, mantidos por uma lista encadeada. No plano pode-se notar o retângulo com bordas pontilhadas, representando a região de interesse para a qual se deseja consultar a pertinência de algum ponto. A Figura 13 ilustra a execução da busca de faixa. O fundo cinza sob o ponto sinaliza a avaliação do ponto. No exemplo, apenas o ponto 26 está contido no intervalo de busca. Porém, dos 27 pontos, foi necessário avaliar os 22 pontos que estavam armazenados nas posições anteriores ao ponto 26.

3.3 Árvore AVL

A árvore AVL é um tipo de árvore binária de busca com a propriedade de ter sua altura *balanceada*, ou seja, proporcional ao logaritmo do número total de elementos. Para alcançar esse balanceamento a árvore AVL assegura que, para cada nó da árvore, a diferença de altura entre as sub-árvores da esquerda e da direita seja no máximo 1. Esse balanceamento é independente da ordem em que os elementos são inseridos ou removidos, o que garante eficiência nas operações de busca.

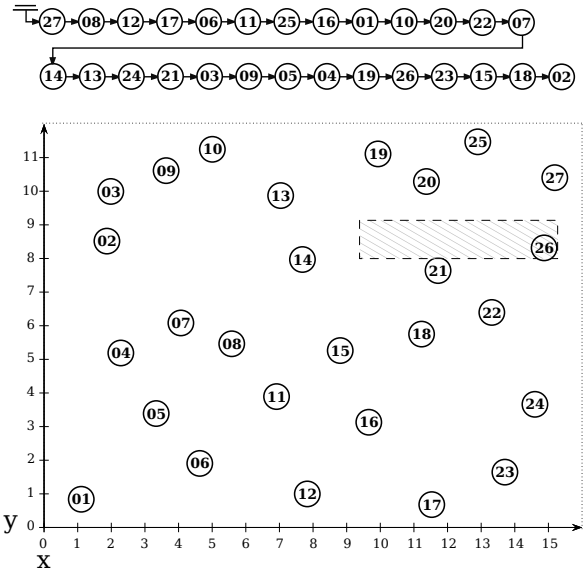


Figura 12 – Pontos no plano mantidos por uma lista encadeada.

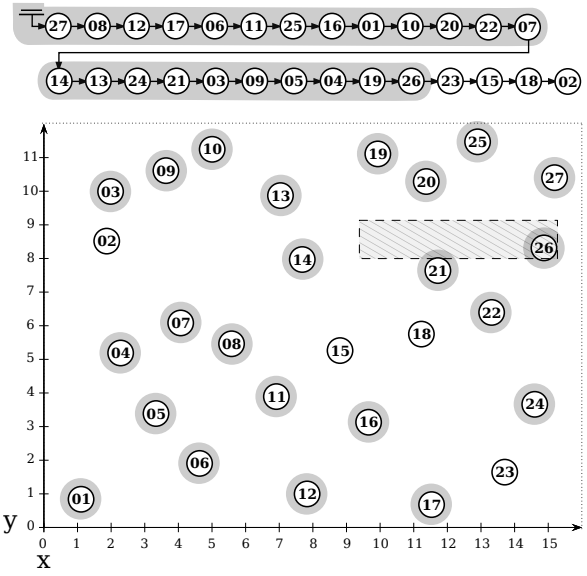


Figura 13 – Execução da operação de busca de faixa em pontos mantidos por uma lista encadeada.

Para que as operações de inserção e remoção de elementos sejam feitas de forma eficiente, é necessário que cada nó mantenha informação sobre o balanceamento entre as suas sub-árvores. Assim, as operações de inserção e remoção de elementos podem ser executadas de forma eficiente, mantendo-se a estrutura balanceada da árvore, realizando, sempre que necessário, operações de reposicionamento dos nós *pais*.

A Figura 14 apresenta uma árvore AVL contendo 7 números inteiros como elementos, tendo o elemento 65 como nó raiz. A parte branca de cada nó representa a informação contida, enquanto que a parte cinza representa dados estruturais da árvore. Cada nó possui espaço para guardar duas referências para nós filhos: um à direita e outro à esquerda. Como qualquer árvore binária de busca, os elementos dispostos à esquerda de um nó são sempre menores que o elemento mantido pelo nó e os elementos dispostos à direita são sempre maiores ou iguais que o elemento mantido pelo nó. O espaço central da região acinzentada contém a informação do balanceamento da árvore: altura da sub-árvore à direita menos a altura da sub-árvore à esquerda, que pode variar de -1 a +1.

Vale observar que a árvore AVL, como qualquer outra árvore binária de busca, mantém uma ordenação crescente dos valores dos elementos, se considerada uma navegação da esquerda para a direita, priorizando os elementos em níveis inferiores às sub-árvores. Tal navegação para a Figura 14 seria 03, 20, 41, 51, 65, 83 e 96. O valor que a árvore considera para a ordenação é chamado de *chave*.

A busca por um elemento em uma árvore AVL é feita da mesma maneira que em uma árvore binária de busca comum. Primeiramente avalia-se o nó raiz. Caso este seja menor que o elemento buscado, prossegue-se avaliando a sub-árvore à esquerda, caso contrário, prossegue-se à sub-árvore à direita. A navegação pelas sub-árvores continua, até que um

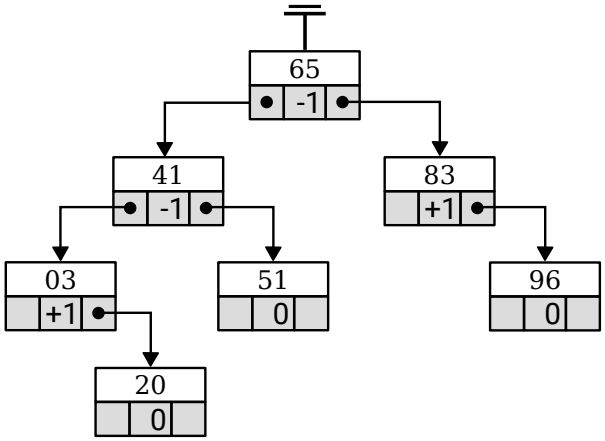


Figura 14 – Exemplo de árvore AVL contendo 7 elementos.

nó com o mesmo valor buscado seja encontrado ou até que se encontre um nó sem a sub-árvore a ser avaliada. Neste último caso, conclui-se que o elemento buscado não existe na árvore.

Para se buscar, por exemplo, o elemento 20 na árvore da Figura 14, inicialmente avalia-se o nó raiz 65. Uma vez que 20 é menor que 65 a busca prossegue para o nó 41 à esquerda. Uma vez que 20 também é menor que 41 a busca prossegue para o nó 03. Como 20 é maior que 03, prossegue-se para a direita, encontrando-se finalmente o elemento 20 buscado. Para se buscar, por exemplo, o elemento 84, serão visitados respectivamente os nós 65, 83 e 96. Visto que 84 é menor que 96 porém o nó 96 não possui sub-árvore à esquerda, a busca se encerra e conclui-se que o elemento 84 não existe na árvore.

Uma vez que a árvore AVL mantém uma ordenação sobre os valores dos elementos, é possível realizar uma busca de faixa com uma maior eficiência que em uma lista encadeada. Porém, convém observar que para se armazenar pontos multidimensionais, a árvore AVL deve considerar apenas uma das dimensões, que será utilizada como chave. As demais dimensões serão desconsideradas pela estrutura. A proposta deste trabalho é que esse fato representa uma potencial deficiência da estrutura quanto à operação de busca de faixas sobre dados multidimensionais, especialmente para uma grande quantidade de dados com mais de duas dimensões, deficiência esta sanada pela árvore k-d.

Para se realizar uma operação de busca de faixa sobre pontos armazenados em uma árvore AVL, o primeiro passo é observar os limites da faixa de busca na dimensão (ou coordenada) utilizada como chave pela árvore, pois a busca se caracteriza por uma varredura de todos os elementos nos limites nesta dimensão. A busca prossegue então localizando o elemento que está mais a esquerda na região de busca, ou seja, que possui o menor valor de chave. A partir desse elemento então é feita uma varredura dos elementos a direita, ou seja, em ordem crescente de valor de chave. Para cada elemento visitado, suas demais coordenadas são verificadas. A busca se encerra ao encontrar algum elemento na região de interesse ou caso a região seja extrapolada pela direita. Neste último caso conclui-se que não existe nenhum elemento na região buscada.

A Figura 15 ilustra uma árvore AVL que considera a componente x com chave e guarda os mesmos 27 pontos das Figuras 12 e 13. Também apresenta a mesma área de interesse (retângulo pontilhado). Vale observar a ordenação que a árvore mantém quanto à distribuição dos pontos ao longo da coordenada x.

A Figura 16 ilustra a operação de busca de faixa utilizando a árvore AVL. As duas setas acinzentadas no plano indicam a região que a árvore AVL deve avaliar para verificar a possível pertinência de um ponto na região de busca (retângulo pontilhado). Os limites na coordenada x da região de busca são $9, 4$ e 15. Vale observar que, apesar da região de busca ser um estreito retângulo horizontal, faz-se necessário verificar os pontos em toda a extensão vertical da região avaliada, pois a árvore não considera o posicionamento vertical dos pontos. O fundo cinza sob o ponto sinaliza a avaliação do ponto. A linha direcionada que percorre a árvore na parte superior da figura indica a sequência em que os nós são avaliados até encontrar o ponto 26. O primeiro passo da execução é encontrar o ponto mais à esquerda da região a ser verificada (ponto 15). Assim que esse ponto é encontrado, inicia-se a varredura da região da esquerda para a direita. Vale observar que durante a varredura, ao se esgotar os nós à esquerda de uma sub-árvore é necessário "voltar" ao nó pai para continuar a varredura na sub-árvore à direita. Após avaliar 14 dos 27 pontos, é encontrado o ponto 26 pertencente à região de busca.

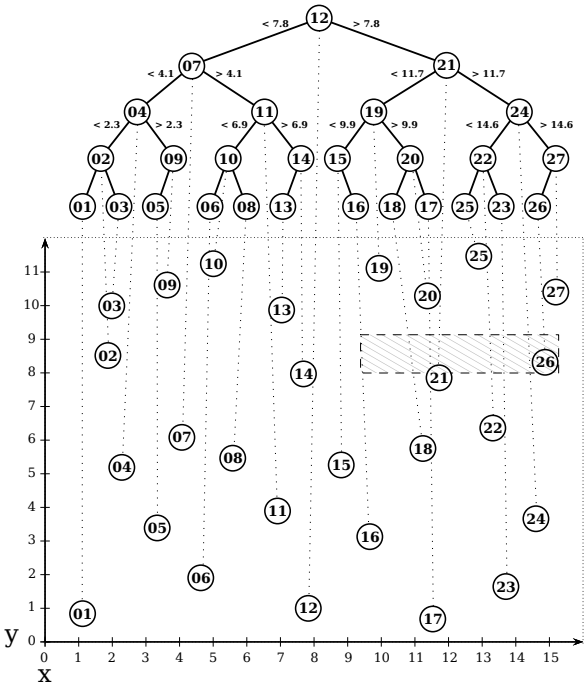


Figura 15 – Pontos no plano mantidos por uma árvore AVL.

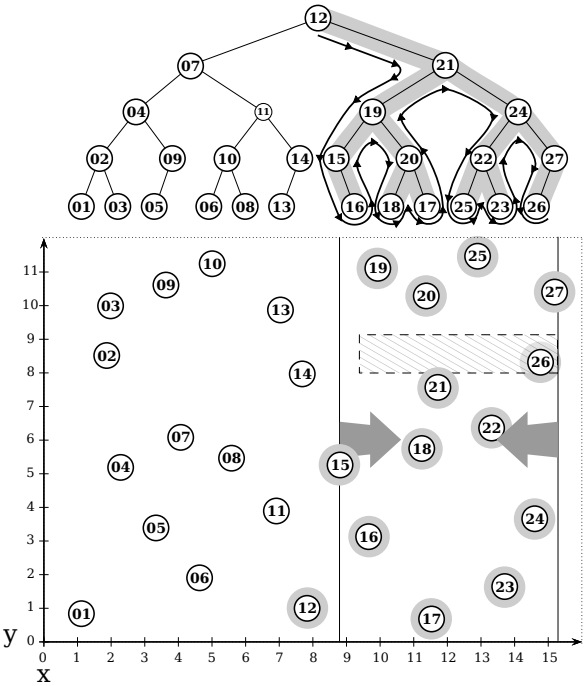


Figura 16 – Execução da operação de busca de faixa em pontos mantidos por uma árvore AVL.

3.4 Árvore k-d

Proposta por Jon Louis Bentley em (BENTLEY, 1975), a árvore k-d é um tipo de árvore binária de construção simples e baixa utilização de memória. Devido a sua simplicidade e eficiência em indexação multidimensional, é uma das estruturas mais utilizadas para operações de verificação de pertinência multidimensional, como é o caso da busca de faixa (PREPARATA; SHAMOS, 2012). Além da operação de busca de faixa, a árvore k-d suporta outras operações como busca de vizinho mais próximo, e por isso é também utilizada em algoritmos de clusterização (KANUNGO et al., 2002; INDYK; MOTWANI, 1998) e renderização gráfica (OWENS et al., 2007).

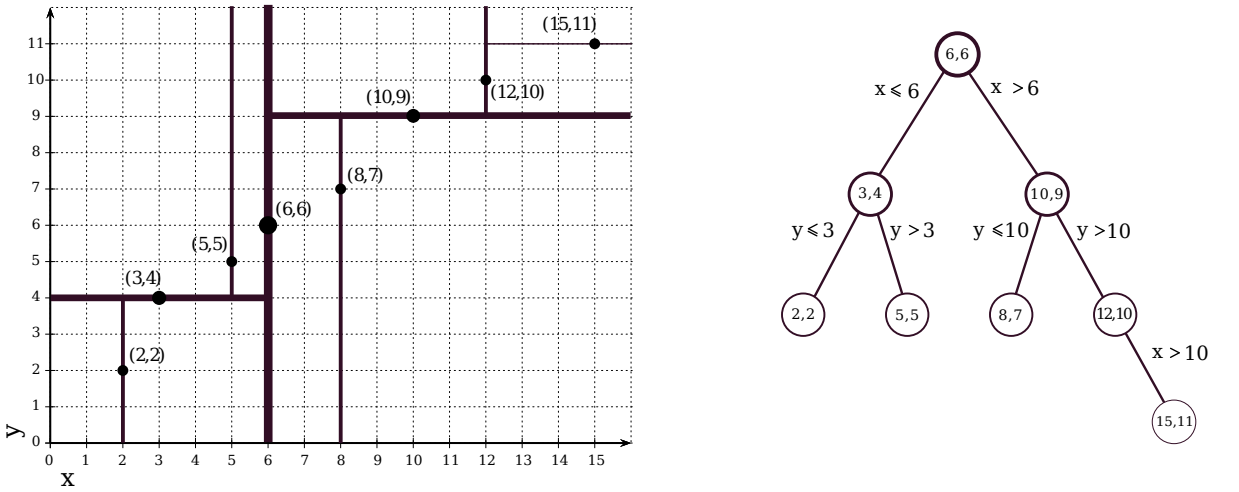
Como qualquer árvore binária, cada nível recursivo da árvore k-d subdivide os dados em duas partes, porém, ao invés de utilizar apenas uma *chave* em todos os níveis da árvore, a árvore k-d utiliza um total de k chaves, fazendo um revezamento circular entre as chaves, à medida que caminha nos níveis da árvore. Dessa forma, os elementos do 1º nível da árvore consideram como chave a 1ª dimensão, o 2º nível da árvore considera a 2ª dimensão, o kº nível considera a kª dimensão, o $k+1$º nível considerada a 1ª dimensão, e assim por diante.

Com relação à eficiência da árvore k-d é importante considerar que não é recomendável escalar de forma arbitrária o número k de dimensões indexadas pela árvore k-d, esperando assim escalar também sua eficiência, mesmo no caso de dados definidos em muitas dimensões. Como regra geral considera-se que uma árvore k-d é adequada para indexar um conjunto com n pontos se n não for muito maior que 2^k (TOTH; O'ROURKE; GOODMAN, 2004), caso contrário, a performance da árvore k-d se assemelhará a de uma busca linear exaustiva, como em uma lista encadeada.

A Figura 17 apresenta um conjunto de pontos dispostos num espaço bi-dimensional (a) e indexados por uma árvore 2-d (b). O primeiro e terceiro nível da árvore 2-d indexa a componente x dos pontos, enquanto o segundo nível indexa a componente y. Cada ponto indexado pela árvore subdivide o espaço em dois de acordo com o valor da componente que está sendo indexada. A subdivisão do espaço é representada na figura por uma linha mais grossa.

A Figura 18 apresenta os mesmos 27 pontos e área de busca (retângulo pontilhado) das Figuras 12, 13, 15 e 16, porém agora os pontos estão mantidos por uma árvore k-d indexando as componentes x e y. Os 1º, 3º e 5º níveis da árvore dividem o plano segundo o valor da componente x enquanto os 2º e 4º níveis dividem o plano segundo o valor da componente y.

A Figura 19 ilustra a operação de busca de faixa utilizando a árvore k-d. As setas acinzentadas no plano indicam a direção da varredura, executada após a avaliação do ponto responsável pela indexação das regiões em questão. Os pontos avaliados na operação estão marcados com um fundo circular acinzentado. As setas junto à árvore na parte



(a) Pontos dispostos num plano bi-dimensional. (b) Pontos indexados por uma 2-d tree.

Figura 17 – Exemplo de pontos indexados por uma árvore k-d.

superior da figura indicam a ordem em que os pontos foram avaliados. Para iniciar a operação de busca de faixa utilizando a árvore k-d é necessário observar os limites tanto no eixo x quanto em y do retângulo de busca. Inicialmente avalia-se a possível pertinência da região de busca em ambos os planos divididos verticalmente pelo ponto 15. Como a região de busca não possui intersecção na região à esquerda do ponto 15 ($x < 8.8$), toda essa região é descartada e o algoritmo segue para a região à direita do plano ($x \geqslant 8.8$), a qual é indexada verticalmente pelo ponto 22. Como a região inferior ao ponto 22 ($y < 6.4$) não possui intersecção com a área de busca, o algoritmo descarta essa região inferior e segue para avaliar a região superior ($y \geqslant 6.4$), a qual é indexada horizontalmente pelo ponto 25. Como ambas as regiões separadas pelo ponto 25 possuem intersecção com a área de busca, o algoritmo deve considerar ambas as regiões, escolhendo uma delas para avaliar primeiro. O algoritmo segue realizando as avaliações de forma sucessiva até encontrar o ponto 26, pertencente à região de busca. Na operação foram avaliados 6 dos 27 pontos. Caso o algoritmo desse prioridade à região a direita do ponto 25, apenas 4 pontos seriam avaliados.

Outra estrutura que suporta a operação de busca de faixa em casos bidimensionais é a quadtree, árvore proposta por Finkel e Bentley em (FINKEL; BENTLEY, 1974) e frequentemente utilizada em processamento de imagens (BERG et al., 1997) e geração de malhas (CHENG; DEY; SHEWCHUK, 2012). A quadtree é uma árvore que divide o espaço em quadrantes. No caso bi-dimensional, por exemplo, cada nó da árvore possui até quatro sub-árvores filhas, representando os 4 quadrantes que fazem interseção com o respectivo ponto. Dessa forma, os nós de uma quadtree que indexa k dimensões possuem até s^k sub-árvores filhas. Segundo a literatura, a utilização da quadtree para indexação de conjunto Paretos mostra-se eficiente apenas em casos bi-objetivos de conjuntos Paretos extensos, acima de 15000 soluções (MOSTAGHIM; TEICH, 2005). Conjectura-se que essa

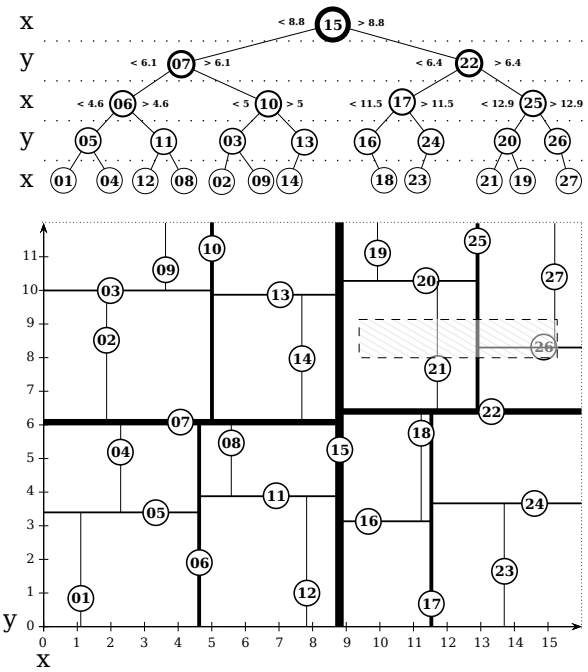


Figura 18 – Pontos no plano mantidos por uma árvore k-d.

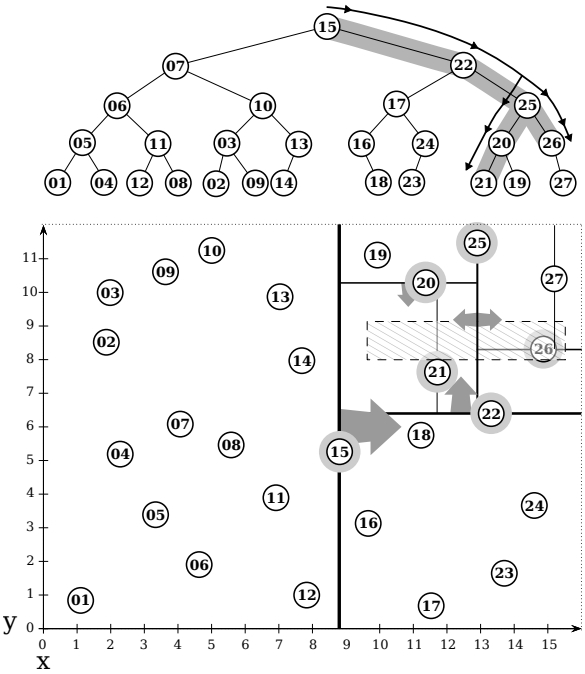


Figura 19 – Execução da operação de busca de faixa em pontos mantidos por uma árvore k-d.

ineficiência é decorrente do alto *overhead* da estrutura, especialmente para os casos com mais de dois objetivos.

Espera-se que a árvore k-d auxilie as operações de verificação de dominância ao descartar uma grande quantidade de soluções, demandando um menor número de comparações entre soluções, melhorando assim a performance dos algoritmos. Infelizmente não se tem uma análise formal de complexidade da busca de faixa utilizando a árvore k-d. Segundo Bentley, a versatilidade da busca de faixa utilizando a árvore k-d torna sua análise formal extremamente difícil (BENTLEY, 1975).

4 Experimentos Computacionais

Experimentos computacionais foram realizados com o objetivo de avaliar a aplicação da proposta de indexação multidimensional ao MOKP, através da observação do número de avaliações de solução e do tempo computacional. Os testes foram divididos em dois grupos visando o contexto exato e o heurístico respectivamente. Em ambos os grupos a performance dos algoritmos utilizando a indexação proposta pela literatura foi comparada à performance utilizando a indexação multidimensional proposta por este trabalho.

Para o primeiro grupo, que visa analisar a aplicação da proposta no contexto exato, foi utilizado o algoritmo Bazgan, por ser considerado pela literatura atual como o algoritmo exato mais eficiente para o MOKP. As instâncias consideradas neste primeiro grupo foram baseadas nas mesmas utilizadas pela literatura em *benchmarks* de algoritmos exatos (BAZGAN; HUGOT; VANDERPOOTEN, 2009; FIGUEIRA et al., 2013; CORREIA; PAQUETE; FIGUEIRA, 2018).

Para o segundo grupo de experimentos, visando analisar a aplicação da proposta no contexto heurístico, foi utilizada a implementação do SCE para o MOKP, proposta neste trabalho. Neste segundo grupo, foram consideradas as instâncias utilizadas pela literatura para *benchmarks* de heurísticas para o MOKP (ZITZLER; THIELE, 1998; ZITZLER; LAUMANNS; THIELE, 2001; DEB et al., 2002; ZHANG; LI, 2007; ZOUACHE; MOUSSAOUI; ABDELAZIZ, 2018).

Ambos os conjuntos são compostos por instâncias de 2 e 3 objetivos. Ambos os algoritmos foram implementados na linguagem C e testados em máquinas Intel® Core™ i5-3570 3.40HGz com 4Gb de RAM com sistema operacional Linux versão 3.19.0 compilados utilizando GCC versão 7.3.1 com parâmetro de otimização -O3.

4.1 Contexto Exato - Algoritmo Bazgan

Para o contexto exato, este trabalho não utilizou exatamente as mesmas instâncias utilizadas pelos autores originais por estas não terem sido divulgadas. As instâncias foram então novamente geradas aleatoriamente seguindo as mesmas regras de distribuição definidas no trabalho original.

As instâncias bi-objetivo são divididas em 4 tipos:

A) Aleatórias: $p_i^j \in [1, 1000], w_i \in [1, 1000]$.

B) Não-conflitantes: $p_i^1 \in [111, 1000]$,
$p_i^2 \in [p_i^1 - 100, p_i^1 + 100]$,
$w_i \in [1, 1000]$.

C) Conflitantes: $p_i^1 \in [1, 1000]$,
$p_i^2 \in [max\{900 - p_i^1; 1\}, min\{1100 - p_i^1, 1000\}]$,
$w_i \in [1, 1000]$.

D) Conflitantes com pesos correlacionados: $p_i^1 \in [1, 1000]$,
$p_i^2 \in [max\{900 - p_i^1; 1\}, min\{1100 - p_i^1, 1000\}]$,
$w_i \in [p_i^1 + p_i^2 - 200, p_i^1 + p_i^2 + 200]$.

onde $\in [a, b]$ denota uma distribuição uniforme aleatória no intervalo $[a, b]$. Para todas as instâncias foi atribuído $W = \frac{1}{2}\left\lfloor \sum_{k=1}^{n} w^k \right\rfloor$. Para cada tipo e valor de n foram geradas dez instâncias. Os valores de n utilizados em cada tipo estão descritos na Tabela 4. O termo "conflitante" refere-se aos valores de p_i^j em cada item: em instâncias conflitantes, itens com altos valores de p^1 tendem a ter baixos valores de p^2 enquanto que itens com baixos valores de p^1 tendem a ter altos valores de p^2. Por sua vez, instâncias não-conflitantes possuem itens cujos valores de p_i^j não possuem relação com o peso w_i,

Devido a este caráter não-conflitante, instâncias do tipo B tendem a possuir itens com valores de eficiência bastante discrepantes entre si. Essa discrepância acaba reduzindo a combinação de itens geradores de soluções eficientes, resultando em conjuntos Pareto de tamanho reduzido. Por este motivo, as instâncias do tipo B são consideradas mais fáceis.

O aspecto conflitante das instâncias do tipo D, por sua vez, dificulta a decisão de quais itens são mais eficientes que outros, resultando em muitas boas combinações de solução e, consequentemente, em conjunto Pareto aumentados. Diante disso, as instâncias do tipo D são consideradas mais difíceis.

Para os experimentos com 3-objetivo considerou-se a generalização introduzida por (BAZGAN; HUGOT; VANDERPOOTEN, 2009) para os tipos A e C e também duas propostas de generalização para os tipo B e D:

A) Aleatórias: $p_i^j \in [1, 1000]$
$w_i \in [1, 1000]$

B) Não-conflitantes: $p_i^1 \in [111, 1000]$,
$p_i^2 \in [p_i^1 - 100, p_i^1 + 100]$,
$p_i^3 \in [p_i^1 - 100, p_i^1 + 100]$,
$w_i \in [1, 1000]$.

C) Conflitantes: $p_i^1 \in [1, 1000]$, $p_i^2 \in [1, 1001 - p_i^1]$
$p_i^3 \in [max\{900 - p_i^1 - p_i^2; 1\}, min\{1100 - p_i^1 - p_i^2, 1001 - p_i^1\}]$
$w_i \in [1, 1000]$.

D) Conflitantes com pesos correlacionados: $p_i^1 \in [1, 1000]$
$p_i^2 \in [1, 1001 - p_i^1]$
$p_i^3 \in [max\{900 - p_i^1 - p_i^2; 1\}, min\{1100 - p_i^1 - p_i^2, 1001 - p_i^1\}]$
$w_i \in [p_i^1 + p_i^2 + p_i^3 - 200, p_i^1 + p_i^2 + p_i^3 + 200]$.

Para todas as instâncias foi atribuído $W = \frac{1}{2}\left\lfloor \sum_{k=1}^{n} w^k \right\rfloor$. A generalização do tipo B foi proposta definindo-se o valor de p_i^3 conforme a mesma distribuição utilizada para definir p_i^2, mantendo assim os valores de p_i^j não-conflitantes. A generalização do tipo D foi proposta aplicando-se para p_i^3 a mesma regra de generalização aplicada no tipo C e, para os pesos, aplicando-se a regra utilizada no caso bi-objetivo para Tipo D, ou seja, w_i tende a ser proporcional à soma dos valores de p_i^j. Para cada tipo e valor de n foram geradas dez instâncias. Os valores de n utilizados em cada tipo estão descritos na Tabela 5. A utilização da árvore k-d foi comparada à utilização da árvore AVL, por ser a estrutura de dados utilizada na proposta original para o algoritmo Bazgan.

A Tabela 4 apresenta a média de tempo de execução em segundos do algoritmo Bazgan para as instâncias bi-objetivo nos casos de utilização da árvore AVL e árvore 2-d. Cada célula refere-se à média das 10 instâncias do respectivo caso. A coluna n apresenta o número de itens na instância enquanto que a coluna $|Par|$ apresenta a média de soluções contidas no conjunto Pareto das respectivas instâncias. A última coluna apresenta o *speedup* da utilização da árvore 2-d em relação a utilização da árvore AVL. As células em destaque possuem os melhores valores de tempo.

Instância			AVL tree	árvore 2-d	
Tipo	n	$\|Par\|$	tempo (s)	tempo (s)	speedup
A	40	38.1	**0.06**	**0.06**	1.0
	60	73.1	1.12	**0.88**	1.3
	80	125.6	19.81	**11.89**	1.7
	100	180.4	165.24	**76.50**	2.2
	120	233.9	708.53	**361.87**	2.0
B	100	3.1	**0.02**	0.08	0.3
	200	10.0	**0.80**	5.09	0.2
	300	24.9	**9.45**	88.30	0.1
	400	36.2	**95.39**	730.04	0.1
	500	53.7	**255.57**	2824.65	0.1
C	20	36.6	**0.00**	**0.00**	1.0
	40	102.8	0.65	**0.42**	1.5
	60	231.9	28.98	**14.09**	2.1
	80	358.0	564.10	**241.54**	2.3
	100	513.8	3756.57	**1605.19**	2.3
D	20	174.9	0.15	**0.12**	1.3
	30	269.3	16.82	**7.60**	2.2
	40	478.0	395.76	**186.67**	2.1
	50	553.4	2459.48	**1417.94**	1.7

Tabela 4 – Tempo computacional médio do algoritmo Bazgan para instâncias bi-objetivo.

Segundo a Tabela 4 observa-se que a árvore 2-d já foi capaz de oferecer redução no tempo computacional, com speedup de até 2.3. Exceto para as instâncias do Tipo B, cujos tempos foram 3 a 10 vezes maior que quando utilizando a árvore AVL. Isto se deve ao menor tamanho dos conjuntos Pareto para este tipo de instância, o que degrada a

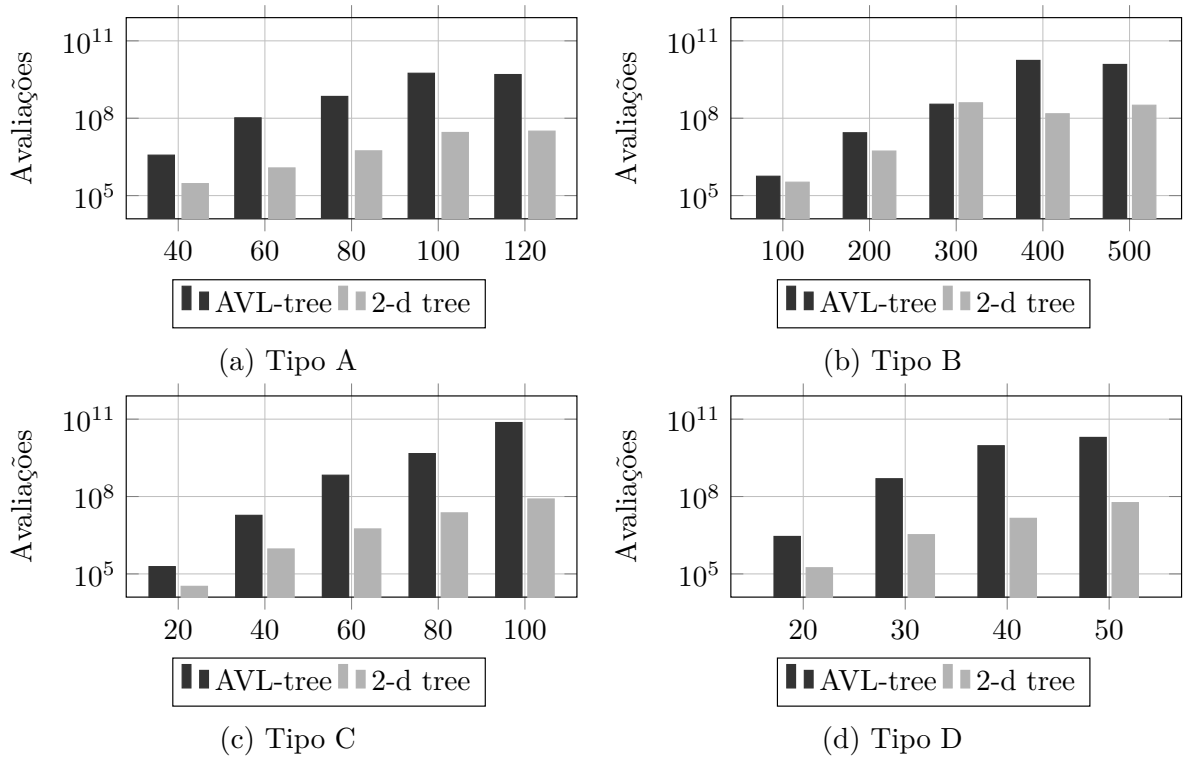


Figura 20 – Número de avaliações médio do algoritmo Bazgan para instâncias bi-objetivo.

performance da árvore 2-d devido ao *overhead* da estrutura. Vale rememorar que, neste algoritmo, o conjunto Pareto é construído de forma incremental durante a execução das iterações, portanto, a quantidade de soluções guardadas durante as primeiras iterações são ainda menores que $|Par|$. Ainda é possível observar na Tabela 4 o reduzido tamanho do conjunto Pareto para instâncias do tipo B, cuja média é de 3.1 para o caso de 100 itens, sendo esta mesma média no mínimo 180.4 para os outros tipos de instâncias.

A Figura 20 apresenta o número médio de avaliações de soluções demandado pelo algoritmo para as instâncias bi-objetivo quando utilizada a árvore AVL (coluna escura) e árvore 2-d (coluna clara). O eixo horizontal corresponde ao número de itens. O eixo vertical representa o número de avaliações em escala logarítmica.

É possível observar na Figura 20 que a utilização da árvore 2-d reduziu consideravelmente o número de avaliações de solução, com exceção apenas no caso com 300 itens do tipo B. Pode-se notar que, nos casos em que se observou altos valores de speedup houve grande redução do número de avaliações; em alguns casos obteve-se uma redução de uma ordem de grandeza. Vale observar que houve alguma redução no número de avaliações para o Tipo B, porém não suficiente para reduzir o tempo computacional. Isto se deve ao reduzido tamanho de seus conjuntos Pareto, para os quais não se torna vantajoso o *overhead* da árvore k-d.

A Tabela 5 apresenta a média de tempo de execução em segundos do algoritmo Bazgan para as instâncias 3-objetivo nos casos de utilização da árvore AVL, árvore 2-d e árvore 3-d. Cada célula se refere à média das 10 instâncias do respectivo caso. A coluna n apresenta o

Instância			AVL tree	árvore 2-d		árvore 3-d	
Tipo	n	$\|Par\|$	tempo (s)	tempo (s)	speedup	tempo (s)	speedup
A	50	557.5	41.2	21.3	1.9	**18.5**	2.2
	60	1240.0	485.9	247.8	1.9	**79.9**	6.0
	70	1879.3	3179.5	1038.0	3.0	**614.5**	5.1
	80	2540.5	6667.9	3796.0	1.7	**2943.9**	2.2
	90	3528.5	24476.5	12916.7	1.8	**3683.7**	6.6
B	100	18.0	**0.1**	0.3	0.3	0.3	0.3
	200	65.4	**11.4**	34.4	0.3	29.1	0.4
	300	214.2	**307.7**	631.5	0.5	583.2	0.5
	400	317.0	**4492.9**	8464.9	0.5	5402.2	0.8
C	20	254.4	0.06	0.05	1.2	**0.03**	2.17
	30	1066.6	9.69	4.18	2.3	**1.30**	7.46
	40	2965.5	471.68	153.21	3.1	**30.50**	15.5
D	20	4087.7	23.6	10.9	2.2	**1.9**	12.5
	30	8834.5	8914.2	3625.3	2.5	**1019.5**	8.7

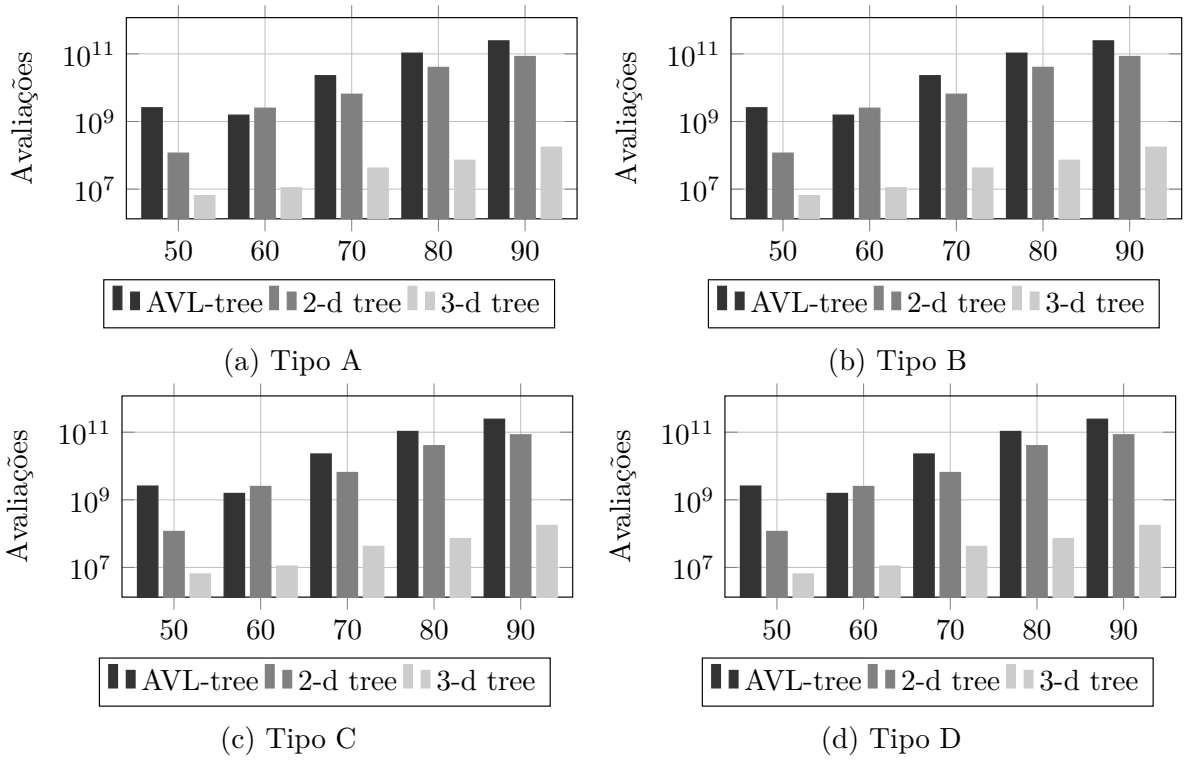
Tabela 5 – Tempo computacional médio do algoritmo Bazgan para instâncias 3-objetivo.

número de itens na instância enquanto que a coluna $|Par|$ apresenta a média de soluções contidas no conjunto Pareto das respectivas instâncias. As células em destaque possuem os melhores valores de tempo.

Conforme a Tabela 5, assim como no caso bi-objetivo, a utilização da árvore k-d resultou em melhoria nos tempos computacionais, exceto para as instâncias do Tipo B. Pode-se observar que o speedup proporcionado pela árvore 2-d para o caso 3-objetivo foi comparável ao caso bi-objetivo. Porém o speedup resultante da utilização da árvore 3-d foi ainda maior, especialmente para instâncias difíceis, chegando a alcançar 15.5.

A Figura 21 apresenta o número médio de avaliações de soluções demandado pelo algoritmo para instâncias 3-objetivo quando utilizada a árvore AVL (coluna direita), árvore 2-d (coluna central) e árvore 3-d (coluna esquerda). O eixo horizontal corresponde ao número de itens. O eixo vertical representa o número de avaliações em escala logarítmica. É possível observar que a utilização da árvore 3-d reduziu consideravelmente o número de avaliações de solução, com exceção de um caso do tipo B. Pode-se notar que, nos casos em que se observou altos valores de speedup houve grande redução do número de avaliações; em alguns casos obteve-se uma redução de uma ordem de grandeza.

É possível notar, segundo a Figura 21, uma leve redução no número de avaliações de solução na maioria dos casos quando utilizada a árvore 2-d e também uma drástica redução no número de avaliações quando utilizada a árvore 3-d, o que explica o speedup alcançado. O resultado se deve à capacidade de indexação da árvore 3-d, a qual indexa informação de todos os 3 valores de objetivo. O speedup foi também possível devido ao tamanho dos conjuntos Pareto, cuja média chegou a alcançar 8834 (Tipo D). Assim como no caso bi-objetivo, a baixa performance da árvore k-d para instâncias do tipo B, tanto no tempo computacional quanto no número de avaliações de solução, se dá pelo reduzido



(a) Tipo A

(b) Tipo B

(c) Tipo C

(d) Tipo D

Figura 21 – Número de avaliações médio do algoritmo Bazgan para instâncias 3-objetivo.

tamanho dos conjuntos Pareto, tornando não proveitosa a utilização da árvore k-d.

Convém observar que o tamanho das instâncias utilizadas nos testes é bem menor que a das instâncias reportadas pelo artigo original. Isto se deu pelo alto tempo computacional demandado pela implementação elaborada para este trabalho, cujo motivo ainda é desconhecido. O fato, porém, não afeta as conclusões quanto à aplicação da proposta, especialmente devido à análise do número de avaliações, sendo esta independente do tempo computacional demandado.

4.2 Contexto Heurístico - Algoritmo SCE

Para o contexto heurístico foram utilizadas o conjunto de instâncias propostas e divulgadas por Zitzler e Thiele (ZITZLER; THIELE, 1998) e utilizadas desde então pela literatura para testar a performance de heurísticas para o MOKPP (ZITZLER; LAUMANNS; THIELE, 2001; DEB et al., 2002; ZHANG; LI, 2007; ZOUACHE; MOUSSAOUI; ABDELAZIZ, 2018). O conjunto é composto por 6 instâncias com valores de n sendo 250, 500 e 750 e m sendo 2 e 3, uma instância para cada caso.

Com o objetivo inicial de verificar a qualidade da heurística SCE proposta para o MOKP, foi realizado um teste inicial, comparando a qualidade das soluções geradas pela heurística com a qualidade das soluções geradas por outros algoritmos da literatura. Os algoritmos utilizados na comparação foram o SPEA-II (ZITZLER; LAUMANNS; THIELE, 2001), NSGA-II (DEB et al., 2002), MOEA/D (ZHANG; LI, 2007) e MOFPA (ZOUACHE;

Parâmetro	Valor	Descrição
N	30	Número de complexos
M	30	Número de indivíduos em cada complexo
P	5	Número de indivíduos em cada subcomplexo
K	400	Número de iterações
K'	30	Número de iterações aplicados a cada evolução de complexo
c	$n/20$	Número de genes carregados no procedimento de cruzamento

Tabela 6 – Valores de parâmetros utilizados no algoritmo SCE.

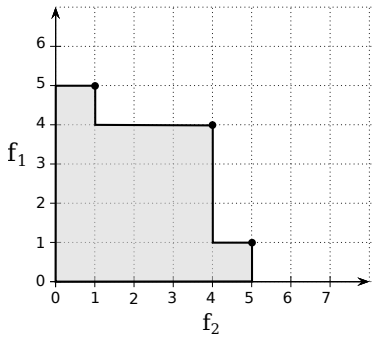


Figura 22 – Exemplo de conjunto Pareto bi-objetivo possuindo 18 unidades hiper-volume (área).

MOUSSAOUI; ABDELAZIZ, 2018). As soluções geradas por estes algoritmo, bem como as instâncias, foram cedidas por Zouache via comunicação particular. Para cada instância, cada um dos algoritmos foi executado 30 vezes, cada uma utilizando uma sequência aleatória diferente, conforme praticado pela literatura.

Os parâmetros utilizados no algoritmo SCE foram os propostos pelo autor da meta-heurística sendo apresentados na Tabela 6. Diferentes combinações de valores de parâmetros próximos aos sugeridos foram testados porém nenhum apresentou melhoria considerável.

A métrica de qualidade utilizada neste caso foi o hiper-volume do conjunto Pareto aproximado (FONSECA; PAQUETE; LÓPEZ-IBÁNEZ, 2006), por ser considerado pela literatura (SCHÜTZE et al., 2016; BEUME; NAUJOKS; EMMERICH, 2007) a métrica mais representativa para a avaliar a qualidade do conjunto Pareto, uma vez que "a velocidade de convergência da taxa de aproximação alcançada ao maximizar o hyper-volume é assintoticamente ótima" conforme apresentado por Bringmann e Friedrich em (BRINGMANN; FRIEDRICH, 2010). A métrica mede o hiper-volume da região contida pelos valores de objetivo das soluções. A Figura 22 apresenta um conjunto Pareto com 3 soluções para um problema bi-objetivo, cujo hiper-volume (unidades de área) é 18. No caso de um problema 3-objetivo, o hiper-volume seria unidades de volume e assim sucessivamente. Outras métricas como distância geracional invertida (VELDHUIZEN; LAMONT, 1998), cobertura de conjunto (ZITZLER; THIELE, 1998) e espaçamento de solução (SCHOTT, 1995) são válidas para evidenciar aspectos específicos do conjunto Pareto, o que não é o objetivo do presente trabalho.

m	n	SPEA2	NSGA-II	MOEA/D	MOFPA	SCE
2	250	90.4	86.3	96.9	**97.8**	93.6
	500	87.6	81.7	96.9	**97.8**	92.7
	750	85.9	79.2	98.4	**99.2**	92.3
3	250	83.3	77.4	99.0	**99.7**	89.4
	500	72.8	65.9	92.9	**93.6**	79.4
	750	77.5	73.3	94.7	**95.2**	79.8

Tabela 7 – Hiper-volume médio alcançado por cada heurística.

Instância			Lista	árvore 2-d		árvore 3-d	
m	n	$\|Par\|$	tempo (s)	tempo (s)	speedup	tempo (s)	speedup
2	250	88.4	**9.3**	13.1	0.71	–	–
	500	106.0	**14.3**	18.3	0.78	–	–
	750	120.4	**18.7**	22.3	0.84	–	–
3	250	705.5	9.8	10.1	0.97	**9.1**	1.08
	500	672.8	15.6	16.0	0.98	**15.2**	1.03
	750	646.0	22.0	24.2	0.91	**21.8**	1.01

Tabela 8 – Tempo computacional médio do algoritmo SCE para instâncias Zouache.

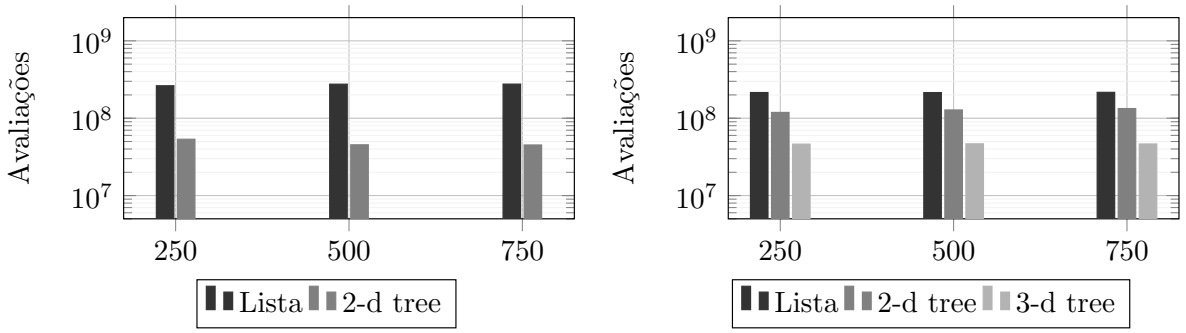
Visando validar a qualidade do algoritmo SCE, a Tabela 7 apresenta os resultados alcançados pelas heurísticas expressos em porcentagem média de hiper volume alcançado, em relação ao maior hiper-volume de solução conhecida para a instância. Cada célula apresenta a média de 30 execuções. Em destaque estão os melhores valores de hiper-volume.

Pode-se observar pela Tabela 7 que, apesar de não apresentar resultados melhores que os do MOEA/D e que os do recente MOFPA, os resultados alcançados pelo SCE foram consistentes, sendo melhores que as heurísticas SPEA2 e NSGA-II.

A utilização da árvore k-d foi comparada à utilização da lista encadeada, por ser a estrutura utilizadas nos algoritmos heurísticos em questão. A Tabela 8 apresenta o tempo computacional médio demandado pela execução do algoritmo SCE para cada uma das 6 instâncias para a utilização da Lista encadeada, árvore 2-d e, no caso 3-objetivo, árvore 3-d. A coluna $|Par|$ apresenta o tamanho médio do conjunto Pareto aproximado dado como resposta pelo algoritmo.

A Figura 23 apresenta o número médio de avaliações de solução para os casos (a) bi-objetivo e (b) 3-objetivo. O eixo horizontal refere-se ao tamanho da instância (número de itens) enquanto que o eixo vertical refere-se ao número médio de avaliações de solução.

Pode-se observar que a árvore 2-d não foi capaz de oferecer melhoria no tempo computacional demandado pelo algoritmo, apesar da redução no número de avaliações de solução. Este resultado se deve ao tamanho reduzido do conjunto Pareto mantido pelo algoritmo, em especial para os casos bi-objetivo. Para os casos 3-objetivo ainda se pode observar um pequeno speedup no caso da árvore 3-d, devido a se ter maiores conjuntos



(a) Instâncias 2-objetivo

(b) Instâncias 3-objetivo.

Figura 23 – Número de avaliações médio do algoritmo SCE para instâncias Zouache.

Pareto, o que não torna favorável o *overhead* adicional da estrutura.

Vale também observar que, para o algoritmo heurístico, a demanda de operações de verificação de dominância é relativamente pequena em relação às outras operações relacionadas ao caráter evolucionário do algoritmo, diferentemente do algoritmo exato, cujo desenvolvimento se baseia em sua maioria na operação de verificação de dominância, o que também explica o insucesso da árvore k-d nesse caso.

5 Conclusão

O presente trabalho propõe a indexação multidimensional das soluções do problema da mochila multiobjetivo, como proposta de aceleração das operações de verificação de dominância de solução. A operação de verificação de dominância é uma das principais operações necessárias para a resolução do problema. A indexação multidimensional das soluções tem por objetivo reduzir o número de avaliações de soluções necessárias para a execução da operação, diminuindo consequentemente o tempo computacional demandado pelo algoritmo.

Há na literatura a proposta de utilização da quadtree como estrutura de dados de indexação de soluções de problemas multiobjetivo. Porém a estratégia mostrou-se eficaz apenas em casos bi-objetivos de conjuntos Paretos consideravelmente extensos. Conjectura-se que essa ineficiência é decorrente do alto *overhead* da estrutura, especialmente em casos com mais de dois objetivos.

Para que a proposta de indexação fosse possível, foi definido no presente trabalho um mapeamento entre a operação de verificação de dominância e o problema da busca de faixa. O problema de busca de faixa consiste em verificar a existência de pontos em uma determinada região do espaço multidimensional. Esse problema é bem conhecido em áreas como computação gráfica, geometria computacional e jogos, onde a árvore k-d é geralmente utilizada como estrutura de dados auxiliar. Sendo assim, a proposta do presente trabalho foi a utilização da árvore k-d como estrutura auxiliar da operação de verificação de dominância.

A aplicação e o comportamento da árvore k-d junto à operação foram discutidos, bem como os da lista encadeada e da árvore AVL, estruturas até então utilizadas pela literatura para este fim. O desempenho da proposta de indexação foi testado no contexto exato e heurístico.

No contexto exato, a proposta foi aplicada ao algoritmo Bazgan, considerado pela literatura como o mais eficiente atualmente para o problema da mochila multiobjetivo. A performance do algoritmo foi comparada através de experimentos computacionais. Para tanto, foi considerada a mesma proposta de instâncias empregada pela literatura, sendo quatro tipos de instâncias bi-objetivo e dois tipos de instâncias 3-objetivo. Para o caso 3-objetivo foram ainda propostas mais dois tipos de instâncias, sendo estas generalizações de seus respectivos casos bi-objetivos.

Os resultados dos experimentos computacionais utilizando o algoritmo Bazgan, mostraram que a proposta de indexação multidimensional possibilitou o speedup de até 2.3 para os casos bi-objetivo, e até 15.5 para casos 3-objetivo. A proposta não se mostrou eficiente apenas em instâncias consideradas fáceis, cujos conjuntos Pareto têm tamanhos consideravelmente reduzidos em relação às demais instâncias. Nesses casos, onde se dá

a manipulação de poucas soluções, o *overhead* computacional da árvore k-d torna sua utilização ineficaz.

No contexto heurístico, a indexação multidimensional foi aplicada ao algoritmo SCE para o MOKP, proposto também neste trabalho. Primeiramente o SCE foi adaptado para resolver problemas multiobjetivo. Para isso, a aptidão das soluções foi definida com base na ordenação em frontes não dominados. A aproximação do conjunto Pareto foi gerada com o auxílio de um arquivo externo. Em seguida, foi estabelecida a implementação específica para o MOKP, definindo-se a operação de geração de solução viável aleatória e a operação de cruzamento de soluções.

A validade da implementação do algoritmo SCE para o MOKP foi primeiramente avaliada e comparada com os principais algoritmos da literatura, não tendo melhores resultados que as heurísticas de estado-da-arte, porém tendo resultados superiores aos das heurísticas mais antigas.

Os experimentos computacionais no contexto heurístico foram realizados sobre o conjunto de 6 instâncias, utilizado pela literatura para avaliar a performance de heurísticas para o MOKP. Apesar de reduzir consideravelmente o número de avaliações de soluções, a utilização da árvore k-d não apresentou eficiência relevante quanto ao tempo computacional na abordagem heurística, tendo pior performance nas instâncias com conjunto Pareto relativamente pequenos.

Pode-se concluir que a utilização da árvore k-d é capaz de reduzir consideravelmente o número de avaliações de soluções na maioria dos casos, além de também reduzir o tempo computacional demandado em casos em que é necessário executar a verificação de dominância em grandes conjuntos de solução. Os resultados evidenciam ser indispensável a utilização de uma estrutura de indexação multidimensional, nos casos de problemas com mais de 2 objetivos com grandes conjuntos de solução. Vale ressaltar que os algoritmos não foram alterados, somente a forma de indexação das soluções.

Notou-se porém, que em casos onde o número de soluções manipuladas é pequeno, a utilização da árvore k-d não é recomendável, pois apesar de apresentar redução no número de avaliações, o *overhead* da estrutura dificulta uma redução no tempo computacional demandado.

Como trabalhos futuros pretende-se verificar a performance da árvore k-d em outros problemas multiobjetivos, bem como comparar com outras estruturas, inclusive com a ND-Tree, proposta em (JASZKIEWICZ; LUST, 2017) como estrutura de dados para auxiliar a operação de verificação de dominância. Pretende-se também investigar a aplicabilidade de outras estruturas de indexação multidimensional ou mesmo a proposta de uma específica para conjuntos Pareto de problemas multiobjetivos.

Pretende-se também aprimorar a implementação do SCE para o MOKP. Uma possibilidade seria o aperfeiçoamento das soluções através da utilização de conhecimentos específicos do problema. Outra possibilidade seria realizando um pré-processamento sobre

o problema inicial, transformando-o em um problema mais simples ou dividindo-o em sub-problemas menores.

Referências

ABDELAZIZ, F. B.; KRICHEN, S.; CHAOUACHI, J. A hybrid heuristic for multiobjective knapsack problems. In: *Meta-heuristics*. [S.l.]: Springer, 1999. p. 205–212.

AGARWAL, P. K.; ERICKSON, J. et al. Geometric range searching and its relatives. *Contemporary Mathematics*, Providence, RI: American Mathematical Society, v. 223, p. 1–56, 1999.

AHO, A. V.; HOPCROFT, J. E. *The design and analysis of computer algorithms*. [S.l.]: Pearson Education India, 1974.

AUGER, A. et al. Hypervolume-based multiobjective optimization: Theoretical foundations and practical implications. *Theoretical Computer Science*, Elsevier, v. 425, p. 75–103, 2012.

BARONI, M. D. V.; VAREJÃO, F. M. A shuffled complex evolution algorithm for the multidimensional knapsack problem. In: *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*. [S.l.]: Springer, 2015. p. 768–775.

BARONI, M. D. V.; VAREJÃO, F. M. A shuffled complex evolution algorithm for the multidimensional knapsack problem using core concept. In: IEEE. *Evolutionary Computation (CEC), 2016 IEEE Congress on*. [S.l.], 2016. p. 2718–2723.

BAZGAN, C.; HUGOT, H.; VANDERPOOTEN, D. Solving efficiently the 0–1 multi-objective knapsack problem. *Computers & Operations Research*, Elsevier, v. 36, n. 1, p. 260–279, 2009.

BENTLEY, J. L. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, ACM, v. 18, n. 9, p. 509–517, 1975.

BERG, M. D. et al. Computational geometry. In: *Computational geometry*. [S.l.]: Springer, 1997. p. 1–17.

BEUME, N.; NAUJOKS, B.; EMMERICH, M. Sms-emoa: Multiobjective selection based on dominated hypervolume. *European Journal of Operational Research*, Elsevier, v. 181, n. 3, p. 1653–1669, 2007.

BHATTACHARJEE, K. K.; SARMAH, S. P. Shuffled frog leaping algorithm and its application to 0/1 knapsack problem. *Applied Soft Computing*, Elsevier, v. 19, p. 252–263, 2014.

BRINGMANN, K.; FRIEDRICH, T. The maximum hypervolume set yields near-optimal approximation. In: ACM. *Proceedings of the 12th annual conference on Genetic and evolutionary computation*. [S.l.], 2010. p. 511–518.

CAPTIVO, M. E. et al. Solving bicriteria 0–1 knapsack problems using a labeling algorithm. *Computers & Operations Research*, Elsevier, v. 30, n. 12, p. 1865–1886, 2003.

CHENG, S.-W.; DEY, T. K.; SHEWCHUK, J. *Delaunay mesh generation*. [S.l.]: CRC Press, 2012.

CORMEN, T. H. et al. *Introduction to algorithms*. [S.l.]: MIT press, 2009.

CORREIA, P.; PAQUETE, L.; FIGUEIRA, J. R. Compressed data structures for bi-objective 0,1-knapsack problems. *Computers & Operations Research*, v. 89, n. Supplement C, p. 82 – 93, 2018. ISSN 0305-0548. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0305054817302150>.

CZYZŻAK, P.; JASZKIEWICZ, A. Pareto simulated annealing—a metaheuristic technique for multiple-objective combinatorial optimization. *Journal of Multi-Criteria Decision Analysis*, Wiley Online Library, v. 7, n. 1, p. 34–47, 1998.

DEB, K. et al. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, IEEE, v. 6, n. 2, p. 182–197, 2002.

DUAN, Q.; SOROOSHIAN, S.; GUPTA, V. Effective and efficient global optimization for conceptual rainfall-runoff models. *Water resources research*, Wiley Online Library, v. 28, n. 4, p. 1015–1031, 1992.

EHRGOTT, M. *Multicriteria optimization*. [S.l.]: Springer Science & Business Media, 2013. v. 491.

EHRGOTT, M.; GANDIBLEUX, X. Bound sets for biobjective combinatorial optimization problems. *Computers & Operations Research*, Elsevier, v. 34, n. 9, p. 2674–2694, 2007.

ELBELTAGI, E.; HEGAZY, T.; GRIERSON, D. A modified shuffled frog-leaping optimization algorithm: applications to project management. *Structure and Infrastructure Engineering*, Taylor & Francis, v. 3, n. 1, p. 53–60, 2007.

FIGUEIRA, J. R. et al. Algorithmic improvements on dynamic programming for the bi-objective {0, 1} knapsack problem. *Computational Optimization and Applications*, Springer, v. 56, n. 1, p. 97–111, 2013.

FINKEL, R. A.; BENTLEY, J. L. Quad trees a data structure for retrieval on composite keys. *Acta informatica*, Springer, v. 4, n. 1, p. 1–9, 1974.

FONSECA, C. M.; PAQUETE, L.; LÓPEZ-IBÁNEZ, M. An improved dimension-sweep algorithm for the hypervolume indicator. In: IEEE. *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*. [S.l.], 2006. p. 1157–1163.

GANDIBLEUX, X.; FREVILLE, A. Tabu search based procedure for solving the 0-1 multiobjective knapsack problem: the two objectives case. *Journal of Heuristics*, Springer, v. 6, n. 3, p. 361–383, 2000.

GAO, J. et al. A quantum-inspired artificial immune system for the multiobjective 0–1 knapsack problem. *Applied Mathematics and Computation*, Elsevier, v. 230, p. 120–137, 2014.

GAREY, M. R.; JOHNSON, D. S. *Computers and intractability*. [S.l.]: wh freeman New York, 2002. v. 29.

INDYK, P.; MOTWANI, R. Approximate nearest neighbors: towards removing the curse of dimensionality. In: ACM. *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. [S.l.], 1998. p. 604–613.

ISHIBUCHI, H.; AKEDO, N.; NOJIMA, Y. Behavior of multiobjective evolutionary algorithms on many-objective knapsack problems. *IEEE Transactions on Evolutionary Computation*, IEEE, v. 19, n. 2, p. 264–283, 2015.

ISHIBUCHI, H.; TSUKAMOTO, N.; NOJIMA, Y. Evolutionary many-objective optimization: A short review. In: IEEE. *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*. [S.l.], 2008. p. 2419–2426.

JASZKIEWICZ, A.; LUST, T. Nd-tree-based update: a fast algorithm for the dynamic non-dominance problem. 2017.

JENKINS, L. A bicriteria knapsack program for planning remediation of contaminated lightstation sites. *European Journal of Operational Research*, Elsevier, v. 140, n. 2, p. 427–433, 2002.

KANUNGO, T. et al. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE transactions on pattern analysis and machine intelligence*, IEEE, v. 24, n. 7, p. 881–892, 2002.

KLAMROTH, K.; WIECEK, M. M. Dynamic programming approaches to the multiple criteria knapsack problem. *Naval Research Logistics*, New York: Wiley in cooperation with the Office of Naval Research, 1987-, v. 47, n. 1, p. 57–76, 2000.

KNUTH, D. E. *Fundamental algorithms: the art of computer programming*. [S.l.: s.n.], 1973.

MARTELLO, S.; TOTH, P. *Knapsack Problems, J.* [S.l.]: Wiley & Sons, Chichester, 1990.

MARTINS, E. d. Q. V.; SANTOS, J. The labeling algorithm for the multiobjective shortest path problem. *Departamento de Matematica, Universidade de Coimbra, Portugal, Tech. Rep. TR-99/005*, 1999.

MARTINS, M. S. et al. Hybrid multi-objective bayesian estimation of distribution algorithm: a comparative analysis for the multi-objective knapsack problem. *Journal of Heuristics*, Springer, p. 1–23, 2017.

MITCHELL, J. E. Branch-and-cut algorithms for combinatorial optimization problems. *Handbook of applied optimization*, p. 65–77, 2002.

MOSTAGHIM, S.; TEICH, J. Quad-trees: A data structure for storing pareto sets in multiobjective evolutionary algorithms with elitism. In: *Evolutionary multiobjective optimization*. [S.l.]: Springer, 2005. p. 81–104.

NEMHAUSER, G. L.; ULLMANN, Z. Discrete dynamic programming and capital allocation. *Management Science*, INFORMS, v. 15, n. 9, p. 494–505, 1969.

OWENS, J. D. et al. A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum*, Blackwell Publishing Ltd, v. 26, n. 1, p. 80–113, 2007. ISSN 1467-8659. Disponível em: <http://dx.doi.org/10.1111/j.1467-8659.2007.01012.x>.

PADBERG, M.; RINALDI, G. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review*, SIAM, v. 33, n. 1, p. 60–100, 1991.

PREPARATA, F. P.; SHAMOS, M. *Computational geometry: an introduction*. [S.l.]: Springer Science & Business Media, 2012.

PUCHINGER, J.; RAIDL, G. R.; PFERSCHY, U. The multidimensional knapsack problem: Structure and algorithms. *INFORMS Journal on Computing*, INFORMS, v. 22, n. 2, p. 250–265, 2010.

ROSENBLATT, M. J.; SINUANY-STERN, Z. Generating the discrete efficient frontier to the capital budgeting problem. *Operations Research*, INFORMS, v. 37, n. 3, p. 384–394, 1989.

SCHOTT, J. R. *Fault Tolerant Design Using Single and Multicriteria Genetic Algorithm Optimization*. [S.l.], 1995.

SCHÜTZE, O. et al. The hypervolume based directed search method for multi-objective optimization problems. *Journal of Heuristics*, Springer, v. 22, n. 3, p. 273–300, 2016.

SEDGEWICK, R. *Algorithms*. [S.l.]: Pearson Education India, 1988.

SILVA, C. G. da; CLÍMACO, J.; FIGUEIRA, J. A scatter search method for bi-criteria {0, 1}-knapsack problems. *European Journal of Operational Research*, Elsevier, v. 169, n. 2, p. 373–391, 2006.

SILVA, C. G. da; FIGUEIRA, J.; CLÍMACO, J. Integrating partial optimization with scatter search for solving bi-criteria {0, 1}-knapsack problems. *European Journal of Operational Research*, Elsevier, v. 177, n. 3, p. 1656–1677, 2007.

TENG, J.-Y.; TZENG, G.-H. A multiobjective programming approach for selecting non-independent transportation investment alternatives. *Transportation Research Part B: Methodological*, Elsevier, v. 30, n. 4, p. 291–307, 1996.

TOTH, C. D.; O'ROURKE, J.; GOODMAN, J. E. *Handbook of discrete and computational geometry*. [S.l.]: CRC press, 2004.

VELDHUIZEN, D. A. V.; LAMONT, G. B. *Multiobjective evolutionary algorithm research: A history and analysis*. [S.l.], 1998.

VISÉE, M. et al. Two-phases method and branch and bound procedures to solve the bi–objective knapsack problem. *Journal of Global Optimization*, Springer, v. 12, n. 2, p. 139–155, 1998.

WEINGARTNER, H. M.; NESS, D. N. Methods for the solution of the multidimensional 0/1 knapsack problem. *Operations Research*, INFORMS, v. 15, n. 1, p. 83–103, 1967.

ZHANG, Q.; LI, H. Moea/d: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on evolutionary computation*, IEEE, v. 11, n. 6, p. 712–731, 2007.

ZHAO, F. et al. A shuffled complex evolution algorithm with opposition-based learning for a permutation flow shop scheduling problem. *International Journal of Computer Integrated Manufacturing*, Taylor & Francis, v. 28, n. 11, p. 1220–1235, 2015.

ZITZLER, E.; LAUMANNS, M.; THIELE, L. Spea2: Improving the strength pareto evolutionary algorithm. 2001.

ZITZLER, E.; THIELE, L. Multiobjective optimization using evolutionary algorithms—a comparative case study. In: SPRINGER. *international conference on parallel problem solving from nature*. [S.l.], 1998. p. 292–301.

ZOUACHE, D.; MOUSSAOUI, A.; ABDELAZIZ, F. B. A cooperative swarm intelligence algorithm for multi-objective discrete optimization with application to the knapsack problem. *European Journal of Operational Research*, Elsevier, v. 264, n. 1, p. 74–88, 2018.