unesp
UNIVERSIDADE ESTADUAL PAULISTA
Instituto de Biociências, Letras e Ciências Exatas
DEPARTAMENTO DE CIÊNCIAS DE COMPUTAÇÃO E ESTATÍSTICA

Uma Abordagem Multiobjetivo para o Problema de Corte de Estoque Unidimensional

André Malvezzi Lopes

Dissertação de Mestrado
Pós-Graduação em Matemática

Rua Cristovão Colombo, 2265
15054-000 - São José do Rio Preto - SP - Brasil
Telefone: (017) 3221-2444
Fax: (017) 3221-2445

Livros Grátis

http://www.livrosgratis.com.br

Milhares de livros grátis para download.

Uma Abordagem Multiobjetivo para o Problema de Corte de Estoque Unidimensional

André Malvezzi Lopes

Dissertação apresentada ao Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Campus de São José do Rio Preto, São Paulo, para a obtenção do título de Mestre em Matemática.

Orientador: Prof. Dr. Silvio Alexandre de Araujo

São José do Rio Preto
Janeiro de 2009

Lopes, André Malvezzi.
Uma abordagem multiobjetivo para o problema de corte de estoque unidimensional / André Malvezzi Lopes. - São José do Rio Preto : [s.n.], 2009.
88 f. : il. ; 30 cm.

Orientador: Silvio Alexandre de Araujo
Dissertação (mestrado) - Universidade Estadual Paulista, Instituto de Biociências, Letras e Ciências Exatas

1. Otimização matemática. 2. Pesquisa operacional. 3. Programação inteira. 4. Otimização multiobjetivo. 5. Algoritmos evolutivos. 6. Problema de corte de estoque. I. Araujo, Silvio Alexandre de. II. Universidade Estadual Paulista, Instituto de Biociências, Letras e Ciências Exatas. III. Título.

CDU – 519.8

Ficha catalográfica elaborada pela Biblioteca do IBILCE
Campus de São José do Rio Preto - UNESP

André Malvezzi Lopes

Uma Abordagem Multiobjetivo para o Problema de Corte de Estoque Unidimensional

Dissertação apresentada para obtenção do título de Mestre em Matemática do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Campus de São José do Rio Preto.

BANCA EXAMINADORA

Orientador
Prof. Dr. Silvio Alexandre Araujo
UNESP - São José do Rio Preto

Primeiro Examinador
Profa. Dra. Helenice de Oliveira Florentino Silva
UNESP - Botucatu

Segundo Examinador
Profa. Dra. Maria do Socorro Nogueira Rangel
UNESP - São José do Rio Preto

São José do Rio Preto, 30 de Janeiro de 2009.

"A mente que se abre a uma nova idéia jamais volta ao seu tamanho original."

Albert Einstein

Aos meus pais, José André e Carmen,
meus avós e meu irmão Lucas,
ofereço.

Ao meu amor, Mirela
dedico.

Agradecimentos

A Deus, por guiar e iluminar meu caminho.

Ao meu orientador Prof. Silvio Alexandre de Araujo, pela sabedoria, compreensão e atenção dedicados a mim.

Aos meus estimados pais José André e Carmen, por todo apoio e sacrifício durante todos estes anos de estudo.

Ao meu irmão Lucas, meus avós Diogo e Conceição, Hermílio e Genoveva, pela compreensão na ausência e pelo apoio incondicional.

A todos os amigos de Campos do Jordão, principalmente Wanderlei, Nádia, Miriana e Milena.

Ao meu amor Mirela, que sempre esteve ao meu lado nos momentos mais difíceis, pelo carinho, paciência, incentivo que foram fundamentais para a realização deste trabalho.

Aos meus amigos, pelo companheirismo em todos os momentos.

A todos os professores e funcionários que, direta ou indiretamente, contribuíram para a concretização desta dissertação.

À CAPES, pelo auxílio financeiro.

Resumo

Este trabalho trata do problema de corte de estoque unidimensional inteiro, que consiste em cortar um conjunto de objetos disponíveis em estoque para a produção de itens menores demandados, de tal forma que se otimize uma ou mais funções objetivos. Foi estudado o caso em que existe apenas um tipo de objeto em estoque em quantidades suficiente para atender a demanda. Três adaptações de um método heurístico baseadas nos conceitos dos algoritmos evolutivos multiobjetivo são propostas para resolver o problema considerando duas funções objetivo conflitantes, a minimização do número de objetos cortados e a minimização do número de diferentes padrões de corte. As adaptações utilizam as idéias presentes no método da Soma Ponderada, no *Vector Evaluated Genetic Algorithm* e no *Multiple Objective Genetic Algorithm*. Estas heurísticas são analisadas resolvendo-se instâncias geradas aleatoriamente.

Palavras-chave: Otimização inteira, problemas de corte de estoque, algoritmos evolutivos, otimização multi-objetivo.

Abstract

This work deals with the one-dimensional integer cutting stock problem, which consist of cutting a set of available objects in stock in order to produce ordered smaller items in such a way as to optimize one or more objective functions. On the case studied there is just one type of object in stock available in sufficient quantity to satisfy the demand. Three adaptations of a heuristic method based on the multi-objective evolutionary algorithms concepts are proposed to solve the problem considering two conflicting objective functions, the minimization of the number of objects to be cut and the minimization of the number of different cutting patterns. The adaptations consider the ideas from the Weighted Sum method, the Vector Evaluated Genetic Algorithm and the Multiple Objective Genetic Algorithm. These heuristics are analyzed by solving randomly generated instances.

Keywords: Integer optimization, cutting stock problem, evolutionary algorithm, multiobjective optimization.

Sumário

Capítulo 1

Introdução

Com a globalização do consumo mundial, as indústrias têm enfrentado um desafio cada vez maior para melhorar sua competitividade e para isso, buscam melhorar seus processos produtivos produzindo mais, com melhor qualidade e menor custo. Um dos fatores predominantes para alcançar tais objetivos é o melhor aproveitamento da matéria prima.

Um problema que aparece em alguns ambientes industriais é o problema de corte de estoque. Foi um dos primeiros a serem estudados em pesquisa operacional e tem grande importância no planejamento da produção de diversas indústrias, tais como as de papel, móvel, vidro, metalúrgica, plástica, têxtil, entre outras. O problema de corte de estoque consiste em cortar os objetos grandes em estoque em itens menores de modo que seja atendida uma demanda. Dependendo dos tipos de itens à serem produzidos estes objetos a serem cortados podem ser de várias dimensões. Ao se cortar os objetos em peças menores pode existir uma perda de material que as indústrias têm um interesse comum em sua redução.

Em algumas indústrias, quando se modifica a forma de cortar os objetos grandes (padrão de corte), é necessário configurar as máquinas para a nova forma de corte (preparação de máquinas). Nestes casos, a redução do número de padrões de corte diferentes também é desejável, pois, um alto número de diferentes padrões de corte, pode aumentar o custo da empresa com relação a preparação das máquinas e pode reduzir o tempo "útil"de produção.

Evitar o desperdício de matéria prima e melhorar o aproveitamento nas preparações das máquinas pode gerar economias significativas, principalmente se a produção é feita em grande escala, e representar uma vantagem decisiva na competição com outras indústrias do setor. Entretanto, os objetivos de minimizar os custos com preparação das máquinas e a perda de matéria prima são conflitantes, ou seja, priorizando um sacrifica-se o outro.

Na prática esse dois objetivos conflitantes são, em geral, considerados empiricamente nas decisões gerenciais. Na literatura de problemas de corte de estoque, embora exista

uma grande quantidade de trabalhos que tratam do problema de minimização da perda, existem poucos trabalhos que consideram simultaneamente a minimização da perda e padrões. Além disso, os trabalhos que tentam minimizar estes dois objetivos, resolvem o problema como sendo mono-objetivo devido a complexidade adicional dos problemas multiobjetivo.

Os problemas de corte de estoque podem ser formulados como problemas de otimização linear inteira e são difíceis de serem resolvidos devido ao grande número de variáveis envolvidas e à restrição de integralidade das variáveis. São problemas pertencentes à classe de problemas NP-difícil e, portanto, obter suas soluções ótimas em tempos computacionais razoáveis não é uma tarefa fácil. Desta forma, torna-se interessante o uso de heurísticas para sua solução.

Neste trabalho, o algoritmo evolutivo desenvolvido em Araujo *et al.* [3], é adaptado para o problema de corte de estoque unidimensional com um tipo de objeto em estoque em quantidade suficiente. Em uma primeira adaptação o problema é tratado como sendo mono-objetivo e em seguida como multiobjetivo. Algumas idéias presentes no método *Vector Evaluated Genetic Algorithm* (VEGA) e também no *Multiple Objective Genetic Algorithm* (MOGA) são utilizadas.

Esta dissertação está organizada da seguinte forma.

No Capítulo 2 são descritos os problemas de corte e empacotamento, as diferentes formulações dos problemas de corte de estoque e alguns trabalhos relevantes presentes na literatura são comentados.

No Capítulo 3 são apresentados os principais conceitos de otimização multiobjetivo que são utilizados no decorrer do trabalho.

No Capítulo 4 são mostrados alguns Algoritmos Evolutivos multiobjetivo presentes na literatura, suas características principais, bem como suas vantagens e desvantagens.

No Capítulo 5 são apresentados os algoritmos desenvolvidos para a resolução do problema, a adaptação do algoritmo desenvolvido por Araujo *et al.* [3] e os algoritmos desenvolvidos com a incorporação das idéias presentes no VEGA e no MOGA, os resultados computacionais obtidos por estes algoritmos são apresentados no Capítulo 6.

No Capítulo 7 encontram-se as conclusões do trabalho e propostas futuras.

Capítulo 2

Problemas de Corte e Empacotamento

Neste capítulo, inicialmente será feita uma classificação dos problemas de corte e empacotamento e, posteriormente tem-se um estudo de problemas de corte de estoque considerando redução de perdas e também redução do número de padrões de corte.

2.1 Introdução

Pequenas melhorias nos processos de corte podem levar a ganhos substanciais e representar uma vantagem decisiva na competição com outras indústrias do setor. Os problemas de empacotamento, em teoria, são similares aos problemas de corte e o objetivo é dispor um conjunto de itens pequenos em um objeto grande, de modo que eles não se sobreponham ou violem as bordas do objeto.

Diferentes tipos de problemas de corte e empacotamento têm sido estudado na literatura, dada a variação dos problemas estudados, uma tipologia ajuda a unificar definições e notações classificando os problemas e facilitando a comunicação entre os pesquisadores. Em 2007, Washer *et al.* [59] aperfeiçoaram a tipologia já existente (Dyckhoff [16]) para que outros problemas de corte de estoque fizessem parte da classificação proposta.

Algumas revisões bibliográfica relacionadas ao problema de corte e empacotamento são Dyckhoff [16], Sweeney e Partenoster [48], Downsland e Downsland [15], Dyckhoff *et al.* [17] e Wäscher *et al.* [59]. Alguns livros específicos ao tema também podem ser citados Martello e Toth [37] e Dyckhoff e Finke [18], assim como edições especiais de jornais, por exemplo, no *European Journal of Operacional Research*, foram publicadas as edições Bischoff e Wäscher [8], Wang e Wäscher [57] e Oliveira e Wäscher [39].

Cinco critérios básicos são utilizados em Wäsher *et al.* [59] para a classificação dos

problemas: dimensionalidade, seleção de itens e objetos, sortimentos de objetos, sortimentos de itens e forma dos itens. Adicionalmente são consideradas algumas variações para tratar casos específicos.

- Na dimensionalidade os problemas se distinguem em unidimendional, bidimensional e tridimensional. Os problemas com dimensão maior que três são considerados por Wäsher *et al.* [59] como uma variante das demais.

  - Problemas Unidimensionais, em que uma das dimensões é relevante para o processo de corte. Por exemplo, no corte de barras de aço.
  - Problemas Bidimensionais, em que duas dimensões são relevantes para a solução do problema. Por exemplo, uma chapa de madeira retangular que deverá ser cortada em peças retangulares menores.
  - Problemas Tridimensionais, em que as três dimensões são relevantes para a solução. Por exemplo, no carregamento de contêineres.
  - N-dimensional com $N > 3$, onde N dimensões são relevantes para a solução. Por exemplo, empacotamento de caixas de comida em fornos para o cozimento (neste caso, o tempo de cozimento representa a quarta dimensão [35]).

  Existem situações em que uma das dimensões relevantes não está fixada *a priori*. Nestes casos, estes problemas podem ser classificados em:

  - Problemas 1.5 dimensionais, em que duas dimensões são relevantes para a solução, porém uma é variável. Por exemplo, no corte de peças de vestuário.
  - Problemas 2.5 dimensionais, onde três dimensões são relevantes sendo uma delas variável. Por exemplo, o problema de se efetuar o carregamento de unidades dentro de caixas abertas, ou seja, as bases estão definidas, mas a altura deverá ser determinada.

- A seleção de itens e objetos são de dois tipos: maximização da produção de item (*output maximization*) em que um conjunto de itens pequenos deve ser designado para um conjunto de objetos grandes, mas este não é suficiente para acomodar todos os itens. Desta maneira todos os objetos serão usados e os itens deverão ser selecionados de forma a obter um conjunto de itens de valor máximo (não há seleção de objetos); o outro tipo é minimização da utilização de objetos (*input minimization*) em que novamente um conjunto de itens deve ser designado para um conjunto de objetos mas diferentemente do tipo anterior, agora os objetos são suficientes para acomodar todos os itens. Todos os itens devem ser designados a um subconjunto de valor mínimo dos objetos utilizados (não há seleção de itens).

- Em relação aos sortimentos de itens são três casos: itens idênticos, sortimentos de itens fracamente heterogêneos e sortimentos de itens fortemente heterogêneos. No primeiro caso todos os itens tem a mesma forma e tamanho. No caso de *"output maximization"* pode-se assumir que o único tipo de item tem uma demanda ilimitada. No caso fracamente heterogêneo os itens podem ser classificados em poucas classes de mesmo tamanho e forma, a demanda para cada tipo de item é relativamente grande e pode ou não ser limitada. Os sortimentos fortemente heterogêneos são caracterizados pelo fato de que poucos itens tem a mesma forma e tamanho, dessa maneira eles são tratados como itens únicos e conseqüentemente têm demanda igual a um. Os problemas com uma variação grande na demanda são tratados como uma variante.

- Os sortimentos de objetos apresentam dois casos: um único tipo de objeto que terá dimensões fixas ou uma ou mais dimensões variáveis e vários tipos de objetos em que diferentes dimensões fixas são consideradas. No caso de vários tipos de objetos, tem-se ainda a classificação em objetos forte e fracamente heterogêneos.

- A forma dos itens é considerada apenas para problemas de mais de uma dimensão e podem variar entre regulares (retângulos, caixas, cilindros, bolas, etc) e irregulares.

Os problemas de corte de estoque segundo Wäsher *et al.* [59] são do tipo de minimização da utilização de objetos, em que itens pequenos fracamente heterogêneos devem ser totalmente designados a um subconjunto de valor mínimo de objetos grandes. Tem-se problemas com um único tipo de objeto ou com vários tipos de objetos.

Um outro aspecto importante presente nos problemas de corte e empacotamento são os padrões de corte.

Definição 2.1.1. *Padrão de corte é a maneira como um objeto em estoque é cortado para a produção de itens demandados. A um padrão de corte associa-se um vetor m-dimensional que contabiliza os itens produzidos*

$$\boldsymbol{a} = (\alpha_1, \alpha_2, \ldots, \alpha_m),$$

em que α_i é a quantidade de itens do tipo i no padrão de corte.

Observe que um vetor $\boldsymbol{a} = (\alpha_1, \alpha_2, \ldots, \alpha_m)$ corresponde a um padrão de corte se e somente se satisfizer as seguintes restrições

$$l_1\alpha_1 + l_2\alpha_2 + \ldots + l_m\alpha_m \leq L \tag{2.1}$$

$$\alpha_1, \alpha_2, \ldots, \alpha_m \geq 0 \text{ e inteiros}, \tag{2.2}$$

em que L é o comprimento do objeto cortado e l_i é o comprimento do item i.

Exemplo 2.1.1. *(Apresentado em Arenales* et al. *[4]) Considere um objeto de comprimento $L = 120$ cm a ser cortado para a produção de 3 tipos de itens de comprimentos $l_1 = 30$ cm, $l_2 = 42$ cm e $l_3 = 45$ cm. Na Figura 2.1 apresentam-se alguns possíveis padrões de cortes (o pedaço rachurado é considerado perda).*

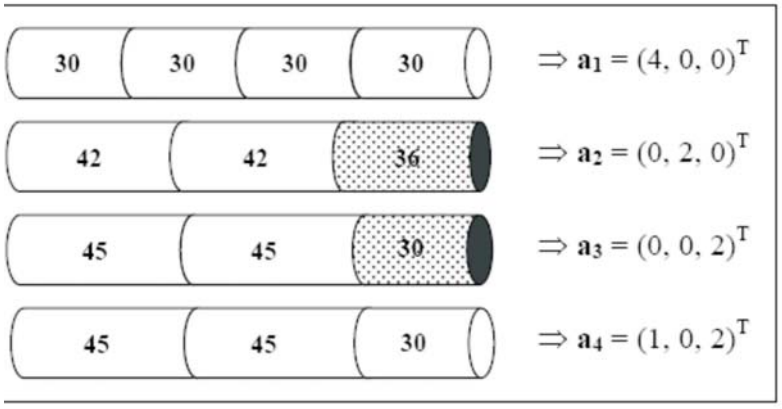


Figura 2.1: Padrões de Corte do Exemplo 2.1.1

Definição 2.1.2. *Um padrão de corte que produza apenas um tipo de item é chamado padrão de corte homogêneo (em outras palavras, um padrão de corte é homogêneo se o vetor associado tem apenas uma coordenada não-nula: $\boldsymbol{a} = (0, \ldots, \alpha_i, \ldots, 0), \alpha_i \neq 0$.).*

Uma solução para o problema de corte de estoque consiste de um conjunto de padrões de corte e das freqüências correspondentes, isto é, o número de vezes que cada padrão tem de ser cortado de forma a atender uma demanda e otimizar uma função objetivo que pode ser, por exemplo, a minimização da perda de material ou a maximização do lucro, ou ainda a minimização de objetivos conflitantes, tais como a perda de material e o número de diferentes padrões de corte (custo de preparação).

2.2 Problemas de Corte de Estoque com um Único Objetivo

A seguir são apresentadas diferentes características dos problemas de corte de estoque unidimensionais com um único objetivo: o problema com um único tipo de objeto em estoque, vários tipos de objeto, objetos limitados, ilimitados, com aproveitamento de sobras e multi-período.

Um Tipo de Objeto em Estoque em Quantidades Ilimitadas

O problema de corte de estoque pode ser formulado como um problema de programação linear inteira. Supondo que hajam m itens, cada padrão j pode ser representado por um vetor coluna a_j, com m elementos, cujo i-ésimo elemento (a_{ij}) indica o número de vezes que o item i ocorre nesse padrão j.

O problema consiste então em considerar todos os padrões viáveis e decidir quantas vezes cada um deles deve ser utilizado de modo a atender a demanda, minimizando, por exemplo, o custo total dos objetos utilizados.

Seja x um vetor decisão cujo elemento x_j é inteiro para $j = 1, \ldots, N$ (onde N é o número de padrões viáveis) e que indica quantas vezes o padrão j é selecionado e, um vetor c cujo elemento c_j é o custo do j-ésimo padrão, com d_i representado a demanda do item i. O problema de corte de estoque pode então ser formulado da seguinte maneira.

$$\text{Minimizar} \quad f(x) = \sum_{j=1}^{N} c_j x_j \tag{2.3}$$

$$\text{sujeito a} \quad \sum_{j=1}^{N} a_{ij} x_j = d_i \quad i = 1, \ldots, m \tag{2.4}$$

$$x_j \geq 0 \text{ e inteiro } j = 1, \ldots, N. \tag{2.5}$$

Alguns trabalhos consideram a restrição de maior ou igual ($\geq$) no lugar da igualdade ($=$) nas restrições (2.4) permitindo assim, a produção excedente de itens. A expressão (2.3) fornece o custo total dos objetos a serem cortados, as restrições (2.4) garantem que toda a demanda seja atendida e as restrições (2.5) garantem que o número de vezes que cada padrão de corte é utilizado é um número inteiro não-negativo.

Alguns trabalhos desenvolveram métodos exatos para este problema, dentre eles Degraeve e Peters [13], Scheithauer e Terno [45] e Vance [51]. Como pode ser observado em (2.5) este é um problema de programação linear inteira. Além disso, existe um grande número de variáveis, devido a grande quantidade de possíveis padrões. Tais características levaram vários autores a desenvolverem diferentes tipos de métodos heurísticos para sua resolução, dentre eles Wäsher e Gau [58], Gilmore e Gomory [23], Stadtler [47], Hinxman [30] e Lopes e Arenales [36].

Adicionalmente, Poldi e Arenales [41] tratam do problema de determinar soluções inteiras considerando o modelo (2.3)-(2.5), dando especial atenção para os problemas com baixa demanda, foram revisados métodos heurísticos conhecidos e algumas variações. Utiliza heurísticas construtivas e também heurísticas residuais.

Uma outra extensão do problema de corte de estoque é proposta por Alem e Arenales [1], onde é considerado apenas um tipo de objeto em estoque em quantidade ilimitada.

Nesta extensão a demanda é aleatória, refletindo diversas possíveis ocorrências. Essas ocorrências são chamadas de cenários e cada um deles possui uma probabilidade associada. Um modelo inicial é proposto, bem como, um algoritmo baseado no método simplex com geração de colunas.

Vários Tipos de Objeto em Estoque em Quantidades Limitadas

Em alguns casos considera-se que existem vários tipos de objetos em estoque em quantidades limitadas. Considere que existam K tipos diferentes de objetos em estoque. O custo de cortar um objeto k segundo um padrão de corte j será c_{jk} para $j = 1, \ldots, N_k$ e $k = 1, \ldots, K$. Considere também L_k como sendo o comprimento de cada objeto de tipo k, o padrão de corte agora será a_{jk} que é o j-ésimo padrão de corte sobre o objeto k, e_k será a disponibilidade do objeto k e, $\mathbf{d}$ o vetor de demandas dos itens. A variável de decisão x_{jk} é o número de vezes que o objeto k é cortado segundo o padrão de corte j. O problema pode ser então formulado da seguinte maneira.

$$\text{Minimizar} \quad f(x) = \sum_{j=1}^{N_1} c_{j1}x_{j1} + \sum_{j=1}^{N_2} c_{j2}x_{j2} + \ldots + \sum_{j=1}^{N_k} c_{jK}x_{jK} \tag{2.6}$$

$$\text{sujeito a} \quad \sum_{j=1}^{N_1} a_{j1}x_{j1} + \sum_{j=1}^{N_2} a_{j2}x_{j2} + \ldots + \sum_{j=1}^{N_k} a_{jK}x_{jK} = d \tag{2.7}$$

$$\begin{aligned} \sum_{j=1}^{N_1} a_{j1}x_{j1} &\leq e_1 \\ \sum_{j=1}^{N_2} a_{j2}x_{j2} &\leq e_2 \\ \ddots \quad &\vdots \\ \sum_{j=1}^{N_k} a_{jK}x_{jK} &\leq e_K \end{aligned} \tag{2.8}$$

$$x_{jk} \geq 0 \text{ e inteiro } j = 1, \ldots, N, k = 1, \ldots, K. \tag{2.9}$$

Na expressão (2.6) tem-se a minimização do custo e considerando as restrições do problema, a restrição (2.7) garante que a quantidade total de itens produzidos seja exatamente igual à demanda. As restrições (2.8) garantem que a quantidade de cada objeto disponível em estoque não seja violada. A última restrição (2.9) garante que a repetição de cada padrão de corte j seja um número inteiro não-negativo.

Alves e Valério de Carvalho [2] utilizam um algoritmo exato para resolver este problema, investigam o uso de inequações duais válidas juntamente com o algoritmo e mostram como o processo de geração de colunas pode ser acelerado em todos os nós da árvore.

Belov e Scheithauer [7] combinam os planos de corte de Chvatal-Gomory e o processo de geração de colunas para tratar deste tipo de problema. Modificações apropriadas são realizadas no processo de geração de colunas e uma heurística de arredondamento é proposta, o procedimento é comparado com o método *branch-and-price* para o problema com um único tipo de objeto em estoque

Poldi e Arenales [40] também estudam este problema mas o foco principal é para os problemas que apresentam baixa demanda, são propostas diferentes heurísticas residuais que são utilizadas para resolver algumas instâncias.

Vários Tipos de Objeto em Estoque em Quantidades Ilimitadas

Considerando os objetos em estoque em quantidades ilimitadas, Gradisar *et al.* [26] propõem um procedimento híbrido para resolução do problema, combinando um método baseado em programação linear e um procedimento seqüencial. Este processo é recomendado quando a troca de um padrão de corte para outro tem um custo baixo. Uma aproximação por decomposição baseada na técnica de geração de colunas para resolver a relaxação do problema e métodos específicos para resolver separadamente os problemas residuais que surgem são propostos por Holthaus [31]. Ao final do método a solução do problema residual é combinada com a obtido pelo arredondamento e um procedimento de melhoria procura por oportunidades de modificar o plano de corte para reduzir o custo total necessário para satisfazer a demanda. Quando todos os objetos em estoque são diferentes Gradisar *et al.* [27] examinam a aplicação da heurística seqüencial, o objetivo é reduzir o custo do material utilizando um plano de corte não muito complexo.

Reaproveitamento de Sobras

Em algumas situações é preferível reaproveitar as sobras de materiais para que sejam usadas posteriormente. Para tratar deste problema, com vários tipos de objetos em estoque e quantidades limitadas objetivando minimizar as perdas, Cherri e Arenales [9] com a finalidade de gerar um conjunto de padrões de corte ideais ou, pelos menos aceitáveis, introduzem algumas alterações em heurísticas clássicas bem conhecidas na literatura tais como a heurística FFD e a gulosa.

Multiperíodos

A demanda pode ocorrer em períodos de um horizonte de planejamento finito, como visto em Poldi e Arenales [42], em que considera-se vários tipos de objeto em estoque,

quantidade limitada com o objetivo de minimizar a soma dos custos com perda de material e estocagem de objetos e itens em todos os períodos.

Considerando T como o número de períodos no horizonte de planejamento, cap_t a capacidade de máquina em cada período, cr_{it} o custo de estocar o item i no período t, cs_{kt} o custo de estocar o objeto k no período t, b_{jkt} o custo de utilizar a máquina para cortar o padrão j do objeto k no período t, r_t o vetor que representa o número de itens de cada tipo antecipados para o período t, s_t o vetor que representa número de objetos de cada tipo que sobram no final do período t, e_t o vetor de disponibilidade de cada tipo de objeto e, x_{jkt} o número de vezes que o objeto k é cortado segundo o padrão de corte j no período t, o problema pode ser modelado da seguinte maneira.

$$\text{Minimizar } f(x) = \sum_{t=1}^{T} \left(\sum_{j=1}^{N_1} c_{j1t}x_{j1t} + \sum_{j=1}^{N_2} c_{j2t}x_{j2t} + \ldots + \sum_{j=1}^{N_k} c_{jKt}x_{jKt} + \sum_{i=1}^{m} cr_{it}r_{it} + \sum_{k=1}^{K} cx_{kt}s_{kt} \right) \tag{2.10}$$

sujeito a

$$\sum_{j=1}^{N_1} a_{j1}x_{j1t} + \sum_{j=1}^{N_2} a_{j2}x_{j2t} + \ldots + \sum_{j=1}^{N_k} a_{jK}x_{jKt} + r_{t-1} - r_t = d, t = 1, \ldots, T \tag{2.11}$$

$$\sum_{k=1}^{K} \sum_{j=1}^{N_k} x_{jkt} - s_{t-1} + s_t = e_t, \qquad t = 1, \ldots, T \tag{2.12}$$

$$\sum_{k=1}^{K} \sum_{j=1}^{N_k} b_{jkt}x_{jkt} \leq cap_t, \qquad t = 1, \ldots, T \tag{2.13}$$

$$x_{jkt} \geq 0 \text{ e inteiro}, r_{it} \geq 0, s_{kt} \geq 0, j = 1, \ldots, N_k, k = 1, \ldots, K, t = 1, \ldots, T. \tag{2.14}$$

Este problema surge como um subproblema quando os problemas de dimensionamento de lotes e de corte são acoplados (Gramani e França [28]), para a resolução deste problema Poldi e Arenales [42] utilizaram o método simplex com geração de colunas para resolver a relaxação linear e então procedimentos heurísticos são utilizados para se obter uma solução para o problema original. Já Trkman e Gradisar [49] consideram este mesmo problema, mas a demanda exata para pedidos futuros não é conhecida, apenas a distribuição de probabilidade é conhecida.

2.3 Problema de Corte de Estoque com Minimização de Perda e Padrões

Dentre os trabalhos que buscam a redução do número de diferentes padrões de corte, Foerster e Wäscher [19] apresentaram uma classe de métodos denominada KOMBI, onde a idéia é fazer combinações de padrões de corte, de forma que a soma das freqüências dos padrões resultantes tem de ser igual a soma das freqüências dos padrões originais a fim de se manter constante a quantidade de objetos utilizados em uma solução. Determinado padrão de corte é substituído por outro equivalente satisfazendo todas as demandas mas com um número menor de padrões de corte distintos.

Yanasse e Limeira [54] apresentam a seguinte extensão do modelo (2.3)-(2.5), onde x_j representa o número de objetos cortados e $\delta(x_j)$ o número de diferentes padrões de corte utilizados.

$$\text{Minimizar} \qquad f(z_1, z_2) = f\left(\sum_{j=1}^{N} x_j, \sum_{j=1}^{N} \delta(x_j)\right) \tag{2.15}$$

$$\text{sujeito a} \qquad \sum_{j=1}^{N} a_{ij} x_j = d_i \quad i = 1, \ldots, m \tag{2.16}$$

$$x_j \geq 0 \text{ e inteiro } j = 1, \ldots, N. \tag{2.17}$$

$$\delta(x_j) \text{ é igual a 1 para } x_j > 0 \text{ e é igual a 0 caso contrário,} \tag{2.18}$$

e propõem um procedimento híbrido para se obter um número reduzido de padrões de corte. Primeiramente são gerados padrões com perda limitada que atendam a demanda em pelo menos 2 itens e este padrão é repetidamente cortado. Em seguida o problema residual é resolvido. Posteriormente, técnicas de redução de padrões de corte são utilizadas. Este processo pode ser utilizado em problemas de qualquer dimensão.

Em Araujo *et al.* [3], uma heurística baseada em algoritmos evolutivos é proposta para o problema de corte de estoque inteiro unidimensional em que há vários tipos de objetos em estoque em quantidade limitada. Neste trabalho também é considerada a minimização da perda de material e do número de padrões de corte usados.

Uma aproximação baseada em metaheurística que incorpora uma técnica de geração de padrões de corte adaptativa é apresentada por Umetani *et al.*[50]. Eles sugerem uma formulação matemática para o problema de corte de estoque unidimensional que considera inicialmente o problema de encontrar um conjunto de padrões $\prod$ ($\prod \subseteq S$, onde S é o conjunto de todos os possíveis padrões) com suas respectivas freqüências $x = (x_1, x_2, \ldots, x_n)$, cujo desvio quadrático da produção com relação à demanda esteja dentro de um limite aceitável. Assim, obtém-se a seguinte formulação.

$$\text{Minimizar} \quad |\prod| \tag{2.19}$$

$$\text{sujeito a} \quad \sum_{i=1}^{m} \left(\sum_{j \in \prod} a_{ij} x_j - d_i \right)^2 \leq D \tag{2.20}$$

$$x_j \in \mathbb{Z}_+, \tag{2.21}$$

onde m é o número de diferentes tipos de itens, $\prod$ é o conjunto dos padrões selecionados, d_i é a demanda associada ao item i, D é o limite tolerável de desvio em relação a demanda, a_{ij} é a freqüência do item do tipo i no padrão j e x_j é a freqüência de corte do padrão j.

O problema descrito anteriormente apresenta dificuldades quanto à sua resolução, pois, o conjunto S de possíveis padrões pode ser muito grande. Para contornar esta dificuldade, Umetani *et al.* [50] propuseram um procedimento que fixa o número N de possíveis padrões distintos para encontrar uma boa solução para o problema. Com isso, o problema é reformulado da seguinte maneira

$$\text{Minimizar} \quad f(\prod, x) = \sum_{i=1}^{m} \left(\sum_{j \in \prod} a_{ij} x_j - d_i \right)^2 \tag{2.22}$$

$$\text{sujeito a} \quad |\prod| = N \tag{2.23}$$

$$x_j \in \mathbb{Z}_+. \tag{2.24}$$

Os autores desenvolvem ainda três algoritmos para reduzir o número de padrões de corte, o primeiro consiste de uma heurística do tipo gulosa, o segundo uma heurística de recombinação de padrões de corte e o terceiro um algoritmo exato.

Um algoritmo exato para reduzir o número de preparações das máquinas é apresentado por Vanderbeck [52], onde o problema é decomposto em múltiplos problemas da mochila limitados. O método *branch-and-bound* com uma técnica de geração de colunas é utilizado para resolver o problema. Em cada nó do método *branch-and-bound* a relaxação linear desta formulação é melhorada adicionando-se inequações. Soluções incumbentes são obtidas usando heurísticas de arredondamento. O procedimento *branch-and-price-and-cut* resultante é usado para produzir soluções ótimas ou aproximadamente ótimas para um conjunto de problemas reais.

O modelo matemático para o problema de redução de padrões em problemas de corte de estoque unidimensional sugerido por Vanderbeck é o seguinte

$$\text{Minimizar} \quad \sum_{k=1}^{K} y_k \tag{2.25}$$

$$\text{sujeito a} \quad \sum_{k=1}^{K} z_k x_{ik} = d_i \qquad i = 1, \ldots, m \tag{2.26}$$

$$\sum_{k=1}^{K} z_k \leq K \tag{2.27}$$

$$z_k \leq K y_k \qquad k = 1, \ldots, K \tag{2.28}$$

$$\sum_{i=1}^{n} l_i x_{ik} \leq L y_k \qquad k = 1, \ldots, K \tag{2.29}$$

$$x_{ik} \in \mathbb{N} \qquad i = 1, \ldots, m \text{ e } k = 1, \ldots, K \tag{2.30}$$

$$y_k \in \{0, 1\} \qquad k = 1, \ldots, K \tag{2.31}$$

$$z_k \in \mathbb{N} \qquad k = 1, \ldots, K, \tag{2.32}$$

onde K é o número máximo de objetos em estoque, m é o número de diferentes tipos de itens, d_i é a demanda associada ao item i, z_k é o número de vezes que o padrão k é usado, y_k é uma variável binária que recebe valor 1 se o padrão k é usado e zero caso contrário, x_{ik} é a freqüência do item tipo i no padrão k, l_i é o comprimento do item i e L é o comprimento do objeto.

A função objetivo (2.25) fornece o número de diferentes padrões de corte que serão usados, as restrições (2.26) garantem que as demandas sejam satisfeitas (restrições não lineares), a restrição (2.27) limita o número de objetos usados, as restrições (2.28) asseguram a própria definição da variável y_k e as restrições (2.29) garantem a viabilidade dos padrões de corte.

Um caso especial para o corte de bobinas foi estudado por McDiarmid [38], onde quaisquer duas bobinas demandadas podem ser cortadas da bobina gigante, mas três não é possível, objetivando minimizar a perda de material e a diminuição dos padrões de corte. Para este caso especial, ele mostrou que o problema de minimização de padrões é NP-difícil e sugeriu métodos heurísticos para resolução, além de reescrever este problema em termos de grafos.

Chu e Antonio [10] tratam de um problema de corte de estoque unidimensional em uma empresa especializada em cortar tubos de metal de várias formas. Nesta empresa só é permitida a produção para atender exatamente a demanda, onde o objetivo não é apenas minimizar a perda mas também minimizar o tempo de corte. Ao tratar desta função multiobjetivo, colocam pesos nos objetivos e somam em uma única função. Os autores desenvolveram dois algoritmos de programação dinâmica, um mais simples e rápido e outro mais elaborado que incorpora aspectos onde é possível planejar a produção de um

dia da indústria. Testes computacionais foram realizados em 16 exemplos fornecidos pela empresa, e o algoritmo mais elaborado melhorou em média 8% os resultados da empresa, chegando em alguns exemplos a 30% de melhora em relação aos cortes feitos manualmente. O tempo computacional não foi muito grande exceto para um exemplo. Os resultados do método foram ainda comparados com a solução ótima de pequenos problemas gerados aleatoriamente e em 8 de 20 exemplos o resultado foi muito próximo dos valores ótimos. Segundo os autores este algoritmo foi implementado na empresa.

Outro trabalho que considera um problema real, é o de Kolen e Spieksma [34], em que uma empresa de médio porte que fabrica abrasivos foi estudada. Estes abrasivos são cortados de bobinas gigantes (jumbo). O trabalho aborda dois tipos de demanda, que são chamadas de demandas exatas quando elas devem ser atendidas exatamente, sem produção extra, e outro tipo que é chamada de demanda aberta onde é permitido um excesso de produção. Existe um limite para o número de itens que podem ser produzidos em cada bobina gigante, que é dado pelo número de facas da máquina. Neste problema também é levado em conta a minimização da perda e a minimização dos diferentes padrões de corte. Os autores optaram por não combinarem estes dois objetivos em uma única função, mas resolver um problema multiobjetivo, isto foi motivado pelo fato de que a fábrica precisava de um conjunto de soluções de alta qualidade, para então uma ser escolhida para a implementação. O conjunto de soluções geradas pelos autores é o conjunto ótimo de Pareto, que é obtido através de um algoritmo *branch-and-bound*, que consiste basicamente de três fases: primeiramente calcula-se o limitante inferior, depois um vetor de freqüências para cada padrão de corte e finalmente para cada elemento do vetor de freqüências é aplicado o *branch-and-bound* para encontrar a perda mínima associada a este vetor. Todas as instâncias testadas (fornecidas pela fábrica) são resolvidas em alguns segundos, isto acontece pelo fato de que elas admitem soluções onde um pequeno número de diferentes padrões de corte são necessários para se obter uma perda mínima, já que todas as instâncias tem um número muito pequeno de diferentes tipos de itens. Quando existem muitas bobinas gigantes e muitos tipos de itens o tempo computacional aumenta muito. Pelos testes realizados, os autores perceberam que com poucos tipos de itens o número de padrões de corte é limitado e o algoritmo leva isto em consideração. De acordo com os autores este algoritmo foi implementado pela empresa e está em uso.

Haessler [29] trabalha com dois objetivos (perda e número de padrões de corte) atribuindo um custo fixo associado a troca de padrões de corte, que tem como objetivo limitar estas trocas. Segundo ele o problema de corte de estoque com um número controlado de padrões pode ser formulado da seguinte maneira,

$$\text{Minimizar} \quad f(x) = c_1 \sum_j t_j x_j + c_2 \sum_j \delta(x_j) \tag{2.33}$$

$$\text{sujeito a} \quad r_l \leq \sum_j a_j x_j \leq r_u \tag{2.34}$$

$$x_j \geq 0 \text{ e inteiros para todo } j, \tag{2.35}$$

onde c_1 é o valor monetário de perda por polegada, c_2 é o custo de mudança de padrões em valores monetários, r_l e r_u são limitantes inferiores e superiores para as demandas dos clientes refletindo uma prática da indústria em geral, t_j é a perda do padrão j, a_{ij} são as componentes do vetor a_j e representam o número de vezes que o item i aparece no padrão j e $\delta(x_j) = 1$ para $x_j > 0$ e 0 caso contrário. Este modelo é conhecido na literatura como um Problema de Carga Fixa.

Note que (2.33) fornece o custo com relação às perdas mais o custo com relação às trocas de padrões. As restrições (2.34) garantem que a produção esteja entre um valor máximo (r_u) e um valor mínimo (r_l) e as restrições (2.35) garantem que o número de vezes que cada padrão de corte é utilizado seja um número inteiro não negativo.

Haessler propôs uma heurística que tenta simultaneamente minimizar o número de padrões de corte e a perda. O algoritmo foi testado em um exemplo de uma indústria de papeis.

Considerando o mesmo problema Walker [56] apresentou um algoritmo baseado em programação linear. Uma solução inicial é encontrada resolvendo a relaxação linear, posteriormente um novo padrão é admitido dentro da base apenas se o custo de preparação de máquina adicional é mais do que compensado pela redução do número de objetos em estoque utilizados. De posse de uma boa solução (pseudo-ótima) a heurística tenta melhorar a solução fazendo troca de padrões de corte.

Capítulo 3

Otimização Multiobjetivo

Na otimização multiobjetivo muitas vezes os objetivos envolvidos no problema são conflitantes, ou seja, cada um possui uma solução ótima diferente, priorizando um, obrigatoriamente o outro é sacrificado. Tem-se então um conjunto de soluções eficientes em que nenhuma solução é a melhor em relação a todos os objetivos. Enquanto no problema de otimização com apenas uma função objetivo, a solução importante é um único ponto ótimo, no multiobjetivo surgem várias soluções eficientes, já que o *trade-off* entre os objetivos conflitantes é importante.

Define-se então, a forma geral de um problema de otimização multiobjetivo (Deb [12]) por:

$$\begin{array}{rll} \text{Minimizar} & f_t(x) & t = 1, 2, \ldots, T \\ \text{sujeito a} & g_k(x) \geq 0 & k = 1, 2, \ldots, K \\ & h_v(x) = 0 & v = 1, 2, \ldots, V \\ & x_i^{(L)} \leq x_i \leq x_i^{(U)} & i = 1, 2, \ldots, n. \end{array} \tag{3.1}$$

O último conjunto de restrições, limita os valores que x_i pode assumir, estes limitantes constituem o espaço de decisão D.

Como pode ser observado, há T funções objetivos $f(x) = (f_1(x), f_2(x), \ldots, f_T(x))^T$, uma das principais diferenças em relação aos problemas com apenas uma função objetivo, é o fato de que, no caso multiobjetivo, as funções objetivo constituem um espaço multidimensional.

3.1 Soluções Ótimas de Pareto

Quando se compara duas soluções e não se pode afirmar qual delas é melhor em relação aos objetivos envolvidos, diz-se que estas soluções são não dominadas (uma em relação

a outra). Como já mencionado anteriormente, quando se tem um conjunto de soluções eficientes para funções objetivo conflitantes, nenhuma delas é melhor em relação a todos os objetivos envolvidos, quando se conhece todas as soluções para um problema e identifica-se as soluções não dominadas em relação a todas as outras, estas soluções são chamadas soluções ótimas de Pareto (Deb [12]), nestas soluções nenhum objetivo pode ser melhorado sem que os demais piorem. A fronteira determinada por estas soluções é chamada fronteira ótima de Pareto, fronteira de Pareto ou frente de Pareto. Veja por exemplo a Figura 3.1.

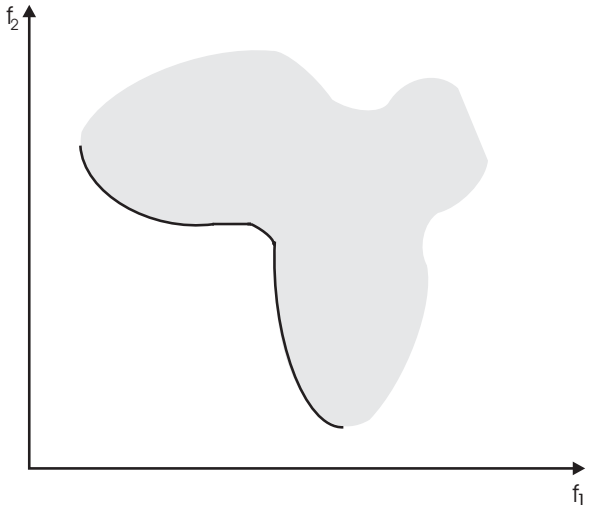


Figura 3.1: Fronteira de Pareto

Quando duas soluções são comparadas e uma delas é melhor que a outra em relação a todos os objetivos envolvidos, dize-se que esta solução domina a outra. Sendo assim existe pelo menos uma solução no conjunto de Pareto que domina qualquer outra solução que não pertence a este conjunto, ou seja, é melhor em relação a todos os objetivos envolvidos.

Pode-se dividir qualquer conjunto finito de soluções factíveis P, nos conjuntos de soluções ótimas de Pareto e não ótimas. Comparando cada solução duas a duas, o conjunto P é dividido em dois conjuntos disjuntos P_1 e P_2, de forma que P_1 contenha todas as soluções não dominadas e P_2 contenha todas as soluções dominadas. O conjunto P_1 é chamado de conjunto não dominado e P_2 de conjunto dominado.

Sendo assim, para que o conjunto P_1 seja não dominado, as seguintes condições devem ser verdadeiras.

- Quaisquer duas soluções de P_1 devem ser não dominadas (uma em relação a outra).
- Qualquer solução que não pertence a P_1 é dominada por pelo menos uma solução de P_1.

Todas as soluções ótimas de Pareto são igualmente importantes e deve-se encontrar o maior número possível destas soluções. Assim pode-se definir duas metas na otimização

multiobjetivo. Encontrar um conjunto de soluções o mais próximo possível da fronteira ótima de Pareto e encontrar um conjunto de soluções não dominadas o mais diverso possível. Desta forma tem-se uma variedade de soluções ótimas com diferentes valores para a função objetivo. Um algoritmo que não encontra um conjunto diverso, é tão bom quanto um algoritmo para uma problema com apenas uma função objetivo. Destas duas metas é que vem a dificuldade adicional da otimização multiobjetivo.

Deve-se ressaltar que existem várias soluções ótimas de Pareto apenas quando as funções objetivos são conflitantes, caso contrário este conjunto terá no máximo um elemento, pois a melhor solução em relação a cada função objetivo é a mesma.

Como no caso mono-objetivo onde tem-se soluções ótimas locais e globais, defini-se também o conjunto ótimo de Pareto local e global.

Definição 3.1.1. *O conjunto não dominado de toda a região factível S é o conjunto ótimo de Pareto.*

Definição 3.1.2. *Se para cada elemento x em P não existe uma solução y (na vizinhança de x, de forma que $||y - x||_\infty \leq \epsilon$, com $\epsilon > 0$) que domina qualquer elemento de P, então os elementos de P constituem o conjunto ótimo de Pareto local.*

3.2 Soluções Especiais

Algumas definições básicas utilizadas em algoritmos da otimização multiobjetivo serão definidas (Deb [12]). A Figura 3.2 ilustra estas soluções especiais para o caso específico de dois objetivos.

Vetor Objetivo Ideal

Para cada parcela de uma função objetivo conflitante, existe uma solução ótima diferente, um vetor construído com o ótimo de cada função é o vetor objetivo ideal.

Definição 3.2.1. *A t-ésima componente do vetor objetivo ideal z^* é a solução para o seguinte problema.*

$$\begin{aligned} &\textit{Minimizar} && f_t(x) \\ &\textit{sujeito a} && x \in S. \end{aligned} \tag{3.2}$$

Sendo assim, se a solução para a t-ésima função objetivo é $x^{*(t)}$ com valor f_t^* para a função objetivo, o vetor ideal é

$$z^* = f^* = (f_1^*, f_2^*, \ldots, f_T^*)^T.$$

Quando se tem objetivos conflitantes este vetor corresponde a um solução inexistente, mas ele é utilizado em muitos algoritmos de otimização multiobjetivo.

Vetor Objetivo Utópico

Alguns algoritmos precisam de uma solução estritamente melhor do que qualquer outra solução (não podendo ser igual), sendo assim, defini-se o vetor objetivo utópico da seguinte maneira.

Definição 3.2.2. *Em um problema de minimização o vetor objetivo utópico z^{**} tem cada uma de suas componentes estritamente menores do que as do vetor ideal, ou $z_t^{**} = z_t^* - \epsilon_t$ com $\epsilon_t > 0$ para todo $t = 1, 2, \ldots, T$.*

Assim como o vetor ideal, este vetor também representa uma solução inexistente.

Vetor Objetivo Nadir

Ao contrário do vetor ideal que para um problema de minimização representa o limitante inferior de cada objetivo, o vetor Nadir z^{nad} representa o limitante superior de cada objetivo no conjunto ótimo de Pareto. Para duas funções objetivo, se $z^{*(1)} = (f_1(x^{*(1)}), f_2(x^{*(1)}))^T$ e $z^{*(2)} = (f_1(x^{*(2)}), f_2(x^{*(2)}))^T$ são coordenadas das soluções mínimas de f_1 e f_2 respectivamente, então o vetor Nadir pode ser estimado como

$$z^{nad} = (f_1(x^{*(2)}), f_2(x^{*(1)}))^T.$$

Este vetor, que é muito difícil de ser calculado, pode representar uma solução existente ou não e pode ser usado para a normalização das funções objetivo da seguinte forma

$$f_t^{norm} = \frac{f_t - z_t^*}{z_t^{nad} - z_t^*}.$$

3.3 Dominância

Nesta seção, define-se formalmente o conceito de dominância e analisa-se suas propriedades (Deb [12]). O operador $\lhd$ será utilizado para denotar qual solução é melhor em relação a um objetivo, por exemplo $i \lhd j$ significa que a solução i é melhor que a solução j em um objetivo específico.

Definição 3.3.1. *Diz-se que uma solução $x^{(1)}$ domina uma solução $x^{(2)}$, se as condições abaixo são verdadeiras.*

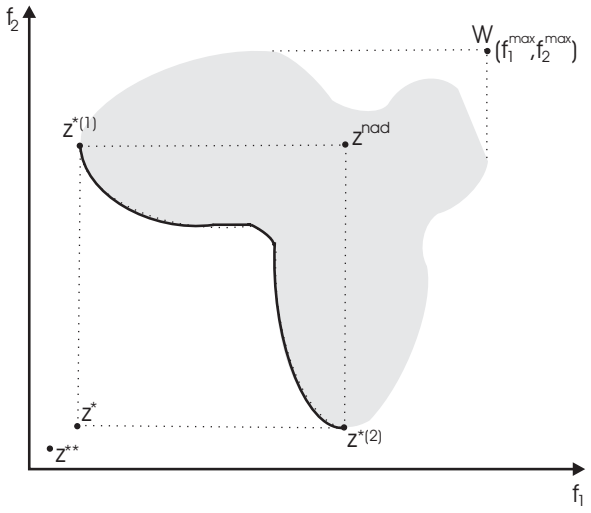


Figura 3.2: Soluções Especiais

i) A solução $x^{(1)}$ *não é pior que a solução* $x^{(2)}$ *em relação a todos os objetivos, ou* $f_t(x^{(1)}) \not\trianglerighteq f_t(x^{(2)})$ *para todo* $t \in \{1, 2, \ldots, T\}$.

ii) A solução $x^{(1)}$ *é estritamente melhor do que* $x^{(2)}$ *em pelo menos um objetivo, ou* $f_t(x^{(1)}) \trianglelefteq f_t(x^{(2)})$ *para pelo menos um* $t \in \{1, 2, \ldots, T\}$.

Se alguma das afirmações acima não for verdadeira a solução $x^{(1)}$ não domina a solução $x^{(2)}$. Se $x^{(1)}$ domina $x^{(2)}$ ($x^{(1)} \preceq x^{(2)}$), valem as seguintes afirmações.

- $x^{(2)}$ é dominada por $x^{(1)}$;
- $x^{(1)}$ não é dominada por $x^{(2)}$, ou;
- $x^{(1)}$ não é inferior a $x^{(2)}$.

Analisando a relação de ordem imposta pela dominância, tem-se que a relação de dominância não é reflexiva, já que qualquer solução p não domina ela mesma pois, (ii) na Definição 3.3.1 não permite que esta propriedade seja satisfeita.

A relação de dominância também é não simétrica, já que $p \preceq q$ não implica que $q \preceq p$, e já que não é simétrica também não é anti-simétrica. Mas se p domina q, então q não domina p, então a relação de dominância é assimétrica.

A relação de dominância é transitiva, já que se $p \preceq q$ e $q \preceq r$, então $p \preceq r$. Outra propriedade importante é que se p não domina q, não significa que q domina p.

A seguir é definido o conceito de dominância forte e fraca.

Definição 3.3.2. *Uma solução* $x^{(1)}$ *domina fortemente* $x^{(2)}$ *(ou* $x^{(1)} \prec x^{(2)}$*), se a solução* $x^{(1)}$ *é estritamente melhor do que* $x^{(2)}$ *em todos os objetivos.*

Definição 3.3.3. *Considerando um conjunto de soluções P, o conjunto de soluções fracamente não dominado P' é aquele onde as soluções não são fortemente dominadas por qualquer outro membro de P.*

Destas duas definições, conclui-se que a cardinalidade do conjunto fracamente não dominado é maior ou igual a cardinalidade do conjunto não dominado.

A seguir apresenta-se um método simples para encontrar o conjunto não dominado de um conjunto de soluções. Neste método, cada solução i é comparada com todas as outras soluções. Se a solução i é dominada por qualquer outra solução, então ela não pertence ao conjunto não dominado. Entretanto, se nenhuma solução domina i, então ela faz parte do conjunto não dominado. Desta forma todas as soluções devem ser comparadas. O procedimento para encontrar o conjunto não dominado de um conjunto P de tamanho N é descrito passo a passo a seguir.

Passo 1 Inicialize o contador $i = 1$ (primeira solução) e crie o conjunto vazio não dominado P'.

Passo 2 Para uma solução $j \in P$ (com $j \neq i$) compare com i, se j domina i então vá para o Passo 4.

Passo 3 Se existem mais soluções em P, incremente j em um e vá para o Passo 2, caso contrário faça $P' = P' \cup \{i\}$.

Passo 4 Incremente i em um. Se $i \leq N$, vá para o Passo 2, caso contrário P' é o conjunto não dominado.

Existem outras maneiras mais eficientes de encontrar este conjunto não dominado que podem ser vistas em Deb [12].

3.4 Classificação de Métodos Multiobjetivo

Na solução de problemas multiobjetivo, dois aspectos importantes podem ser levados em conta, a busca de soluções e a tomada de decisões. O primeiro aspecto refere-se ao processo de otimização na qual a busca é direcionada para as soluções ótimas de Pareto. Como no caso da otimização mono-objetivo, a busca pode tornar-se difícil devido ao tamanho e a complexidade do espaço de busca, podendo inviabilizar o uso de métodos exatos. O segundo aspecto refere-se a tomada de decisões e envolve a seleção de um critério adequado para a escolha de uma solução do conjunto ótimo de Pareto. É necessário que o decisor faça uma ponderação (*trade-off*) dos objetivos conflitantes. A partir do ponto

de vista do decisor, os métodos de otimização multiobjetivo podem ser classificados em três categorias, Métodos *a-priori*, Métodos *a-posteriori*, Métodos iterativos.

Os métodos *a-priori* são caracterizados pela participação do decisor antes do processo de busca de soluções, ou seja, antes de resolver o problema. Arroyo [5] apresenta dois tipos de métodos *a-priori*. No primeiro, os objetivos do problema são combinados em um único objetivo. Isto requer a determinação explícita de pesos para refletir a preferência de cada objetivo. A vantagem deste método é que podem ser aplicadas estratégias clássicas de otimização mono-objetivo sem nenhuma modificação. No segundo, os objetivos são classificados em ordem decrescente de prioridade. Feito isto, o problema é resolvido para o primeiro objetivo sem considerar os demais. A seguir, o problema é resolvido para o segundo objetivo sujeito ao valor ótimo encontrado para o primeiro objetivo. Esse processo é continuado até que o problema seja resolvido para o último objetivo sujeito aos valores ótimos dos outros objetivos.

Nos métodos *a-posteriori*, o processo de decisão é feito logo após a realização da busca de soluções (que é feita através da escolha de algoritmos adequados). A busca é feita, considerando-se que todos os objetivos são de igual importância. Ao final do processo da busca tem-se um conjunto de soluções aproximadas ou ótimas de Pareto. A partir deste conjunto, o responsável pelas decisões deve selecionar uma solução que representa a solução adequada do problema.

Já nos métodos iterativos o responsável pela decisão intervém durante o processo de otimização articulando preferências e guiando a busca para regiões onde existam soluções de interesse.

3.5 Métodos Clássicos

A maior dificuldade em otimização multiobjetivo é a existência de objetivos conflitantes, pois, nenhuma das soluções factíveis otimiza simultaneamente todos os objetivos, o problema nos dá um número de soluções eficientes, conhecidas como soluções ótimas de Pareto. Já que não se pode dizer qual destas soluções é melhor, é importante que se encontre o maior número possível de soluções ótimas de Pareto.

Alguns dos algoritmos convertem os problemas de otimização multiobjetivo em problemas de otimização mono-objetivo, através de procedimentos definidos pelos usuário, Deb [12] chama estes algoritmos de Métodos Clássicos.

A seguir são apresentados alguns métodos clássicos que têm sido aplicados para resolver problemas multiobjetivo. De acordo com a classificação da Seção 3.4, estes métodos são classificados como *a-priori*.

3.5.1 Método da Soma Ponderada

No método da soma ponderada cada objetivo é multiplicado por um peso e então todos são somados em uma única função objetivo. Este método é o mais simples e o método clássico mais utilizado na otimização multiobjetivo.

O peso de cada função objetivo é escolhido de acordo com sua importância. Em geral todos os objetivos são normalizados e uma função $F(x)$ é formada somando todos os objetivos, resultando no seguinte problema de otimização mono-objetivo.

$$\begin{aligned} \text{Minimizar} \quad & F(x) = \sum_{t=1}^{T} w_t f_t(x) & & \\ \text{sujeito a} \quad & g_k(x) \geq 0 & & k = 1, 2, \ldots, K \\ & h_v(x) = 0 & & v = 1, 2, \ldots, V \\ & x_i^{(L)} \leq x_i \leq x_i^{(U)} & & i = 1, 2, \ldots, n, \end{aligned} \tag{3.3}$$

onde $w_t \in [0, 1]$ é o peso da t-ésima função objetivo, já que o problema não se altera se todos os pesos forem multiplicados por uma constante. Normalmente todos os pesos são escolhidos de forma que $\sum_{t=1}^{T} w_t = 1$. Para o problema (3.3) tem-se os seguintes teoremas.

Teorema 3.5.1. *A solução para o problema (3.3) é ótima de Pareto se o peso é positivo para todos os objetivos.*

Este teorema é válido para qualquer problema multiobjetivo de minimização. Entretanto, não implica que qualquer solução ótima de Pareto possa ser obtida usando um vetor peso positivo.

Teorema 3.5.2. *Se x^* é uma solução ótima de Pareto para um problema de otimização multiobjetivo convexo, então existe um vetor positivo diferente de zero w tal que x^* é solução para o problema dado em (3.3).*

Este teorema sugere que para um problema convexo, qualquer solução ótima de Pareto pode ser encontrada por este método. Esta é sua principal vantagem, além de ser intuitivo e fácil de utilizar. Em problemas onde existem objetivos tanto de maximização quanto de minimização, todos os objetivos devem ser convertidos no mesmo tipo.

Diferentes vetores pesos não fornecem necessariamente diferentes soluções ótimas de Pareto. Em alguns problemas, podem existir múltiplas soluções ótimas para um vetor peso específico e cada uma dessas soluções podem representar soluções diferentes no conjunto ótimo de Pareto. Entretanto, quando se tem um espaço não convexo, algumas soluções ótimas de Pareto podem não ser encontradas.

Este método é muito eficiente computacionalmente, e pode ser usado para gerar uma solução fortemente não dominada que pode ser usada como ponto de partida para outros métodos. A dificuldade com este método é determinar apropriadamente os pesos quando não se tem informações suficientes para o problema.

3.5.2 Método do ϵ-restrito

Para contornar as dificuldades observadas no método anterior, em relação à espaços não convexos, tem-se o método ϵ-restrito. Este método mantém apenas uma função objetivo (geralmente a mais importante) e restringe as outras por um valor pré-definido. O problema modificado pode ser dado na seguinte forma.

$$\begin{array}{rll} \text{Minimizar} & f_\mu(x) & \\ \text{sujeito a} & f_t(x) \leq \epsilon_t & t = 1, 2, \ldots, T \text{ e } t \neq \mu \\ & g_k(x) \geq 0 & k = 1, 2, \ldots, K \\ & h_v(x) = 0 & v = 1, 2, \ldots, V \\ & x_i^{(L)} \leq x_i \leq x_i^{(U)} & i = 1, 2, \ldots, n. \end{array} \tag{3.4}$$

O teorema a seguir nos mostra a grande vantagem deste método, tanto para espaços convexos quanto para não convexos.

Teorema 3.5.3. *Se existir uma única solução para o problema (3.4), esta é ótima de Pareto para qualquer vetor* $\epsilon = (\epsilon_1, \ldots, \epsilon_{\mu-1}, \epsilon_{\mu+1}, \ldots, \epsilon_T)$.

Outra vantagem deste método é que diferentes soluções ótimas de Pareto podem ser encontradas usando diferentes valores de ϵ_t, ele geralmente encontra soluções fracamente não dominadas, entretanto se a solução ótima é única, então esta solução é fortemente não dominada. Como desvantagem, a solução para o problema (3.4) depende muito da escolha do vetor ϵ, e cada componente deste vetor deve estar entre o valor mínimo e máximo da função objetivo correspondente. Para se obter valores adequados de ϵ, otimização mono-objetivo pode ser utilizada para cada função objetivo. Quanto maior o número de objetivos, mais elementos terá o vetor ϵ, requerendo mais informações.

3.5.3 Método da Métrica Ponderada

Este método também combina várias funções objetivo em uma única função mono-objetivo. Para pesos não negativos, a distância entre qualquer solução x e o vetor ideal z^* (ponto de referência), utilizando a norma l_p, pode ser minimizada da seguinte maneira.

$$\begin{array}{rll} \text{Minimizar} & l_p(x) = (\sum_{t=1}^{T} w_t |f_t(x) - z_t^*|^p)^{\frac{1}{p}} & \\ \text{sujeito a} & g_k(x) \geq 0 & k = 1, 2, \ldots, K \\ & h_v(x) = 0 & v = 1, 2, \ldots, V \\ & x_i^{(L)} \leq x_i \leq x_i^{(U)} & i = 1, 2, \ldots, n. \end{array} \tag{3.5}$$

O parâmetro p pode assumir qualquer valor entre 1 e ∞. Quando $p = 1$ é usado, o problema (3.5) é equivalente ao problema (3.3). Quando $p = 2$, tem-se a distância Euclidiana. Para um valor suficientemente grande para p, tem-se um problema especial denominado problema ponderado de Tchebycheff (Deb [12]) que utiliza a norma do máximo.

$$\begin{array}{rll} \text{Minimizar} & l_\infty(x) = max_{t=1}^{T} w_t |f_t(x) - z_t^*| & \\ \text{sujeito a} & g_k(x) \geq 0 & k = 1, 2, \ldots, K \\ & h_v(x) = 0 & v = 1, 2, \ldots, V \\ & x_i^{(L)} \leq x_i \leq x_i^{(U)} & i = 1, 2, \ldots, n. \end{array} \tag{3.6}$$

Para este problema, tem-se o seguinte teorema.

Teorema 3.5.4. *Seja x^* uma solução ótima de Pareto. Então existe um vetor peso positivo tal que x^* é solução do problema ponderado de Tchebycheff (3.6), onde o ponto de referência é o vetor objetivo utópico z^{**}.*

A métrica de Tchebycheff garante encontrar todas as soluções ótimas de Pareto quando z^* é o vetor objetivo utópico. Uma desvantagem desse método é que as funções objetivo devem ser normalizadas, sendo assim necessário conhecer os valores máximos e mínimos para cada função. Este método também requer o vetor ideal z^*, então todos os T objetivos devem ser independentemente otimizados antes de resolver o problema multiobjetivo.

3.5.4 Método de Benson

Um outro método similar ao da soma ponderada é o Método de Benson, em que o ponto de referência é escolhida como uma solução factível, não ótima de Pareto. Escolhida uma solução factível z^0, a diferença não negativa $(z_t^0 - f_t(x))$ de cada objetivo é calculada e a soma destas é maximizada.

$$\begin{array}{rll} \text{Maximizar} & \displaystyle\sum_{t=1}^{T} \max(0, (z_t^0 - f_t(x))) & \\ \text{sujeito a} & f_t(x) \leq z_t^0 & t = 1, 2, \ldots, T \\ & g_k(x) \geq 0 & k = 1, 2, \ldots, K \\ & h_v(x) = 0 & v = 1, 2, \ldots, V \\ & x_i^{(L)} \leq x_i \leq x_i^{(U)} & i = 1, 2, \ldots, n. \end{array} \tag{3.7}$$

Com este método é possível obter diferentes soluções ótimas de Pareto ao se atribuir pesos às diferenças. O uso do vetor nadir, z^{nad}, como o ponto escolhido pode ser útil. Se z^0 for escolhido apropriadamente, este método pode ser usado para encontrar soluções ótimas de Pareto em regiões não convexas.

O problema (3.7) tem um grande número de restrições e a função objetivo é não diferenciável.

3.5.5 Método da Função de Utilidade

No Método da Função de Utilidade o usuário fornece uma função de utilidade $U : \mathbb{R}^T \to \mathbb{R}$ que envolve todos os T objetivos. Esta função deve ser válida em toda região factível e o objetivo é maximizar o valor da função da seguinte forma.

$$\begin{array}{rll} \text{Maximizar} & U(f(x)) & \\ \text{sujeito a} & g_k(x) \geq 0 & k = 1, 2, \ldots, K \\ & h_v(x) = 0 & v = 1, 2, \ldots, V \\ & x_i^{(L)} \leq x_i \leq x_i^{(U)} & i = 1, 2, \ldots, n. \end{array} \tag{3.8}$$

Aqui, $f(x) = (f_1(x), f_2(x), \ldots, f_T(x))^T$. A função de utilidade fornece uma maneira de interação entre diferentes objetivos, por exemplo, entre duas soluções i e j, se $U(f(i)) > U(f(j))$ a solução i é melhor que a solução j.

Teorema 3.5.5. *Seja a função utilidade $U : \mathbb{R}^T \to \mathbb{R}$ estritamente decrescente. Seja f^{**} o valor máximo de U. Então, f^{**} é ótima de Pareto.*

A idéia deste método é simples e eficiente, se a função de avaliação for adequada. Entretanto, a qualidade da solução obtida depende da escolha desta função.

3.5.6 Programação de Metas

A idéia principal é encontrar soluções que atinjam uma meta s_t para um ou mais objetivos. Se não existe nenhuma solução que atinja a meta em todos os objetivos, tenta-se encontrar soluções que minimizem o desvio em relação a meta. Se existe uma solução que satisfaça

a meta, o objetivo da programação de metas é identificar esta solução. O problema pode ser formulado da seguinte maneira.

$$\begin{aligned} \text{Minimizar} \quad & |f_t(x) - s_t| \\ \text{sujeito a} \quad & x \in S \end{aligned} \tag{3.9}$$

Esta pode ser um técnica muito eficiente computacionalmente se forem conhecidas as metas e se elas estiverem na região factível.

Ainda em programação de metas, tem-se a programação de metas ponderada, onde uma função objetivo é composta pelo desvio de cada um dos T objetivos e pode ser formulado da seguinte maneira.

$$\begin{aligned} \text{Minimizar} \quad & \sum_{t=1}^{T}(\alpha_t p_t + \beta_t n_t) & \\ \text{sujeito a} \quad & f_t(x) - p_t + n_t = s_t & t = 1, 2, \ldots, T \\ & x \in S & \\ & n_t, p_t \geq 0, & t = 1, 2, \ldots, T, \end{aligned} \tag{3.10}$$

onde os parâmetros α_t e β_t são os pesos, para desvios positivos e negativos do t-ésimo objetivo.

Na programação de metas lexicográfica, diferentes metas são classificadas de acordo com sua importância e é resolvida uma seqüencia de problemas de programação de metas. Primeiramente, apenas as metas mais importantes e restrições correspondentes são consideradas na formulação. Se existirem múltiplas soluções, então outro problema é formulado com a próxima meta, utilizando a meta anterior como restrição. Este processo continua até que se encontre uma única solução, geralmente, uma única solução ótima de Pareto é encontrada.

Outro método é a programação de metas Min-Max, que é similar a programação de metas ponderada, mas ao invés de minimizar a soma ponderada dos desvios, agora o maior desvio em qualquer uma das metas é minimizado. Tem-se então o seguinte problema não linear.

$$\begin{aligned} \text{Minimizar} \quad & d & \\ \text{sujeito a} \quad & \alpha_t p_t + \beta_t n_t \leq d, & t = 1, 2, \ldots, T \\ & f_t(x) - p_t + n_t = s_t & t = 1, 2, \ldots, T \\ & x \in S & \\ & n_t, p_t \geq 0 & t = 1, 2, \ldots, T. \end{aligned} \tag{3.11}$$

3.5.7 Considerações Finais

Analisando todos estes métodos clássicos de otimização multiobjetivo, pode-se salientar algumas dificuldades.

- Os métodos encontram apenas uma solução ótima de Pareto em cada execução;
- Nem todas as soluções ótimas de Pareto podem ser encontradas por alguns métodos em um problema não-convexo;
- Todos os métodos precisam de um conhecimento do problema, como pesos, valor para ϵ ou metas adequadas.

Eles sugerem uma maneira de converter um problema de otimização multiobjetivo em um problema mono-objetivo, que deve ser então resolvido com um algoritmo apropriado. Na maioria dos casos a solução encontrada é ótima de Pareto, mas para encontrar N diferentes soluções ótimas de Pareto, pelo menos N diferentes problemas mono-objetivo devem ser resolvidos, mudando os parâmetros. Alguns algoritmos não garantem encontrar soluções em toda a região ótima de Pareto. Além disso, cada um destes métodos envolve muitos parâmetros que devem ser definidos pelo usuário.

Por outro lado eles apresentam algumas vantagens, e por isso são usados para resolver problemas reais. A prova de convergência é a principal vantagem destes métodos.

Capítulo 4

Algoritmos Evolutivos Multiobjetivo

Este capítulo é baseado em [12] e nele são apresentadas algumas características dos Algoritmos Evolutivos, bem como, as principais adaptações necessárias para utilizar estes algoritmos em problemas multiobjetivo.

Os Algoritmos Evolutivos (AE) imitam princípios da evolução natural para atingir uma solução ótima. Uma população de soluções é utilizada em cada iteração, ao invés de apenas uma, com isso, o resultado final também é uma população de soluções. Se um problema de otimização tem apenas uma solução ótima tenta-se buscar que os membros da população do algoritmo evolutivo convirjam para este ótimo. Entretanto, se um problema de otimização tem várias soluções eficientes, como no caso multiobjetivo, os algoritmos evolutivos podem ser usados para encontrar em sua população final diferentes soluções eficientes. Esta maneira de explorar a estrutura dos algoritmos evolutivos para encontrar várias soluções ótimas tem-se mostrado uma ferramenta muito poderosa para resolução de problemas de otimização multiobjetivo. Vale lembrar que a dificuldade em resolver tais tipos de problemas não é apenas a complexidade como no caso de problemas mono-objetivo, mas também devido a busca de diferentes soluções não dominadas que crescem com o número de objetivos do problema

Dentre os algoritmos evolutivos tem-se os Algoritmos Genéticos que são métodos flexíveis inspirados na evolução natural das espécies, e têm a capacidade de produzir soluções de boa qualidade para problemas complexos de grande porte, em tempo computacional viável. A eficiência dos algoritmos genéticos pode ser comprovada pela grande quantidade de trabalhos publicados com aplicações em diversas áreas.

Quando um algoritmo genético é aplicado a um problema de otimização, cada solução do problema deve ser codificada ou representada na forma de uma estrutura finita (vetor, matriz, etc). Em seguida, devem ser definidos os operadores genéticos de seleção, recombinação, mutação e estratégias de elitismo.

No desenvolvimento de algoritmos genéticos para otimização multiobjetivo, existem

algumas questões importantes que devem ser levadas em consideração. Para guiar a busca nas direções das soluções ótimas de Pareto deve-se ter em mente que para calcular o nível de aptidão (*fitness*) das soluções da população e para fazer a escolha de soluções (seleção) que participarão na reprodução de novas soluções, múltiplos objetivos devem ser considerados. Assim, a preservação da diversidade na população passa a ter importância fundamental para evitar uma convergência prematura e obter um conjunto de soluções não dominadas bem distribuídas.

4.1 Conceitos Básicos e Operadores Genéticos

Nos algoritmos genéticos cada solução é um indivíduo e as características destes indivíduos são representadas por seus cromossomos tal como na molécula de DNA.

Após definir a forma de representação da solução, deve-se gerar uma população inicial. Neste ponto podem ser incorporados aos algoritmos genéticos algumas heurísticas, tais como, heurísticas gulosas.

É necessário que as soluções sejam avaliadas, isto é feito através do cálculo do *fitness* que é atribuído a cada solução, esta cálculo leva em conta o valor de cada função objetivo.

Em seguida um critério de parada deve ser verificado, se ele não é satisfeito a população é modificada pelos operadores genéticos e uma nova (de preferência melhor) população é criada. O contador de gerações é incrementado, indicando que uma geração do algoritmo foi completada. A Figura 4.1 apresenta um fluxograma dos procedimentos citados.

4.1.1 Operador de Reprodução ou Seleção

O objetivo principal da reprodução ou seleção é selecionar os melhores indivíduos que poderão gerar novas soluções, isto pode ser feito atribuindo probabilidades maiores aos melhores indivíduos, ou fazendo cópias das soluções mais promissoras eliminando as piores, mantendo o tamanho da população constante. Alguns métodos mais utilizados são o método do torneio, método da proporção (roleta) e o método da seleção por *ranking*.

No método do torneio, duas soluções escolhidas na população são comparadas, a melhor é escolhida e colocada em uma população intermediária de onde serão selecionados os indivíduos para fazerem a recombinação (crossover). Duas outras soluções são escolhidas novamente e a melhor é selecionada. Cada solução participa duas vezes do torneio, a melhor solução da população vencerá duas vezes, tendo duas cópias na população intermediária, similarmente a pior solução perderá duas vezes e será eliminada da população

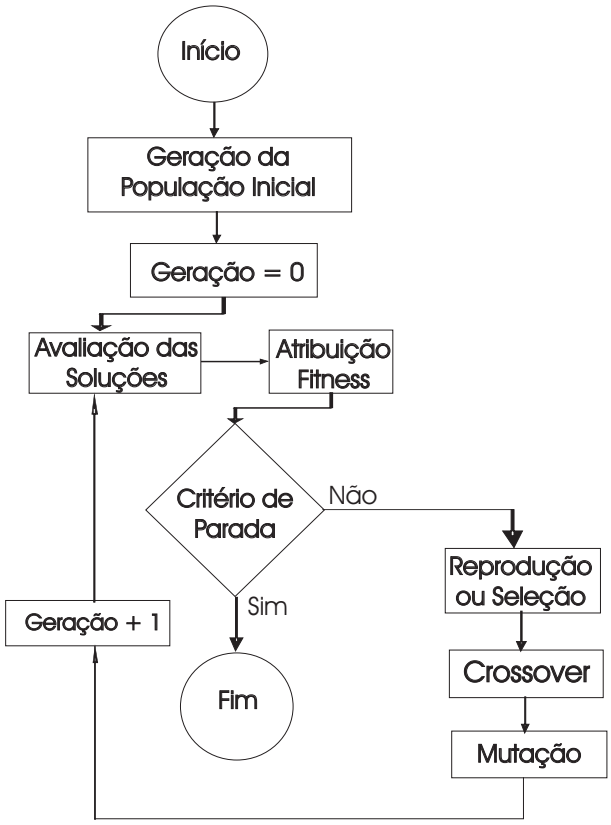


Figura 4.1: Fluxograma Algoritmo Genético

intermediária. Desta maneira, qualquer solução terá nenhuma, uma ou duas cópias na população intermediária.

No método da proporção, são criadas cópias de cada solução um número de vezes proporcional ao seu *fitness*. Se o *fitness* médio de uma população é F_{med}, uma solução com *fitness* F_i terá um valor esperado de $\frac{F_i}{F_{med}}$ cópias. Pode-se pensar neste método como uma roleta (método da roleta), dividida em N (tamanho da população) espaços de tamanhos proporcionais ao *fitness* de cada membro da população. Então a roleta é girada N vezes, escolhendo assim N indivíduos para a população intermediária. Assim a probabilidade do indivíduo i ser selecionado é

$$p_i = \frac{F_i}{\sum_{j=1}^{N} F_j}.$$

Como neste método a escolha é repetida muitas vezes tem-se o problema da variância. Para contornar este problema a probabilidade p_i pode ser multiplicada pelo tamanho da população, calculando assim um número esperado de cópias para cada solução, que é a parte inteira do número esperado de cópias. Então o método da roleta é aplicado utilizando a parte fracionária para selecionar os indivíduos restantes.

Além da grande complexidade computacional, o método da proporção pode apresentar

problemas de convergência. Por exemplo, se na população uma solução tem um alto valor de *fitness* comparado as outras, a probabilidade de escolher esta solução será muito próxima de um, dominando a população intermediária com suas cópias. Por outro lado, se na população todas as soluções tiverem *fitness* muito próximos, todas as soluções terão uma probabilidade similar, fazendo com que cada solução tenha apenas uma cópia na população intermediária, o que é equivalente a não se utilizar um procedimento de reprodução ou seleção.

Este problema pode ser evitado utilizando o método da seleção por *ranking*. Primeiramente cada solução é ordenada de acordo com seu *fitness*, da pior (*rank* 1) para a melhor (*rank* N). A cada membro é atribuído um *fitness* igual ao valor do seu *rank*, então o método da proporção é utilizado e N soluções são escolhidas para fazerem parte da população intermediária.

4.1.2 Operador de Recombinação (*Crossover*)

O *crossover* é aplicado na população intermediária, a reprodução ou seleção não cria nenhuma nova solução, apenas faz cópias das melhores soluções em detrimento das piores, a criação de novas soluções é feita através do *crossover* e da mutação. O *crossover* mais comum é quando dois indivíduos da população intermediária são selecionados e então eles transmitem suas características para a criação de dois indivíduos que farão parte da nova geração. O número de indivíduos necessários para se produzir novas soluções e o número de novas soluções criadas podem variar.

Não se pode garantir que o *crossover* entre indivíduos gerará soluções melhores que os pais, mas existem grandes chances de que isto aconteça, já que os pais selecionadas não são soluções arbitrárias escolhidas na população, mas são aquelas que sobreviveram a algum processo de reprodução ou seleção. Assim é esperado que elas transmitam boas características à nova geração.

4.1.3 Operador de Mutação

A mutação é necessária para que a diversidade da população seja mantida. Cada solução quando criada terá uma probabilidade de sofrer uma pequena perturbação, ou seja, esse indivíduo é modificado. A seleção ou reprodução identifica bons indivíduos, o *crossover* seleciona aleatoriamente indivíduos da população intermediária para que um ou mais indivíduos sejam criados. Então a mutação causa uma pequena perturbação nas novas soluções para que a diversidade seja mantida (enfatizada) e de preferência se tornem soluções melhores.

4.2 *Fitness Sharing*

A qualidade das soluções é representada por uma função de avaliação denominada *fitness*, esta função representa quão boa é esta solução. Em otimização multiobjetivo, geralmente a função de *fitness* leva em consideração todos os objetivos envolvidos no problema.

Manter a diversidade da população é crucial para encontrar um conjunto de soluções não dominadas cujos pontos estejam bem distribuídos ao longo de toda fronteira de Pareto. A função de *fitness* pode auxiliar na tentativa de se manter a diversidade da população, evitando a convergência para regiões pequenas do conjunto ótimo de Pareto, para isso muitos métodos utilizam o *fitness sharing* para manter a diversidade da população.

Proposto por Goldberg e Richardson [25], o *fitness sharing* consiste em dividir a população em diferentes nichos de acordo com a proximidade das soluções. A idéia desta técnica é penalizar o *fitness* de soluções que estão muito próximas, esta proximidade é determinada através de uma função no espaço de decisões ou no espaço objetivo, denominada função *sharing*

$$Sh(d_{ij}) = \begin{cases} 1 - \left(\dfrac{d_{ij}}{\sigma_{share}}\right)^{\alpha}, & \text{se } d_{ij} \leq \sigma_{share} \\ 0, & \text{caso contrário} \end{cases} \tag{4.1}$$

onde d_{ij} é uma distância entre duas soluções i e j da população, α é um parâmetro que normalmente é 1 ou 2, e σ_{share} é a distância máxima permitida entre quaisquer duas soluções.

A função (4.1) assume valores no intervalo $[0, 1]$, dependendo dos valores de d e de σ_{share}. Se $d = 0$ (o que significa que duas soluções são idênticas ou a distância entre elas é zero), $Sh(d) = 1$, o que significa que as soluções receberão um valor máximo de penalização. Por outro lado, se $d \geq \sigma_{share}$ (o que significa que duas soluções estão a pelo menos uma distância de σ_{share} uma da outra), $Sh(d) = 0$, ou seja, estas duas soluções não recebem penalização.

Se o valor desta função é calculado para todos os membros da população (inclusive ela mesma) e somados, tem-se a medida *niche* que é calculada para a $i-$ésima solução da seguinte forma

$$\eta c_i = \sum_{j=1}^{N} Sh(d_{ij}), \quad i = 1, \ldots, N. \tag{4.2}$$

A medida *niche* é uma estimativa da quantidade de soluções que rodeiam a $i-$ésima solução. O novo *fitness* da solução é calculado por

$$F_i' = \frac{F_i}{\eta c_i}. \tag{4.3}$$

A seguir, o procedimento do *fitness sharing* é esquematizado.

Passo 1 Para cada solução $x^{(i)}$ da população atribua um *fitness* inicial F_i.

Passo 2 Para cada solução $x^{(i)}$ da população calcule o valor de $Sh(d_{i,j})$ utilizando a equação (4.1).

Passo 3 Para cada solução $x^{(i)}$ da população calcule a medida *niche* utilizando a equação (4.2).

Passo 4 Para cada solução $x^{(i)}$ da população calcule o novo *fitness* utilizando a equação (4.3).

A equação (4.1) introduz dois parâmetros, α e σ_{share}, a escolha de α não tem muito influência na performance do método, entretanto o parâmetro σ_{share} influencia bastante. Este valor pode ser estimado da seguinte maneira, primeiramente utiliza-se d_{ij} como sendo a distância Euclidiana

$$d_{ij} = \sqrt{\sum_{k=1}^{n} (x_k^{(i)} - x_k^{(j)})^2}. \tag{4.4}$$

E σ_{share} é estimado da seguinte maneira

$$\sigma_{share} = \frac{r}{\sqrt[n]{q}}, \tag{4.5}$$

com

$$r = \frac{1}{2}\sqrt{\sum_{k=1}^{n} (x_k^{(U)} - x_k^{(L)})^2}, \tag{4.6}$$

e q quantidade de soluções ótimas que se deseja encontrar.

Desta forma a estimativa para σ_{share} é

$$\sigma_{share} = \frac{1}{2}\frac{\sqrt{\sum_{k=1}^{n} (x_k^{(U)} - x_k^{(L)})^2}}{\sqrt[n]{q}}. \tag{4.7}$$

4.3 Algoritmos Evolutivos Que Não Utilizam Abordagem de Pareto

A principal diferença entre os métodos clássicos multiobjetivo e os algoritmos evolutivos é que a cada iteração uma população de soluções é gerada, que é uma grande vantagem para resolver problemas de otimização multiobjetivo. Esta população pode ser explorada para enfatizar as soluções não dominadas e simultaneamente preservar a diversidade destas soluções utilizando um operador de nicho. Desta forma algumas boas soluções podem ser encontradas e mantidas na população. Depois de algumas gerações, este processo pode levar a população a convergir para soluções próximas da fronteira ótima de Pareto e com uma boa diversidade.

4.3.1 Vector Evaluated Genetic Algorithm (VEGA)

O nome deste algoritmo vem do fato de que ele avalia um vetor (ao invés de uma função escalar), em que cada elemento do vetor representa uma função objetivo, este algoritmo foi proposto por Schaffer [44] em 1984.

Suponha que o problema tenha T objetivos, o algoritmo divide a população a cada iteração em T subpopulações aleatoriamente. Cada subpopulação recebe um *fitness* baseado em um objetivo diferente. Desta forma, cada uma das T funções objetivo é utilizada para avaliar uma parte da população.

Este algoritmo é particularmente útil para problemas em que as funções objetivo assumem valores com diferentes ordens de magnitude. Como todos os membros de uma subpopulação recebem um *fitness* baseado em um objetivo particular, restringindo a seleção a apenas uma subpopulação, boas soluções correspondentes aquela função objetivo serão enfatizadas. Além do que, como as soluções não são comparadas para funções objetivo diferentes, não há disparidade entre os valores. Em geral, o método da proporção é utilizado no processo de seleção.

A seguir são apresentados os procedimentos do algoritmo. Considere T funções objetivo e uma população de tamanho N.

Passo 1 Faça o contador das funções objetivo $i = 1$ e defina $k = N/T$.

Passo 2 Para cada solução $j = 1 + (i - 1) * k$ até $j = i * k$ atribua um *fitness*

$$F(x^{(j)}) = f_i(x^{(j)}).$$

Passo 3 Execute o método da proporção na subpopulação i para selecionar k indivíduos para fazerem parte da população intermediária.

Passo 4 Se $i = T$, vá para o Passo 5. Caso contrário, incremente i em um e vá para o Passo 2.

Passo 5 Na população intermediária realize *crossover* e mutação pra criar uma nova população.

O algoritmo enfatiza soluções que são boas em um objetivo. Para que haja soluções intermediárias, Schaffer [44] permitiu o *crossover* entre quaisquer soluções (não levando em conta a subpopulação), para que a solução gerada possa ser razoavelmente boa entre dois ou mais objetivos.

O funcionamento do algoritmo é ilustrado na Figura 4.2.

A principal vantagem do VEGA é que ele é simples de implementar e tem a tendência de encontrar soluções próximas da melhor solução de cada objetivo.

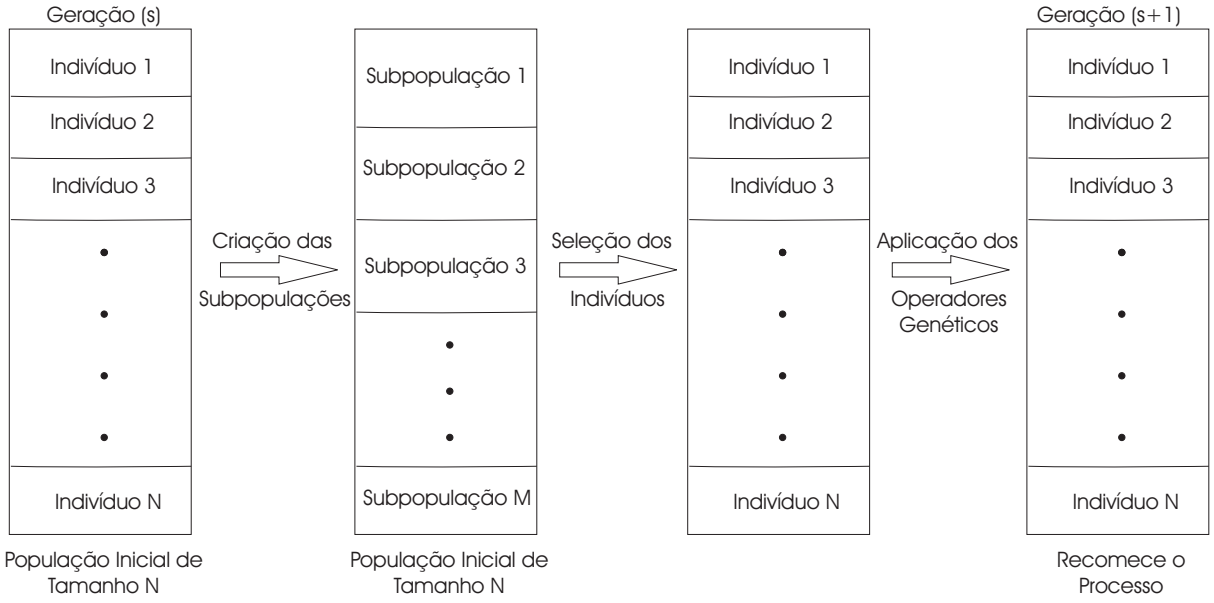


Figura 4.2: Visão Geral do Algoritmo VEGA

Apesar de o *crossover* ser permitido entre quaisquer indivíduos, inclusive os melhores em cada objetivo, não é encontrado um conjunto diverso de soluções, eventualmente, o VEGA converge para os melhores indivíduos em cada objetivo.

Schaffer [44] percebeu que uma das maneiras de manter a diversidade da população na região ótima de Pareto era enfatizar as soluções não dominadas. Uma das maneiras de dar uma importância maior para estas soluções é lhes dar uma probabilidade maior de ser escolhida. No método da proporção, uma solução com *fitness* f_i é selecionada com uma probabilidade $\dfrac{F_i}{\sum_{j=1}^{N} F_j}$. Então, a solução i pode ser enfatizada adicionando um termo ϵ_i a esta probabilidade, mas deve-se tomar o cuidado para que esta quantidade extra seja subtraída de alguma outra solução na população.

Após encontrar as soluções não dominadas uma quantidade ϵ é subtraída de cada solução dominada, reduzindo assim sua importância. Para uma subpopulação de N' membros tendo ρ' soluções não dominadas, seria então uma redução total de $(N'-\rho')\epsilon$ de todas as $(N'-\rho')$ soluções dominadas. Este valor é então redistribuído de forma igualitária entre as ρ' soluções não dominadas, adicionando um valor $\dfrac{N'-\rho'}{\rho'}\epsilon$ a cada uma.

4.3.2 Algoritmo Genético de Pesos Aleatórios (Random Weighted GA)

Este algoritmo foi proposto por Ishibuchi e Murata [33], nele são utilizados pesos aleatórios e o *fitness* de cada solução é a soma ponderada dos objetivos. Os passos envolvidos são descritos a seguir.

Passo 1 É gerada uma população inicial aleatória.

Passo 2 São calculados os valores dos T objetivos para cada indivíduo da população. As soluções não dominadas são armazenadas em uma população separada, denominada NDOM, e a população atual é denominada ATUAL.

Passo 3 Sejam M o número de soluções não dominadas em NDOM e N o tamanho de ATUAL, então são escolhidos $(N - M)$ pares de pais, da seguinte maneira. Sejam $r_1, r_2, \ldots, r_k$, k números aleatórios no intervalo $[0, 1]$. O *fitness* para cada solução é

$$f(x^{(i)}) = \sum_{j=1}^{T} w_j^{(i)} f_j(x^{(i)}),$$

onde T é o número de objetivos e

$$w_j^{(i)} = \frac{r_j}{(r_1 + r_2 + \ldots + r_T)},$$

para $j = 1, 2, \ldots, T$. Isto assegura que os pesos são todos positivos e que

$$\sum_{j=1}^{T} w_j^{(i)} = 1.$$

Os pais são então selecionados com uma probabilidade

$$p(x^{(i)}) = \frac{f(x^{(i)}) - f_{min}(ATUAL)}{\sum_{x \in ATUAL} (f(x) - f_{min}(ATUAL))},$$

onde $f_{min}(ATUAL)$ é o menor *fitness* da população ATUAL.

Passo 4 O *crossover* é aplicado aos $(N - M)$ pares de pais e a mutação é aplicada à nova população.

Passo 5 São selecionadas, aleatoriamente, E soluções de $NDOM$. Então elas são adicionadas às $(N - M)$ soluções geradas no passo anterior para que uma população de tamanho N seja construída.

Passo 6 É aplicada uma busca local para todos os N indivíduos de ATUAL.

Passo 7 Verifica se um determinado critério é atingido, caso contrário retorna ao Passo 2.

Neste algoritmo, a diversidade das soluções não dominadas é mantida de duas maneiras: usando um vetor peso aleatório para avaliar cada solução, enfatizando assim soluções que podem levar a diferentes soluções na região ótima de Pareto; e usando uma operação que substitui uma parte da população por indivíduos de uma população externa de soluções não dominadas. A ênfase para soluções próximas da fronteira ótima de Pareto é dada quando é utilizado, no momento da reprodução ou seleção, o método da proporção com peso nas funções. Entretanto, como outros métodos que utilizam peso, ele também não encontra soluções ótimas de Pareto em problemas não convexos.

4.4 Algoritmos Evolutivos com Abordagem de Pareto

Muitos algoritmos incorporam a idéia sugerida por Goldberg [24] (alguns deles serão apresentados a seguir), de utilizar um *ranking* das soluções não dominadas para direcionar a população para a fronteira de Pareto nos problemas de otimização multiobjetivo. A idéia básica é encontrar um conjunto de soluções não dominadas pelo resto da população, estes indivíduos recebem então o maior *rank* e são tirados da população. Em seguida outro conjunto de soluções não dominadas é encontrado na população restante e recebe um *rank* menor que o anterior, este processo continua até que a população esteja totalmente classificada. Goldberg também sugeriu o uso de algum tipo de técnica de nicho para que o algoritmo não convirja para um único ponto. Um mecanismo como o *fitness sharing* mantém indivíduos por toda a fronteira de soluções não dominadas.

O principal problema com o método de *ranking* de Pareto é que não há um algoritmo eficiente para se identificar as soluções não dominadas. Os algoritmos tradicionais apresentam grande queda de performance quando o número de indivíduos da população e o número de objetivos aumentam. Quando se utiliza o *fitness sharing* é necessário estimar o valor de σ_{share}, o que não é fácil.

Apesar de alguns poucos aspectos negativos a performance destes métodos é muito boa (Deb [12]).

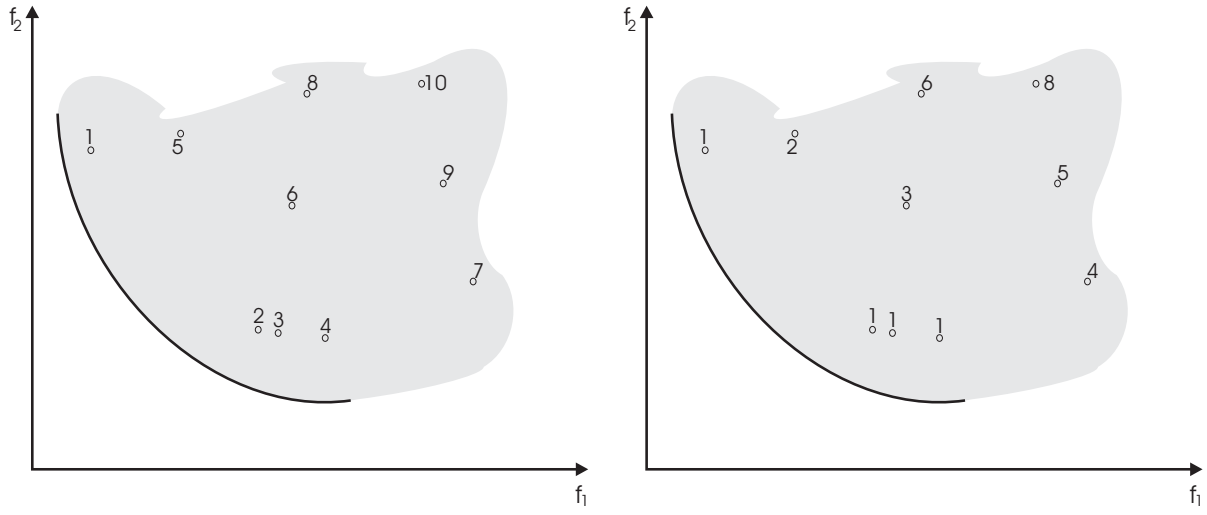
4.4.1 Multiple Objective Genetic Algorithm (MOGA)

Fonseca e Fleming [21] propuseram um algoritmo em que é usado o esquema de *rank*, mas de uma maneira diferente. Este algoritmo enfatiza as soluções não dominadas e

simultaneamente mantém a diversidade das soluções. O *rank* (τ_i) de cada solução será o número de indivíduos na população atual que dominam este indivíduo (η_i) mais um,

$$\tau_i = 1 + \eta_i.$$

Desta maneira, todas as soluções não dominadas recebem *rank* 1, enquanto que os indivíduos dominados sofrem uma penalização. Com a população classificada, o *fitness* é atribuído para cada indivíduo, isto é feito através de uma interpolação do melhor *rank* (*rank* 1) para o pior (*rank* $n \leq N$), geralmente é utilizada uma função linear e as soluções não dominadas são enfatizadas. Um exemplo de classificação da população por *rank* pode ser observado na Figura 4.3



(a) Soluções (b) *Rank* das Soluções

Figura 4.3: Classificação por *Rank*

Para manter a diversidade Fonseca e Fleming [21] também utilizaram o *fitness sharing*, mas de uma maneira diferente pois, levam em conta o valor das funções objetivo ao invés das variáveis e utilizam o valor de $\alpha = 1$. Desta forma, a distância entre duas soluções i e j é calculada da seguinte maneira

$$d_{ij} = \sqrt{\sum_{k=1}^{T} \left(\frac{f_k^{(i)} - f_k^{(j)}}{f_k^{max} - f_k^{min}} \right)^2}, \tag{4.8}$$

onde f_k^{max} e f_k^{min} são respectivamente os valores máximos e mínimos da $k-$ésima função objetivo. Para a solução i, d_{ij} é calculado para toda solução j (inclusive ela mesma) que tenha o mesmo *rank* de i. Para calcular a função $Sh(d)$, utiliza-se a equação (4.1) com $\alpha = 1$, então a medida *niche* para cada solução i é calculada somando os valores das funções de *sharing*

$$\eta c_i = \sum_{j=1}^{\mu(\tau_i)} Sh(d_{ij}), \tag{4.9}$$

onde $\mu(\tau_i)$ é o número de soluções que tenham *rank* τ_i.

O novo *fitness* é então calculado dividindo o *fitness* da solução pela medida *niche*, quando isto é feito o *fitness* da cada solução diminui e uma redução maior ocorre com as soluções que são muito similares à outras soluções. Para que a média do *fitness* de cada *rank* seja mantida tal qual era antes do cálculo do novo *fitness*, estes valores devem ser recalculados. Depois disso os operadores genéticos podem ser aplicados na população.

Os procedimentos de atribuição do *fitness* presentes no MOGA, em detalhes, são esquematizados a seguir e na Seção 5.3 será apresentado um exemplo de utilização do algoritmo.

Passo 1 Escolha um valor adequado para σ_{share}. Inicialize $\mu(j) = 0$ para todos os possíveis *ranks*, $j = 1, \ldots, N$. Defina um contador de soluções $i = 1$.

Passo 2 Calcule o número de soluções (η_i) que dominam a solução i. Calcule o *rank* da $i-$ésima solução como sendo $\tau_i = 1 + \eta_i$. Incremente o contador que guarda o número de soluções no *rank* τ_i em um, isto é, $\mu(\tau_i) = \mu(\tau_i) + 1$.

Passo 3 Se $i < N$, incremente i em um e vá para o Passo 1. Caso contrário, vá para o Passo 4.

Passo 4 Identifique o maior *rank* τ^*, verificando o maior τ_i que tenha $\mu(\tau_i) > 0$. Em seguida calcule o *fitness* para cada solução

$$F_i = N - \sum_{k=1}^{\tau_i - 1} \mu(k) - 0,5(\mu(\tau_i) - 1), \quad i = 1, \ldots, N. \tag{4.10}$$

Esta equação atribui um mesmo valor de *fitness* para todas as soluções de mesmo *rank*. Defina um contador de *rank* $\tau = 1$.

Passo 5 Para cada solução i no *rank* τ, calcule a medida *niche* ηc_i utilizando as outras soluções de mesmo *rank* e a equação (4.9). Calcule o novo *fitness* $F'_j = F_j/\eta c_j$. Para que o *fitness* médio seja mantido, recalcule o novo *fitness* da seguinte maneira

$$F'_j \leftarrow \frac{F_j \mu(\tau)}{\sum_{k=1}^{\mu(\tau)} F'_k} F'_j. \tag{4.11}$$

Passo 6 Se $\tau < \tau^*$, incremente τ em um e vá para o passo 5. Caso contrário, o processo está completo.

Uma das principais vantagens do MOGA é que a atribuição do *fitness* é feita de maneira simples, além do que a função *sharing* é calculada em relação aos valores das

funções objetivos e não das variáveis o que é mais simples. Por estes motivos o MOGA pode ser facilmente aplicado a vários tipos de problemas de otimização combinatória e permite encontrar uma boa diversidade nas soluções ótimas de Pareto em relação aos valores das funções objetivo.

Apesar de ser um método eficiente e relativamente fácil de ser implementado, sua performance depende muito de uma escolha adequada do valor de σ_{share}. Fonseca e Fleming [21] desenvolveram um forma de calcular este valor.

O valor de σ_{share} deve ser escolhido de forma que as soluções estejam uniformemente distribuídas na fronteira ótima de Pareto. Se for utilizado o *fitness sharing* em relação às funções objetivo, utiliza-se a distância Euclidiana para calcular a distância entre qualquer duas soluções. Então, se a fronteira ótima de Pareto é conhecida, deseja-se ter todas as N (tamanho da população) soluções uniformemente espaçadas na fronteira ótima de Pareto. Sabendo o perímetro (L), pode-se calcular $\sigma_{share} = L/N$. Desta forma, cada nicho teria uma solução. Entretanto, a dificuldade é que a região ótima de Pareto não é conhecida a princípio.

No método proposto, o σ_{share} é calculado em cada geração, entre os membros da população. Primeiramente deve-se encontrar os limitantes superiores e inferiores da cada uma das funções objetivo (l_i e u_i). Desta forma todas as soluções da população estão em um hipervolume $V = \prod_{i=1}^{T}(u_i - l_i)$ (onde T é a quantidade de objetivos. Em cada função objetivo é adicionado o valor de σ_{share} e encontrado um novo hipervolume $V' = \prod_{i=1}^{T}(u_i - l_i + \sigma_{share})$. A diferença entre estes dois hipervolumes é $\Delta V = V' - V$ e todas as N soluções devem estar nesta diferença. Como cada nicho ocupa um hipervolume σ_{share}^T e há N destes hipervolumes, é utilizado a seguinte equação para calcular σ_{share}

$$\Delta V = \prod_{i=1}^{T}(u_i - l_i + \sigma_{share}) - \prod_{i=1}^{T}(u_i - l_i) = N(\sigma_{share})^T. \tag{4.12}$$

Quando tem-se dois objetivos ($T = 2$), a equação acima se reduz a

$$\sigma_{share} = \frac{u_2 - l_2 + u_1 - l_1}{N - 1}. \tag{4.13}$$

Se for utilizado um único σ_{share} para todas as funções objetivo, então elas devem ser normalizadas, desta forma a equação (4.12) pode ser reescrita da seguinte forma

$$(1 + \sigma'_{share})^T - 1 = N(\sigma'_{share})^T. \tag{4.14}$$

Para $T = 2$, a equação acima fica

$$\sigma'_{share} = \frac{2}{(N - 1)}. \tag{4.15}$$

A equação (4.15) permite calcular um valor normalizado e constante para σ'_{share} que não precisa ser calculado em cada geração. Entretanto, ao invés de utilizar a distância Euclidiana, uma distância Euclidiana normalizada deve ser calculada

$$d'_{ij} = \sqrt{\sum_{k=1}^{T} \left(\frac{f_k^{(i)} - f_k^{(j)}}{u_k - l_k} \right)^2}. \tag{4.16}$$

Agora o cálculo da distância é um pouco diferente, pois os limitantes u_k e l_k mudam em cada geração, antes estes limitantes eram fixos. Os autores relatam que as duas maneiras, estática e dinâmica apresentaram resultados similares.

4.4.2 Non-Dominated Sorting Genetic Algorithm (NSGA)

O NSGA foi proposto por Srinivas e Deb [46] e é baseado em várias camadas de classificação dos indivíduos. Também utiliza uma estratégia de *sharing* que preserva a diversidade entre as soluções de cada camada de soluções não dominadas.

O primeiro passo no NSGA é classificar a população P. As soluções são classificadas em classes mutuamente exclusivas (ou conjuntos não dominados) P_j, $j = 1, 2, \ldots, \rho$, onde

$$P = \bigcup_{j=1}^{\rho} P_j. \tag{4.17}$$

É importante ressaltar que entre quaisquer dois indivíduos de uma mesma classe, não se pode concluir que um é melhor ou pior que o outro em relação a todos os objetivos.

Assim que a classificação é feita, fica claro que todas as soluções pertencentes ao primeiro conjunto, isto é, as que estão em P_1, pertencem ao melhor conjunto não dominado na população. As próximas melhores pertencem ao segundo conjunto, membros de P_2, e assim por diante. Obviamente, as piores soluções são aquelas pertencentes ao último conjunto, membros de P_ρ, em que ρ é o número de diferentes conjuntos de soluções não dominadas da população.

As soluções que pertencem a primeira classe são aquelas mais próximas da fronteira ótima de Pareto, dessa maneira elas recebem o maior *fitness*, progressivamente vão se atribuindo *fitness* piores para as soluções das outras classes.

Qualquer solução i do primeiro (ou melhor) conjunto não dominado recebe um *fitness* $F_i = N$ (tamanho da população). Este valor fixo N é utilizado pelo fato de que todas as soluções no primeiro conjunto não dominado são igualmente importantes em relação a proximidade com a fronteira ótima de Pareto e o mesmo *fitness* é atribuído a todas estas soluções.

Atribuir um *fitness* maior para as soluções pertencentes aos melhores conjuntos não dominados garante uma pressão para que sejam selecionadas soluções na fronteira ótima de Pareto. Entretanto, para ter uma diversidade entre as soluções, é necessário que se use algum mecanismo para se preservar a diversidade. No NSGA o *fitness* é degradado com base no número de soluções vizinhas. Para que uma solução que esteja sozinha em uma região não seja perdida durante o processo, é preciso ter certeza que soluções em regiões menos povoadas sejam adequadamente enfatizadas. Para isso é utilizado o *fitness sharing*.

Para cada solução i em uma classe P_s, a distância Euclidiana normalizada d_{ij} para outra solução j é utilizada

$$d_{ij} = \sqrt{\sum_{k=1}^{|P_s|} \left(\frac{x_k^{(i)} - x_k^{(j)}}{x_k^{max} - x_k^{min}} \right)^2}. \tag{4.18}$$

As distâncias são calculadas e utilizadas para calcular a função *sharing*, utilizando a equação (4.1) com $\alpha = 2$. A função *sharing* assume valores entre zero e um, dependendo da distância d_{ij}. Qualquer solução j que está a uma distância maior do que σ_{share} da $i-$ésima solução não contribui em nada para o valor da função *sharing*. Após todas as $|P_s|$ funções *sharing* serem calculadas, elas são somadas para o cálculo da medida *niche* ηc_i da $i-$ésima solução. A medida *niche* denota o número de soluções na vizinhança da $i-$ésima solução, incluindo ela mesma. Se não existe nenhuma outra solução em um raio σ_{share} da solução, a medida *niche* para esta solução será um. Se por outro lado, todas as $|P_s|$ soluções da classe estão muito próximas uma das outras comparadas com σ_{share}, a medida *niche* de qualquer solução na classe será próxima de $|P_s|$.

Por último o *fitness* da $i-$ésima solução é reduzido de acordo com a medida *niche* e o novo *fitness* é obtido $F_i' = \frac{F_i}{\eta c_i}$. O processo de degradar o *fitness* de uma solução que esta rodeada de muitas outras soluções ajuda a enfatizar as soluções que estão em regiões menos povoadas, este é o intuito do mecanismo que preserva a diversidade entre as soluções.

Após este procedimento ser realizado para uma classe, o menor valor entre os novos *fitness* é reduzido e é atribuído às soluções da próxima classe. Isto garante que nenhuma solução em uma classe terá um novo *fitness* menor do que as soluções em uma classe pior. Então o *fitness sharing* é aplicado às soluções das próximas classes até que todas as soluções tenham recebido o novo *fitness*.

No operador de reprodução ou seleção são feitas cópias na população intermediária proporcionais ao novo *fitness*, desta maneira, as soluções da primeira classe tem maior chance de sobreviver do que a das outras classes, o que permite uma busca por regiões não dominadas e resulta em uma convergência rápida. Por outro lado, a função *sha-*

ring também garante que soluções em regiões menos povoadas tenham mais cópias na população intermediária.

A seguir o processo de atribuição de *fitness* do NSGA é descrito.

Passo 1 Escolha um σ_{share} apropriado, um valor pequeno e positivo ϵ e inicialize $F_{min} = N+\epsilon$. Defina um contador $j = 1$.

Passo 2 Classifique a população P de acordo com a não dominância.

Passo 3 Para cada $q \in P_j$

Passo 3a Atribua um *fitness* $F_j^{(q)} = F_{min} - \epsilon$.

Passo 3b Calcule a medida *niche* somente entre as soluções de P_j utilizando a equação (4.2).

Passo 3c Calcule o novo *fitness* $F_j'^{(q)} = F_j^{(q)}/\eta c_q$.

Passo 4 Faça $F_{min} = \min(F_j'^{(q)} : q \in P_j)$ e $j = j + 1$.

Passo 5 Se $j < \rho$, vá para o passo 3. Caso contrário, o procedimento está completo.

A maior vantagem do NSGA é a atribuição de *fitness* de acordo com conjuntos não dominados. Já que conjuntos melhores são enfatizados o NSGA chega cada vez mais próximo da fronteira ótima de Pareto. O *fitness sharing* é feito nas variáveis ao invés das funções objetivo, para garantir uma maior distribuição de indivíduos e permitir que múltiplas soluções existam. Esta técnica é mais ineficiente (tanto computacionalmente quanto na qualidade das soluções na fronteira de Pareto encontradas) do que o MOGA, e muito mais sensível a escolha do σ_{share}. Pode-se realizar o *fitness sharing* nas funções objetivo e também estimar o valor de σ_{share} como proposto por Fonseca e Fleming [21] .

Na figura 4.4 tem-se um fluxograma do NSGA.

4.4.3 *Niched Pareto Genetic Algorithm* (NPGA)

O NPGA foi proposto por Horn *et al.* [32] e também é baseado no conceito de não dominância e o operador de reprodução ou seleção utilizado é o método do torneio. Neste algoritmo a estratégia de nicho é atualizada durante o funcionamento do método. No momento da seleção, duas soluções i e j são escolhidas aleatoriamente da população de pais P. Então uma subpopulação T de tamanho t_{dom} ($\ll N$) é escolhida e cada uma das soluções i e j são comparadas com cada uma das soluções da subpopulação. Se das duas soluções, uma domina todas as soluções da subpopulação e a outra é dominada por pelo menos uma solução, então a primeira é escolhida. Entretanto se as duas soluções i

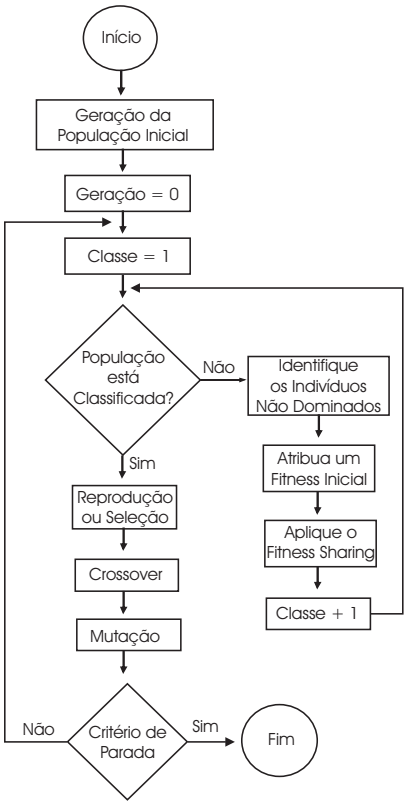


Figura 4.4: Fluxograma do NSGA

e j são dominadas por pelo menos uma solução da subpopulação ou não são dominadas por nenhuma outra elas são comparadas com a população de filhos Q. Cada solução é colocada na população de filhos e a medida *niche* é calculada, a que tiver o menor valor vence. No início do procedimento, quando não há ainda uma população de filhos uma das duas soluções é selecionada aleatoriamente, o mesmo processo é repetido para outro par de soluções para se escolher o segundo pai, destes dois são gerados os filhos. A partir do terceiro torneio o procedimento original é utilizado.

Para quaisquer duas soluções i e j, e a atual população de filhos Q, o seguinte procedimento de seleção, denominado Torneio-NPGA (i, j, Q), é utilizado.

Passo 1 Escolha uma subpopulação T_{ij} de tamanho t_{dom} da população de pais P.

Passo 2 Calcule α_i, o número de soluções em T_{ij} que domina i. Calcule α_j, o número de soluções em T_{ij} que domina j.

Passo 3 Se $\alpha_i = 0$ e $\alpha_j > 0$, então a solução i é a vencedora. O processo está completo.

Passo 4 Caso contrário, se $\alpha_i > 0$ e $\alpha_j = 0$, então a solução j é a vencedora. O processo está completo.

Passo 5 Caso contrário, se $|Q| < 2$, o indivíduo i ou j é escolhido vencedor com uma probabilidade de $0,5$. O processo está completo. Alternativamente, a medida *niche* ηc_i e ηc_j são calculadas inserindo i e j na população de filhos Q, independentemente. Com o σ_{share}, a medida *niche* é calculada como sendo o número de filhos ($k \in Q$) com uma distância d_{ik} de no máximo σ_{share} de i. A distância d_{ik} é a distância Euclidiana entre as soluções i e k no espaço das funções objetivo

$$d_{ik} = \sqrt{\sum_{t=1}^{T} \left(\frac{f_t^{(i)} - f_t^{(k)}}{f_t^{max} - f_t^{min}} \right)^2}. \tag{4.19}$$

Sendo f_t^{max} e f_t^{min} os valores máximos e mínimos da $t-$ésima função objetivo.

Passo 6 Se $\eta c_i \leq \eta c_j$, a solução i é a vencedora. Caso contrário, a solução j é a vencedora.

Uma característica importante do NPGA é que não há a necessidade da atribuição de um *fitness* para cada solução. A seleção por torneio prefere soluções não dominadas e quando esta relação não pode ser estabelecida, os pais que residem em regiões menos povoadas na população dos filhos é escolhida, o que possibilita uma procura por soluções próximas a frente de Pareto mantendo uma diversidade de soluções.

A seguir o processo completo do NPGA é esquematizado.

Passo 1 Misture a população P, faça $i = 1$, e $Q = \emptyset$.

Passo 2 Faça o torneio e encontre o primeiro pai, p_1 =Torneio-NPGA $(i, i+1, Q)$.

Passo 3 Faça $i = i + 2$ e encontre o segundo pai, p_2 =Torneio-NPGA $(i, i+1, Q)$.

Passo 4 Faça o *crossover* entre $p1$ e $p2$ e crie os filhos c_1 e c_2. Faça mutação em c_1 e c_2.

Passo 5 Atualize a população de filhos $Q = Q \bigcup \{c_1, c_2\}$.

Passo 6 Faça $i = i + 1$. Se $i < N$ vá para o passo 2. Caso contrário, se $|Q| = \frac{N}{2}$, misture P, faça $i = 1$ e vá para o passo 2. Caso contrário, o processo está completo.

Além do parâmetro σ_{share}, o NPGA também é sensível ao valor de t_{dom}. Apesar dos autores não sugerirem nenhuma forma de determinar o valor de t_{dom}, seus experimentos mostraram que ele deve ser menor que o tamanho da população. Se o valor escolhido for muito pequeno o método pode não enfatizar satisfatoriamente as soluções não dominadas da população, por outro lado, se o valor é muito grande a complexidade computacional será muito alta.

Capítulo 5

Algoritmos Evolutivos para o Problema de Corte de Estoque Unidimensional

Em Araujo *et al.* [3] foi proposto um Algoritmo Evolutivo para o problema de corte de estoque unidimensional para a minimização simultânea da perda e do número de padrões diferentes. Além disso, o problema considerado tem vários objetos diferentes em estoque em quantidades limitadas (2.6)-(2.9). Araujo *et al.* [3] trataram este problema multiobjetivo transformando-o em um problema mono-objetivo pelo Método da Soma Ponderada, ou seja, associando-se pesos a cada objetivo.

Neste capítulo, inicialmente será apresentado o algoritmo proposto por Araujo *et al.* [3], considerando algumas adaptações para o caso onde se tem apenas um tipo de objeto em estoque (2.3)-(2.5) em quantidade ilimitada. Posteriormente, as idéias presentes no método proposto por Araujo *et al.* [3] são consideradas em conjunto com as idéias propostas no método VEGA (veja na Seção 4.3.1) resultando num método multiobjetivo. Por fim são consideradas as idéias presentes no método MOGA (Seção 4.4.1) juntamente com o método proposto por Araujo *et al.* [3].

5.1 Um Algoritmo Evolutivo: Proposta Inicial

Um indivíduo representa uma solução factível para o problema de corte de estoque, onde cada gene é representado por um padrão de corte a_j e o número de vezes que ele é cortado x_j, ou seja, o gene é dado por

$$\begin{pmatrix} a_{1j} \\ a_{2j} \\ \vdots \\ a_{mj} \\ x_j \end{pmatrix}. \tag{5.1}$$

Então, um indivíduo é representado como uma matriz onde a coluna representa o padrão de corte (e o número de vezes que ele é cortado) e o número de linhas é igual ao número de diferentes tipos de itens mais uma linha que representa o número de vezes que o padrão de corte é utilizado,

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1s} \\ a_{21} & a_{22} & \dots & a_{2s} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{ms} \\ x_1 & x_2 & \dots & x_s \end{pmatrix}. \tag{5.2}$$

5.1.1 População Inicial

A população inicial é um conjunto de várias soluções factíveis (conjunto de indivíduos), o tamanho desta população é um parâmetro a ser estabelecido no algoritmo evolutivo. Para geração da população inicial, dois aspectos foram considerados, o tempo computacional e a diversidade. Para construir o indivíduo foi utilizada a heurística de repetição exaustiva (*Repeat Exhaustion Reduction* RER) proposta por Hinxman [30] que pode ser resumida da seguinte maneira

Passo 1 Construa um padrão de corte;

Passo 2 Utilize este padrão de corte quantas vezes for possível, sem exceder a demanda;

Passo 3 A demanda é atualizada e o procedimento é repetido até que a demanda seja satisfeita (tem-se então uma solução factível inteira).

Dois métodos foram utilizados para se construir os padrões de corte no Passo 1.

- O primeiro utiliza a heurística FFD (*First Fit Decreasing Heuristic*), onde os itens são ordenados de forma não crescente em relação ao comprimento.
- O segundo utiliza um procedimento randômico para escolher os itens que farão parte do padrão de corte. Assim diferentes indivíduos podem ser construídos.

A seguir é esquematizado o primeiro método em detalhes.

Algoritmo 1: RER *Heuristic* Utilizando FFD (RER1)

Passo 1 Ordene os itens demandados em ordem não crescente de acordo com o comprimento. Sem perda de generalidade assume-se que $l_1 \geq l_2 \geq \ldots \geq l_m$.

Passo 2 Inicialize os seguinte parâmetros.

Passo 2.1 $r_i = d_i,\ i = 1, 2, \ldots, m$, (demanda residual para cada item).

Passo 2.2 $D = \sum_{i=1}^{m} r_i$, (D é a demanda residual total).

Passo 2.3 $L =$ comprimento do objeto em estoque.

Passo 2.4 $j = 1$, (primeiro padrão de corte).

Passo 3 Enquanto $D > 0$ (existe demanda residual)

Passo 3.1 Faça $i = 1$; Espaço $= L$.

Passo 3.2 Enquanto $i \leq m$ e Espaço $\geq l_m$ faça

Passo 3.2.1 $a_{ij} = min\left\{\left\lfloor \frac{\text{Espaço}}{l_i} \right\rfloor, r_i\right\}$.

Passo 3.2.2 Espaço $=$ Espaço $- a_{ij} * l_i$.

Passo 3.2.3 $i = i + 1$.

Passo 3.3 Determine o número de vezes que o padrão de corte deve ser utilizado

$$x_j = min\left\{\left\lfloor \frac{r_i}{a_{ij}} \right\rfloor,\ \text{onde}\ \ a_{ij} \neq 0,\ \text{para}\ i = 1, 2, \ldots, m\right\}.$$

Passo 3.4 Atualize a demanda

$$D = D - \sum_{i=1}^{m} x_j a_{ij},$$

$r_i = r_i - x_j a_{ij},\ i = 1, 2, \ldots, m$ e faça $j = j + 1$.

Para ilustrar o funcionamento desta heurística será considerado o seguinte exemplo que se encontrada entre os exemplares gerados para os testes computacionais.

Exemplo 5.1.1. *Neste exemplo será mostrado a criação de um indivíduo com a heurística RER1, a Tabela 5.1 apresenta a identificação de cada item, seu comprimento, sua demanda e a demanda residual inicial. Será considerado que o comprimento dos objetos em estoque é* 1000*.*

Os itens já se encontram ordenados de forma não crescente de acordo com o comprimento, a demanda residual total é $D = 100$ *(soma da demanda de cada item). Como a*

Item	Comprimento	Demanda	Demanda Residual
1	684	13	13
2	667	13	13
3	606	17	17
4	594	17	17
5	563	11	11
6	529	4	4
7	510	5	5
8	403	6	6
9	372	10	10
10	209	4	4

Tabela 5.1: Características dos Itens

demanda residual total não é nula primeiramente são inicializadas as variáveis Espaço $=$ 1000 *(que é o comprimento do objeto utilizado do qual podem ser cortados os itens),* $i = 1$ *(primeiro item a ser alocado - item atual) e* $j = 1$ *(primeiro padrão de corte).*

Como o último item ainda não foi atingido e ele ainda pode ser alocado no objeto, deve ser calculado o número de vezes que o item atual será cortado. Pela restrição do tamanho do item ele poderia ser cortado apenas uma vez, já pela restrição da demanda residual poderia ser cortado 13 vezes, portanto $a_{11} = 1$.

Agora é necessário a atualização do espaço disponível na barra, Espaço $= 316$*, agora o contador de itens deve ser incrementado, o próximo item a ser inserido (próximo item que cabe na barra) é o décimo, ou seja,* $i = 10$ *é o item atual. Este item pode ser alocado apenas uma vez na barra, pela restrição de espaço, então* $a_{10\,1} = 1$*. Desta forma está montado o primeiro padrão de corte do indivíduo, que produz uma vez o item 1 e uma vez o item 10 este padrão de corte pode ser utilizado apenas 4 vezes, pois o item 10 tem uma demanda de 4 unidades enquanto que o item 1 tem uma demanda de 13 unidades, ou seja,* $x_1 = 4$.

Agora a demanda total e a demanda residual devem ser atualizadas, $D = 92$*, na Tabela 5.2 é apresentada a nova demanda residual dos itens.*

Este processo deve ser repetido até que toda a demanda seja atendida.

Item	Comprimento	Demanda	Demanda Residual
1	684	13	9
2	667	13	13
3	606	17	17
4	594	17	17
5	563	11	11
6	529	4	4
7	510	5	5
8	403	6	6
9	372	10	10
10	209	4	0

Tabela 5.2: Dados do Problema após Utilização de um Padrão Gerado pela RER1

Com o método RER1 apenas um indivíduo será construído, o restante da população inicial será construída com o segundo método, que é uma adaptação do método anterior onde os itens atuais são escolhidos aleatoriamente para fazer parte do padrão de corte, desta forma vários indivíduos são construídos. Apenas o início do Passo 3 é modificado. A seguir é detalhado o procedimento modificado.

Algoritmo 2: Heurística Aleatória (RER2)

Passo 3 Enquanto $D > 0$ (existe demanda residual)

Passo 3.1 Sorteie um item i aleatoriamente; Faça Espaço $= L$.

Passo 3.2 Enquanto Espaço $\geq l_m$ faça

Passo 3.2.1 Se Espaço $\geq l_i$ faça;

Passo 3.2.1.1 $a_{ij} = min\left\{\left\lfloor \frac{\text{Espaço}}{l_i} \right\rfloor, r_i\right\}$.

Passo 3.2.1.2 Espaço $=$ Espaço $- a_{ij} * l_i$.

Passo 3.2.2 Sorteie um novo item i', mas este deve estar entre $i+1$ e m.

Passo 3.2.3 $i = i'$.

Passo 3.3 Determine o número de vezes que o padrão de corte deve ser utilizado

$$x_j = min\left\{\left\lfloor \frac{r_i}{a_{ij}} \right\rfloor, \text{ onde } a_{ij} \neq 0, \text{ para } i = 1, 2, \ldots, m\right\}.$$

Passo 3.4 Atualize a demanda

$$D = D - \sum_{i=1}^{m} x_j a_{ij},$$

$r_i = r_i - x_j a_{ij},\ i = 1, 2, \ldots, m$ e faça $j = j + 1$.

O tamanho da população inicial é uma parâmetro que deve ser definido no algoritmo. Pode-se observar também que a população inicial é ordenada de forma não decrescente de acordo com o valor do *fitness*.

Utilizaremos os mesmos dados do Exemplo 5.1.1 para ilustrar o funcionamento desta segunda heurística.

Exemplo 5.1.2. *As características deste exemplo encontram-se na Tabela 5.1. A demanda total deve ser calculada* $D = 100$ *e a variável* j *inicializada* ($j = 1$).

Como a demanda total não é zero um item atual deve ser sorteado, $i = 2$ *e a variável de espaço de ver inicializada* $Espaço = 1000$.

Como a variável espaço é maior que o comprimento do último item, deve ser calculada a quantidade de vezes que o item atual será cortado no padrão de corte que está sendo construído. Este item pode ser cortado apenas uma vez da barra e sua demanda é de 13 unidades, portanto $a_{21} = 1$, *e* $Espaço = 333$.

Um novo item deve ser sorteado, mas entre o terceiro e o último item, então $i = 4$, *mas o comprimento do item atual é* 594 *não podendo assim ser inserido neste padrão de corte. Desta forma um novo item é sorteado* $i = 9$, *como também não é possível alocá-lo neste padrão o próximo item sorteado é* $i = 10$.

O item atual pode ser cortado apenas uma vez, dessa forma $a_{10\,1} = 1$. *Assim está montado o primeiro padrão de corte do indivíduo, que produz uma vez o item 2 e uma vez o item 10 este padrão de corte pode ser utilizado apenas 4 vezes, pois o item 10 tem uma demanda de 4 unidades enquanto que o item 2 tem uma demanda de 13 unidades, ou seja,* $x_1 = 4$.

Agora a demanda total e a demanda residual devem ser atualizadas, $D = 92$, *na tabela 5.3 é apresentada a nova demanda residual dos itens.*

Este processo deve ser repetido até que toda a demanda seja atendida.

5.1.2 Reprodução ou Seleção

O processo de seleção é realizado para escolher um indivíduo que participará da geração de uma nova solução. A seleção de um indivíduo será de forma gulosa/aleatória, isto é, quanto melhor for o indivíduo (menor *fitness*) maior será a probabilidade dele ser escolhido. Para facilitar o processo de seleção, utiliza-se o fato de que a população está ordenada de forma

Item	Comprimento	Demanda	Demanda Residual
1	684	13	13
2	667	13	9
3	606	17	17
4	594	17	17
5	563	11	11
6	529	4	4
7	510	5	5
8	403	6	6
9	372	10	10
10	209	4	0

Tabela 5.3: Dados do Problema após Utilização de um Padrão Gerado pela RER2

não decrescente em relação ao *fitness* e escolhe um indivíduo pertencente a um intervalo com uma certa probabilidade. Esta probabilidade de seleção de um indivíduo é um dos parâmetros a serem determinados no algoritmo evolutivo.

5.1.3 *Crossover*

O processo de *crossover* é um pouco diferente do usual, já que geralmente dois pais são selecionados para gerar um ou dois filhos, até criar uma nova geração da população. No método desenvolvido por Araujo *et al.* [3], cada indivíduo é gerado por vários pais e após gerar um indivíduo, este já é inserido na população.

Para se criar um novo indivíduo, após o processo de seleção, um padrão de corte deste indivíduo selecionado é escolhido aleatoriamente e então tenta-se colocar este padrão de corte no indivíduo que está sendo criado. O processo de seleção é repetido novamente, escolhendo um novo indivíduo, um padrão de corte é escolhido aleatoriamente e inserido no novo indivíduo. Isto é feito repetidamente tomando-se o cuidado para que não se adicione um padrão de corte que contenha um item com a demanda já satisfeita. Este procedimento termina quando um parâmetro g é atingido, onde g é o número máximo de tentativas de inserir um padrão de corte em um indivíduo. Este parâmetro g deve ser previamente definido e é baseado nas características da população (considera-se a média do número de padrões de corte dos indivíduos da população mais uma certa porcentagem desta média). Se após g tentativas a demanda residual é nula, então tem-se um indivíduo factível. Caso contrário executa-se o processo de Adaptação ou Mutação, que consiste em

utilizar novamente a heurística RER1 considerando apenas a demanda residual.

5.1.4 Inserção do Novo Indivíduo na População

Caso o novo indivíduo seja melhor que a pior solução da população atual, então o pior indivíduo é eliminado e o novo é inserido na população. Como a população está ordenada de forma não decrescente o *fitness* do novo indivíduo é comparado com o restante da população, se for menor ou igual que alguma solução, o novo indivíduo entrará nesta posição "empurrando"todos os outros, eliminando assim o último indivíduo (pior *fitness*). Se o valor do *fitness* for igual, um critério de desempate é utilizado, que é o número de padrões de corte do indivíduo.

5.1.5 Critério de Parada

O critério de parada é mais um parâmetro a ser definido no algoritmo evolutivo. Este critério é o número máximo de iterações e cada iteração, representa a geração de um indivíduo. Se o critério de parada for atendido a solução final é determinada selecionando-se o primeiro elemento da população de indivíduos. O indivíduo que está na primeira posição será o indivíduo que possui o melhor *fitness* entre os demais. Caso não seja satisfeito o critério de parada o procedimento é repetido.

5.1.6 *Fitness*

O *fitness* utilizado em Araujo *et al.* [3] leva em consideração dois aspectos, o número de objetos cortados e o número de diferentes padrões de corte utilizados. O *fitness* pode ser usado para dar mais ênfase para algum objetivo específico, para isso utiliza-se os parâmetros β_1 e β_2. Estes parâmetros devem ser determinados no algoritmo evolutivo e ao determiná-los o problema multiobjetivo é transformado num problema mono-objetivo. Assim, o *fitness* é dado por

$$\beta_1 \frac{\sum_{i=1}^{n} x_i}{\rho_{MAX}} + \beta_2 \frac{\sum_{i=1}^{n} \delta(x_i)}{\rho_{MAX}}, \tag{5.3}$$

onde ρ_{MAX} é a demanda total do problema e normaliza as parcelas no intervalo $[0, 1]$.

5.1.7 Visão Geral do Algoritmo

Basicamente, no primeiro passo do algoritmo evolutivo, uma população inicial é gerada (apenas indivíduos factíveis). No segundo passo tem-se o processo de Seleção e também o *crossover*, os melhores indivíduos são selecionados com uma probabilidade pré-definida,

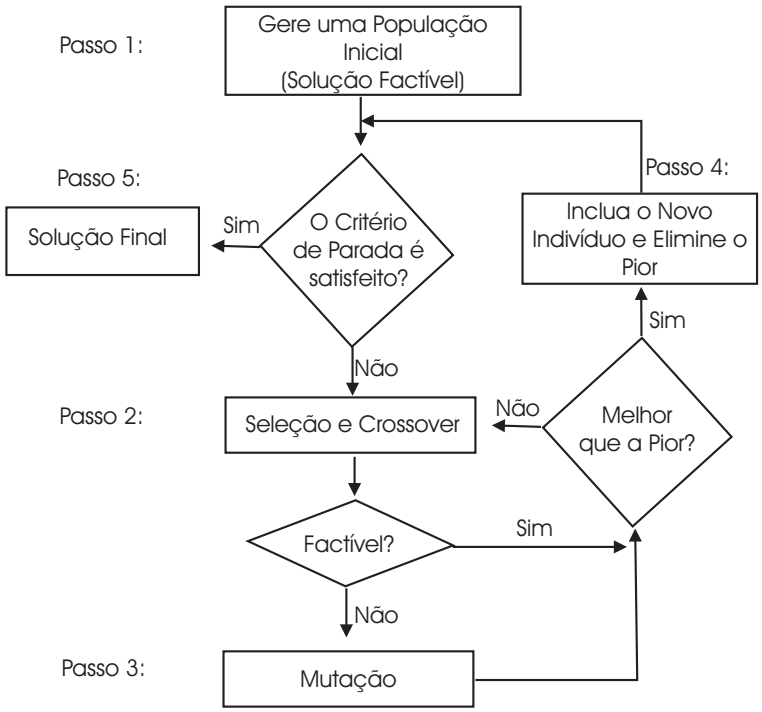


Figura 5.1: Visão Geral do Algoritmo

então as soluções selecionadas trocam informações entre si para que um novo indivíduo seja criado.

O novo indivíduo criado pode ser factível ou não, a factibilidade é checada e caso for negativa o indivíduo passará por um processo de mutação, tornando-se assim factível. Verifica-se então se a nova solução encontrada é melhor do que a pior solução da população, se for, ele será inserido na posição correspondente e o pior indivíduo será eliminado. Se o critério de parada não for satisfeito, o processo é repetido, caso contrário obtém-se uma solução final. A seguir são esquematizados os passos do algoritmo evolutivo.

Os seguintes parâmetros devem ser definidos para a aplicação do algoritmo: tamanho da população inicial; a probabilidade de escolha de indivíduos melhores para criação de novas soluções; número de tentativas de inserção de padrões de corte pelo algoritmo (g); critério de parada (número máximo de iterações) e os valores de β_1 e β_2 na função *fitness*.

No exemplo a seguir o funcionamento do algoritmo é ilustrado.

Exemplo 5.1.3. *Os dados deste exemplo são os mesmos dos exemplos anteriores e podem ser visualizados na Tabela 5.1. Na tabela 5.4 é mostrada a população inicial de 10 indivíduos criados pela heurística RER1 e RER2, já ordenados em relação ao fitness. Este fitness foi calculado utilizando os valores de* $\beta_1 = 0,5$ *e* $\beta_2 = 0,5$ *transformando o problema multiobjetivo em um problema mono-objetivo. Na segunda coluna é apresentado o número de objetos cortados* (x_i) *e na terceira coluna o número de diferentes padrões de*

corte ($\delta(x_i)$).

Indivíduo	x_i	$\delta(x_i)$	*Fitness*
1	80	10	0,450000018
2	80	10	0,450000018
3	80	10	0,450000018
4	82	10	0,460000008
5	83	10	0,465000004
6	84	10	0,469999999
7	85	10	0,475000024
8	85	10	0,475000024
9	87	10	0,485000014
10	88	10	0,49000001

Tabela 5.4: População Inicial

O próximo passo é verificar o critério de parada, que obviamente não foi atingido (já que esta é a primeira iteração), devemos então passar para o próximo passo que é a Seleção e o Crossover, após este processo um novo indivíduo será criado.

O indivíduo criado apresenta um número de objetos cortados de 68 e um número de diferentes padrões de corte de 7. O próximo passo é a verificação da factibilidade, deve ser verificado se a demanda residual é nula o que não ocorre neste caso, portanto deve ser realizado o processo de Mutação que utiliza a heurística RER1, levando em consideração somente a demanda residual.

Após este processo o indivíduo criado apresenta 84 objetos cortados, 10 diferentes padrões de corte e 0,469999999 de fitness. Este novo indivíduo será inserido na sexta posição eliminando o décimo indivíduo.

Na Tabela 5.5 é apresentada a solução do problema após 1500 iterações.

Indivíduo	x_i	$\delta(x_i)$	*Fitness*
1	80	9	0,444999993

Tabela 5.5: Única Solução Final após 1500 Iterações Utilizando $\beta_1 = 0,5$ e $\beta_2 = 0,5$

5.2 Incorporação das Idéias Presentes no VEGA

Para tratar o problema de corte de estoque unidimensional inteiro com objetivos conflitantes (número de objetos cortados e número de diferentes padrões de corte) como um problema multiobjetivo, foi incorporado ao algoritmo de Araujo *et al.* [3] algumas idéias presentes no método VEGA (Seção 4.3.1), com o intuito de obter um conjunto de soluções (não dominadas) para que o decisor possa escolher a que melhor lhe convém no momento.

A população inicial será criada da mesma forma, com as heurísticas RER1 e RER2, mas após a criação esta população é dividida em duas subpopulações, uma que será ordenada de forma não decrescente de acordo com o número de objetos cortados, sendo o número de diferentes padrões de corte o critério de desempate (subpopulação 1). E outra ordenada também de forma não decrescente mas de acordo com o número de diferentes padrões de corte, similarmente, tendo o número de objetos cortados como critério de desempate (subpopulação 2).

Para o processo de seleção primeiramente uma subpopulação é escolhida aleatoriamente, desta subpopulação é então selecionado um indivíduo de forma gulosa/aleatória, ou seja, a probabilidade de escolha será maior para indivíduos melhores.

Após a seleção, passa-se para o *crossover*. Do indivíduo selecionado é escolhido um padrão de corte aleatoriamente, este padrão de corte é inserido no indivíduo que será criado. O processo de seleção é então repetido selecionando-se outro indivíduo e mais um padrão de corte é colocado no indivíduo que está sendo criado. Este procedimento se repete até que o parâmetro g seja atingido, sempre tomando-se o cuidado para que não se escolha um padrão de corte que contenha um item com demanda já satisfeita.

Caso obtenha um indivíduo com demanda residual não nula, ou seja um indivíduo infactível, é necessário que ele passe por um processo de mutação, onde será usada novamente a heurística RER1 considerando apenas a demanda residual.

De posse do novo indivíduo deve-se decidir em qual população ele será inserido, isto é feito da seguinte forma. Primeiramente este indivíduo é comparado com os indivíduos da subpopulação 1, que está ordenada de forma não decrescente em relação ao número de objetos cortados e tendo o número de diferentes padrões de corte como desempate, verifica-se qual seria sua posição nesta subpopulação. Em seguida ele é comparado com os indivíduos da subpopulação 2 que está em ordenada em ordem não decrescente em relação ao número de diferentes padrões de corte e tendo como critério de desempate o número de objetos cortados. Compara-se as duas posições, na subpopulação que o novo indivíduo tiver uma posição melhor (menor) ele será inserido, eliminando o pior (último) indivíduo desta subpopulação, em caso de empate a subpopulação é escolhida aleatoriamente. Com o objetivo de analisar a eficiência e o impacto deste procedimento no tempo de execução do problema, um processo alternativo foi utilizado, em que a população que o indivíduo

deverá ser inserido é escolhido de forma aleatória.

O critério de parada utilizado é novamente o número máximo de gerações permitidas. Quando este critério é atingido o algoritmo identifica em cada subpopulação as soluções não dominadas, e estas são apresentadas ao decisor.

Neste algoritmo são utilizadas duas funções *fitness*, uma para cada subpopulação, que consistem em cada uma das funções objetivo. Na subpopulação 1 é utilizada a função objetivo que retorna o número de objetos cortados, e na subpopulação 2 a que retorna o número de diferentes padrões de corte. Esta é uma vantagem deste algoritmo, pois, toda solução já tem o número de objetos cortados e o número de diferentes padrões de corte, não sendo necessário um cálculo de uma nova função de avaliação, acabando também com a necessidade da normalização da função *fitness*, pois, objetivos diferentes nunca são comparados.

Resumidamente, no primeiro passo do algoritmo, uma população inicial é gerada (apenas indivíduos factíveis), esta população é então dividida em duas subpopulações. Depois, no segundo passo tem-se o processo de Seleção e também o *crossover*, os melhores indivíduos são selecionados com uma probabilidade pré-definida, então as soluções selecionadas trocam informações entre si para que um novo indivíduo seja criado.

O novo indivíduo criado pode ser factível ou não, a factibilidade é checada e caso for negativa o indivíduo passará por um processo de mutação, tornando-se assim factível. Verifica-se então se a nova solução encontrada é melhor do que a pior solução de cada subpopulação, se for, ele será inserido na posição correspondente na subpopulação onde estiver melhor posicionada (alternativamente de forma aleatória) e o pior indivíduo será eliminado. Se o critério de parada não for satisfeito, o processo é repetido, caso contrário o algoritmo determina em cada subpopulação as soluções não dominadas e estas são apresentadas ao decisor como as soluções finais.

Uma variação deste algoritmo também foi implementada, já que muitas vezes as soluções não dominadas apresentam o mesmo número de objetos cortados e o mesmo número de diferentes padrões de corte, apesar de serem soluções diferentes pois apresentam padrões de corte diferentes, bem como, suas freqüências. Nesta variação quando o indivíduo é inserido em uma subpopulação, se ele apresentar o mesmo número de objetos cortados e o mesmo número de diferentes padrões de corte que algum indivíduo da subpopulação onde será inserido ele então substitui este indivíduo.

A seguir é apresentado um exemplo para ilustrar o funcionamento do algoritmo, a configuração escolhida é a que compara em qual subpopulação o indivíduo deve ser inserido e que substitui o indivíduo caso apresente o mesmo número de objetos cortados e de diferentes padrões de corte.

Exemplo 5.2.1. *Os dados utilizados neste exemplo encontram-se na Tabela 5.1. Pri-*

meiramente a população inicial é criada utilizando-se as heurísticas RER1 e RER2, e divididos aleatoriamente em duas subpopulações, que são apresentadas na Tabela 5.6.

	Subpopulação 1		Subpopulação 2	
Indivíduo	x_i	$\delta(x_i)$	x_i	$\delta(x_i)$
1	80	10	80	10
2	84	10	80	10
3	85	10	82	10
4	87	10	83	10
5	88	10	85	10

Tabela 5.6: Subpopulações Iniciais para o VEGA

A subpopulação 1 está ordenada de forma não decrescente pelo número de objetos cortados, sendo o número de diferentes padrões de corte o critério de desempate. A subpopulação 2 está ordenada também de forma não decrescente mas de acordo com o número de diferentes padrões de corte, similarmente, tendo o número de objetos cortados como critério de desempate

O critério de parada não é satisfeita, pois, esta é a primeira iteração, então é feita a Seleção e o Crossover para a construção de um novo indivíduo. O novo indivíduo criado neste exemplo apresenta o número de 64 objetos cortados e o número de 6 diferentes padrões de corte, este indivíduo não é factível pois a demanda ainda não foi totalmente atingida. Deve ser então realizado o processo de Mutação.

Após o processo de Mutação o indivíduo criado apresenta o número de 84 objetos cortados e 11 diferentes padrões de corte. Analisando em que subpopulação o indivíduo deve ser inserido deve-se analisar a posição em que ele seria inserido em cada subpopulação. Analisando a subpopulação 1 percebe-se que o novo indivíduo empata com o segundo indivíduo no número de objetos cortados, deve-se então analisar o número de diferentes padrões de corte e dessa forma o novo indivíduo deveria ser inserido na posição 3 da subpopulação 1 eliminando o quinto indivíduo. Analisando a subpopulação 2 percebe-se que o número de diferentes padrões de corte do novo indivíduo é pior que o quinto indivíduo, desta forma ele não pode ser inserido nesta subpopulação. Conclui-se então que o novo indivíduo deve ser inserido na subpopulação 1.

Na Tabela 5.7 são apresentadas as duas subpopulações finais do problema após 1500

iterações.

	Subpopulação 1		Subpopulação 2	
Indivíduo	x_i	$\delta(x_i)$	x_i	$\delta(x_i)$
1	80	8	80	8
2	80	9	80	9
3	80	10	81	9
4	80	11	84	9
5	81	10	90	9

Tabela 5.7: Dez Soluções Finais após 1500 Iterações

Analisando estas duas subpopulações como um todo percebe-se que existem duas soluções não dominadas, uma em cada subpopulação, apesar das duas soluções apresentam o mesmo número de objetos cortados e o mesmo número de diferentes padrões de corte, elas são diferentes, ou seja, os padrões utilizados são diferentes.

5.3 Incorporação das Idéias Presentes no MOGA

Neste método a população inicial também é criada através das heurísticas RER1 e RER2, e então o procedimento de atribuição de fitness é feito da mesma forma do MOGA (Seção 4.4.1), utilizando o procedimento do *fitness sharing*.

Primeiramente devem ser calculados o *rank* de cada solução, que é o número de soluções que as dominam mais um, priorizando assim as soluções não dominadas. Em seguida um *fitness* é atribuído para todas as soluções, de forma que as soluções de mesmo *rank* possuam obrigatoriamente o mesmo *fitness*. O próximo passo é o cálculo da medida *niche* e do *shared fitness* que é utilizado para a classificação da população.

A população é ordenada de forma não crescente em relação ao *shared fitness* de cada indivíduo. A escolha das soluções que contribuirão para a construção de um novo indivíduo é feita utilizando o método da roleta, em que a probabilidade da escolha de cada solução é baseada no seu *shared fitness*.

Novamente quando um indivíduo é sorteado, um padrão de corte é escolhido aleatoriamente para fazer parte do novo indivíduo, isto é repetido até que um parâmetro g seja atingido. Caso obtenha um indivíduo com demanda residual não nula, ou seja um indivíduo infactível, é necessário que ele passe por um processo de mutação, em que será usada novamente a heurística RER1 considerando apenas a demanda residual.

Neste momento tem-se um total de $N + 1$ indivíduos, para que se determine em que posição o novo indivíduo deve ser inserido. Novamente deve-se fazer o procedimento de *fitness sharing* nos $N + 1$ indivíduos. Em seguida a população é ordenada não crescentemente em relação a *shared fitness* e o último indivíduo é eliminado.

O critério de parada do algoritmo é novamente o número de iterações.

Para ilustrar o funcionamento do algoritmo será utilizado o exemplo abaixo. Maiores detalhes do método e algumas notações estão presentes na Seção 4.4.1

Exemplo 5.3.1. *Os dados utilizados neste exemplo podem ser visualizados na Tabela 5.1. A população inicial é criada utilizando novamente as heurísticas RER1 e RER2, e então deve ser realizado o processo de fitness sharing, dessa forma o rank de cada solução deve ser calculado. O rank de cada solução é o número de soluções que a dominam mais um. A Tabela 5.8 mostra o rank da população inicial.*

Indivíduo	x_i	$\delta(x_i)$	τ_i
1	80	10	1
2	80	10	1
3	80	10	1
4	82	10	4
5	83	10	5
6	84	10	6
7	85	10	7
8	85	10	7
9	87	10	9
10	88	10	10

Tabela 5.8: *Rank* da População Inicial

Outra informação necessária é a quantidade de soluções por rank. Observa-se por exemplo na Tabela 5.8 que existem três soluções de rank 1 ($\tau_i = 1$), ou seja, $\mu(1) = 3$ e nenhuma solução de rank 2 ($\tau_i = 2$), ou seja, $\mu(2) = 0$.

Agora deve ser calculado o fitness para cada solução utilizando a equação (4.10), de forma que as soluções de mesmo rank possuam o mesmo fitness. Por exemplo, calculando o fitness do indivíduo 7 ou 8, todas as soluções de rank 7 terão o seguinte fitness

$$F_7 = 10 - \sum_{k=1}^{6} \mu(k) - 0,5(2-1) \tag{5.4}$$

$$F_7 = 10 - 3 - 0 - 0 - 1 - 1 - 1 - 0,5 \tag{5.5}$$

$$F_7 = 3,5. \tag{5.6}$$

O próximo passo é o cálculo da medida niche (ηc_i), para isso deve-se calcular a função sharing ($Sh(d_{ij})$) e conseqüentemente a distância (d_{ij}) entre todas as soluções de mesmo rank.

Para o cálculo da distância entre as soluções (inclusive dela mesma que será zero) primeiramente deve-se calcular os limitantes inferiores e superiores para cada função objetivo, neste caso $\mu_1 = 88$, $l_1 = 80$, $\mu_2 = 10$ e $l_2 = 10$. Note que os limitantes da segunda função objetivo são iguais, neste caso, esta parcela assume o valor 0 no somatório. A seguir utiliza-se a equação (4.16) para o cálculo da distância. Neste exemplo a distância entre quaisquer duas soluções de mesmo rank é zero, pois seus valores são os mesmos. Considere por exemplo o cálculo da distância entre as soluções 1 e 2.

$$d_{12} = \sqrt{\left(\frac{80-80}{88-80}\right)^2 + 0} = 0. \tag{5.7}$$

Agora deve-se calcular a função de sharing para todas as distâncias calculadas utilizando-se a equação (4.1). Como todas as distâncias são zero, qualquer função sharing assumirá o valor 1.

O próximo passo é o cálculo da medida niche de cada solução utilizando a equação (4.9), que é calculada somando-se o valor da função sharing de todas as distâncias da solução em questão em relação às soluções de mesmo rank. Para exemplificar será calculado a medida niche para a solução 7.

$$\eta c_7 = \sum_{j=1}^{\mu(\tau_7)=2} Sh(d_{ij}) = Sh(d_{77}) + Sh(d_{78}) = 2. \tag{5.8}$$

Para o cálculo do shared fitness deve-se dividir o fitness obtido pela equação (4.10) pela medida niche, para o sétimo indivíduo tem-se

$$F'_7 = F_7/\eta c_7 = 3,5/2 = 1,75.$$

Observe que, para a solução 7 tem-se $F_7 = 3,5$ *e* $F'_7 = 1,75$*, para a solução 1 tem-se* $F_1 = 9$ *e* $F'_1 = 3$*. Para que seja mantido o fitness médio da população, deve-se utilizar a equação (4.11). A seguir será calculado, por exemplo o novo shared fitness das soluções 1 e 7.*

$$F'_1 \leftarrow \frac{F_1 \mu(1)}{\sum_{k=1}^{\mu(1)} F'_k} F'_1 \Rightarrow F'_1 \leftarrow \frac{9 * 3}{9} * 3 = 9.$$

$$F'_7 \leftarrow \frac{F_7 \mu(7)}{\sum_{k=1}^{\mu(7)} F'_k} F'_7 \Rightarrow F'_7 \leftarrow \frac{3,5 * 2}{3,5} * 1,75 = 3,5.$$

Pode-se perceber que este procedimento mantém o fitness médio original da população, neste caso os valores do fitness são restaurados por influência da função sharing, pois as soluções de mesmo rank são iguais.

Tem-se então calculado o shared fitness e a população é então ordenada de forma não crescente em relação à ele, a Tabela 5.9 mostra a população inicial criada e seu respectivo shared fitness.

Indivíduo	x_i	$\delta(x_i)$	*Shared Fitness*
1	80	10	9
2	80	10	9
3	80	10	9
4	82	10	7
5	83	10	6
6	84	10	5
7	85	10	3,5
8	85	10	3,5
9	87	10	2
10	88	10	1

Tabela 5.9: *Shared Fitness* da População Inicial

Passa-se então para o processo de Seleção e Crossover onde um novo indivíduo é criado, ele apresenta o número de 73 objetos cortados e 7 diferentes padrões de corte, porém este indivíduo não é factível, devendo passar pelo processo de Mutação.

Após o processo de Mutação o indivíduo apresenta o número de 83 objetos cortados e 10 diferentes padrões de corte, agora a população é de 11 indivíduos, deve-se realizar novamente o processo de fitness sharing para reclassificar a população e eliminar o último indivíduo. A Tabela 5.10 mostra os 11 indivíduos reclassificados.

Indivíduo	x_i	$\delta(x_i)$	*Shared Fitness*
1	80	10	10
2	80	10	10
3	80	10	10
4	82	10	8
5	83	10	6,5
6	84	10	5
7	85	10	3,5
8	85	10	3,5
9	87	10	2
10	88	10	1
NOVO	83	10	6,5

Tabela 5.10: População Reclassificada

Dessa forma o novo indivíduo entrará na quinta posição, eliminando o décimo indivíduo. A Tabela 5.11 apresenta a população final após 1500 iterações.

Percebe-se neste exemplo que a população convergiu para uma única solução.

Indivíduo	x_i	$\delta(x_i)$	*Shared Fitness*
1	80	9	6,5
2	80	9	6,5
3	80	9	6,5
4	80	9	6,5
5	80	9	6,5
6	80	9	6,5
7	80	9	6,5
8	80	9	6,5
9	80	9	6,5
10	80	9	6,5

Tabela 5.11: Dez Soluções Finais após 1500 Iterações

Capítulo 6

Resultados Computacionais

Neste capítulo, os resultados computacionais dos três métodos descritos no Capítulo 5 são comparados. Inicialmente, é apresentado o gerador aleatório utilizado para geração dos dados avaliados e tem-se uma descrição dos parâmetros utilizados em cada método.

Os algoritmos evolutivos apresentados foram desenvolvidos em linguagem C na ferramenta Bloodsheed Dev-C++ 4.9.9.2. Os testes computacionais foram realizados utilizando um computador com processador AMD Athlon (64 X2 Dual Core Processor 2.81 GHz) com 1.87 GB de memória RAM.

6.1 Geração dos Dados

O conjunto de dados utilizados nos testes computacionais foram gerados de acordo com o gerador aleatório CUTGEN1 proposto por Gau and Wäscher [22]. Dezoito classes com instâncias aleatórias foram geradas, assim como em Foerster e Wäscher [19], Umetani *et al.* [50] e em Yanasse e Limeira [54]. As classes combinam diferentes valores dos parâmetros m (diferentes tipos de itens) que assume valores 10, 20 ou 40, além dos parâmetros *dbar* que é a demanda média $d_i, i = 1, 2, \ldots, m$ de cada item e v_1 e v_2. O tamanho dos objetos em estoque foi considerado como sendo 1000, e o comprimento dos itens l_i foi gerado aleatoriamente no intervalo $[v_1 L, v_2 L]$.

Para cada classe foram geradas e resolvidas 100 instâncias, totalizando 1800 problemas que foram gerados de acordo com as características da tabela 6.1.

6.2 Parâmetros

Nos algoritmos desenvolvidos, é necessário que alguns parâmetros sejam fixados.

Na primeira adaptação realizada no algoritmo evolutivo de Araujo *et al.* [3] em que foi

Classe	m	v_1	v_2	dbar
1	10	0.01	0.2	10
2	10	0.01	0.2	100
3	20	0.01	0.2	10
4	20	0.01	0.2	100
5	40	0.01	0.2	10
6	40	0.01	0.2	100
7	10	0.01	0.8	10
8	10	0.01	0.8	100
9	20	0.01	0.8	10
10	20	0.01	0.8	100
11	40	0.01	0.8	10
12	40	0.01	0.8	100
13	10	0.2	0.8	10
14	10	0.2	0.8	100
15	20	0.2	0.8	10
16	20	0.2	0.8	100
17	40	0.2	0.8	10
18	40	0.2	0.8	100

Tabela 6.1: Características das Classes

utilizado o método da soma ponderada, nos testes computacionais os parâmetros foram fixados da seguinte maneira.

- A população inicial foi fixada em 10 ($N = 10$).
- O número máximo de iterações foi fixado em 1500.
- O percentual para a escolha dos indivíduos, ou seja, a probabilidade de que os indivíduos sejam escolhidos entre os 50% melhores foi de 50%.
- O parâmetro g que é definido como sendo o número máximo de tentativas de inserir um padrão de corte em um indivíduo, considera a média do número de padrões de corte dos indivíduos mais uma porcentagem dessa média, que foi definida em 20%, ou seja $g = 1,2$.

- Os parâmetros β_1 e β_2 variam em cada teste como será mostrado posteriormente.

No segundo algoritmo desenvolvido, quando são inseridas as idéias presentes no VEGA, os parâmetros foram utilizados da seguinte maneira.

- A população inicial foi fixada em 10 e 20.
- O número máximo de iterações foi fixado em 1500, 3000, 5000 e 10000.
- O percentual foi de 50%.
- O parâmetro g foi definido como sendo $g = 1, 2$.

No terceiro algoritmo desenvolvido, quando são inseridas as idéias presentes no MOGA, os seguintes parâmetros foram utilizados.

- A população inicial foi fixada em 10 e 20.
- O número máximo de iterações foi fixado em 1500, 3000 e 5000.
- O parâmetro g foi definido como sendo $g = 1, 2$.
- O parâmetro σ_{share} foi definido como sendo $\sigma_{share} = \dfrac{2}{(N-1)}$, como proposto por Fonseca e Fleming [21].

6.3 Resultados

Na Tabela 6.2 são apresentados a média dos limitantes inferiores para cada classe, assim como os resultados obtidos por Yanasse e Limeira [54], em que $\overline{x}_j$ representa a média do número de objetos cortados de cada classe e $\overline{\delta}(x_j)$ a média do número de diferentes padrões de corte para cada classe.

6.3.1 Resultados Algoritmo Evolutivo Utilizando Método Soma Ponderada

Os primeiros resultados apresentados nas Tabelas 6.3 e 6.4 são referentes a adaptação do algoritmo apresentado por Araujo *et al.* [3] utilizando as idéias do método da soma ponderada. Em cada simulação do algoritmo foram utilizados diferentes valores para β_1 e β_2 para que fosse verificada a influência destes parâmetros bem como para a comparação com os algoritmos multiobjetivo desenvolvidos.

	Limitantes		Yanasse	
Classe	$\overline{x}_j$	$\overline{\delta}(x_j)$	$\overline{x}_j$	$\overline{\delta}(x_j)$
1	10,97	1,66	11,56	3,31
2	109,73	1,66	110,40	6,95
3	21,57	2,55	22,17	4,96
4	215,39	2,55	215,98	10,32
5	42,48	4,25	42,99	7,63
6	424,21	4,25	424,89	13,31
7	49,95	4,99	51,69	7,66
8	499,33	4,99	502,23	9,62
9	93,34	9,24	99,49	13,64
10	931,95	9,24	948,41	18,21
11	176,59	16,92	195,67	24,60
12	1763,18	16,92	1847,42	33,23
13	63,22	6,67	64,20	8,93
14	632,09	6,67	633,26	10,51
15	119,35	11,76	123,90	16,28
16	1191,75	11,76	1197,66	19,89
17	224,55	21,98	244,02	29,76
18	2242,29	21,98	2268,30	37,90
Média	489,55	8,89	500,24	15,37

Tabela 6.2: Limitantes Inferiores

Como nos próximos algoritmos multiobjetivo o resultado é uma população de indivíduos, para que no primeiro algoritmo desenvolvido também fosse obtida uma população de indivíduos, os parâmetros variam no intervalo $[0, 1]$ com um incremento de $0,1$ sempre com $\beta_1 + \beta_2 = 1$, resultando em um total de 11 simulações, o que pode ser entendido com tendo uma população final de 11 indivíduos.

As soluções não dominadas (em relação às soluções das Tabelas 6.3 e 6.4) em cada classe são apresentadas em negrito, o tempo é apresentado em segundos, pode ser observado que o tempo computacional de cada simulação não é grande, mas em cada simulação apenas um indivíduo é a solução do problema, para um total de 11 indivíduos são necessárias as 11 simulações com a variação dos parâmetros resultando em um tempo de

	S1 $\beta_1 = 1,0\,\beta_2 = 0$		S2 $\beta_1 = 0,9\,\beta_2 = 0,1$		S3 $\beta_1 = 0,8\,\beta_2 = 0,2$		S4 $\beta_1 = 0,7\,\beta_2 = 0,3$		S5 $\beta_1 = 0,6\,\beta_2 = 0,4$	
Classe	$\overline{x}_j$	$\overline{\delta}(x_j)$	$\overline{x}_j$	$\overline{\delta}(x_j)$	$\overline{x}_j$	$\overline{\delta}(x_j)$	$\overline{x}_j$	$\overline{\delta}(x_j)$	$\overline{x}_j$	$\overline{\delta}(x_j)$
1	11,57	8,02	11,58	6,4	11,58	6,41	11,58	6,37	**11,57**	**6,35**
2	**110,86**	**13,26**	**110,91**	**10,39**	**111,05**	**10,11**	**111,06**	**10**	**111,13**	**9,84**
3	22,19	15,46	**22,18**	**11,14**	**22,17**	**11,21**	22,2	11,46	**22,2**	**11,1**
4	216,81	26,7	**216,8**	**22,51**	**216,94**	**21,86**	**217,07**	**21,63**	**217,21**	**21,4**
5	43,08	29,42	**43,07**	**22,11**	43,09	22,26	43,08	22,14	43,09	22,19
6	426,18	54,15	**426,18**	**49,68**	**426,35**	**48,2**	**426,49**	**47,15**	**426,51**	**46,98**
7	**50,27**	**9,65**	**50,29**	**8,17**	**50,29**	**8,17**	**50,28**	**8,19**	**50,34**	**8,15**
8	500,28	11,12	500,27	9,93	500,24	9,96	500,3	9,96	**500,3**	**9,78**
9	**93,8**	**18,39**	**93,82**	**14,93**	93,87	15,01	93,85	14,94	**93,98**	**14,81**
10	**934,11**	**22**	**934,19**	**20,22**	934,46	19,87	**934,4**	**19,94**	**934,41**	**19,77**
11	177,35	36,66	**177,35**	**30,86**	**177,43**	**30,39**	177,47	30,41	**177,59**	**29,78**
12	1768,6	42,93	1768,7	40,77	1768,64	40,6	1768,75	40,04	**1768,35**	**40,02**
13	**63,51**	**9,84**	63,55	9,15	**63,53**	**9,12**	**63,55**	**9,1**	**63,55**	**9,1**
14	**632,73**	**10,71**	632,93	10,34	632,93	10,31	632,95	10,31	633,03	10,27
15	**119,79**	**19,04**	**119,83**	**16,59**	**119,81**	**16,61**	**119,85**	**16,57**	**119,88**	**16,48**
16	1193,84	20,77	1193,8	19,67	1193,65	19,68	1193,76	19,67	**1193,52**	**19,67**
17	225,16	36,75	225,14	31,87	**225,13**	**31,83**	**225,23**	**31,74**	**225,3**	**31,55**
18	2246,53	40,54	2246,5	38,77	2246,72	38,69	2246,5	38,48	2246,49	38,55
Média	490,93	23,63	490,95	20,75	490,99	20,57	491,02	20,45	491,03	20,32
Tempo	561,5 s		499,547 s		503,812 s		517,391 s		500,062 s	

Tabela 6.3: Resultados Algoritmo Evolutivo Parte 1

5565, 76 segundos.

Analisando as Tabelas 6.3 e 6.4 pode-se perceber que entre as simulações realizadas a que possui maior número de indivíduos não dominados são as simulações S5 e S6 (em 15 das 18 classes) que utilizam os parâmetros $\beta_1 = 0,6$, $\beta_2 = 0,4$ e $\beta_1 = 0,5$, $\beta_2 = 0,5$ respectivamente, podendo-se concluir que quando a importância entre os dois objetivos é balanceada, indivíduos melhores (em relação a dominância) são criados. Se forem com-

	S6 $\beta_1 = 0,5\, \beta_2 = 0,5$		S7 $\beta_1 = 0,4\, \beta_2 = 0,6$		S8 $\beta_1 = 0,3\, \beta_2 = 0,7$		S9 $\beta_1 = 0,2\, \beta_2 = 0,8$		S10 $\beta_1 = 0,1\, \beta_2 = 0,9$		S11 $\beta_1 = 0\, \beta_2 = 1$	
Classe	$\overline{x}_j$	$\overline{\delta}(x_j)$	$\overline{x}_j$	$\overline{\delta}(x_j)$	$\overline{x}_j$	$\overline{\delta}(x_j)$	$\overline{x}_j$	$\overline{\delta}(x_j)$	$\overline{x}_j$	$\overline{\delta}(x_j)$	$\overline{x}_j$	$\overline{\delta}(x_j)$
1	**11,59**	**6,33**	11,6	6,36	11,65	6,4	11,68	6,36	11,8	6,35	14,11	6,35
2	111,33	9,92	**111,44**	**9,71**	111,51	9,82	**111,9**	**9,63**	112,02	9,68	**162,91**	**9,61**
3	22,18	11,33	22,18	11,22	22,19	11,37	22,22	11,22	22,2	11,18	23,68	11,2
4	**217,42**	**21,28**	**217,71**	**20,96**	217,79	21,03	217,95	21,02	**218,37**	**20,94**	267,33	20,98
5	43,09	22,07	43,11	22	**43,11**	**21,8**	43,11	21,92	**43,09**	**21,92**	44,39	21,96
6	**426,75**	**45,61**	**426,97**	**45,46**	**427,15**	**44,62**	427,38	44,77	427,85	44,62	**498,65**	**44,47**
7	**50,45**	**7,97**	**50,67**	**7,95**	**50,75**	**7,88**	**50,79**	**7,89**	**51,7**	**7,77**	54,87	7,77
8	**500,23**	**9,85**	**500,44**	**9,71**	**500,63**	**9,69**	**500,77**	**9,63**	**502,03**	**9,52**	**564,72**	**9,29**
9	**94,09**	**14,78**	**94,35**	**14,58**	**94,61**	**14,48**	**94,86**	**14,44**	**95,51**	**14,19**	99,89	14,34
10	**934,75**	**19,56**	**934,83**	**19,39**	**935,43**	**19**	**935,94**	**18,97**	**938,05**	**18,62**	**1038,31**	**18,24**
11	**177,86**	**29,64**	178,01	29,78	**178,34**	**29,51**	**178,68**	**29,28**	**179,38**	**29,27**	184,11	29,57
12	**1768,99**	**39,55**	**1769,46**	**39,14**	**1769,92**	**38,82**	**1771,31**	**38,56**	**1773,98**	**38,27**	**1884,11**	**37,46**
13	**63,68**	**9,03**	**63,77**	**8,89**	**63,95**	**8,85**	**64,2**	**8,77**	64,43	8,77	**68,36**	**8,59**
14	**632,92**	**10,21**	**633,14**	**10,16**	633,24	10,15	**633,15**	**10,07**	**634,68**	**9,92**	**715,9**	**9,65**
15	**120,03**	**16,38**	**120,24**	**16,17**	**120,57**	**16,03**	120,77	16,11	**121,78**	**15,81**	126,43	15,83
16	**1193,87**	**19,5**	**1193,99**	**19,37**	**1194,06**	**19,28**	**1194,64**	**19,14**	**1197,22**	**18,79**	**1303,18**	**18,25**
17	**225,44**	**31,33**	**225,67**	**30,98**	**226,06**	**30,64**	**226,47**	**30,59**	**227,32**	**30,38**	**233,65**	**30,35**
18	**2246,21**	**38,34**	2246,7	38,37	**2247,17**	**38,15**	**2247,57**	**37,64**	**2250,74**	**37,22**	**2406,19**	**36,36**
Média	491,16	20,15	491,35	20,01	491,56	19,86	491,86	19,78	492,90	19,62	538,38	19,46
Tempo	502,625 s		493,734 s		486,765 s		490,375 s		479,86 s		530,093 s	

Tabela 6.4: Resultados Algoritmo Evolutivo Parte 2

paradas as médias destas simulações com a média dos limitantes inferiores da Tabela 6.2, pode ser observado que o número de objetos cortados está muito próximo do limitante. Se forem ainda comparadas com os resultados obtidos por Yanasse e Limeira [54] também presentes na Tabela 6.2 pode-se observar que o número de objetos cortados do algoritmo proposto é melhor, o que não ocorre com o número de diferentes padrões de corte, isso se deve ao fato de que no algoritmo de Yanasse e Limeira [54] é permitido o excesso de produção, o que não é permitido no algoritmo proposto. Ainda a média das soluções de quase todas as simulações são não dominadas pela média das soluções do algoritmo de Yanasse e Limeira [54], a única média dominada é a da simulação S11.

Quando se deseja enfatizar um único objetivo (obrigatoriamente o outro é sacrificado), como era de se esperar, as simulações que dão peso total para apenas um único objetivo apresentam melhores resultados (no objetivo em questão). No número de objetos cortados (x_i) a simulação que apresenta melhor resultado é a S1, já no número de diferentes padrões de corte ($\delta(x_i)$) é a S11.

6.3.2 Resultados VEGA

Nas Tabelas 6.5 e 6.6 são apresentados os resultados do algoritmo proposto com as incorporações das idéias presentes no VEGA. Esta tabela apresenta o resultado para diferentes versões do algoritmo, todas consideram os seguintes parâmetros.

- A população inicial foi fixada em 10.
- O número máximo de iterações foi fixado em 1500.
- O percentual foi de 50%.
- O parâmetro g foi definido como sendo $g = 1, 2$.

Na configuração V1 após um novo indivíduo ser criado, ele é inserido em uma subpopulação onde tem uma posição melhor e não é verificado se ele tem o mesmo valor nas duas funções objetivo que algum outro indivíduo desta subpopulação. Na configuração V2 isto também ocorre, mas a escolha da subpopulação onde o indivíduo será inserido é aleatória, esta mudança foi feita para analisar o impacto da escolha da subpopulação no tempo computacional.

Já na configuração V3 o indivíduo é inserido na subpopulação onde tem uma posição melhor e caso exista algum indivíduo com o mesmo valor das funções objetivos este é substituído pela nova solução, visando assim uma maior diversidade da população. Na configuração V4 a escolha da subpopulação é aleatória mas é mantido a substituição do indivíduo que apresente mesmo valor nas duas funções objetivo.

Novamente as soluções em negrito são as não dominadas (em relação as Tabelas 6.5 e 6.6) por nenhuma outra configuração do algoritmo, as Tabelas 6.5 e 6.6 apresentam ainda o número médio de soluções não dominadas em cada classe (NDOM) e também a média de diferentes soluções não dominadas em todas as classes (Média NDIF).

	VEGA V1					VEGA V2				
	Subpopulação 1		Subpopulação 2			Subpopulação 1		Subpopulação 2		
Classe	$\overline{x}_j$	$\overline{\delta}(x_j)$	$\overline{x}_j$	$\overline{\delta}(x_j)$	NDOM	$\overline{x}_j$	$\overline{\delta}(x_j)$	$\overline{x}_j$	$\overline{\delta}(x_j)$	NDOM
1	11,59	6,45	12,47	6,30	8,08	11,57	6,56	11,91	6,40	7,33
2	111,08	11,38	116,15	9,90	4,29	111,10	11,30	116,75	9,88	4,45
3	23,29	11,33	23,52	11,21	8,62	23,29	11,85	23,40	11,72	7,12
4	**217,07**	**23,92**	**223,26**	**21,56**	2,61	217,15	23,99	**225,55**	**21,46**	2,84
5	**45,23**	**22,52**	**45,37**	**22,34**	4,12	45,24	22,72	**45,37**	**22,38**	5,18
6	**426,58**	**49,54**	**466,41**	**45,10**	2,65	426,77	49,53	**472,98**	**45,00**	2,97
7	**54,55**	**9,31**	56,99	8,17	5,71	**54,55**	**9,16**	56,47	8,26	5,40
8	500,86	10,58	524,77	9,30	4,96	500,82	10,57	519,16	9,34	5,05
9	98,85	16,57	101,01	14,78	5,12	98,89	16,66	**100,60**	**14,83**	4,62
10	935,75	20,87	**959,27**	**18,69**	3,32	**935,56**	**21,16**	964,79	18,76	3,50
11	**186,72**	**33,07**	**189,47**	**30,36**	2,67	186,74	33,41	**189,56**	**30,33**	3,26
12	1770,95	41,54	**1812,53**	**38,12**	2,91	1771,10	41,99	**1802,47**	**38,23**	3,05
13	81,24	9,85	84,01	9,04	6,88	81,23	9,89	83,71	9,02	6,62
14	633,89	10,63	661,71	9,78	6,86	633,83	10,62	657,11	9,79	6,35
15	**126,14**	**17,71**	128,73	16,09	5,25	126,20	17,77	128,40	16,13	4,91
16	1195,63	20,35	1226,90	18,61	4,36	1195,59	20,41	1227,27	18,57	4,54
17	237,13	33,91	240,85	31,71	3,24	237,08	34,42	**240,36**	**31,68**	3,43
18	2249,18	39,85	2293,34	37,26	3,39	2249,37	39,91	**2299,45**	**37,12**	3,60
Média	494,76	21,63	509,26	19,91	4,72	494,78	21,77	509,18	19,94	4,68
Tempo	681,45 s					665,39 s				
Média NDIF	1,76					1,75				

Tabela 6.5: Configurações VEGA Parte 1

As configurações que apresentam uma maior quantidade de médias não dominadas (em relação aos resultados de cada classe - veja destaques em negrito) são as configurações V3 e V4 (28 e 21 respectivamente), podendo-se concluir que os melhores resultados são obtidos quando se substitui as soluções com mesmo valor nas duas funções objetivos e

	VEGA V3					VEGA V4				
	Subpopulação 1		Subpopulação 2			Subpopulação 1		Subpopulação 2		
Classe	$\overline{x}_j$	$\overline{\delta}(x_j)$	$\overline{x}_j$	$\overline{\delta}(x_j)$	NDOM	$\overline{x}_j$	$\overline{\delta}(x_j)$	$\overline{x}_j$	$\overline{\delta}(x_j)$	NDOM
1	**11,56**	**5,84**	**11,96**	**5,74**	2,06	11,58	5,86	**11,94**	**5,75**	2,12
2	**111,06**	**10,88**	**115,02**	**9,79**	1,95	**111,09**	**10,62**	117,18	9,79	2,18
3	**23,30**	**11,19**	**23,45**	**10,98**	1,83	**23,29**	**11,21**	23,53	11,09	1,96
4	**217,18**	**23,44**	**226,32**	**21,39**	1,98	217,11	23,98	226,04	21,66	2,27
5	45,23	22,67	45,64	22,66	1,57	45,24	22,76	45,74	22,54	1,87
6	**426,61**	**49,01**	**463,12**	**45,17**	2,25	426,72	49,38	**438,30**	**45,41**	2,26
7	**54,57**	**8,64**	56,03	8,11	2,20	**54,57**	**8,57**	**55,60**	**8,06**	2,33
8	500,78	10,14	**513,84**	**9,28**	2,11	**500,74**	**10,07**	**511,39**	**9,30**	2,25
9	**98,86**	**15,96**	**100,95**	**14,77**	2,01	**98,85**	**16,04**	**101,06**	**14,64**	2,36
10	**935,74**	**20,47**	**963,72**	**18,53**	2,33	**935,60**	**20,80**	962,03	18,74	2,59
11	186,72	33,14	189,47	30,51	2,08	**186,76**	**32,88**	**189,31**	**30,47**	2,18
12	**1770,92**	**41,51**	**1807,02**	**38,17**	2,55	1771,04	41,76	1810,32	38,27	2,72
13	**81,21**	**9,49**	**83,36**	**8,84**	2,46	**81,23**	**9,48**	**83,18**	**8,94**	2,69
14	633,81	10,37	**646,23**	**9,69**	2,50	**633,58**	**10,35**	647,50	9,70	2,63
15	**126,20**	**16,99**	**128,08**	**16,04**	2,09	**126,20**	**17,15**	128,34	16,06	2,29
16	**1195,59**	**19,96**	**1220,39**	**18,49**	2,21	**1195,72**	**19,88**	1225,60	18,53	2,46
17	**237,08**	**33,47**	**240,55**	**31,64**	2,02	237,15	33,58	**240,81**	**31,61**	2,35
18	**2249,05**	**39,61**	2299,49	37,17	2,54	**2249,18**	**39,42**	**2291,96**	**37,15**	2,52
Média	494,75	21,26	507,48	19,83	2,15	494,76	21,32	506,10	19,87	2,34
Tempo	684,67 s							671,58 s		
Média NDIF	2,15							1,86		

Tabela 6.6: Configurações VEGA Parte 2

também que a escolha da subpopulação deve ser feita baseada na melhor posição do novo indivíduo, já que a configuração com esta característica apresenta melhor resultado e um tempo computacional equivalente.

Analisando o número de soluções não dominadas (NDOM) pode-se concluir que as configurações V1 e V2 possuem um número maior de soluções não dominadas, mas quando se olha para o número médio de diferentes soluções não dominadas, percebe-se novamente que a configuração V3 apresenta melhor resultado.

Como a configuração V3 mostrou-se superior, foi analisada uma variação nos parâmetros que pode ser observada nas Tabelas 6.7 e 6.8. No teste T1 foi fixada uma população inicial de 10 indivíduos e o número máximo de 5000 iterações, no teste T2 a população também foi de 10 indivíduos mas com o número máximo de 10000 iterações, no teste T3 a população inicial foi de 20 indivíduos e um número máximo de 3000 iterações e finalmente o teste T4 também foi realizado com uma população de 20 indivíduos e o número máximo de 5000 iterações. Estes valores foram escolhidos de forma que o tempo total de execução não ultrapassasse o tempo total de execução do primeiro algoritmo evolutivo apresentado quando são variados os parâmetros β_1 e β_2.

Como pode ser observado, a média de soluções não dominadas em cada classe é bem superior no teste T2, isso se deve ao fato de em relação ao teste T1 o número de iterações dobrou, possibilitando uma maior convergência. Os testes T3 e T4 como tem uma população maior (o dobro em relação a T2) necessitariam de um número maior de iterações para obter uma maior convergência. Já o número de soluções não dominadas de cada teste foi muito próximo.

6.3.3 Resultados MOGA

Na Tabela 6.9 são apresentados os resultados do algoritmo com a incorporação das idéias presentes no MOGA, no teste M1 a população inicial foi fixada em 10 indivíduos e um número máximo de 1500 iterações, no teste M2 a população também foi de 10 indivíduos mas com um número máximo de 3000 iterações, no teste M3 a população foi de 20 indivíduos e um número máximo de 3000 iterações, e finalmente no teste M4 a população também de 20 indivíduos com um número máximo de 5000 iterações, sempre com o cuidado no momento da escolha dos parâmetros para que o tempo total de execução do algoritmo não ultrapassasse o tempo de execução do primeiro algoritmo evolutivo apresentado quando são variados os parâmetros β_1 e β_2.

Como pode ser observado os testes M2 e M4 apresentaram um maior número de média de soluções não dominadas por classe, apesar do número médio de diferentes soluções não dominadas do teste M3 ser ligeiramente maior.

Pode-se concluir que o melhor resultado é o apresentado pelo teste M4 já que a quantidade de médias não dominadas em cada classe é de 15 das 18 classes enquanto que o teste M2 apresenta 11 médias não dominadas. Em relação ao número de objetos cortados e de diferentes padrões de corte o teste M4 é apenas ligeiramente melhor. Por outro lado o tempo computacional gasto pelo teste M4 é muito maior.

	VEGA V3 T1					VEGA V3 T2				
	Subpopulação 1		Subpopulação 2			Subpopulação 1		Subpopulação 2		
Classe	$\overline{x}_j$	$\overline{\delta}(x_j)$	$\overline{x}_j$	$\overline{\delta}(x_j)$	NDOM	$\overline{x}_j$	$\overline{\delta}(x_j)$	$\overline{x}_j$	$\overline{\delta}(x_j)$	NDOM
1	11,56	5,64	12,00	5,53	2,08	11,56	5,61	12,00	5,49	2,07
2	111,00	10,31	113,67	9,26	2,09	**111,00**	**10,09**	**113,08**	**9,07**	2,09
3	23,28	10,70	23,43	10,60	2,02	**23,27**	**10,60**	23,39	10,51	2,02
4	217,07	22,48	222,14	20,46	1,96	**217,04**	**21,81**	**222,04**	**19,86**	1,91
5	45,23	21,03	45,33	20,94	1,65	**45,22**	**20,25**	**45,52**	**20,13**	1,85
6	426,41	47,86	443,71	43,87	2,23	**426,34**	**47,01**	**434,85**	**43,33**	2,11
7	54,55	8,59	55,71	8,02	2,22	54,54	8,56	**55,56**	**8,01**	2,20
8	500,74	9,96	**511,84**	**9,11**	2,13	**500,72**	**9,87**	**513,80**	**9,01**	2,17
9	98,85	15,61	100,83	14,31	2,14	98,84	15,45	**100,63**	**14,13**	2,16
10	935,33	20,16	**958,15**	**18,08**	2,32	**935,18**	**20,00**	**959,23**	**17,87**	2,36
11	186,67	31,90	188,88	29,43	2,02	**186,66**	**31,44**	**188,88**	**28,80**	2,04
12	1770,01	40,94	1800,84	37,48	2,41	**1769,71**	**40,56**	**1794,16**	**37,15**	2,34
13	81,18	9,46	83,19	8,84	2,44	**81,18**	**9,44**	83,12	8,83	2,42
14	633,73	10,31	**645,01**	**9,63**	2,49	**633,73**	**10,24**	**645,28**	**9,60**	2,52
15	126,20	16,85	128,13	15,82	2,15	126,17	16,81	128,08	15,73	2,17
16	1195,43	19,51	1219,03	18,20	2,22	**1195,38**	**19,42**	**1217,41**	**18,07**	2,20
17	237,02	32,70	239,61	30,71	2,16	**236,99**	**32,34**	**239,45**	**30,39**	2,16
18	2248,38	38,88	2295,86	36,52	2,33	**2248,22**	**38,56**	**2284,98**	**36,21**	2,26
Média	494,59	20,71	504,85	19,27	2,17	494,54	20,45	503,41	19,01	2,17
Tempo	2296,13 s							4665,78 s		
Média NDIF	1,75							1,73		

Tabela 6.7: Variação nos Parâmetros VEGA Parte 1

6.3.4 Comparação Entre os Algoritmos

Para uma melhor comparação entre os algoritmos, foram selecionados as melhores configurações dos três algoritmos e apresentados na Tabela 6.10.

Pode-se concluir que a quantidade de médias não dominadas de cada classe é muito semelhante (médias destacadas em negrito) apesar do teste S6 levar vantagem em relação aos outros métodos.

Os três testes também apresentam valores muito próximos em relação ao número de objetos cortados e ao número de diferentes padrões de corte, não havendo nenhum que

	VEGA V3 T3					VEGA V3 T4				
	Subpopulação 1		Subpopulação 2			Subpopulação 1		Subpopulação 2		
Classe	$\overline{x}_j$	$\overline{\delta}(x_j)$	$\overline{x}_j$	$\overline{\delta}(x_j)$	NDOM	$\overline{x}_j$	$\overline{\delta}(x_j)$	$\overline{x}_j$	$\overline{\delta}(x_j)$	NDOM
1	11,55	5,33	11,90	5,26	2,04	**11,55**	**5,23**	**11,77**	**5,14**	2,04
2	110,96	10,61	118,98	9,46	2,06	**110,91**	**10,27**	116,84	9,14	2,06
3	23,28	10,83	23,84	10,67	1,83	**23,28**	**10,36**	**23,91**	**10,32**	1,83
4	217,02	23,29	231,86	21,11	2,14	**216,97**	**22,81**	223,33	20,63	2,14
5	45,22	22,92	46,08	22,84	1,46	45,22	22,35	45,71	22,17	1,46
6	426,53	48,56	488,02	44,42	2,62	426,41	48,01	467,40	43,92	2,62
7	54,53	8,51	56,31	7,89	2,24	**54,51**	**8,42**	**56,24**	**7,79**	2,24
8	500,58	10,04	**512,34**	**9,11**	2,24	**500,55**	**9,95**	**512,61**	**9,04**	2,24
9	98,81	15,32	101,36	14,37	2,05	**98,80**	**15,08**	101,08	14,16	2,05
10	935,49	20,26	959,98	18,37	2,50	**935,39**	**19,93**	959,79	18,15	2,50
11	186,72	31,94	190,06	30,28	2,12	**186,68**	**31,44**	189,52	29,67	2,12
12	1770,44	41,04	1809,02	38,06	2,83	1770,18	40,76	1806,89	37,70	2,83
13	81,18	9,39	**83,02**	**8,82**	2,72	**81,18**	**9,36**	**83,34**	**8,79**	2,72
14	633,56	10,33	647,90	9,57	3,23	**633,56**	**10,28**	**647,74**	**9,55**	3,23
15	126,10	16,68	128,27	15,75	2,22	**126,09**	**16,56**	**127,96**	**15,66**	2,22
16	1194,95	20,01	1220,73	18,44	2,49	**1194,78**	**19,77**	**1217,29**	**18,31**	2,49
17	237,05	32,90	239,86	31,41	2,14	237,02	32,42	239,67	30,91	2,14
18	2249,29	39,21	2297,77	36,87	2,73	2248,72	38,93	2296,89	36,56	2,73
Média	494,62	20,95	509,29	19,59	2,31	494,54	20,66	507,11	19,31	2,31
Tempo	3204,88 s					5463,36 s				
Média NDIF	1,92					1,90				

Tabela 6.8: Variação nos Parâmetros VEGA Parte 2

tenha uma média geral melhor em todos os objetivos.

A quantidade de soluções não dominadas só pode ser analisada nos testes VEGA V3 T2 e MOGA M4, já que estes dois algoritmos fornecem como resultado uma população de soluções. Neste quesito o número médio de soluções não dominadas em cada classe é maior no MOGA M4, mas na média de diferentes soluções não dominadas o VEGA V3 T2 é ligeiramente maior.

Quanto ao tempo gasto na resolução dos problemas o teste S6 é muito menor, entretanto este método obtém apenas uma solução enquanto os outros métodos obtém uma

	MOGA M1			MOGA M2			MOGA M3			MOGA M4		
Classe	$\overline{x}_j$	$\overline{\delta}(x_j)$	NDOM	$\overline{x}_j$	$\overline{\delta}(x_j)$	NDOM	$\overline{x}_j$	$\overline{\delta}(x_j)$	NDOM	$\overline{x}_j$	$\overline{\delta}(x_j)$	NDOM
1	11,83	6,3	10,00	11,81	6,3	10,00	**11,72**	**6,04**	20,00	**11,8**	**6,03**	20,00
2	113,98	10,103	7,64	**111,92**	**9,833**	8,57	**112,95**	**9,49**	10,63	**112,06**	**9,56**	14,79
3	23,33	11,481	9,55	**23,299**	**11,408**	9,79	23,339	10,705	18,41	**23,321**	**10,686**	19,48
4	222,75	21,525	3,05	**221,44**	**20,794**	3,23	**220,49**	**21,307**	3,17	222,22	20,837	3,14
5	45,318	22,135	4,80	**45,314**	**21,358**	6,60	45,455	21,373	4,15	**45,422**	**20,628**	7,80
6	434,35	45,261	2,77	**431,54**	**44,654**	3,20	436,5	45,884	3,11	432,57	45,326	2,74
7	55,154	8,493	10,00	55,055	8,527	10,00	**55,055**	**8,399**	19,38	**54,946**	**8,413**	19,84
8	505,95	9,635	8,97	**504,57**	**9,565**	9,32	504,58	9,574	17,43	**504,85**	**9,464**	18,06
9	99,4	14,806	8,10	99,526	14,616	9,40	99,476	14,476	14,38	**99,378**	**14,385**	17,44
10	948,07	18,818	5,79	948,96	18,536	6,02	948,74	18,685	8,42	**946,99**	**18,504**	10,00
11	188,38	29,958	3,84	**187,65**	**29,045**	4,84	188,59	29,727	3,86	188,2	29,085	4,87
12	1794	38,98	4,22	1790,4	38,49	5,59	1794,2	38,647	4,61	**1789,5**	**38,141**	4,46
13	82,36	9,28	9,84	**82,224**	**9,275**	10,00	82,202	9,306	20,00	**82,125**	**9,301**	20,00
14	638,74	10,083	9,77	**639,96**	**10,003**	9,98	**637,16**	**10,083**	19,44	**638,2**	**10,023**	19,88
15	126,94	16,421	9,72	126,75	16,37	9,68	**126,53**	**16,271**	18,93	**126,55**	**16,24**	19,39
16	1204,9	18,974	8,26	**1204**	**18,934**	8,55	1204,4	18,863	14,58	**1204,2**	**18,672**	14,81
17	238,75	31,08	5,25	**238,33**	**30,439**	6,98	238,9	30,819	8,79	**238,52**	**30,327**	10,59
18	2276	37,071	5,16	2271,9	36,764	5,60	**2270,2**	**37,018**	5,92	**2270,2**	**36,483**	7,34
Média	500,56	20,02	7,04	499,70	19,72	7,63	500,02	19,81	11,96	499,50	19,56	13,04
Tempo	830,89 s			1642,55 s			3065,42 s			5510,94 s		
Média NDIF	1,65			1,62			1,69			1,65		

Tabela 6.9: Resultados para o MOGA

população de soluções. Analisando estes dados pode-se concluir que os três métodos apresentam desempenhos equivalentes.

Para ilustrar a disposição das soluções não dominadas no espaço de soluções um exem-

	S6 $\beta_1 = 0,5\,\beta_2 = 0,5$		VEGA V3 T2						MOGA M4		
			Subpopulação 1		Subpopulação 2						
Classe	$\overline{x}_j$	$\overline{\delta}(x_j)$	$\overline{x}_j$	$\overline{\delta}(x_j)$	$\overline{x}_j$	$\overline{\delta}(x_j)$	NDOM	$\overline{x}_j$	$\overline{\delta}(x_j)$	NDOM
1	11,59	6,33	**11,56**	**5,61**	**12,00**	**5,49**	2,07	11,8	6,03	20,00
2	**111,33**	**9,92**	**111,00**	**10,09**	**113,08**	**9,07**	2,09	**112,058**	**9,56**	14,79
3	**22,18**	**11,33**	**23,27**	**10,60**	**23,39**	**10,51**	2,02	23,321	10,686	19,48
4	**217,42**	**21,28**	**217,04**	**21,81**	**222,04**	**19,86**	1,91	222,223	20,837	3,14
5	**43,09**	**22,07**	**45,22**	**20,25**	**45,52**	**20,13**	1,85	45,422	20,628	7,80
6	**426,75**	**45,61**	**426,34**	**47,01**	**434,85**	**43,33**	2,11	**432,574**	**45,326**	2,74
7	**50,45**	**7,97**	54,54	8,56	55,56	8,01	2,20	54,946	8,413	19,84
8	**500,23**	**9,85**	500,72	9,87	**513,80**	**9,01**	2,17	**504,849**	**9,464**	18,06
9	**94,09**	**14,78**	98,84	15,45	**100,63**	**14,13**	2,16	**99,378**	**14,385**	17,44
10	**934,75**	**19,56**	935,18	20,00	**959,23**	**17,87**	2,36	**946,994**	**18,504**	10,00
11	177,86	29,64	186,66	31,44	188,88	28,80	2,04	188,2	29,085	4,87
12	**1768,99**	**39,55**	1769,71	40,56	**1794,16**	**37,15**	2,34	**1789,54**	**38,141**	4,46
13	**63,68**	**9,03**	81,18	9,44	**83,12**	**8,83**	2,42	82,125	9,301	20,00
14	**632,92**	**10,21**	633,73	10,24	**645,28**	**9,60**	2,52	**638,201**	**10,023**	19,88
15	**120,03**	**16,38**	126,17	16,81	**128,08**	**15,73**	2,17	**126,552**	**16,24**	19,39
16	**1193,87**	**19,5**	**1195,38**	**19,42**	**1217,41**	**18,07**	2,20	**1204,2**	**18,672**	14,81
17	**225,44**	**31,33**	236,99	32,34	239,45	30,39	2,16	**238,522**	**30,327**	10,59
18	**2246,21**	**38,34**	2248,22	38,56	**2284,98**	**36,21**	2,26	**2270,19**	**36,483**	7,34
Média	491,16	20,15	494,54	20,45	503,41	19,01	2,17	499,50	19,56	13,04
Tempo	502,625 s		4665,78 s					5510,94 s		
Média NDIF	-		1,73					1,65		

Tabela 6.10: Comparações entre os Algoritmos

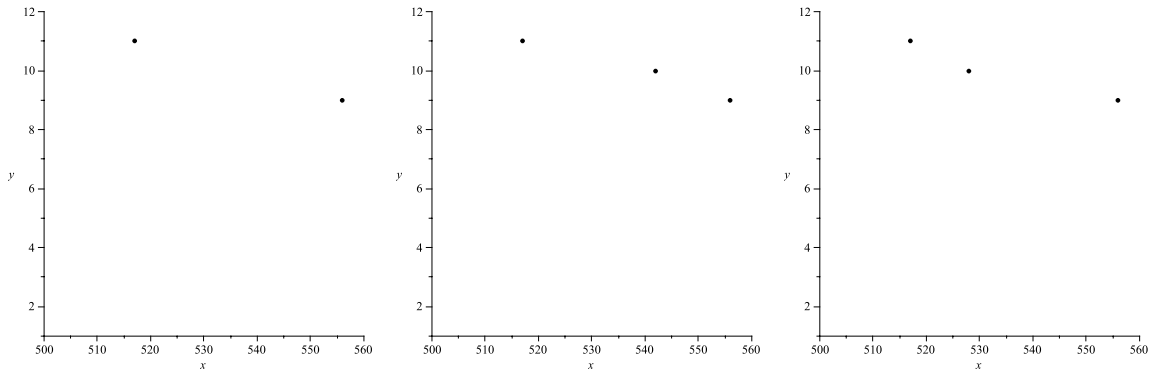
plo que também se encontra entre os testes realizados e que tem seus dados mostrados na Tabela 6.11 foi resolvido utilizando os três algoritmos desenvolvidos, os gráficos com as soluções não dominadas obtidas em cada algoritmo são apresentados na Figura 6.1.

No eixo x de cada gráfico está o número de objetos cortados, e no eixo y o número de diferentes padrões de corte utilizados. Na Figura 6.1(a) é apresentado o gráfico com as duas soluções não dominadas quando se utiliza o primeiro algoritmo evolutivo com a

Item	Comprimento	Demanda
1	749	39
2	618	179
3	502	72
4	480	140
5	455	162
6	452	108
7	427	105
8	374	61
9	354	28
10	304	106

Tabela 6.11: Características dos Itens

variação dos parâmetros β_1 e β_2, como no método da soma ponderada. Na Figura 6.1(b) é apresentado o gráfico com as três soluções não dominadas para o mesmo problema utilizando o algoritmo com as incorporações das idéias presentes no VEGA. E na Figura 6.1(c) é apresentado o gráfico das três soluções não dominadas também para o mesmo problema utilizando o algoritmo com as incorporações das idéias presentes no MOGA.



(a) Soma Ponderada (b) VEGA (c) MOGA

Figura 6.1: Soluções Não Dominadas

Pode ser observado que a curva do gráfico do MOGA se assemelha mais com a curva de Pareto desejada, pois uma das soluções do VEGA é dominada por uma das soluções não dominadas do MOGA.

Capítulo 7

Conclusões e Propostas Futuras

Neste trabalho foi abordado o problema de corte de estoque unidimensional inteiro tendo apenas um único tipo de objeto em estoque em quantidade suficiente para atender toda a demanda. Foi considerada a minimização de duas funções objetivo conflitantes: número de objetos cortados e número de diferentes padrões de corte utilizados.

Inicialmente foi feita uma adaptação do algoritmo de Araujo *et al.* [3] que trata do mesmo problema mas considera vários tipos de objeto em estoque em quantidades limitadas. Nesta adaptação o algoritmo trata o problema como sendo mono-objetivo, pois são dados pesos as funções objetivo, cabe ao decisor escolher estes pesos de forma a atender da melhor forma suas necessidades, o algoritmo fornece uma única solução.

Buscando contornaras dificuldades da utilização de duas funções objetivos conflitantes foram estudados os algoritmos evolutivos multiobjetivo, já que são ferramentas mais adequadas para tratar com estes tipos de problemas, pois fornecem ao decisor um conjunto de boas soluções. Quando se utiliza os algoritmos evolutivos multiobjetivo procura-se um conjunto de soluções não dominadas, ou seja, onde não se pode concluir qual delas é melhor, ficando esta escolha ao decisor de acordo com suas prioridades no momento. Mais ainda, se busca um conjunto de soluções conhecidas como soluções ótimas de Pareto ou soluções o mais próximo possível da frente de Pareto.

Para se obter um conjunto de soluções com a primeira adaptação apresentada foi necessário utilizar uma variação nos pesos das duas funções objetivo, como é feito no método da soma ponderada, apesar do algoritmo evolutivo apresentado possuir um tempo de execução rápido em cada simulação, quando se tenta obter um conjunto de soluções são necessárias uma variação nos pesos e conseqüentemente uma nova simulação para cada nova solução que se deseje encontrar. Os resultados obtidos com a variação dos parâmetros foram apresentados no Capítulo 6.

Buscando eliminar a necessidade da variação dos pesos do algoritmo e também da necessidade de várias simulações do algoritmo foram buscadas novas alternativas. A escolha

natural seria a utilização dos conceitos presentes nos algoritmos evolutivos multiobjetivo, o método escolhido inicialmente foi o *Vector Evaluated Genetic Algorithm* (VEGA) proposto por Schafer [44] em 1984, onde a população inicial é dividida em subpopulações de acordo com o número de funções objetivo (neste caso duas), onde cada uma se preocupa com apenas uma função objetivo.

Neste novo algoritmo apresentado foi dada uma especial atenção as soluções não dominadas, pois estas geralmente são mais interessantes ao decisor, diferentes configurações do algoritmo foram testadas, bem como uma variação nos parâmetros buscando um método que fornecesse um conjunto de boas soluções. Novamente os testes realizados foram apresentados no Capítulo 6.

Com o objetivo de testar um método multiobjetivo diferente e comparar os resultados foi proposto um novo algoritmo que incluía idéias presentes no *Multiple Objective Genetic Algorithm* (MOGA) proposto por Fonseca e Fleming [21]. As principais contribuições deste método foram a utilização de um *rank* para classificar as soluções de acordo com sua dominância e também a utilização do *fitness sharing*, ambas na tentativa de melhorar a diversidade das soluções obtidas. Foram realizados testes com diferentes parâmetros e apresentados no Capítulo 6.

Todos os algoritmos apresentados se mostraram eficientes principalmente na minimização do número de objetos cortados, em relação ao número de diferentes padrões de corte o desempenho não foi tão bom já que os algoritmos não permitem a produção em excesso.

As seguintes passos futuros são propostos

- Buscar uma melhoria na eficiência dos algoritmos para encontrar um maior número de soluções não dominadas;

- Realizar modificações no algoritmo que inclui idéias presentes no MOGA para reduzir o tempo computacional, já que é necessário a utilização do *fitness sharing* duas vezes a cada iteração, testar também a mudança da estratégia de substituição populacional *Steady State* pela geracional;

- Estudar o impacto da mudança da igualdade ($=$) na restrição de atendimento da demanda pela da desigualdade ($\geq$) tratando o excesso de produção como perda, já que esta mudança pode causar um impacto significativo na redução no número de diferentes padrões de corte. Para ilustrar considere por exemplo o caso onde $L = 8$, $l_1 = 5$ e $l_2 = 2$ com $d_1 = 1$ e $d_2 = 2$. Para satisfazer a demanda são necessários no mínimo 2 objetos e uma solução poderia ser um padrão que corte uma vez o item 1 e uma vez o item 2 e um segundo padrão que corte uma vez o item 2, satisfazendo assim a demanda com igualdade e utilizando dois padrões de corte

distintos. Não é possível encontrar uma solução ótima para o problema que utilize apenas um padrão de corte (considerando a igualdade na demanda). No entanto se for utilizada a restrição de desigualdade ($\geq$) pode-se cortar duas vezes o segundo padrão de corte e dessa forma tem-se uma solução com 2 objetos cortados e apenas um padrão de corte distinto. Considerando ainda a desigualdade pode-se incluir a heurística KOMBI [19] para a melhoria de um indivíduo na tentativa de tornar os métodos mais eficientes na minimização do número de diferentes padrões de corte utilizados;

- Extender os algoritmos para o caso com vários tipos de objetos em estoque.

Referências Bibliográficas

[1] ALEM, D.J.; ARENALES, M.N. O problema de corte com demanda estocástica: formulação matemática e método de solução. In: XIII Congreso Latino-Iberoamericano de Investigación Operativa, Montevidéu, 2006.

[2] ALVES, C.; CARVALHO, J.M.V. de. A stabilized branch-and-price-and-cut algorithm for the multiple length cutting stock problem, *Computers & Operations Research*, v. 35, p. 1315-1328, 2008.

[3] ARAUJO, S.A. de; HEIS A.; CONSTANTINO, A.A.; POLDI, K.C. An evolutionary algorithm to the one-dimensional cutting stock problem with multiple stock lengths. In: XIII CLAIO, Montevideo, 2006.

[4] ARENALES, M.N.; MORABITO, R.; YANASSE, H.H. Problemas de corte e empacotamento. In: XXXVI Simpósio Brasileiro de Pesquisa Operacional, São João Del-Rei, 2004.

[5] ARROYO, J.E.C. *Heurísticas e metaheurísticas para otimização combinatória multiobjetivo*. 2002. 256 f. Tese (Doutorado em Engenharia Elétrica) - Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas, Campinas, 2002.

[6] BELOV, G.; SCHEITHAUER, G. A branch-and-cut-and-price algorithm for one-dimensional stock cutting and two-dimensional two-stage cutting, *European Journal of Operational Research*, v. 171, p. 85-106, 2006.

[7] BELOV, G.; SCHEITHAUER, G. A cutting plane algorithm for the one-dimensional cutting stock problem with multiple stock lengths, *European Journal of Operational Research*, v.141, p. 274-294, 2002.

[8] BISCHOFF, E.E.; WÄSCHER, G. Cutting and packing, *European Journal of Operation Research*, v. 84, p. 503-505, 1995.

[9] CHERRI, A.C.; ARENALES, M.N. Heurísticas para o problema de corte de estoque com sobras de material reaproveitáveis. In: XIII Congreso Latino-Iberoamericano de Investigación Operativa, Montevideo, 2006.

[10] CHU, C.; ANTONIO, J. Approximation algorithms to solve real-life multicriteria cutting stock problems, *Operational Research*, v. 47, nº 4, p. 495-508, 1999.

[11] COELLO, C.A.C. An updated survey of ga-based multi-objective optimization techniques, *ACM Computing Surveys*, v. 32, n. 2, p.109-143, 2000.

[12] DEB, K. *Multi-objective optimization using evolutionary algorithms*. England: John Wiley & Sons Ltd., 2004.

[13] DEGRAEVE, M.; PETERS, M. Optimal integer solutions to industrial cutting stock problems: part 2, Benchmark Results, *Informs Journal on Computing*, v. 15, n. 1, p. 58-81, 2003.

[14] DIEGEL, A.; MILLER, G.; MONTOCCHIO, E.; SCHALKWYK, S. VAN; DIEGEL, O. Enforcing minimum run length in the cutting stock problem, *European Journal of Operational Research*, v. 171, p. 708-721, 2006.

[15] DOWSLAND, K.; DOWSLAND, W.B. Packing problems, *European Journal of Operational Research*, v. 56, p. 2-14, 1992.

[16] DYCKHOFF, H. A typology of cutting and packing problems, *European Journal of Operational Research*, v. 44, n. 2, p. 145-159, 1990.

[17] DYCKHOFF, H.; SCHEITHAUER, G.; TERNO, J. Cutting and packing. In: Amico, M., Maffioli, F., Martello, S., (eds.), Annoted bibliographies in combinatorial optimization. New York: John Wiley & Sons, p.393-414, 1997.

[18] DYCKHOFF, H.; FINKE, U. Cutting and packing in production and distribution - a typology and bibliography. New York: Physica-Verlag, Heidelberg, 258 p., 1992.

[19] FOERSTER, H.; WÄSHER, G. Pattern reduction in one-dimensional cutting stock problem, *International Journal of Operational Research*, v. 38, p. 1657-1676, 2000.

[20] FOERSTER, H.; WÄSHER, G. Simulated annealing for order spread minimisation in sequencing cutting patterns, *European Journal of Operational Research*, v. 110, p. 272-281, 1998.

[21] FONSECA, C.M.; FLEMING, P.J. Genetic algorithms for multiobjective optimization: fomulations, discussion and generalization. In Genetic Algorithms: Proceedings

of the fifth international conference on genetic algorithms (University of Illinois), p. 416-423, San Mateo, 1993.

[22] GAU, T.; WÄSHER, G. CUTGEN1: A problem generator for the standard one-dimensional cutting stock problem, *European Journal of Operational Research*, v. 84, p. 572-579, 1995.

[23] GILMORE, P.C.; GOMORY, R.E. A linear programming approach to the cutting stock problem, *Operations Research*, v. 9, p. 848-859, 1961.

[24] GOLDBERG, D.E. *Genetic algorithms in search, optimization, and machine learning*. Redwood City: Addison-Wesley Publishing Company, 1989.

[25] GOLDBERG, D.E. Genetic algorithms with sharing for multimodal function optimization. In: Proccedings of the First International Conference on Genetic Algorithms and Their Applications, p. 41-49, Cambridge, 1987.

[26] GRADISAR, M.; RESINOVIC, G.; KLJAJIC, M. A hybrid approach for optimization of one-dimensional cutting, *European Journal of Operational Research*, v. 119, p. 719-728, 1999.

[27] GRADISAR, M.; KLJAJIC, M.; RESINOVIC, G.; JESENKO, J. A sequential heuristic procedure for one-dimensional cutting, *European Journal of Operational Research*, v. 114, p. 557-568, 1999.

[28] GRAMANI, M.C.N; FRANÇA, P.M. The combined cutting stock and lot-sizing problem in industrial processes, *European Journal of Operational Research*, v. 174, p. 509-521, 2006.

[29] HAESSLER, R. W. Controlling cutting pattern changes in one-dimensional trim problems, *Operations Research*, v. 23, n. 3, p. 483-493, 1975.

[30] HINXMAN, A. The trim-loss and assortment problems: a survey, *European Journal of Operational Research*, v. 5, p. 8-18, 1980.

[31] HOLTHAUS, O. Decomposition approaches for solving the integer one-dimensional cutting stock problem with different types of standard lengths, *European Journal of Operational Research*, v. 141, p. 295-312, 2002.

[32] HORN, J.; NAFPLOITIS, N.; GODBERG, D. A niched Pareto genetic algorithm for multiobjective optimization. In: Proceedings of the First IEEE Conference on Evolutionary Computation, p. 82-87, Orlando, 1994.

[33] ISHIBUCHI, H.; MURATA, T. Multi-objective genetic local search algorithm. In: Procceedings of the 1996 IEEE International Conference on Evolutionary Computation, p. 119-124, 1996.

[34] KOLEN, A.W.J.; SPIEKSMA, F.C.R. Solving a bi-criterion cutting stock problem with open-ended demand: a case study, *The Jounal of the Operational Research Society*, v. 51, n. 11, p. 1238-1247, 2000.

[35] LIMEIRA, M.S. *Redução do número de padrões em problemas de corte de estoque*. 2003. 176 f. Tese (Doutorado em Computação Aplicada), Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2005.

[36] LOPES, M.A.L.P.; ARENALES, M.N. Estabilização da geração de colunas aplicada ao problema de corte de estoque unidimensal, XXXVII Simpósio Brasileiro de Pesquisa Operacional, Gramado, 2005.

[37] MARTELLO, S.; TOTH, P. *Knapsack problems:* algorithms and computer implementations. New York: J. Wiley & Sons, 1990. West Sussex.

[38] MCDIARMID, C. Pattern minimisation in cutting stock problems, *Discrete Applied Mathematics*, v. 98, p. 121-130, 1999.

[39] OLIVEIRA, J.F.; WÄSCHER, G. (Eds.), Cutting and packing, *European Journal of Operacional Research*, v.183, 2007.

[40] POLDI, K.C.; ARENALES, M.N. Heuristics for the one-dimensional cutting stock problem with limited multiple stock lengths, *Computers and Operations Research*, v. 36, p. 2074-2081, 2009.

[41] POLDI, K.C.; ARENALES, M.N. Heurísticas para o problema de corte de estoque unidimensal inteiro, *Pesquisa Operacional*, v. 26, n. 3, p. 473-492, 2006.

[42] POLDI, K.C.; ARENALES, M.N. O problema de corte de estoque multi-períodos, XXXVII Simpósio Brasileiro de Pesquisa Operacional, Gramado, 2005.

[43] RIEHME, J.; SCHEITHAUER, G.; TERNO, J. The solution for two-stage guillotine cutting stock problems havin extremely varying order demands, *European Journal of Operational Research*, v. 91, p. 543-552, 1995.

[44] SCHAFFER, J.D. *Some experiments in machine learning using vector evaluated genetic algorithms*. 1984. 186 f. Tese (Doutorado), TN: Vanderbilt University, Nashville, 1984.

[45] SCHEITHAUER, G.; TERNO, J. The modified integer round-up property of the one-dimensional cutting stock problem, *European Journal of Operational Research*, v. 84, p. 562-571, 1995.

[46] SRINIVAS, N.; DEB, K. Multiobjective optimization using nondominated sorting in genetic algorithms, *Journal of Evolutionary Computation*, v. 2, n. 3, p. 221-248, 1994.

[47] STADTLER, H. A one-dimensional cutting stock problem in aluminium industry and its solutions, *European Journal of Operational Research*, v. 44, p. 209-223, 1990.

[48] SWEENET, P.; PATRNOSTER, E. Cutting and packing problems: a categorized, application oriented research bibliography, *Journal of the Operation Research Society*, v. 43, p. 691-706, 1992.

[49] TRKMAN, P.; GRADISAR, M. One-dimensional cutting stock optimization in consecutive time periods, *European Journal of Operational Research*, v. 179, p. 291-301, 2007.

[50] UMETANI, S.; YAGIURA, M.; Ibaraki, T. One-dimensional cutting stock problem to minimize the number of different patterns, *European Journal of Operational Research*, v. 146, p.388-402, 2003.

[51] VANCE, P. Branch-and-prince algorithms for the one-dimensional cutting stock problem, *Computational Optimization and Applications*, v. 9, p. 212-228, 1998.

[52] VANDERBECK, F. Exact algorithm for minimising the number of setups in the one-dimensional cutting stock problem, *Operations Research*, v.48, n. 6, p. 915-926, 2000.

[53] VANDERBECK, F. A nested decomposition approach to a three-stage, two-dimensional cutting-stock problem, *Management Science*, v. 47, n. 6, p. 864-879, 2001.

[54] YANASSE, H.H.; LIMEIRA, M.S. A hybrid heuristic to reduce the number of different patterns in cutting stock problems, *Computers & Operations Research*, v. 33, p. 2744-2756, 2006.

[55] YANASSE, H.H.; LAMOSA, M.J.P. An integrated cutting stock and sequencing problem, *European Journal of Operational Research*, v. 183, p. 1353-1370, 2007.

[56] WALKER, W. E. A heuristic adjacent extreme point algorithm for the fixed charge problem, *Management Science*, v. 22, n. 5, p. 587-596, 1976.

[57] WANG, P.Y., WÄSCHER, G., (Eds.), Cutting and packing, *European Journal of Operation Research*, v. 141, 2002.

[58] WÄSHER, G.; GAU, T. Heuristics for the integer one-dimensional cutting stock problem: a computational study, *OR Spektrum*, v. 18, p. 131-144, 1996.

[59] WÄSHER, G.; HAYβER, H.; SCHUMANN, H. An improved typology of cutting and packing problems, *European Journal of Operational Research*, v. 183, p. 1109-1130, 2007.

Autorizo a reprodução xerográfica para fins de pesquisa.

São José do Rio Preto, _____/_____/____

_____
Assinatura

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo