Universidade Estadual de Campinas
Instituto de Matemática Estatística e Computação Científica
Departamento de Matemática Aplicada

Algoritmos Genéticos e o Problema de Corte Multiobjetivo

Daniel Tressi da Silva

Mestrado em Matemática Aplicada

Orientador: Prof. Dr. Antonio Carlos Moretti

Co-Orientador: Prof. Dr. Roberto Andreani

Trabalho Financiado pelo CNPq

Campinas

15 de Maio de 2009

Algoritmos Genéticos e o Problema de Corte Multiobjetivo

Este exemplar corresponde à redação final da dissertação devidamente corrigida e defendida por **Daniel Tressi da Silva** e aprovada pela comissão julgadora.

Campinas, 15 de Maio de 2009.

Prof. Dr: Antonio Carlos Moretti
Orientador

Prof. Dr: Roberto Andreani
Co-orientador

Banca Examinadora:

Prof. Dr. Antonio Carlos Moretti.
Prof. Dr. Aníbal Tavares de Azevedo.
Profa. Dra. Márcia Aparecida Gomes Ruggiero.

Dissertação apresentada ao Instituto de Matemática, Estatística e Computação Científica, UNICAMP, como requisito parcial para obtenção do Título de **MESTRE em MATEMÀTICA APLICADA.**

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP
Bibliotecária: Maria Fabiana Bezerra Müller – CRB8 / 6162

Silva, Daniel Tressi da
Si38a Algoritmos genéticos e o problema de corte multiobjetivo/Daniel Tressi da Silva -- Campinas, [S.P. : s.n.], 2009.

Orientador : Antonio Carlos Moretti ; Roberto Andreani.
Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Matemática, Estatística e Computação Científica.

1.Programação multiobjetiva. 2.Problema de corte. 3. Algoritmos genéticos. 4.Fronteira eficiente. I. Moretti, Antonio Carlos. II.Andreani, Roberto. III. Universidade Estadual de Campinas. Instituto de Matemática, Estatística e Computação Científica. IV. Título.

Título em inglês: Genetic algorithms and the cutting stock problem.

Palavras-chave em inglês (Keywords): 1. Multiobjective programming. 2. Cutting stock problem. 3. Genetic algorithms. 4. Efficient frontier.

Área de concentração: Pesquisa Operacional

Titulação: Mestre em Matemática Aplicada

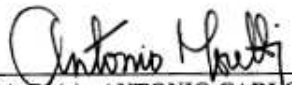
Banca examinadora: Prof. Dr. Antonio Carlos Moretti (IMECC – UNICAMP)
Prof. Dr. Roberto Andreani (IMECC - UNICAMP)
Prof. Dr. Aníbal Tavares de Azevedo (FEG - UNESP)
Profa. Dra. Márcia Aparecida Gomes Ruggiero (IMECC - UNICAMP)

Data da defesa: 15/05/09

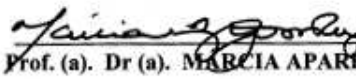
Programa de Pós-Graduação: Mestrado em Matemática Aplicada

Dissertação de Mestrado defendida em 15 de maio de 2009 e aprovada

Pela Banca Examinadora composta pelos Profs. Drs.



Prof.(a). Dr(a). ANTONIO CARLOS MORETTI



Prof. (a). Dr (a). MARCIA APARECIDA GOMES RUGGIERO

Prof. (a). Dr (a). ANIBAL TAVARES DE AZEVEDO

Agradecimentos

O principal e mais importante agradecimento é dedicado a meus pais, Josephina e Auster, pelo amor incondicional, carinho, dedicação e confiança ao longo dos anos de Unicamp e agora, em São Paulo. Agradeço também à Giseli e Clawrence, pelo apoio e compreenção durante esta passagem de minha vida.

Aos amigos que fiz em Campinas e que me acompanharam nesta longa jornada e tenho muita estima: Fellipe (Capial), por todo apoio, ter me ajudado e hospedado em sua casa na última fase do Mestrado. Aos companheiros de república, Regis, pelas incontáveis horas de conversas ora sérias, ora engraçadas. Carlos, pelas preciosas dicas de informática e ter me ensinado que existe vida inteligente fora dos sistemas operacionais pagos. Aos amigos de balada e suporte *psicológico*, Letícia, Chris e André, por terem me levado para casa em segurança quando me empolgava nas comemorações. Ulisses e Poli, por incontáveis horas de churrascos, baladas e histórias depois que o *Quarteto* foi formado. Elton, que além estar sempre presente nas festas e fazer parte do quarteto, muito me ajudou com dicas valiosas de programação em Matlab.

Aos amigos de São Paulo pelo apoio, incentivo e amizade ao retornar para casa novamente: Ademir (Varal), Igor (Barbal), Renato (Bull Halley), Leoneo (Derek), Marcelo (Boça), Clayton (Veio), Leornado (Leh). À Cesar (Ney), pelo incentivo a inclusão das últimas modificações neste trabalho.

Ao meu orientador Moretti e co-orientador Andreani, pelas críticas e sugestões que me possibilitaram um grande crescimento acadêmico e pessoal.

Ao CNPq - Conselho Nacional de Desenvolvimento Científico e Tecnológico pelo apoio financeiro.

Resumo

Nesta dissertação, estudamos algoritmos genéticos para resolver o problema de corte unidimensional multiobjetivo, onde minimizamos o desperdício dos objetos processados e o número de padrões distintos denominado custo de setup. Primeiro, realizamos uma codificação baseada em grupos desenvolvida por Falkenauer e, em seguida, aplicamos o algoritmo genético multiobjetivo SPEA2 para obter a Fronteira de Eficiente do problema.

Palavras-chave: Programação Multiobjetivo, Problema de Corte, Algoritmos Genéticos, Fronteira Eficiente.

Abstract

In this dissertation we studied genetic algorithms to solve the unidimensional multiobjective cutting stock problem, where we minimize the wastage of processed objects and the distinct number of patterns used, called setup cost. First, we make a group based codification derived by Falkenauer and, after that, we apply the multiobjective genetic algorithm SPEA2 to obtain problem's Efficient Frontier.

Keywords: Multiobjective Programming, Cutting Stock Problem, Genetic Algorithms, Efficient Frontier.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Nesta dissertação, fazemos o estudo de um problema clássico em Programação Linear denominado *Problema de Corte e Empacotamento*. O problema consiste em cortar rolos ou peças de tamanhos maiores em tamanhos menores, de modo que as sobras destes cortes sejam as menores possíveis e atendam uma certa demanda.

O ponto principal deste estudo é incorporar ao problema outros aspectos como o tempo de setup para a realização dos cortes além de minimizar o desperdício total oriundo dos cortes realizados. Para adicionar estas funções ao problema, este foi reformulado como um problema multiobjetivo e assim, passamos a considerar dois ou mais objetivos simultaneamente durante o processo de otimização.

Como ferramenta de otimização, será utilizado o Algoritmo Evolutivo (AE) denominado SPEA2 [29], que possui características desejadas para o nosso problema em estudo. Dado que este algoritmo engloba muitas características de algoritmos evolutivos e em particular, algoritmos genéticos, discutiremos os principais aspectos desta ferramenta para que a construção do algoritmo SPEA2 possa ser detalhada de forma coerente.

Neste capítulo, faremos uma breve introdução sobre o problema de corte, apresentando as motivações e conceitos que serão desenvolvidos nos capítulos seguintes.

1.1 Motivação

Na última década, algoritmos genéticos vêm sendo estudados na otimização de problemas cuja as variáveis de decisão são inteiras. Além dos algoritmos genéticos, outros tipos de algoritmos evolutivos têm sido aplicados e desenvolvidos para resolver diversos tipos e variações desta classe de problemas.

Nosso objetivo neste trabalho é reformular o problema de corte para que este incorpore mais de uma função objetivo e assim, obtermos um problema de corte multiobjetivo para analisar a variedade das soluções obtidas. Neste processo, aplicaremos o Algoritmo SPEA2 descrito em [29] para trabalhar diretamente com o problema multiobjetivo, pois, este é um algoritmo evolutivo desenvolvido para este tipo de problema.

Dada a natureza dos problemas multiobjetivo, obtemos um conjunto de soluções ótimas em apenas uma execução do algoritmo, ao invés de obter uma única solução ao final do processo de otimização. Com este procedimento, aumentamos o número de opções ao final do processo de otimização e assim, a decisão final pode ser tomada escolhendo-se uma das soluções obtidas.

1.2 Problemas de Corte

O problema de corte foi formulado pela primeira vez em 1939 pelo economista russo Kantorovich [13]. Tradicionalmente as soluções propostas para este problema se dividem em duas categorias: *Programação Linear Inteira* (PLI) e *Métodos Heurísticos*. Desta última categoria, iremos trabalhar com algoritmos genéticos, que possui características desejadas para o problema em questão.

Existem diversas variações do problema de corte segundo conceitos básicos como dimensionalidade, tipos de pedidos, variações de corte e outros. Neste trabalho, estudaremos o problema de corte unidimensional, ou seja, os cortes considerados são feitos somente em relação ao comprimento da peça.

O problema de corte é classificado como um problema de Programação Linear Inteira

(PLI), pois, as variáveis de decisão do problema são inteiras e positivas. As primeiras tentativas de resolução para este problema consistiam em relaxar as condições de integrabilidade das variáveis e resolver o problema aplicando o método simplex. Obtido o resultado, a solução era truncada para o valor inteiro mais próximo e ajustes pós-otimização eram feitos, se necessário, para atender o restante da demanda. Métodos heurísticos também foram desenvolvidos para aumentar a eficiência do processo de arredondamento de variáveis. (veja por exemplo , [30]).

Mas foi com Gilmore e Gomory [8] que surgiu o primeiro método eficaz de solução para o problema. Este método é conhecido como *Método de Geração de Colunas*. Devido a natureza do problema de corte, as restrições do problema ficam determinadas pela escolha do número de padrões de corte montados para o problema, formando uma matriz onde cada coluna determina um padrão de corte. Com o método de geração de colunas, novos padrões vão sendo adicionados ao problema anterior a cada iteração do método simplex revisado. Durante este processo, um novo padrão é encontrado resolvendo-se um *Problema da Mochila*. Este novo método possibilitou a resolução de problemas relativamente grandes com uma considerável diminuição de tempo.

1.3 Programação Multiobjetivo e os Algoritmos Evolutivos

É comum em um modelo matemático de otimização, encontrarmos definidas diversas funções objetivo para um mesmo problema. Geralmente, escolhe-se uma função por vez ou, as funções são escalarizadas dando-se um peso para cada objetivo e, aplica-se um método de otimização ao problema associado. A vantagem da programação multiobjetivo é poder incorporar diversas funções objetivo simultaneamente ao problema. Porém, grande parte dos métodos usados para tratar dos problemas multiobjetivo consiste em adaptar algoritmos usados para resolver problemas mono-objeitvo como descrito em [19], pois, são raros os métodos analíticos que trabalham diretamente com o problema em sua forma multiobjetivo.

Uma alternativa quem vem sendo estudada e utilizada nestes últimos anos é a aplicação de algoritmos evolutivos para tratar do problema multiobjetivo. Nesta classe, destacamos os algoritmos evolutivos multiobjetivo (AEMO), onde podemos citar como exemplo: (Non-Dominant Sorting Genetic Algorithm-II) NSGA2 [25], (Strength Pareto Evolutionary Algorithm-II) SPEA2 [29] e (Vector Evaluated Genetic Algorithm) VEGA [29]. A vantagem de aplicar estes métodos ao problema multiobjetivo está no fato de não ser necessário o cálculo de gradientes ou matrizes Jacobianas. Durante o processo de otimização são utilizados apenas informações relativas aos valores das funções objetivo e viabilidade de cada solução. Esta é a nossa motivação para o uso de um AEMO para otimizar problemas multiobjetivos.

1.4 Estrutura do Trabalho

Neste capítulo fizemos uma introdução dos tópicos que serão abordados com mais detalhes nos capítulos seguintes. O restante do trabalho está organizado na seguinte estrutura: no Capítulo 2 apresentamos os conceitos e definições de programação multiobjetivo, descrevendo seus aspectos e tipos de algoritmos aplicados para este tipo de problema. Em seguida, apresentamos o problema de corte com algumas variações e sua reformulação multiobjetivo.

No Capítulo 3 apresentamos e desenvolvemos os principais conceitos de algoritmos genéticos, pois, o algoritmo adotado neste trabalho tem como base a estrutura de algoritmos genéticos usados para otimizar o problema de corte em sua forma clássica. Também neste capítulo, descrevemos o Algoritmo SPEA2, que será utilizado para otimizar o problema de corte multiobjetivo. No Capítulo 4, apresentamos detalhes sobre a aplicação do SPEA2 para o problema em estudo dando detalhes de implementação sobre os procedimentos necessários para cada operador do algoritmo. No Capítulo 5 apresentamos os testes númericos, analisamos os resultados e apresentamos as conclusões e perspectivas futuras no Capítulo 6.

Capítulo 2

Descrição do Problema

Neste capítulo, apresentamos as definições e terminologias que serão utilizadas no decorrer deste trabalho, assim como o problema de corte com algumas variações e modelos para introduzirmos o Problema de Corte Multiobjetivo.

2.1 Programação Multiobjetivo

Quando estudamos programação matemática, estamos interessados em modelar e resolver problemas científicos encontrados em nosso cotidiano, como por exemplo, minimizar o desperdício em uma certa linha de produção ou maximizar o lucro obtido por uma empresa em um certo intervalo de tempo. Os ramos da programação matemática mais conhecidos são, Programação Linear (PL) e Programação Não-Linear (PNL). Tradicionalmente, os problemas de programação matemática possuem a seguinte forma:

$$\begin{aligned} Min \quad & f(x) \\ s.a \quad & x \in S \end{aligned} \tag{2.1.1}$$

onde $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ e S é o conjunto de soluções factíveis.

Definimos a solução ótima para o problema (2.1.1) como sendo $x^* \in S$ tal que $f(x^*) \leq f(x)$, para todo $x \in S$. O conceito de solução ótima local é definido da seguinte maneira:

dada uma vizinhança $V(x^*, \epsilon) \subseteq S$ em torno de x^*, tal ponto é uma solução local se $f(x^*) \leq f(x)$, para todo $x \in V(x^*, \epsilon)$.

Os métodos de otimização tradicionais são aplicados com o intuito de encontrar tanto soluções locais quanto globais para estes tipos de problema, mas temos garantia de convergência ao minimizador global somente para problemas convexos.

A Programação Multiobjetivo (PMO) é o ramo da matemática onde são estudadas soluções e métodos de otimização para problemas com mais de uma função objetivo. Muitos problemas em otimização admitem diferentes objetivos, em geral conflitantes. Usualmente selecionamos apenas um objetivo e otimizamos o problema. Os modelos de PMO permitem levar em conta todos os possíveis objetivos do problema. Tais modelos possuem a seguinte forma geral:

$$\begin{aligned} Min \quad & \{f_1(x), f_2(x), ..., f_m(x)\} \\ s.a \quad & x \in S \end{aligned} \tag{2.1.2}$$

onde $f_i : \mathbb{R}^n \longrightarrow \mathbb{R}$ e $i \geq 2$. Primeiro, devemos estabelecer alguns conceitos iniciais como descritos em [29], para posteriormente caracterizar a solução procurada para este tipo de problema.

As soluções factíveis deste problema são descritas por um vetor $x = (x_1, x_2, ..., x_n)$ no espaço de decisão S. A função definida por $w(x) = \{f_1(x), f_2(x), ..., f_m(x)\}$, com $w : S \longrightarrow W$, onde W é a imagem de S pela função $w(x)$, calcula o valor de uma solução no espaço dos objetivos W, dado pelo vetor objetivo $w = (w_1, w_2, ..., w_m)$. Para este tipo de problema, temos em geral um conjunto de soluções ótimas conhecidas como Conjunto Eficiente. Um exemplo deste mapeamento de soluções pode ser visto na figura (2.1). Para classificar as soluções ótimas do Problema (2.1.2), devemos introduzir o conceito de dominância, uma vez que estamos sempre tratando de minimizar duas ou mais funções objetivo.

Dados dois vetores objetivo w_1 e w_2, dizemos que w_1 *domina fortemente* w_2, $(w_1 \succ w_2)$, se $w_1^i < w_2^k$ para algum i, k e $w_1^i \leq w_2^j$ para todo $i, j = 1, ..., m$ com $j \neq k$, ou seja, se

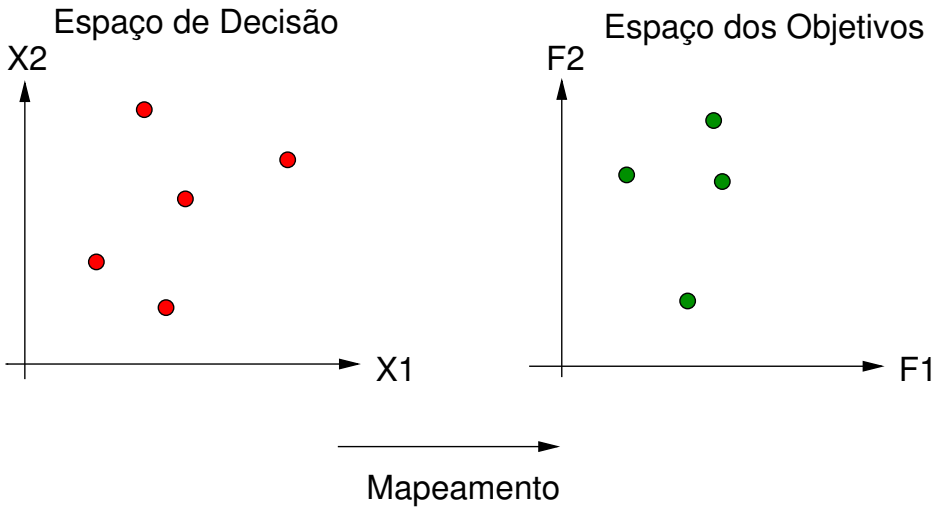


Figura 2.1: Mapeamento para o espaço dos objetivos

nenhuma componente de w_1 é maior ou igual que a componente correspondente de w_2 e pelo menos uma é menor. Dizemos que w_1 *domina fracamente* w_2, $(w_1 \succeq w_2)$ se $w_1^i \leq w_2^j$ para todo $i, j = 1, ..., m$. Neste sentido, dizemos que as soluções ditas como ótimas ou localmente ótimas, são aquelas que não são dominadas por nenhuma outra.

O conjunto de soluções ótimas no espaço de decisão S é denotado pelo Conjunto Pareto $S^* \subseteq S$ e, a imagem no espaço dos objetivos por Fronteira Pareto ou Fronteira Eficiente $W^* = g(S^*) \subseteq W$. Neste trabalho, usamos tanto os termos Fronteira Pareto quanto Fronteira Eficiente. Na prática, procuramos por um conjunto de soluções que não são fracamente dominantes entre si.

Para obter a Fronteira Pareto, métodos que transformam as funções objetivos como uma única função são os mais conhecidos e aplicados, pois, podemos usar algoritmos já conhecidos de problemas de otimização mono-objetivo. Estas estratégias são conhecidas como técnicas de escalarização, e dentre estas podemos citar:

- Método da somas ponderadas;
- Métodos de penalidade;
- Métodos baseados em funções de utilidade;

- Método do critério global.

2.1.1 Exemplo de Aplicação

Considere o problema multiobjetivo [7] apresentado a seguir:

$$\begin{aligned} Min \quad & \{x^2, (x-1)^2\} \\ s.a \quad & x \in \mathbb{R} \end{aligned} \tag{2.1.3}$$

De acordo com as definições acima, notamos que o Conjunto Pareto é formado pelo intervalo $[0, 1]$, pois, as imagens das soluções neste intervalo não são dominantes entre si. Esta imagem do conjunto $[0, 1]$ no espaço dos objetivos é a Fronteira Eficiente do problema. A seguir, apresentamos analiticamente como surgem estes conjuntos.

Para obter a Fronteira Pareto escolhemos o método das somas ponderadas. Este método consiste em reformular o Problema (2.1.2) como um problema mono-objetivo da seguinte forma,

$$\begin{aligned} Min \quad & \sum_{i}^{m} \lambda_i f_i(x) \\ s.a \quad & x \in S \end{aligned} \tag{2.1.4}$$

onde escolhemos $\lambda_i \geq 0$ tal que $\sum_{i}^{m} \lambda_i = 1$. O problema acima pode ser resolvido com o uso de métodos tradicionais para problemas mono-objetivos variando-se os valores de λ para obter os pontos da Fronteira Pareto. Aplicando este método ao nosso exemplo, podemos encontrar a Fronteira Eficiente resolvendo o seguinte problema,

$$\begin{aligned} Min \quad & f(x, \lambda) = \lambda x^2 + (1-\lambda)(x-1)^2 \\ s.a \quad & x \in \mathbb{R} \end{aligned} \tag{2.1.5}$$

onde calculamos a derivada da função acima com relação a variável x para obtermos,

$$f'_{\lambda}(x,\lambda) = 2\lambda x + 2(1-\lambda)(x-1) \tag{2.1.6}$$

onde temos $f'_{\lambda}(\overline{x}) = 0$ para $\overline{x} = 1 - \lambda$. Variando os valores de λ entre 0 e 1, e calculando a imagem de $\overline{x}$ por $f(\overline{x}, \lambda)$, obtemos a Fronteira Eficiente apresentada na figura (2.2),

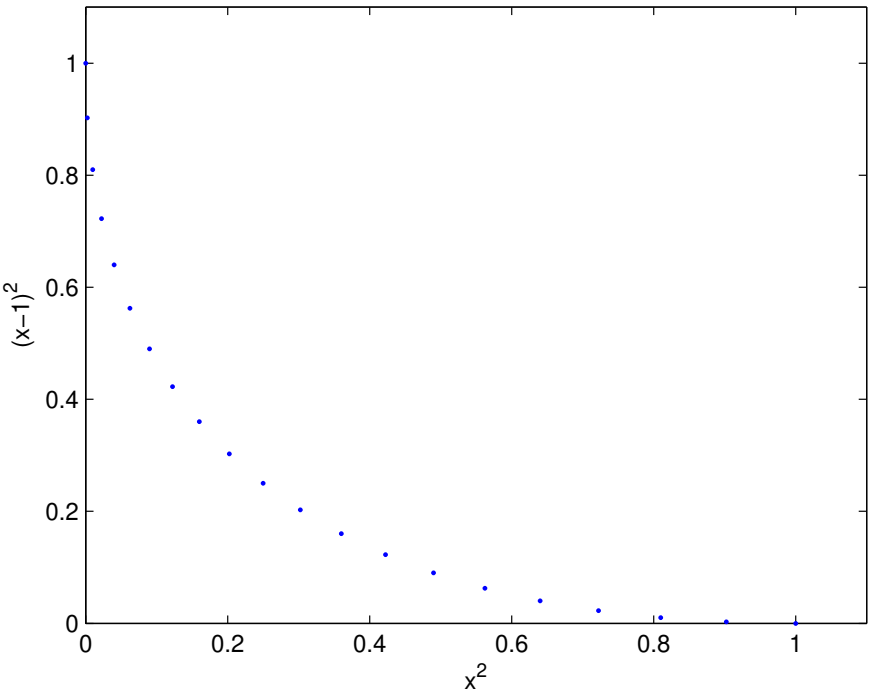


Figura 2.2: Fronteira Pareto

Apesar de ser o método mais comum de PMO, não temos garantia de obter uma boa distribuição da Fronteira Eficiente , pois, isto só ocorre se a Fronteira Eficiente for convexa. Existem ainda adaptações deste método [19], cujo objetivo é obter uma melhor distribuição da Fronteira Pareto, mas, tais adaptações só podem ser aplicadas de forma eficiente quando a fronteira desejada é convexa e, portanto, sua aplicação também é limitada.

Os métodos que se concentram em trabalhar diretamente com o problema na forma multiobjetivo, ainda não são os mais aplicados como, por exemplo, o *Método Simplex Multiobjetivo*. Uma alternativa aos métodos analíticos é o uso da *Computação Evolutiva*. Durante os últimos anos, algoritmos evolutivos como SPEA e SPEA2 descritos em [29]

vem sendo aplicados com sucesso em problemas multiobjetivo onde o espaço de busca não é convexo ou quando os objetivos são difíceis de se lidar como, por exemplo, um problema cujas as variáveis são inteiras ou, um problema misto onde temos tanto variáveis inteiras quanto contínuas. Aqui, descreveremos o SPEA2 como uma ferramenta alternativa para obter a Fronteira Pareto.

2.2 Problemas de Corte

Os problemas de corte (1/V/I/R por Dyckhoff [4]) consistem em cortar peças menores a partir de peças maiores com o objetivo de atender uma quantidade fixa de pedidos (*Demanda*). No presente trabalho, as matérias-prima serão denotadas por *Peças* e os tipos de pedidos como *Itens*. Após conhecer o total de pedidos, cada peça é cortada para se obter uma quantidade específica de itens, podendo apresentar uma sobra residual, que denotaremos por *Desperdício*. Para melhor ilustração, apresentamos o exemplo a seguir.

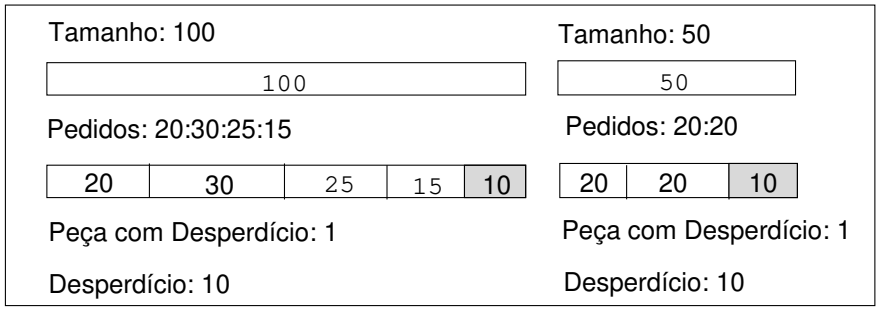


Figura 2.3: Exemplos de esquemas de corte

Neste exemplo, temos duas peças de tamanhos diferentes: 100 e 50. A primeira seqüência de corte é (20,30,25,15), que apresenta desperdício igual a 10. No segundo padrão de corte temos (20,20), o que resulta também em 10 como desperdício.

2.2.1 Opções de Matéria-Prima

Nos problemas de corte, temos duas possibilidades de tamanho para a matéria-prima: simples ou múltipla. Para o problema de corte simples, temos somente uma opção de

peça para ser utilizada. No caso de múltiplas escolhas, dispomos de mais de um tamanho de peça em estoque e devemos escolher quais e quantas peças serão necessárias para satisfazer a lista de pedidos. É intuitivo que mais escolhas implicam em melhor otimização do problema. A figura (4.3) mostra um exemplo deste processo:

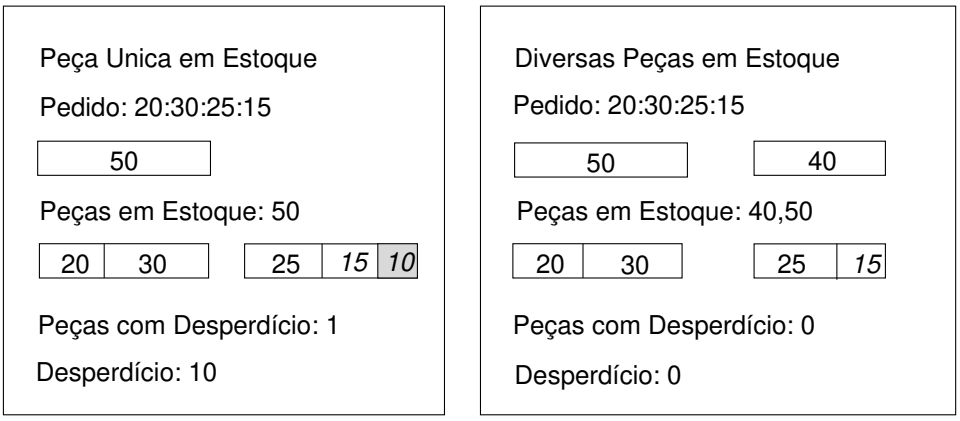


Figura 2.4: Opções de peças

Um aspecto importante a ser brevemente discutido é a dimensionalidade dos problemas de corte.

2.2.2 Dimensionalidade

Dimensionalidade em um problema de corte, indica quantas dimensões são necessárias para cortar uma peça e obter os itens de uma lista de pedidos. Essencialmente temos os seguintes tipos de classificação:

1. Unidimensional;
2. Bidimensional;
3. Tridimensional;
4. Multidimensional.

À medida em que o número de dimensões aumenta, os problemas de corte vão se tornando cada vez mais complexos de serem resolvidos, aumentando seu custo computa-

cional. Como problemas unidimensionais, podemos citar o problema de corte já descrito anteriormente. Para o segundo tipo, temos o corte de uma chapa retangular em chapas menores e para o terceiro caso, temos o carregamento de contêiners. Podemos pensar o quarto tipo como sendo um problema de corte do terceiro tipo onde levamos em conta o tempo como variável adicional.

2.2.3 Pilhas Abertas

Pilhas abertas (*Open Stacks*) assumem importância quando estamos considerando que dispomos de espaço limitado para armazenar os ítens que são parcialmente cortados. Neste caso, incluímos com o desperdício o custo de armazenamento dos ítens parcialmente abertos. Porém, pela própria natureza do problema, o desperdício e o custo de estoque dos ítens abertos são objetivos conflitantes.

Para um problema suficientemente grande, tal aspecto torna-se importante durante o processo de corte. A idéia básica de um Problema de Corte com Pilhas Abertas é seqüenciar os padrões de corte de modo a minimizar o número de itens parcialmente cortados. O exemplo a seguir ilustra como se dá este processo, dado que dispomos de um número suficiente de peças de tamanho 35. Considere a seguinte tabela de pedidos:

Tamanho	3	5	6	7	8	9
Demanda	2	4	1	4	2	2

Tabela 2.1: Tabela de demandas.

A tabela (2.2) mostra como funciona o seqüenciamento dos padrões de corte:

Padrão de Corte	1	2	3	4
Seqüência	(5,6,7,8,9)	(5,7,3)	(9,7,5,3)	(7,5,8)
Itens Abertos	(5,7,8,9)	(5,7,8,9,3)	(5,7,8)	(/)
Número de Itens	4	5	3	0

Tabela 2.2: Seqüenciamento de cortes.

Após o primeiro corte, cinco peças são cortadas e um item é fechado (item 6), assim o número de itens em aberto é quatro. Para o segundo corte, um novo item é aberto mas nenhum é fechado e portanto, temos cinco itens abertos, no terceiro corte dois itens são fechados (itens 3 e 9) restando agora três abertos. Finalmente, no último corte todos os itens são fechados e a demanda é atendida.

2.2.4 Custo de Setup

Quando uma peça é cortada, usamos um determinado padrão de corte que pode não ser o mesmo que será aplicado na próxima peça. Assim, devemos reconfigurar as lâminas ou facas da máquina que realizam o corte do novo padrão a ser utilizado. Ao executarmos este procedimento, se atribuírmos um custo ao tempo necessário para tal procedimento, estaremos incluindo o custo de setup em um Problema de Corte como descrito em Moretti [23]. Quanto mais padrões de corte, maior será o custo de setup.

2.3 Formulações

Nesta secão apresentamos as formulações para o Problema de Corte onde consideramos os diversos aspectos apresentados anteriormente, além de apresentar a formulação clássica do Problema de Corte que é amplamente utilizada na literatura. Estas formulações nos servirão de base para reformular o Problema de Corte e realizar a abordagem multiobjetivo do problema.

2.3.1 Problema de Corte Clássico

Nesta parte, descrevemos a seguir o modelo clássico abordado pela literatura. Para escrever o modelo baseando-se em [23], vemos o Problema de Corte da seguinte forma: Os padrões de corte são descritos em forma de uma matriz A onde cada elemento da coluna contém o número de vezes que cada item será cortado. Assim, indicamos por x_j como sendo o número de vezes que cada padrão de corte j é utilizado. Cada vez que

um padrão de corte é utilizado, uma nova matéria-prima é usada. Desta forma, a função objetivo minimiza o número de peças empregadas para atender a demanda, o que equivale a minimizar o desperdício. O problema é formulado da seguinte maneira,

$$\begin{aligned} Min \quad & \sum_{j=1}^{n} x_j \\ s.a \quad & \sum_{j=1}^{n} a_{ij} x_j = d_i \quad i = 1, ..., m \\ & x_j \in \mathbb{N} \quad j = 1, ..., n, \end{aligned} \tag{2.3.1}$$

onde x_j acima indica o número de vezes em que o padrão de corte j é usado, a_{ij} indica quantas vezes cada item i é cortado no padrão de corte j e d_i corresponde a demanda a ser satisfeita para cada item i. Para este problema, existem diversos métodos heurísticos e soluções baseadas em algoritmos de programação linear, sendo a mais usada o método de geração de colunas de Gilmore e Gomory.

2.3.2 Problema de Corte - Segunda Formulação

Além do modelo clássico descrito acima, existem formulações alternativas que tentam incorporar a geração de padrões de corte ao problema. Considerando um problema padrão, assumimos que dispomos de um número suficiente de peças de tamanho L para satisfazer os pedidos abaixo,

Tamanhos	w_1	w_2	...	w_m
Demandas	d_1	d_2	...	d_m

Tabela 2.3: Tabela de Pedidos.

Neste modelo minimizamos o desperdício, onde a primeira restrição garante que o tamanho dos itens cortados não ultrapasse o tamanho da peça L no padrão j, e a segunda restrição garante que a demanda é atendida na igualdade,

$$
\begin{aligned}
Min \quad & \sum_{j=1}^{n} Lz_j - \sum_{i=1}^{m} w_i d_i \\
s.a \quad & \sum_{i=1}^{m} x_{ij} w_i \leq Lz_j \quad j = 1, ..., n \\
& \sum_{j=1}^{n} x_{ij} = d_i \quad i = 1, ..., m \\
& x_{ij} \in \mathbb{N} \quad j = 1, ..., n \quad i = 1, ..., m \\
& z_j \in \mathbb{N} \quad j = 1, ..., n,
\end{aligned}
\tag{2.3.2}
$$

onde

m : número de itens pedidos.

n : número de padrões de corte.

L : tamanho da peça para corte.

w_i : tamanho do item i.

x_{ij} : número de itens i cortados na peça j.

d_i : demanda do item i.

z_j : número de vezes que o padrão j será utilizado.

2.3.3 Problema de Corte com Pilhas Abertas

O modelo para pilhas abertas possuí algumas modificações na função objetivo. Basicamente a nova função objetivo é descrita como ,

$$
\begin{aligned}
Min \quad & \sum_{j=1}^{n} Lz_j - \sum_{i=1}^{m} w_i d_i + \sum_{j=1}^{n} y(x_{ij}) \\
s.a \quad & \sum_{i=1}^{m} x_{ij} w_i \leq Lz_j \quad j = 1, ..., n \\
& \sum_{j=1}^{n} x_{ij} = d_i \quad i = 1, ..., m \\
& x_{ij} \in \mathbb{N} \quad j = 1, ..., n \quad i = 1, ..., m \\
& z_j \in \mathbb{N} \quad j = 1, ..., n,
\end{aligned}
\tag{2.3.3}
$$

onde o último termo da função objetivo é definido por,

$$y(x_{ij}) = \begin{cases} 0 & \text{se } \sum_{i=1}^{m} x_{ij} = 0 \text{ ou } \sum_{i=1}^{m} x_{ij} = d_i \quad j = 1, ..., n, \\ 1 & C.C. \end{cases}$$

Em relação ao modelo anterior, apenas adicionamos $y(x_{ij})$ que indica o número de itens abertos no padrão de corte j.

2.3.4 Problema de Corte com Custo de Setup

O problema de corte (2.3.2) é modificado para que passe a considerar o custo de reconfiguração das maquinas para cada padrão de corte. As modificações necessárias são:

$$Min \quad \sum_{j=1}^{n} Lz_j - \sum_{i=1}^{m} w_i d_i + \sum_{j=1}^{n} s(z_j), \tag{2.3.4}$$

onde o último termo é dado por,

$$s(z_j) = \begin{cases} 1 & \text{se } z_j > 0 \\ 0 & C.C. \end{cases}$$

O termo $s()$ varia de acordo com o número de padrões de corte, quanto mais padrões em comum, menor será o custo de setup e além disso, as restrições anteriores devem ser adicionadas ao problema. Os modelos descritos acima servem como base para a reformulação do Problema de Corte levando-se em conta os objetivos acima combinados onde, construiremos o problema de corte como um problema multiobjetivo.

2.3.5 Problema de Corte Multiobjetivo

Para considerar os demais objetivos ao invés de, trabalhar com diversos problemas sendo otimizados simultaneamente, podemos construir um versão multiobjetivo do problema de corte. Primeiro, definimos cada objetivo da seguinte maneira:

1. $f_1(x) = \sum_{j=1}^{n} Lz_j - \sum_{i=1}^{m} w_i d_i$ - Desperdício.

2. $f_2(x) = \sum_{j=1}^{n} y(x_{ij})$ - Pilhas abertas.

3. $f_3(x) = \sum_{j=1}^{n} s(z_j)$ - Setup

onde $y(x_{ij})$ e $s(z_j)$ são definidos como nos modelos anteriores. O problema multiobjetivo é formulado como:

$$\begin{aligned} Min \quad & \{f_1(x), f_2(x), f_3(x)\} \\ s.a \quad & \sum_{i=1}^{m} x_{ij} w_i \leq L z_j \quad j = 1, ..., n \\ & \sum_{j=1}^{n} x_{ij} = d_i \quad i = 1, ..., m \\ & x_{ij} \in \mathbb{N} \quad j = 1, ..., n \quad i = 1, ..., m \\ & z_j \in \mathbb{N} \quad j = 1, ..., n, \end{aligned} \tag{2.3.5}$$

O modelo multiobjetivo pode ser formulado para considerar apenas dois objetivos:

$$\begin{aligned} Min \quad & \{f_p(x), f_q(x)\} \\ s.a \quad & \sum_{i=1}^{m} x_{ij} w_i \leq L z_j \quad j = 1, ..., n \\ & \sum_{j=1}^{n} x_{ij} = d_i \quad i = 1, ..., m \\ & x_{ij} \in \mathbb{N} \quad j = 1, ..., n \quad i = 1, ..., m \\ & z_j \in \mathbb{N} \quad j = 1, ..., n, \end{aligned} \tag{2.3.6}$$

onde $p, q = 1, ..., 3$, $p \neq q$. A vantagem neste tipo de formulação é poder combinar os objetivos dois a dois para obtermos um conjunto mais amplo de soluções para o problema de corte.

Capítulo 3

Conceitos, Algoritmos e Soluções

Os problemas de corte estão entre os mais tradicionais em Pesquisa Operacional. Basicamente, as soluções propostas para o problema de corte unidimensional podem ser divididas em dois grupos: soluções exatas e heurísticas. Nesta seção, vamos descrever os algoritmos que serão a base para tratar o modelo de problema de corte multiobjetivo.

3.1 Algoritmo Genético (AG)

3.1.1 Introdução

Grande parte dos problemas científicos podem ser formulados como um problema de otimização. Os problemas de otimização possuem o seguinte formato: dados uma função $f : \mathbb{R}^n \longrightarrow \mathbb{R}^m$, e um espaço de busca $S \subseteq \mathbb{R}^n$, o problema pode ser formulado como

$$\begin{aligned} Min \quad & f(x) \\ s.a \quad & x \in S. \end{aligned}$$

Os problemas de otimização são classificados de acordo com sua função objetivo, restrições e tipo de variável de decisão. Por exemplo, se a função objetivo e as restrições são lineares, este problema é classificado como um problema de programação linear . Dentre

essas classes existem divisões que são características de cada problema.

Basicamente os métodos de otimização mais utilizados são: os númericos, enumerativos e probabilísticos. Os métodos numéricos englobam praticamente todos os algoritmos tradicionais baseados em variações da direção do gradiente e da direção de Newton. A idéia é, dado uma solução inicial, calculamos o gradiente da função objetivo para a partir dessa informação, obter o ponto seguinte. Também podemos analisar a concavidade da função e calcular suas derivadas parciais para decidir qual será o ponto seguinte. Geralmente, as derivadas parciais são aproximadas numéricamente a cada iteração, como ocorre por exemplo, no Método Quasi-Newton. Para tais métodos são garantidos convêrgencia local e global sob certas condições. Estes métodos estão preparados para ser eficientes na busca de pontos estacionários, geralmente mínimos locais. Também mínimos globais são achados com freqüência mas, a tarefa de achar o mínimo global é um desafio importante a ser trabalhado.

Os métodos enumerativos seguem uma idéia bastante intuitiva: examina-se os pontos do espaço em busca do ótimo local ou global. Percebemos que conforme cresce a dimensão do problema, o método torna-se impraticável. Os métodos probabilísticos utilizam a escolha probabilística como ferramenta para a busca dirigida. Um exemplo desta técnica [20] são os *Algoritmos Genéticos* (AGs).

AGs são processos de busca aleatória estruturada que imitam o processo de evolução biológica. O algoritmo inicia com um conjunto de estimativas (*População*) codificadas chamadas de *Indivíduos* e, a cada uma desta estimativas é atribuído um valor que mostra a qualidade da solução. Esse valor é chamado de *Mérito*. Com base nestes valores, indivíduos são selecionados para serem combinados entre si gerando uma nova população. Essa analogia com o processo biológico sugere que tal procedimento conduzirá a soluções mais eficientes para problemas mais complexos. Estes métodos representam uma alternativa na busca do mínimo global aos métodos clássicos de programação matemática. A eficiência dos AGs está em sua adequação ao problema que queremos resolver.

Uma idéia muito comum é combinar ambos os métodos como uma espécie de algoritmo híbrido onde, utilizam-se o gradiente e o esquema probabilístico dos AGs. Por

exemplo, podemos combinar Têmpera Simulada com o Método de Newton para garantir boas soluções. Nosso objetivo é aplicar um algoritmo genético para obter uma aproximação das soluções ótimas do problema de corte multiobjetivo.

3.1.2 Características de um Algoritmo Genético

Os Algoritmos Genéticos possuem características que se destacam dos métodos tradicionais. O método trabalha em um espaço de soluções codificadas onde, o valor de mérito atribuído a cada indivíduo depende do valor da função objetivo calculada para cada um destes. Com base neste valor é que se dará a seleção e busca em um espaço de soluções codificadas. Esta busca é feita através de um ou mais operadores genéticos para obtermos novas soluções candidatas. Os operadores mais utilizados são *Cruzamento* e *Mutação*, que serão descritos com mais detalhes na proxima seção. Durante o processo descrito acima, não é utilizado nenhuma informação sobre derivadas parciais ou gradiente, o que faz com que a aplicação mais natural dos AGs seja em problemas onde as funções objetivo não possuem derivadas ou, sejam descontínuas por partes. Uma aplicação muito comum dos AGs é a otimização de funções multimodais com muitos picos e vales, pois, os métodos tradicionais de otimização convergem para um ponto crítico local.

Quando os AGs são aplicados para a resolução de um problema, os operadores cruzamento e mutação atuam em conjunto para aumentar a eficiência do método. Se um AG utilizar somente o operador Cruzamento, o espaço codificado de busca ficará limitado e, neste caso, a Mutação permite que novos locais sejam exploradas, o que ajuda a impedir a convergência para um ótimo local.

Porém, existem variações de AGs que fazem uso somente do operador mutação. Tais algoritmos são aplicados como uma alternativa, quando o Cruzamento demanda um alto custo computacional comprometendo o desempenho do algoritmo. A princípio, podemos pensar que os AGs trabalham aleatóriamente no espaço de soluções codificadas. O método baseia-se na probabilidade de encontrar melhores soluções em uma determinada região de busca. Além disso, os operadores Cruzamento e Mutação combinados mantêm a diversidade do conjunto de soluções quando estamos longe do ótimo global e conservam

as características dos melhores indivíduos.

3.1.3 Principais Aspectos de um Algoritmo Genético

Nesta seção, daremos uma visão geral da estrutura básica de um AG ilustrando a função dos operadores cruzamento, mutação e seleção sem entrar em detalhes de implementação. O primeiro passo a ser definido é como o espaço de busca será codificado de acordo com os parâmetros do problema. Tradicionalmente, uma maneira de efetuar tal codificação é usar o alfabeto binário $A = \{0, 1\}$. Cada indivíduo é representado por um cromossomo que é uma seqüência de caracteres escolhidos para a representação de cada solução. No caso da representação binária, um cromossomo é uma seqüência de $0s$ e $1s$. Escolhida a representação, o próximo passo é determinar o tamanho do espaço de busca das soluções que varia para cada tipo de problema.

Um aspecto importante dos AGs é a seleção ambiental. Para desempenhar esse papel, aplicamos o operador *Seleção* que imita o processo de evolução Darwiniana selecionando os melhores indivíduos da população. Assim como ocorre na natureza, os melhores indivíduos irão gerar descendentes. O primeiro passo é atribuir um valor de aptidão ou mérito que refletirá o quanto o indivíduo está adaptado ao ambiente. É comum neste caso, usar a função objetivo com algumas modificações para atribuir os valores de mérito para cada indivíduo. Estas modificações são necessárias, pois, não é uma boa estratégia usar diretamente a função objetivo já que alguns operadores de seleção baseiam-se em probabilidades e portanto não devemos admitir valores negativos de mérito. Existem diversas maneiras de se promover esta seleção sendo as mais utilizadas o *Torneio* e a *Roleta* [20].

Além disso, é necessário definir uma condição de término para o algoritmo. Geralmente, é utilizado como condição de parada o número máximo de gerações, ou seja, quando o número de gerações atingir um valor pré-definido, o algoritmo é encerrado.

3.1.4 Mutação e Cruzamento

Vamos agora dar mais detalhes sobre os operadores Cruzamento e Mutação. No cruzamento, o procedimento consiste em selecionar dois indivíduos da população e recombiná-los para se obter dois novos indivíduos. Este processo é denominado *Recombinação Cromossômica*.

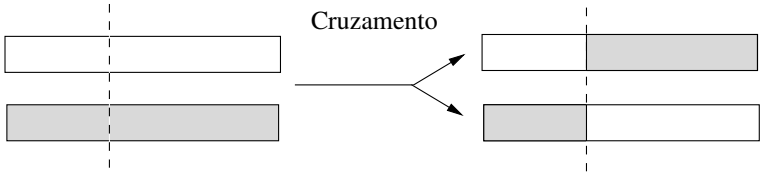


Figura 3.1: Recombinação Cromossômica.

Vamos ilustrar o cruzamento com um exemplo. Dada a seguinte população $P = \{22, 50, 23, 71, 56, 32\}$, suponha que os indivíduos selecionados sejam $P_1 = 50$ e $P_2 = 23$. Para codificar as soluções, escolhemos a base 2 com uma representação de 6 bits. O próximo passo é escolher aleatóriamente uma posição i tal que $i \in [1, 5]$ para recombinar P_1 e P_2. Suponha $i = 3$, neste caso, os indivíduos selecionados são escritos na base 2 como:

$$P_1 = 110|010$$

$$P_2 = 010|111.$$

Agora trocamos as informações a partir da posição selecionada para obter novos indivíduos F_1 e F_2 dados por:

$$F_1 = 110|111$$

$$F_2 = 010|010.$$

Convertendo para a base 10, temos $F_1 = 55$ $F_2 = 18$ que serão incorporados na população.

O operador Mutação atua diretamente na representação do indivíduo alterando um bit j selecionado gerando um novo indivíduo. Dada a população anterior, suponha que $I = 71$ seja o indivíduo selecionado. Sua representação binária é:

$$I = 1000111.$$

Se o bit selecionado for $j = 5$, então $I(j) = 1$, e basta definir $I(j) = 0$ para obter um novo indivíduo:

$$I^{'} = 1000011.$$

Assim obtemos $I^{'} = 67$ que será incorporado na população. Os operadores ilustrados acima atuam em conjunto a cada iteração. A frequência com que é aplicada cada um destes operadores também deve ser levada em conta. Geralmente, para cada um atribuímos uma probabilidade de ocorrência sendo padrão atribuir uma probabilidade entre 50% e 75% para o Cruzamento e 0.05% para a Mutação. O motivo para tal escolha se deve ao seguinte fato [20]: O Cruzamento atua recombinando indivíduos para obter melhores soluções enquanto a Mutação fornece espaços alternativos de busca. Se atribuímos uma frequência alta para a Mutação, não teremos um espaço de busca definido , pois, o algoritmo pode se comportar como um método de busca aleatório , e portanto, a convergência do algoritmo será comprometida ou o algoritmo pode não convergir.

3.1.5 Roleta

Tal procedimento consiste em atribuir probabilidades de seleção a cada indivíduo com base nos valores de mérito. Para isto, devemos ter valores de mérito positivos. Dados m indivíduos, cada um com valor de mérito f_i, definimos novos valores de mérito como,

$$f_i^{'} = f_i + max(abs(f_i)) + \beta \tag{3.1.1}$$

onde $\beta > 0$. A função de mérito é escalarizada da maneira acima, pois, para montar a Roleta, é necessário que estes valores sejam todos positivos e não nulos conforme descrevemos a seguir. Assim, a probabilidade de cada indíviduo ser escolhida é calculada por,

$$P_i = \frac{f_i^{'}}{\sum_{i=1}^{m} f_i^{'}} \quad .$$

Obtidas as probabilidades P_i, estas são ordenadas em ordem crescente. Suponha que os valores tenham sido ordenados da seguinte forma,

$$(P_2, P_1, P_5, ..., P_{j-2}, P_j),$$

onde $j \in (1, ..., m)$. A partir desta ordenação considere os intervalos $(0, P_2]$, $(P_1, P_5]$, até $(P_{j-2}, P_j]$. Um número α é gerado aleatóriamente e verifica-se em qual intervalo se encaixa o número. Se $\alpha \in (0, P_2]$, então o indivíduo $i = 2$ é escolhido, ou seja, se $\alpha \in [P_k, P_l]$ então $i = l$, onde $k, l \in (1, ..., m)$. Este procedimento é equivalente a *girar* a roleta uma vez.

Portanto para completar a população com o restante dos indivíduos, basta utilizar a roleta o número de vezes necessário. A roleta também pode ser executada sem reposição, ou seja, uma vez escolhido um indivíduo j, este é excluído e então as probabilidades dos indivíduos são calculadas novamente.

3.1.6 Torneio

O Torneio é uma forma mais simples e com custo computacional inferior à roleta. Quando aplicamos o Torneio como forma de seleção, não é necessário efetuar nenhuma alteração nos valores de mérito dos indivíduos. No Torneio, um certo número de indivíduos é escolhido ao acaso e dentre estes, escolhe-se aquele que apresenta o melhor valor de mérito.

Por exemplo, se o problema onde está sendo aplicado o AG for de minimização, podemos utilizar como função de mérito o valor da própria função objetivo, pois, aquele que tiver o menor valor de mérito será o vencedor. Portanto, não há necessidade de escalarizar os valores de mérito. Quanto ao número de indivíduos escolhidos no Torneio, este pode variar com o número de iteração ou ser fixo durante todo o processo. A escolha do número de indivíduos escolhidos no Torneio tem grande influência na velocidade de convergência do AG.

3.1.7 Estrutura de um Algoritmo Genético

Na figura (3.2) exibimos uma possível estrutura de um algoritmo genético ilustrando a seqüência dos operadores :

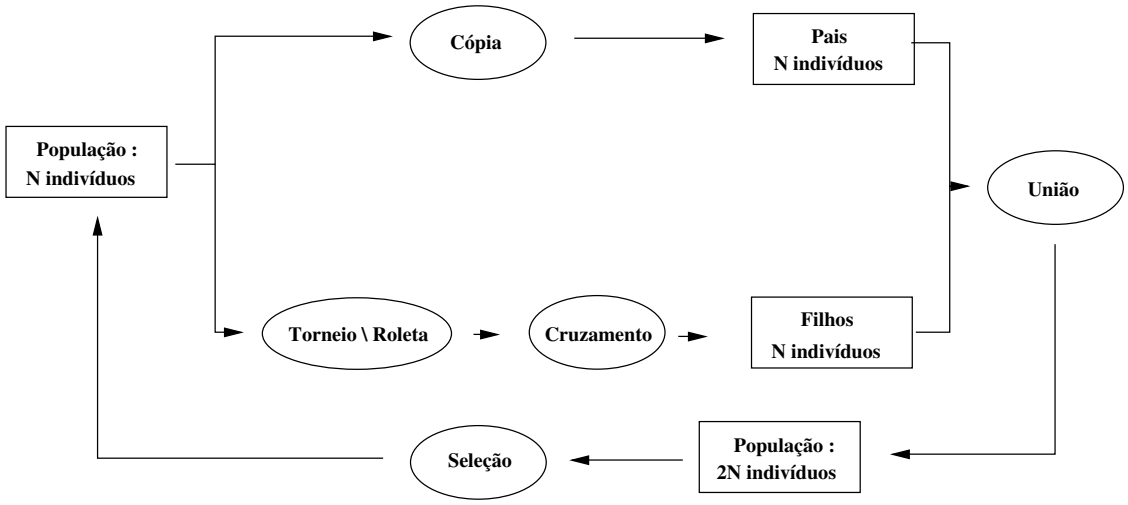


Figura 3.2: Exemplo de estrutura de um AG.

3.1.8 Exemplo de Aplicação

Para ilustrar os conceitos descritos acima, vamos aplicar um AG na minimização de uma função $f(x)$. Vamos descrever a aplicação do AG ao problema:

$$\begin{aligned} Min \quad & \frac{-sen(4\pi x)}{4\pi x} \\ s.a \quad & x \in [-4, 4] \end{aligned} \tag{3.1.2}$$

por ser uma função multimodal, métodos contínuos não podem ser aplicados eficientemente para se obter o ótimo global.

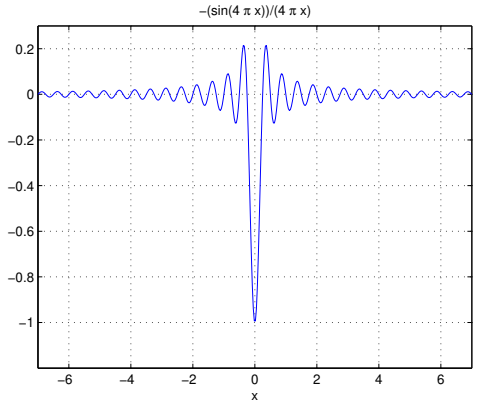


Figura 3.3: Gráfico da função objetivo.

Observamos que o mínimo global ocorre em $x = 0$ com $f(x) = -1$. Para representar uma solução factível x, escolhemos o alfabeto binário $A = \{0, 1\}$ com um número de bits L. O número de bits fica determinado de acordo com a precisão escolhida. No caso, escolhemos $L = 20$ que corresponde a uma precisão de 10^{-6} , pois, $8/2^{20} = 7.6294 \times 10^{-6}$. O espaço de busca $-4 \leq x \leq 4$ será codificado em $0 \leq \overline{x} \leq 2^{L-1}$.

A transformação para este intervalo é feita desta maneira, pois, podemos trabalhar com os operadores genéticos de maneira mais simples uma vez que, cada solução no intevalo codificado é positiva. Tais operadores podem ser aplicados diretamente sem nenhuma restrição. A relação entre as soluções x e $\overline{x}$ fica definida da seguinte forma:

O mapeamento $x \longrightarrow \overline{x}$ é dado por

$$\overline{x} = \frac{x - x_{min}}{x_{max} - x_{min}}(2^L - 1) + 0.5.$$

A operação inversa é definida como

$$x = \frac{\overline{x}}{2^L - 1}(x_{max} - x_{min}) + x_{min}.$$

O mapeamento é feito quando aplicamos os operadores cruzamento e mutação onde, a operação inversa é feita quando calculamos os valores de mérito das soluções. Durante a implementação do método, não é necessário converter $\overline{x}$ para base binário se a linguagem suportar operadores binários. Na linguagem Matlab, podemos combinar os operadores *bitand* e *bitor* para realizar as operações com alfabeto binário. O esquema abaixo mostra a linha geral do algoritmo implementado.

AG - Estrutura Geral

Dados de entrada:

- M - Tamanho da população.
- T_{max} - Número máximo de gerações.
- P_{mut} - Probabilidade de ocorrer Mutação.

1. Gerar a população inicial M.
2. Selecionar indivíduos para efetuar o Cruzamento através do Torneio.
3. Efetuar o Cruzamento nos indivíduos selecionados.
4. Aplicar o operador Mutação nos descendentes.

5. Selecionar indivíduos na união de Pais+Filhos através do Torneio para obter os indivíduos que continuam na próxima geração.

A população adotada para este problema possui tamanho $N = 32$ e utilizamos um gerador de números aleatórios uniforme para obter os indivíduos iniciais dentro do intervalo $0 \leq \overline{x} \leq 2^{L-1}$. Os operadores de recombinação genética aplicados são os mesmos descritos nas sessões anteriores já que a codificação adotada é a mesma dos exemplos descritos. O desempenho do algoritmo é ilustrado pelo gráfico apresentado na figura (3.4), no qual mostramos o valor da função objetivo para o melhor indivíduo a cada geração. Vemos que antes de atingir 10 iterações o algoritmo já convergiu para a solução do problema.

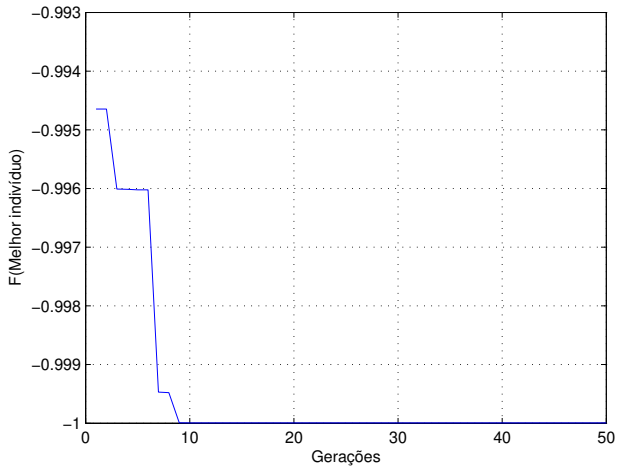


Figura 3.4: Valores da função objetivo para o melhor indivíduo.

A ordem dos operadores em nossa implentação não corresponde ao padrão básico adotado pelos AGs. O operador Seleção é aplicado duas vezes a cada geração, a primeira quando selecionamos os indivíduos para gerar descendentes e a segunda, quando selecionamos os indivíduos que passarão para a próxima geração na união entre pais e filhos. Através deste procedimento, aceleramos a convergência do método garantindo a seleção dos melhores indivíduos a cada geração.

3.2 SPEA2

Dada a natureza dos problemas de programação multiobjetivo, vimos que a grande maioria dos algoritmos clássicos consiste em transformar o modelo em um problema mono-objetivo e aplicar métodos já conhecidos para obter o Conjunto Pareto. Na figura (3.5), podemos notar que o algoritmo SPEA2 possui uma estrutura diferente de um algoritmo genético clássico. Neste contexto, um ramo da computação evolutiva que vem ganhando destaque é o campo dos *Algoritmos Evolutivos Multiobjetivos* (AEMO). Nesta seção descrevemos os principais aspectos do Algoritmo SPEA2 [29].

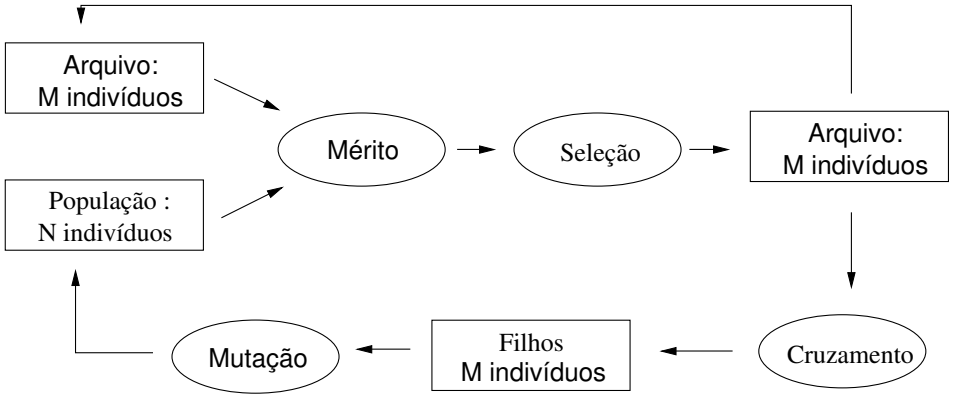


Figura 3.5: Estrutura geral: SPEA2.

3.2.1 Visão Geral

Apesar do SPEA2 contar com o mesma estrutura básica de um AG, existem outros detalhes e modificações que tornam este algoritmo um pouco mais complexo. Dado que nosso objetivo é obter uma boa aproximação para a Fronteira Pareto, desejamos obter uma quantidade razoável de soluções diversificadas e, minimizar a distância entre as mesmas.

Tal aproximação já é por si mesma, multiobjetivo e conflitante. O primeiro aspecto adicional considerado consiste em não perder boas soluções durante as gerações do processo. Esta estratégia é chamada de *Elitismo*.

Devemos garantir que se obtemos uma solução da Fronteira Pareto durante a execução

do algoritmo, esta fique armazenada durante toda a execução do algoritmo. Além do elitismo, desejamos obter uma boa diversificação das soluções. Para efetuar este processo, são usadas funções ou estimadores de densidade.

O aspecto mais importante considerado em um AEMO é a função de mérito, pois, é a partir dela que as soluções serão classificadas. Daí sua importância, pois, o valor de mérito será utilizado durante processos de recombinação e Mutação. Vamos descrever cada um deste processos com mais detalhes nas seções seguintes.

3.2.2 Diversidade

Muitos métodos evolutivos incorporam em suas funções de mérito estimadores de densidade para obter uma distribuição uniforme da Fronteira Pareto. Como estamos trabalhando com métodos que atuam diretamente com a aproximação da Fronteira Pareto, é natural que o conjunto de soluções deve atingir o máximo de diversidade possível para aumentar o número de escolhas do decisor. Eis alguns dos estimadores mais utilizados por tais métodos [19]:

1. Núcleo;
2. Histograma;
3. Vizinhança Local.

O primeiro estimador define uma vizinhança em torno de um indivíduo i e sua distância Euclidiana entre outros indivíduos j é calculada e posteriormente são somadas. Esta soma representa o estimador de densidade para o indivíduo. O histograma define uma grade em torno de um indivíduo e o estimador de densidade é dado pelo número de pontos contidos nessa grade. Tal grade pode ser fixa ou variar de acordo com cada geração, o que torna o estimador mais robusto e eficiente. O último dos estimadores, vizinhança local, calcula a distância entre todos os indivíduos e a k-ésima distância é escolhida. O estimador é dado pelo inverso da distância escolhida. Este último é o escolhido para ser aplicado durante

a otimização com o SPEA2. A medida de distância utilizada neste trabalho é calculada usando a distância Euclidiana no espaço dos objetivos.

3.2.3 Elitismo

O elitismo está relacionado com o fato de boas soluções se perderem durante o processo de otimização. Perder uma boa solução significa maior tempo gasto na otimização do problema. Uma maneira de contornar este problema é combinar a população atual e seus descendentes aplicando algum processo determinístico para montar a nova população da geração seguinte.

Uma alternativa é criar uma segunda população chamada de Arquivo, onde os melhores indivíduos são armazenados e a cada geração este Arquivo vai sendo atualizado. No caso de um AEMO, o critério de seleção consiste em comparar a dominância entre as soluções e aqueles que não são dominantes entre si serão copiados para o Arquivo. Durante o processo de otimização, somente o Arquivo participa do processo de seleção para formar a população seguinte.

Os critérios baseados em dominância são aplicados em conjunto com o estimador de densidade adotado. Ao final do algoritmo as soluções contidas no Arquivo são declaradas como a aproximação final do processo de otimização.

3.2.4 Função de Mérito

A função de mérito no caso de um AEMO deve levar em conta os diversos objetivos do problema. A forma de tratamento dos objetivos pela função de mérito está diretamente ligada com a eficiência do algoritmo, logo, devemos ter cautela em nossa escolha. Algumas funções de mérito usam critérios como soma ponderada ou alternam entre os objetivos durante o processo de otimização. O critério adotado pelo SPEA2 é baseado no conceito de dominância onde, para cada indivíduo é levado em conta tanto as soluções dominadas, quanto seus dominadores.

Para cada indivíduo i no Arquivo A_t e na População P_t, é atribuído um valor de força $S(i)$ que representa o número de soluções dominadas por este indivíduo:

$$S(i) = |\{j | j \in P_t \cup A_t \wedge i \succ j\}|, \tag{3.2.1}$$

onde $|.|$ indica a cardinalidade do conjunto e $\succ$ indica a dominância com relação a outros indivíduos j. A partir deste valor, calculamos o valor de mérito simples:

$$R(i) = \sum_{j \in P_t \cup A_t, j \succ i} S(j). \tag{3.2.2}$$

O valor de mérito simples é calculado pela soma da força de seus dominadores considerando a População e Arquivo. Nota-se que $R(i) = 0$ indica um indivíduo que não é dominado por nenhum outro, enquanto que um valor alto de $R(i)$ indica uma solução dominada por muitos indivíduos.

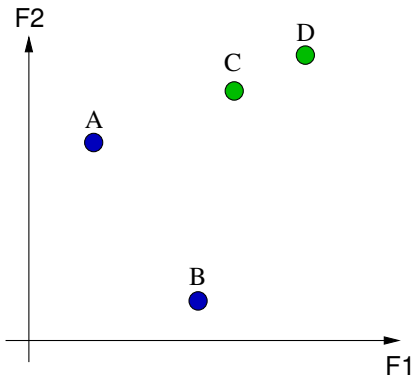


Figura 3.6: Exemplo de soluções.

Na figura (3.6) temos um exemplo de conjunto de soluções. De acordo com as definições acima temos:

- S(A)=2: {C,D}, S(B)=2: {C,D}, S(C)=1: {D} e S(D)=0.
- R(A)=0, R(B)=0, R(C)=4: {A,B} e R(D)=5: {A,B,C}.

Ainda que o valor de mérito simples seja adequado para este tipo de avaliação, este pode falhar se em determinado momento a maioria dos indivíduos não for dominantes

entre si. Para contornar este problema, incorporamos o valor de densidade em nossa avaliação:

$$D(i) = \frac{1}{\sigma_k^i + 2} \quad . \tag{3.2.3}$$

Para cada indivíduo i, as distâncias Euclidianas para todos indivíduos j no espaço dos objetivos é calculada e ordenada de forma crescente em uma lista. Então, o k-ésimo elemento da lista fornece o valor de σ_k^i. Adotamos neste trabalho, como padrão, o valor $k = \sqrt{\text{Total de Indivíduos}}$.

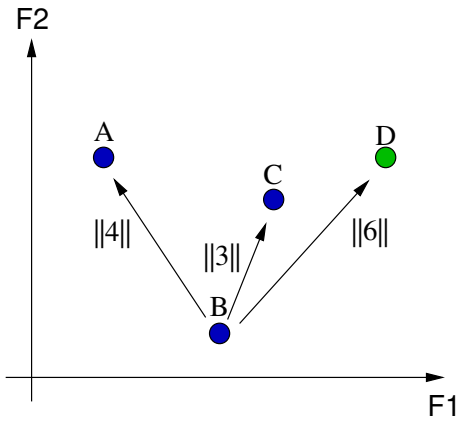


Figura 3.7: Exemplo de distâncias entre as soluções.

Na figura (3.7) temos um exemplo do cálculo destas distâncias com relação a uma solução (B). O vetor de distâncias para esta solução é $dist_B = (3, 4, 6)$. Como $k = 2$, temos $\sigma_2^B = 4$ de acordo com o vetor de distâncias.

Agora, podemos calcular do valor de mérito para cada indivíduo como:

$$F(i) = R(i) + D(i) \tag{3.2.4}$$

Um valor de $F(i) < 1$ indica quais soluções são não dominadas, sendo estas, as soluções que fazem parte da Fronteira Eficiente.

3.2.5 Seleção Ambiental

A troca de indivíduos entre a População P_t e o Arquivo A_t é feita através de um processo determinístico. O Arquivo A_{t+1} para a próxima geração é preenchido da seguinte forma:

$$A_{t+1} = \{i | i \in P_t \cup A_t \wedge F(i) < 1\} \tag{3.2.5}$$

Se o número de soluções não-dominadas tem exatamente a cardinalidade do Arquivo ($|A_{t+1}| = N$), então a seleção ambiental está concluída. Agora, veremos os casos restantes, onde o número de soluções não-dominadas é insuficiente ($|A_{t+1}| < N$) ou sobram soluções ($|A_{t+1}| > N$). No primeiro caso, completamos as $(N - |A_{t+1}|)$ com soluções dominadas presentes na união da População anterior com seu respectivo Arquivo ($P_t \cup A_t$). As $(N - |A_{t+1}|)$ primeiras soluções são copiadas completando-se o Arquivo A_{t+1}.

No segundo caso, o processo de truncagem é feito como base no estimador de densidade. Neste processo, priorizamos as soluções que possuem maior distância ao seu k-ésimo vizinho, com o objetivo de obter um conjunto de soluções não-dominadas mais variadas para melhor representar a Fronteira Eficiente. De acordo com a definição da função de densidade em (3.2.3), quanto maior essa distância, menor é o valor de mérito de um indivíduo. Após ordenarmos os indivíduos não-dominantes, estes vão sendo transferidos para o Arquivo A_{t+1} até ser completamente preenchido ($|A_{t+1}| = N$).

3.2.6 Estrutura do SPEA2

A estrutura do algoritmo SPEA2 segue o padrão de um AG simples mas com algumas modificações em relação ao processo de seleção. O algoritmo segue o esquema apresentado abaixo:

SPEA2 - Estrutura do Algoritmo

Dados de entrada:

- M - Tamanho da População.
- N - Tamanho do Arquivo, com $N < M$.
- T_{max} - Número máximo de gerações.

1. Inicialização: Criar uma População inicial P_0 e um Arquivo vazio A_0. Faça t=0.
2. Valor de mérito: Calcular o valor de mérito para os indivíduos em P_t e A_t.
3. Seleção ambiental: Copiar todos os indivíduos não-dominados em A_t e P_t para A_{t+1}. Se $|A_{t+1}| < N$, completar com soluções dominadas de A_t e P_t. Caso contrário, reduzir o tamanho do Arquivo através do processo de truncagem.
4. Critério de parada: Se $t = T_{max}$ ou outro critério de parada é satisfeito, faça $A^* = A_{t+1}$ e finalizar o algoritmo.
5. Torneio: Promover o Torneio no Arquivo com reposição para escolher os indivíduos que irão gerar descendentes.
6. Variabilidade: Aplicar a recombinação e a mutação nos indivíduos selecionados no passo anterior e atribuir o resultado para P_{t+1}. Faça $t = t + 1$ e volte ao Passo 2.

Capítulo 4

Aplicação ao Problema de Corte

Neste capítulo, daremos detalhes sobre a aplicação do Algoritimo SPEA2 para o problema de corte multiobjetivo. Apresentamos os detalhes sobre a codificação das soluções e descrevemos os operadores de variabilidade genética Mutação e Cruzamento aplicados a nossa estrutura de codificação.

4.1 Estrutura do Algoritmo

Como foi visto nas seções anteriores, os AGs seguem de maneira semelhante a mesma estrutura de implementação. A codificação do problema e os operadores genéticos seguem as definições trabalhadas em Falkenauer [6] e Leduino [22]. Neste trabalho escolhemos a representação em grupo para fazer a implementação do problema.

4.1.1 Representação em Ordem

A representação de uma solução factível para o problema de corte multiobjetivo segue as definições contidas em Falkenauer [6]. Para a representação em ordem, construímos uma lista com todos os itens que serão cortados e, cada corte é feito de forma que o tamanho dos itens acumulados não ultrapasse o tamanho da peça para corte em questão. Este método heurístico de codificação é o mesmo desenvolvido em [18]. O exemplo de representação

a seguir ilustra o processo considerando que dispomos de um número suficiente de peças para realizar os cortes.

Tamanhos	3	4	5	6	7	8
Demanda	1	1	6	2	1	1

Tabela 4.1: Tabela de Pedidos.

Cromossomo: $(5:4:5:6:7:8:6:5:5:8:5:5:3)$.

Tamanhos de peças disponíveis: 15.

Itens	Peça
5,4,5	15
6,7	15
8,6	15
5,5	15
8,5	15
5,3	15

Tabela 4.2: Codificação em ordem.

Quando dispomos de peças em estoque de tamanhos variados, o processo de codificação se torna um pouco mais complexo. Primeiro, criamos uma lista com os itens pedidos. Em seguida, tomamos a maior peça disponível para corte e efetuamos um corte de maneira que o número de itens acumulados não ultrapasse a peça. Agora com este grupo em mãos, recalculamos o desperdício para as outras peças disponíveis. Aquela que obtiver o menor desperdício forma um novo padrão corte. Considere o seguinte exemplo:

Cromossomo: $(5:4:5:6:7:8:6:5:5:8:5:5:3)$.

Tamanhos de peças disponíveis: 12,13,22.

Primeiro, escolhemos a maior peça, que tem tamanho 22. Para esta peça, de acordo com a definição de corte para representação em ordem, obtemos $\{5,4,5,6\}$ como grupo e desperdício 2. Deste grupo, tomamos a peça de tamanho 12 e obtemos $\{5,4\}$ com

Itens	Peça
5,4,5,6	22
7,8,6	22
5,5	12
8,5	13
5,3	12

Tabela 4.3: Codificação em ordem para peças variadas em estoque.

desperdício igual a 3. Para a peça de tamanho 13 obtemos $\{5, 4\}$ com desperdício 4. Com base nestas informações, o melhor corte é $\{5, 4, 5, 6\}$ pois é o que apresenta menor desperdício. O resultado final deste processo é apresentado na tabela (4.3).

4.1.2 Operadores de Variabilidade Genética

Os operadores Mutação e Cruzamento são aplicados para promover de forma gradativa e eficiente, uma melhoria na qualidade das soluções ou indivíduos ao longo das gerações. Para o operador Cruzamento, aplicamos o *Order Based Crossover* (OBX), cujo foco se baseia na ordem relativa das informações. Este operador também é aplicado ao Problema do Caixeiro Viajante em [20], que trabalha da seguinte maneira: selecionamos um número pré-determinado de posições com base nas fórmulas de probabilidades definidas em (4.1.1) para aplicamos uma permutação aleatória das mesmas em ambos os pais. O esquema abaixo ilustra o processo:

$$Pai_1 = \boxed{\text{A}}\boxed{\text{B}}\boxed{\text{C}}\boxed{\text{D}}\boxed{\text{E}}\boxed{\text{F}}\boxed{\text{G}} \quad ,$$

$$Pai_2 = \boxed{\text{C}}\boxed{\text{E}}\boxed{\text{G}}\boxed{\text{A}}\boxed{\text{D}}\boxed{\text{F}}\boxed{\text{B}} \quad .$$

Assuma como escolhidas as posições $\{2, 4, 5\}$ e que a permutação escolhida seja:

$$5 \longrightarrow 4$$
$$4 \longrightarrow 2$$
$$2 \longrightarrow 5 \quad .$$

Assim, obtemos dois novos indivíduos:

$$Filho_1 = \boxed{\text{A}}\boxed{\text{D}}\boxed{\text{C}}\boxed{\text{B}}\boxed{\text{E}}\boxed{\text{F}}\boxed{\text{G}} \quad ,$$

$$Filho_2 = \boxed{\text{C}}\boxed{\text{E}}\boxed{\text{G}}\boxed{\text{D}}\boxed{\text{A}}\boxed{\text{F}}\boxed{\text{B}} \quad .$$

O operador OBX permite promover a variabilidade genética da População sem a necessidade de verificar se o novo indivíduo é factível, pois, a permutação dos genes mantém a nova solução factível. Outros operadores como *OX Crossover* e *Position-Based Crossover* [20], efetuam a troca de genes entre os pais para gerar os filhos. Neste caso, é comum obter um indivíduo não factível, pois, retirar partes de ambos os pais e combiná-los pode criar um novo vetor com um número de itens acima ou abaixo da demanda. Considere o problema abaixo com peças de tamanho 12 disponíveis para corte como exemplo:

Itens	3	5	6	7
Quantidades	2	2	2	1

Sendo os pais escolhidos em uma determinada geração dados por:

$$Pai_1 = \boxed{5}\boxed{3}\boxed{6}\boxed{6}\boxed{3}\boxed{7}\boxed{5} \quad ,$$

$$Pai_2 = \boxed{6}\boxed{7}\boxed{6}\boxed{5}\boxed{3}\boxed{3}\boxed{5} \quad .$$

Assumindo que as genes escolhidos são $\{3, 4\}$, o primeiro filho é obtido efetuando-se a troca de material genético entre os pais selecionados:

$$Filho_1 = \boxed{6}\boxed{7}\boxed{6}\boxed{6}\boxed{3}\boxed{3}\boxed{5} \quad .$$

Trata-se de uma solução infactível, pois, há um item a mais na lista (6) e um a menos (5). Para tornar factível este indivíduo, é necessário aplicar um procedimento de correção. Por este motivo, o operador Mutação é alterado para se encaixar de maneira eficiente em nossa definição.

A Mutação adotada difere do conceito clássico, pois, ao invés de modificar um determinado gene do cromossomo, o procedimento é baseado no conceito de troca (Swap) entre uma posição e outra. Dado um indivíduo, todas as posições do vetor são percorridas com uma probabilidade P_j relativo ao gene j. O cálculo de P_j é feito de forma que um item em um padrão de corte que apresenta desperdício elevado tenha maior probabilidade de ser escolhido. O valor de P_j é definido em [10] por:

$$P_j = \frac{(1 - \frac{k}{k_{max}})\beta w_j - w_{max}}{\sum_{i=1}^{n}((1 - \frac{k}{k_{max}})\beta w_i + w_{max})} \quad , \tag{4.1.1}$$

onde k_{max} é o número máximo de gerações, w_j é o desperdício do padrão de corte j, w_{max} é o maior desperdício presente na solução e β é um fator de escala, sendo adotado em [18] como 100. Caso o gene j seja escolhido, ocorrerá uma troca de posição entre o gene i com o gene j, sendo $i \neq j$. Um exemplo de permutação é mostrado no indivíduo Υ a seguir:

$$\Upsilon = \boxed{7}\boxed{12}\boxed{1}\boxed{5}\boxed{8}\boxed{9}\boxed{5} \quad .$$

Assuma que a posição escolhida para troca seja a $2^{\underline{a}}$, que será permutada com a $7^{\underline{a}}$, escolhida aleatóriamente. O novo indivíduo então é:

$$\Upsilon' = \boxed{7}\boxed{5}\boxed{1}\boxed{5}\boxed{8}\boxed{9}\boxed{12} \quad .$$

Normalmente este procedimento é realizado duas vezes com o mesmo item i, e portanto, temos duas trocas de posições. Tal procedimento é denominado *3-Point Swap (3PS)*.

Outra variação de Mutação consiste em escolher um gene j aleatóriamente, removê-lo e, o inserir após um gene semelhante i, ou seja, escolhemos um item e percorremos o vetor solução até encontrar um item do mesmo tipo para que este seja inserido logo a frente. Considere o indivíduo a seguir,

$$\pi = \boxed{7}\boxed{8}\boxed{1}\boxed{5}\boxed{8}\boxed{9}\boxed{5} \quad .$$

Caso a posição 2 tenha sido escolhida, inserimos o item 8 após um item semelhante obtendo o seguinte indivíduo,

$$\pi' = \boxed{7}\boxed{1}\boxed{5}\boxed{8}\boxed{8}\boxed{9}\boxed{5} \quad .$$

Este operador é denominado *Search and Reinsert* (SRI). Ambos tipos de Mutação (3PS e SRI) são utilizados em conjunto no algoritmo.

4.1.3 Representação em Grupo

Neste tipo de codificação, cada gene do cromossomo representa um padrão de corte. As idéias apresentadas em [6] e [22] são adaptadas para as características do problema de corte. A grande vantagem deste tipo de codificação em relação a representação em ordem é evitar a redundância de representação de soluções. Considerando um problema padrão, assumimos que dispomos de um número suficiente de peças de tamanho L para satisfazer a os pedidos da tabela (4.4).

Itens	w_1	w_2	...	w_m
Quantidades	d_1	d_2	...	d_m

Tabela 4.4: Tabela de Pedidos.

A idéia da codificação em grupo consiste em conservar as melhores soluções ao longo das gerações do algoritmo, pois, o que determina os custos de setup e desperdício é o conjunto de padrões de corte presente na solução e não a ordem relativa dos itens cortados. Os trabalhos de Falkenauer [6] e Leduino [22] mostram que este tipo de codificação é a melhor abordagem para estes tipos de problema. A codificação ideal para este problema seria trabalhar com um número de padrões de corte igual a m ou menor. Esta não é uma tarefa complicada quando estamos trabalhando com folga, ou seja, quando pode haver excesso de demanda, porém, estamos trabalhando sem folgas, atendendo exatamente a demanda. Por este motivo, gerar no máximo m padrões de corte para atender toda a demanda na igualdade se torna uma tarefa difícil.

Por este motivo, trabalharemos com no máximo $2 \times m$ padrões de corte, pois, nos primeiros m padrões aplicamos uma heurística para gerar estes padrões iniciais e, em seguida, corrigimos os itens restantes nos m padrões de corte posteriores. Assim, teremos certeza que a demanda é atendida sem haver sobras. Como exemplo, considere que temos um número suficiente de peças de tamanho 12 para atender as demandas abaixo,

Itens	5	6	7
Quantidades	2	3	1

Tabela 4.5: Tabela de Pedidos.

Cada gene do cromossomo contém $m = 3$ elementos, pois, cada elemento indica quantas vezes cada item é cortado no padrão em questão. Considere o exemplo de cromossomo a seguir,

$$\text{Cromossomo: } [(2,0,0)|(0,2,0)|(0,0,1)|(0,1,0)|(0,0,0)|(0,0,0) : 1,1,1,1,0,0].$$

No problema acima temos três tamanhos de itens para corte, assim, o cromossomo possuí seis genes, cada gene com três elementos e mais seis posições para informar quantas vezes cada padrão de corte é usado. Portanto, se temos m padrões de corte, o vetor que representa o cromossomo terá $2m^2 + 2m$ elementos. A parte importante do cromossomo está nos m primeiros padrões de corte, pois, gostaríamos que no máximo fossem usados m padrões. Os outros m padrões são usados para eventuais correções como é mostrado no cromossomo acima, uma vez que, após os três primeiros padrões, ainda restou cortar um item de tamanho 6, atendendo assim, toda a demanda na igualdade. Para gerar as soluções iniciais, utilizamos uma adaptação da heurística *First Fit* (FF).

4.1.4 First Fit Adaptada

A heurística First Fit descrita em [6] consiste em escolher os itens de uma lista e colocá-los no primeiro padrão disponível. A diferença em nossa implementação está no fato de ao formarmos um padrão, usarmos ele o máximo de vezes possível. Vamos utilizar o cromossomo gerado acima como exemplo. Para cada gene do cromossomo, procedemos da seguinte maneira,

First Fit Adaptada: Geração da População Inicial

1. Cada elemento do gene representa o número de vezes que cada item será cortado no padrão de corte.

2. Para o primeiro elemento $x(1)$, escolhemos uniformemente um número entre 1 e $d(1)$, onde, $d(1)$ indica quantas vezes o item 1 deve ser cortado. O mesmo raciocínio vale para um elemento $x(i)$ de um determinado gene, ou seja, geramos um número de forma uniforme entre 1 e $d(i)$.

3. Assuma que o valor escolhido do primeiro elemento seja $x(1) = 2$. Agora, verificamos se essa quantidade não viola o tamanho máximo dos padrões que é $L = 12$ no caso. Dado que o primeiro item tem tamanho 5, temos $2 \times 5 = 10 < 12$, ou seja, não há problema com este valor. Caso houvesse problema, diminuiríamos este valor até o corte do padrão ficar dentro do limite da peça ou o consideraríamos zero caso isto não seja possível. Para os dois elementos seguintes, a única possibilidade é zero, pois, isto não viola o corte máximo do padrão em questão.

4. Desta maneira, temos o primeiro gene: $(2, 0, 0)$. Resta agora contarmos quantas vezes podemos utilizá-lo. No caso, notamos que é possível utilizar este padrão somente uma vez para não ocorrer excesso de demanda. A seguir, atualizamos os valor do vetor de demandas d. Após o primeiro padrão, temos $d = (0, 3, 1)$ como demanda restante. Para os próximos genes, repetimos o processo descrito acima.

5. Caso seja obtida uma solução onde tenhamos utilizados os $2 \times m$ padrões de corte e ainda resta demanda a ser satisfeita, eliminamos a solução e repetimos o processo novamente, pois, estamos interessados em soluções com no máximo $2 \times m$ padrões.

4.1.5 Cruzamento em Grupos

O operador cruzamento atua nos m primeiros padrões de corte, deixando os m padrões de corte restantes para efetuar as correções, se necessário. Neste procedimento, escolhemos um ponto entre os m primeiros padrões de corte e, efetuamos a troca de genes entre os pais selecionados para obter dois novos indivíduos. Os genes restantes servem para efetuar a correção, tornando os novos indivíduos factíveis se necessário. Vamos ilustrar este processo considerando os indivíduos selecionados abaixo,

$$Pai_1\text{: } [(2,0,0)|(0,1,0)|(0,0,1)|(0,0,0)|(0,0,0)|(0,0,0) : 1,3,1,0,0,0].$$

$$Pai_2\text{: } [(2,0,0)|(0,2,0)|(0,0,1)|(0,1,0)|(0,0,0)|(0,0,0) : 1,1,1,1,0,0].$$

Para fazer o cruzamento, podemos escolher dois pontos de corte entre os três padrões de corte principais. Suponha que o ponto escolhido seja o primeiro. A parte destinada a correção por enquanto será completada com zeros, juntamente com a parte onde são contados os padrões de corte. Então, o primeiro filho recebe o primeiro gene do Pai_1 e os outros dois do Pai_2,

$$Filho_1\text{: } [(2,0,0)|(0,2,0)|(0,0,1)|(0,0,0)|(0,0,0)|(0,0,0) : 0,0,0,0,0,0].$$

Feito este primeiro passo, falta corrigir a solução para que ela se torne factível se necessário. Primeiro, percorremos os genes verificando se não há excesso de demanda, caso haja, diminuímos os elementos dos genes em questão até que a demanda seja factível. Após o gene ter sido corrigido, contamos quantas é possível utilizar este padrão de corte e atualizamos o vetor de demandas. Este processo é repetido para os m primeiros genes do indivíduo. Após este primeiro passo, o indivíduo possui o seguinte aspecto,

$$Filho_1\text{: } [(2,0,0)|(0,2,0)|(0,0,1)|(0,0,0)|(0,0,0)|(0,0,0) : 1,1,1,0,0,0].$$

Caso a demanda não seja atendida até este processo, por exemplo, ainda temos $d(2) = 1$, onde d é o vetor de demandas, então o padrão 2 após os 3 primeiros recebe 1 na posição 2 e é utilizado 1 vez. Para o primeiro filho obtemos o seguinte cromossomo,

$$Filho_1\text{: } [(2,0,0)|(0,2,0)|(0,0,1)|(0,0,0)|(0,1,0)|(0,0,0) : 1,1,1,0,1,0].$$

Repetindo este processo para o segundo filho obtemos,

$$Filho_2\text{: } [(2,0,0)|(0,1,0)|(0,0,1)|(0,0,0)|(0,0,0)|(0,0,0) : 1,3,1,0,0,0].$$

O processo de crossover pode ser resumido da seguinte maneira: escolhemos um ponto de troca entre os m primeiros genes que representam os m primeiros padrões e, em seguida, efetuamos a troca dos genes após o ponto de corte até o gene m e, corrigimos as sobras de demandas restantes. Se $d(j) = k$, fazemos o padrão j após os m primeiros receber 1 na posição j e este será utilizado k vezes. Portanto, cada padrão extra é reponsável pela correção de um item ainda em aberto. Este processo de correção também é aplicado durante o processo de mutação.

4.1.6 Mutação em Grupos

A mutação é um processo feito de maneira semelhante ao cruzamento. Dada uma probabilidade P_{mut}, percorremos cada gene do cromossomo que será alterado de acordo com esta probabilidade. Considere o indivíduo abaixo,

$$I\text{: } [(2,0,0)|(0,1,0)|(0,0,1)|(0,0,0)|(0,0,0)|(0,0,0) : 1,3,1,0,0,0].$$

Caso o segundo gene seja escolhido para sofrer a mutação, todos os elementos diferentes de zero serão diminuídos de uma unidade. Assim temos o indivíduo intermediário,

$$I'\text{: } [(2,0,0)|(0,0,0)|(0,0,1)|(0,0,0)|(0,0,0)|(0,0,0) : 0,0,0,0,0,0].$$

Agora, aplicamos a mesma correção feita para o cruzamento para obtermos o novo indivíduo,

$$F\text{: } [(2,0,0)|(0,0,0)|(0,0,1)|(0,0,0)|(0,1,0)|(0,0,0) : 1,0,1,0,3,0].$$

Tornamos então o novo indivíduo factível que passa a fazer parte da nova população.

4.2 Seleção Ambiental

Cabe ao processo de Seleção Ambiental determinar quais indivíduos são os melhores e que seguirão para a próxima geração. É nesta parte que o algoritmo SPEA2 difere de outros AGs. Como já foi descrito no capítulo anterior, o algoritmo trabalha operando na População e em um Arquivo onde são armazenadas as soluções mais promissoras, no caso, soluções não-dominadas. Nesta fase do algoritmo, a seleção é feita baseada no valor de mérito das soluções, conforme foi discutido no capítulo anterior. Um aspecto em relação a Seleção Ambiental que deve ser abordado diz respeito ao processo de determinar quais indivíduos serão selecionados para gerar descendentes. O papel de executar tal procedimento pertence ao Torneio. A forma mais simples de torneio é escolher um número pré-determinado de indivíduos e, aquele que possuir o menor valor de mérito (3.2.4) será declarado vencedor. O número de indivíduos que disputam o torneio tem grande influência na convergência do algoritmo. Abaixo temos o procedimento aplicado na execução do torneio no SPEA2.

Algoritmo Torneio

- M - Tamanho da População.
- N - Tamanho do Arquivo ($N < M$).
- Dim_T - Número de indivíduos escolhidos para competir no Torneio.

1. Selecionar Dim_T indivíduos em A_t aleatóriamente e escolher aquele que tiver o melhor valor de mérito (3.2.4). Esta seleção é feita com reposição.
2. Aplicar o torneio M vezes, pois, os M indivíduos selecionados do Arquivo irão gerar M descendentes que serão transferidos para a nova População.

4.3 Cálculo do Valor de Mérito

Durante processo de codificação e ordem, é possível obter todas as informaçãos necessárias para avaliar um indivíduo e lhe atribuir um valor de adequabilidade com relação a outras soluções utilizando a função de mérito (3.2.4) definida no capítulo anterior, baseada no conceito de dominância entre os indivíduos.

4.4 Condição de Parada

A condição de parada usualmente adotada durante a execução de um AG é baseada no número máximo de iterações pré-determinado pelo usuário. Outra alternativa comum como critério de parada consiste em observar a variação do valor de mérito do melhor indivíduo da População. Se o valor permanecer constante durante um certo número de gerações, o algoritmo é encerrado, pois, assume-se que o ótimo local ou global já foi atingido. No caso do SPEA2, este último critério não tem sentido de aplicacão, pois, o objetivo é trabalhar no espaço das funções objetivo para obter uma aproximação da Fronteira Pareto. Assim, não há um valor de mérito como em um AG simples. Temos uma variedade de valores de mérito para os pontos de tal aproximação que variam de geração em geração. Adotamos então como critério de parada o número máximo de iterações.

4.5 Detalhes de Implementação

Nesta secão apresentamos a seqüência estruturada do algoritmo SPEA2 aplicada ao problema de corte.

SPEA2: Aplicação ao Problema de Corte

Dados de entrada:

- M - Dimensão da População P_t.
- N - Dimensão do Arquivo A_t, com $N < M$.
- T - Vetor contendo os itens pedidos.
- Q - Vetor contendo as respectivas quantidades de cada item em T_i.
- L - Vetor que contém as opções de peças.
- T_{max} - Número máximo de iterações do algoritmo.

1. Escolher quais funções objetivo serão otimizadas durante a execução do algoritmo.
2. Gerar a População inicial P_t com base nos dados contidos em T e Q e criar um Arquivo vazio A_t. Faça $t = 0$.
3. Calcular o valor de mérito dos indivíduos presentes em P_t e A_t.
4. Copiar os indivíduos não dominados para A_{t+1}.
5. Aplicar o Torneio para selecionar os melhores indivíduos em A_{t+1}.
6. Nos indivíduos selecionados, aplicar o Cruzamento em Grupos e a Mutação em Grupos.
7. Copiar os descendentes para a nova População P_{t+1}.
8. Passar para próxima geração fazendo $t = t + 1$ e retornar ao passo 3.

Capítulo 5

Resultados Numéricos

5.1 Experimentos Computacionais

Os testes numéricos realizados utilizaram a linguagem de programação MATLAB 7.0, implementados em uma máquina com processador AMD ATHLON 3.2 GHz, 512 de memória RAM e sistema operacional Windows XP.

Alguns dos problemas usados nos testes foram gerados através do **Cutgen1** [31], enquanto outros pertencem a uma classe específica dos problemas **Fiber** [22]. Para todos os testes realizados, adotamos como critério de parada um número de iterações pré-definido. Para efeitos de padronização, o valor adotado foi de 400 iterações.

Para a execução dos testes, foram feitos alguns ajustes que têm por objetivo evitar erros de precisão e tornar sua execução mais simples para que possamos extrair o máximo de informações sobre a performance do método. A seguir, temos os ajustes computacionais adotados:

1. Adotamos o conceito de dominância fraca, ou seja, dados dois pontos no espaço dos objetivos $x = (x_1, ..., x_n)$ e $y = (y_1, ..., y_n)$, dizemos que x domina fracamente y se $x_i \leq y_i$, para $i = 1, ..., n$;

2. A segunda População denominada Arquivo, onde são guardadas as melhores soluções,

é exatamente 35% da dimensão da População principal. Assim, a População é composta de 40 indivíduos e o Arquivo de 16 indivíduos;

3. No cálculo da função de mérito (3.2.4) para um indivíduo x, não foi utilizado o valor exato da definição $fit(x) \leq 1$ como critério de dominância, mas sim $fit(x) \leq 1.0002$ como critério para evitar erros numéricos durante a execução do método;

4. Para o operador de seleção Torneio, escolhemos 40% do total da População para competir e adotamos $P_{mut} = 0.3\%$. Também existe um valor mínimo de 2 indivíduos, garantindo o funcionamento do operador para populações menores.

Os critérios e parâmetros adotados nos testes foram escolhidos com base em publicações, tutoriais e livros, nos quais, problemas semelhantes são resolvidos utilizando-se AGs como em [20]. Os parâmetros usados para gerar o problemas através do Cutgen1 são definidos da seguinte maneira,

- m - Tamanho do problema

- $v_1 = \dfrac{min(l_1, ...l_n)}{L}$ Limite inferior relativo

- $v_2 = \dfrac{max(l_1, ...l_n)}{L}$ Limite superior relativo

- $\bar{d} = \dfrac{D}{m}$ Densidade média.

onde l_i é a demanda relativa ao item i, L é o tamanho do rolo mestre e D é a demanda total. Ao todo foram geradas 10 classes, combinando-se os parâmetros da seguinte forma,

- O parâmetro v_1 foi fixado em 0.0001 e v_2 varia entre 0.25 e 1.00.
- A demanda média varia entre baixa $\overline{d} = 05$, média $\overline{d} = 10$ e alta $\overline{d} = 20$.
- O número de itens foi fixado em $m = 25$.

- O tamanho do rolo-mestre foi fixado em $L = 10000$.
- Ao todo são 10 classes, cada uma com 5 problemas.

Desta forma, obtemos 3 classes com itens pequenos ($v_1 = 0.0001, v_2 = 0.25$), 3 classes com itens variados ($v_1 = 0.0001, v_2 = 0.50$) e 4 classes com itens grandes ($v_1 = 0.0001, v_2 = 0.75, v_2 = 1.00$). Para cada classe, foram gerados 5 problemas, totalizando 50. A tabela abaixo resume os parâmetros adotados em cada classe.

Classe	m	L	v_1	v_2	$\overline{d}$	Semente
01	25	10000	0.0001	0.25	05	2502505
02	25	10000	0.0001	0.25	10	2502510
03	25	10000	0.0001	0.25	20	2502505
04	25	10000	0.0001	0.50	05	2505005
05	25	10000	0.0001	0.50	10	2505510
06	25	10000	0.0001	0.50	20	2505520
07	25	10000	0.0001	0.75	05	2507505
08	25	10000	0.0001	0.75	10	2507510
09	25	10000	0.0001	0.75	20	2507520
10	25	10000	0.0001	1.00	05	2510005

Tabela 5.1: Classes geradas e seus respectivos parâmetros.

Os problemas Fiber são divididos em dois tipos de acordo com o tamanho da peça disponível para corte (5180 ou 9080). Estes tipos são iguais com relação ao tamanho dos problemas, sendo a diferença dada pelo tamanho do Rolo-Mestre. A demanda total de itens pedidos aumenta seqüencialmente para os dois tipos.

Como exemplo de Fronteira Eficiente gerada pelo SPEA2, considere o seguinte problema de corte unidimensional,

Itens	3	4	5	6	7	8	9	10
Quantidades	2	8	5	7	8	5	5	8

Tabela 5.2: Tabela de Pedidos.

Para atender os pedidos acima, dispomos de um número suficiente de peças de tamanho 15. Para resolvermos os problemas deste capítulo, utilizamos as funções definidas em (2.3.5) como objetivos nos testes feitos neste trabalho. A Fronteira Eficiente que contém a porcentagem de desperdício e o custo de setup para o problema é mostrada a seguir,

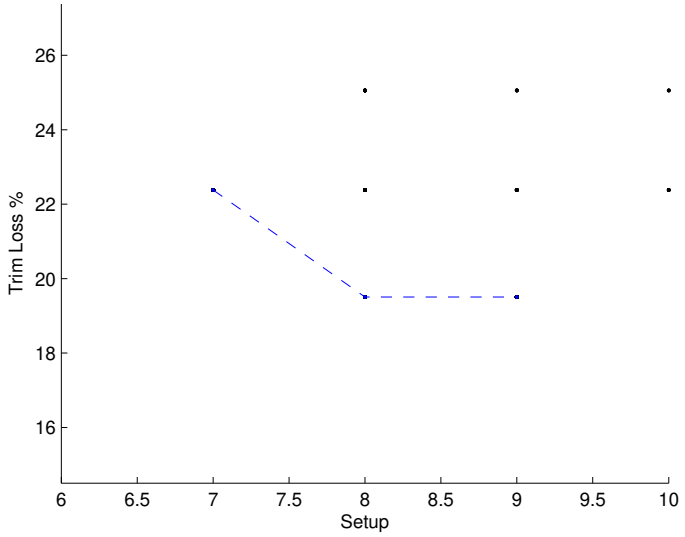


Figura 5.1: Fronteira Eficiente - SPEA2

A seguir, apresentamos os resultados obtidos para as classes geradas pelo Cutgen1 e Fiber respectivamente. As tabelas que comparam nossos resultados com os problemas **Fibers** mostram resultados de outros métodos mas que não atendem necessariamente a igualdade de demanda, pois, as restrições sobre demanda destes métodos não são de igualdade, podendo ocorrer excesso de produção. Ressaltamos que os problemas de corte testados têm suas demandas satisfeitas na igualdade, não havendo excesso de itens produzidos. Portanto, a solução obtida já é inteira não havendo a necessidade de um procedimento de integralização. Exemplos de resultados e algumas das soluções obtidas podem

ser encontradas no Apêndice, para que os resultados obtidos possam ser validados. Ainda no Apêndice, pode ser encontrada uma descrição mais detalhada de alguns problemas.

5.1.1 Resultados Cutgen1

Ao todo foram feitos testes com 10 diferentes classes, cada uma com 5 problemas. Em cada classe é apresentado o conjunto de parâmetros que foram utilizados para gerar seus respectivos problemas. Os resultados obtidos usam o algoritmo SPEA2 implementado nesta trabalho. A Fronteira Eficiente dada pela porcentagem de desperdício (%Trim Loss) e o Custo de Setup é apresentada nas tabelas abaixo.

- **Classe 01** ($v_1 = 0.001, \quad v_2 = 0.25, \quad \overline{d} = 05, \quad m = 25$)

Classe 01 $m = 25$	Setups	%Desperdício
Problema 01	16	2.93
	17	2.93
Problema 02	12	8.94
	13	3.58
	14	3.58
Problema 03	10	3.14
	11	3.14
	12	3.14

- **Classe 01** ($v_1 = 0.001, \quad v_2 = 0.25, \quad \overline{d} = 05, \quad m = 25$)

Classe 01 $m = 25$	Setups	%Desperdício
Problema 04	14	5.23
	15	5.23
Problema 05	12	1.61
	13	1.61
	14	1.61

Tabela 5.3: Setups e porcentagem de desperdícios obtidos Cutgen1 (Classe 01).

- **Classe 02** ($v_1 = 0.001, \quad v_2 = 0.25, \quad \overline{d} = 10, \quad m = 25$)

Classe 02 $m = 25$	Setups	%Desperdício
Problema 01	19	2.29
	20	2.29
	21	2.29
Problema 02	20	3.96
	21	1.30
	22	1.30
	23	1.30
Problema 03	21	2.29
	22	2.29
	23	2.29

- **Classe 02** ($v_1 = 0.001, \quad v_2 = 0.25, \quad \overline{d} = 10, \quad m = 25$)

Classe 02 $m = 25$	Setups	%Desperdício
Problema 04	16	4.30
	17	4.30
	18	4.30
Problema 05	19	0.94
	22	0.94
	23	0.94

Tabela 5.4: Setups e porcentagem de desperdícios obtidos Cutgen1 (Classe 02).

- **Classe 03** ($v_1 = 0.001, \quad v_2 = 0.25, \quad \overline{d} = 20, \quad m = 25$)

Classe 03 $m = 25$	Setups	%Desperdício
Problema 01	26	59.34
	27	24.83
	28	11.47
	31	1.63
Problema 02	25	1.63
	28	1.63
Problema 03	24	1.95
	25	1.95
	30	1.02
Problema 04	27	6.68
	31	3.81
	32	3.81
Problema 05	24	4.91
	25	1.28

Tabela 5.5: Setups e porcentagem de desperdícios obtidos Cutgen1 (Classe 03).

Para os resultados das Classes 01 e 02, as soluções encontradas apresentam o custo de setup abaixo do número básico de padrões m, o que mostra que o método implementado atingiu soluções satisfatórias para as duas primerias classes.

- **Classe 04** ($v_1 = 0.001$, $v_2 = 0.50$, $\overline{d} = 05$, $m = 25$)

Classe 04 $m = 25$	Setups	%Desperdício
Problema 01	19	1.91
	21	1.91
	22	1.91
	24	1.91
Problema 02	17	1.21
	18	1.21
	19	1.21
Problema 03	17	4.23
	18	0.93
	19	0.93
	20	0.93
Problema 04	16	3.46
	17	3.46
	18	3.46
Problema 05	17	2.41
	18	2.41

Tabela 5.6: Setups e porcentagem de desperdícios obtidos Cutgen1 (Classe 04).

Na Classe 03 ocorre o aumento da demanda média de $\overline{d} = 5$ para $\overline{d} = 20$. Os resultados obtidos se encontram um pouco acima do número básico de padrões $m = 25$. Como já havíamos mencionado anteriormente, em alguns problemas, encontrar boas soluções satisfazendo a demanda na igualdade é uma tarefa difícil. Para os problemas da classe 04, temos uma demanda média de $\overline{d} = 5$ e os resultados se encontram nos mesmos padrões das Classes 01 e 02, ou seja, todos abaixo de 25. Mesmo quando a demanda média aumenta,

as soluções obtidas não chegam a ser tão afetadas durante a execução do método.

- **Classe 05** ($v_1 = 0.001$, $v_2 = 0.50$, $\overline{d} = 10$, $m = 25$)

Classe 05 $m = 25$	Setups	%Desperdício
Problema 01	26	7.03
	27	6.01
Problema 02	22	2.96
	23	1.69
	24	1.69
Problema 03	22	8.24
	23	5.04
	24	5.04
Problema 04	22	1.80
	23	1.80
	24	1.80
Problema 05	22	1.62

Tabela 5.7: Setups e porcentagem de desperdícios obtidos Cutgen1 (Classe 05).

- **Classe 06** ($v_1 = 0.001$, $v_2 = 0.50$, $\overline{d} = 20$, $m = 25$)

Classe 06 $m = 25$	Setups	%Desperdício
Problema 01	25	1.16
	27	1.16
Problema 02	28	27.08
	29	24.72
	32	2.58
Problema 03	26	4.72
	30	2.83
	31	1.86
	32	1.86
Problema 04	26	2.92
	27	2.92
	34	2.20
Problema 05	26	41.16
	27	7.11
	28	1.94
	31	1.26

Tabela 5.8: Setups e porcentagem de desperdícios obtidos Cutgen1 (Classe 06).

Nas Classes 05 e 06 ocorre o mesmo fenômeno das classes anteriores, ocorrendo um pequeno aumento no número dos padrões de corte.

- **Classe 07** ($v_1 = 0.001, \quad v_2 = 0.75, \quad \overline{d} = 05, \quad m = 25$)

Classe 07 $m = 25$	Setups	%Desperdício
Problema 01	18	4.18
Problema 02	23	17.44
	24	11.85
Problema 03	18	6.82
	19	4.99
Problema 04	19	11.36
	20	11.36
Problema 05	21	21.19
	22	16.37
	23	4.70

Tabela 5.9: Setups e porcentagem de desperdícios obtidos Cutgen1 (Classe 07).

- **Classe 08** ($v_1 = 0.001, \quad v_2 = 0.75, \quad \overline{d} = 10, \quad m = 25$)

Classe 08 $m = 25$	Setups	%Desperdício
Problema 01	22	12.83
	23	12.10
Problema 02	20	2.41
	21	2.41
	22	2.41

Na Classe 07 o resultado já podia ser esperado, pois, os parâmetros são semelhantes as duas primeiras classes.

- **Classe 08** ($v_1 = 0.001, \quad v_2 = 0.75, \quad \overline{d} = 10, \quad m = 25$)

Classe 08 $m = 25$	Setups	%Desperdício
Problema 03	24	22.15
	25	8.06
	28	8.06
Problema 04	23	11.08
	24	4.06
	25	2.78
	27	2.78
Problema 05	24	3.86

Tabela 5.10: Setups e porcentagem de desperdícios obtidos Cutgen1 (Classe 08).

- **Classe 09** ($v_1 = 0.001, \quad v_2 = 0.75, \quad \overline{d} = 20, \quad m = 25$)

Classe 09 $m = 25$	Setups	%Desperdício
Problema 01	25	46.74
	26	30.49
	27	18.79
	28	8.36
	30	4.63
	32	4.63

- **Classe 09** ($v_1 = 0.001, \quad v_2 = 0.75, \quad \overline{d} = 20, \quad m = 25$)

Classe 09 $m = 25$	Setups	%Desperdício
Problema 02	25	1.48
Problema 03	23	27.62
	24	7.29
Problema 04	23	1.84
	24	1.22
Problema 05	20	3.17

Tabela 5.11: Setups e porcentagem de desperdícios obtidos Cutgen1 (Classe 09).

- **Classe 10** ($v_1 = 0.001, \quad v_2 = 1.00, \quad \overline{d} = 05, \quad m = 25$)

Classe 10 $m = 25$	Setups	%Desperdício
Problema 01	20	5.73
Problema 02	22	15.51
Problema 03	21	10.85
Problema 04	24	19.42
Problema 05	26	13.44
	22	13.44

Tabela 5.12: Setups e porcentagem de desperdícios obtidos Cutgen1 (Classe 10).

Na Classe 10 obtemos bons resultados mas não muito variados. Basta observar a figura (5.11) na próxima seção para notar que as fronteiras eficientes desta classe apresentam em sua maioria somente uma solução não dominada por problema.

5.1.2 Gráficos Cutgen1

A seguir, exibimos um exemplo de Fronteira Eficiente para cada classe de problemas gerados pelo Cutgen1 [31]. Os gráficos foram selecionados com base na distribuição dos pontos

da Fronteira Eficiente, ou seja, são os que apresentam uma variabilidade considerável de pontos uma vez que trata-se de soluções inteiras e portanto, algumas fronteiras podem simplesmente conter apenas uma solução eficiente.

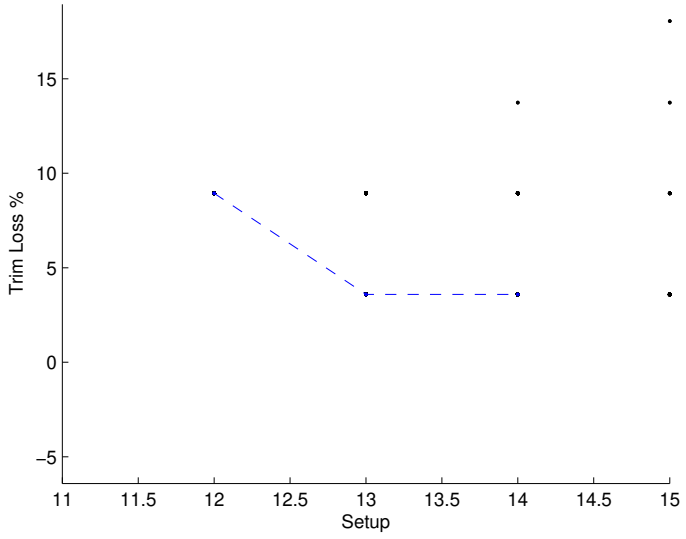


Figura 5.2: Classe 01 - Problema 2

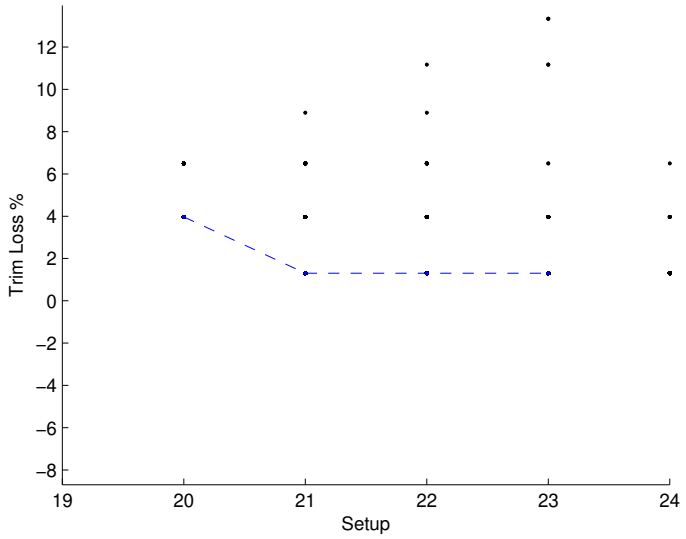


Figura 5.3: Classe 02 - Problema 2

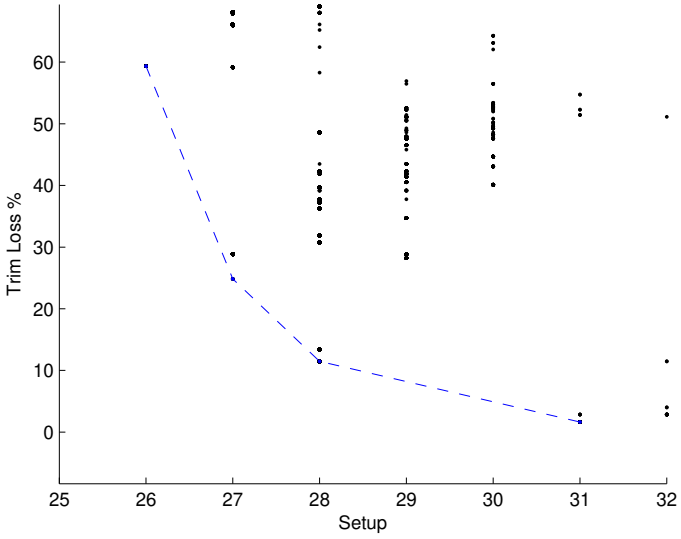


Figura 5.4: Classe 03 - Problema 1

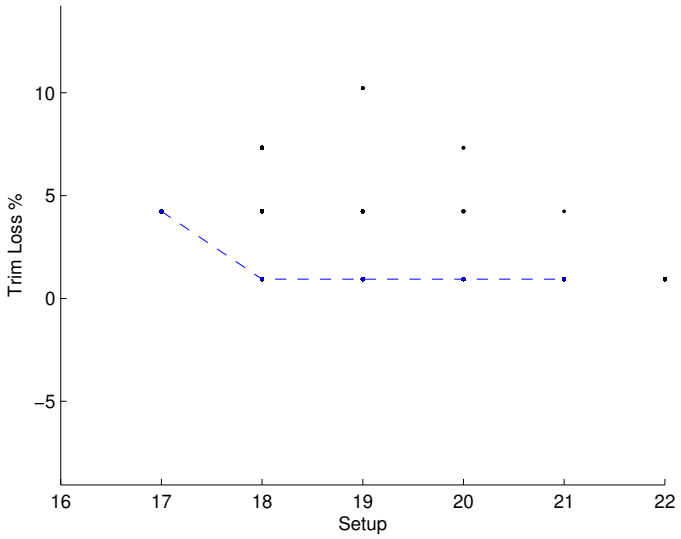


Figura 5.5: Classe 04 - Problema 3

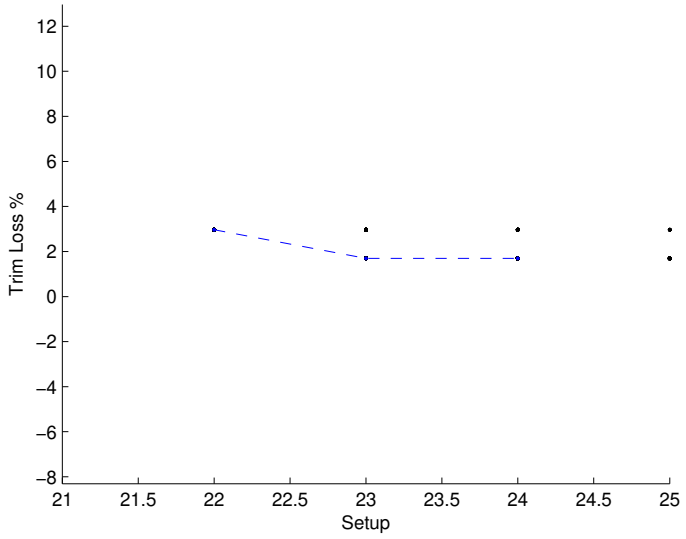


Figura 5.6: Classe 05 - Problema 2

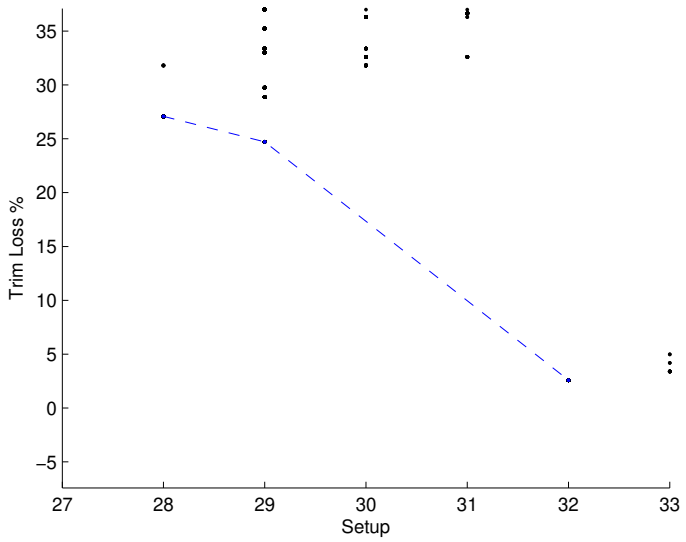


Figura 5.7: Classe 06 - Problema 2

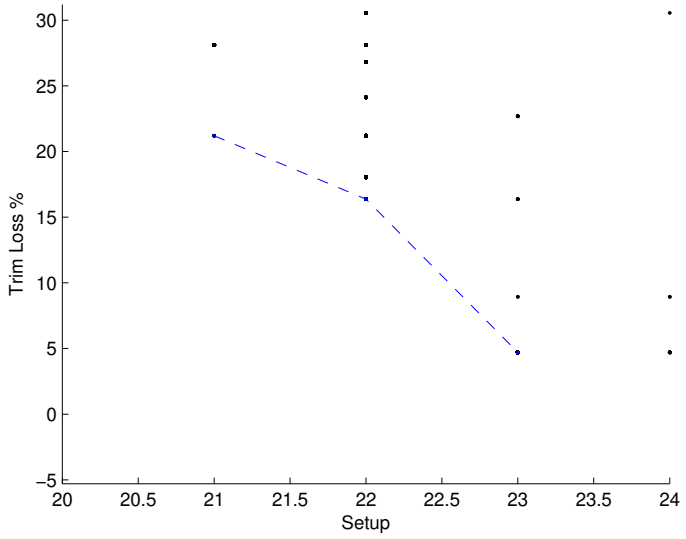


Figura 5.8: Classe 07 - Problema 5

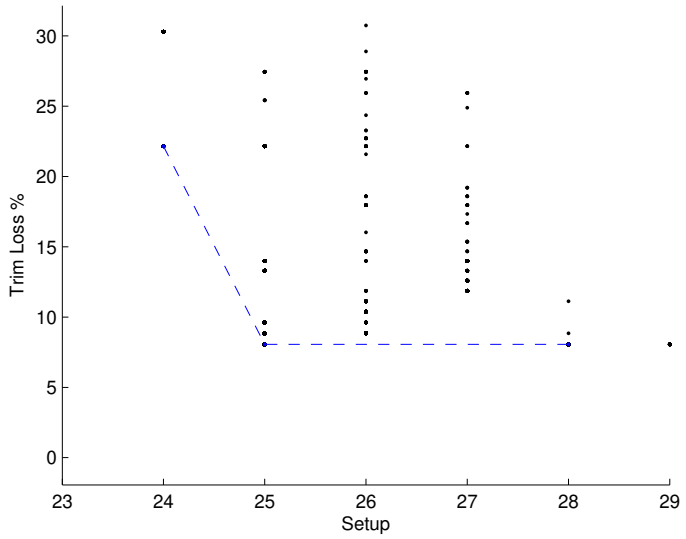


Figura 5.9: Classe 08 - Problema 3

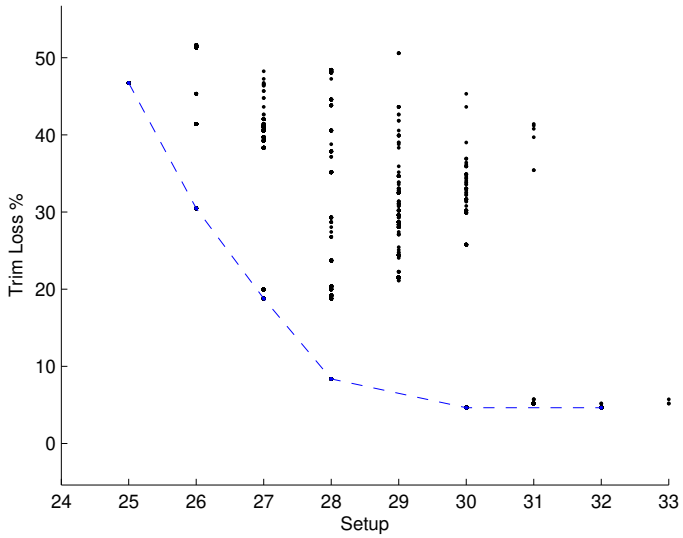


Figura 5.10: Classe 09 - Problema 1

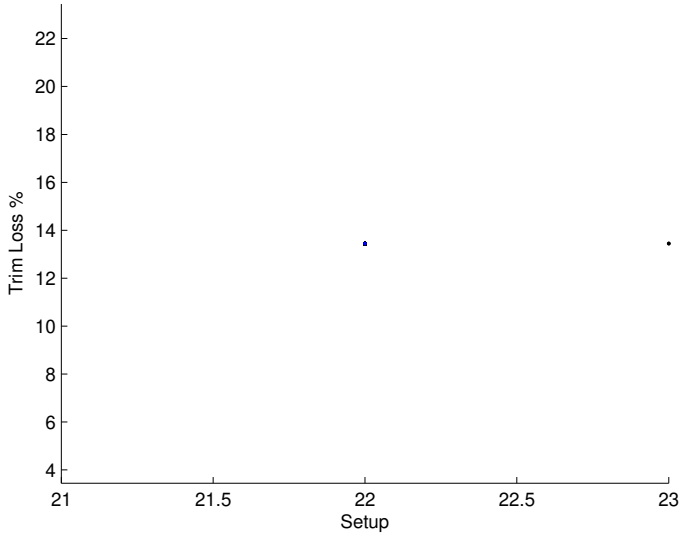


Figura 5.11: Classe 10 - Problema 5

5.1.3 Resultados Fiber

Abaixo exibimos as Fronteiras Eficientes para os problemas do tipo Fiber de maneira semelhante aos problemas gerados pelo Cutgen1. Nas tabelas temos o custo de setup em comum para cada método e suas respectivas porcentagens de desperdício. Os métodos ILS [16] e Crawla [16] não são diretamente multiobjetivos, pois, trabalham escalarizando os objetivos do problema. Os métodos Symbio [22] e SPEA2 [29] são métodos que trabalham diretamente com o problema na forma multiobjetivo.

As soluções médias ou de melhor trade-off são marcadas nas tabelas. Os resultados originais usados para a comparação podem ser encontrados neste trabalho.

Problema	Setups	ILS	Crawla	Symbio	SPEA2
Fiber 06	7	5.19			10.21
	6	5.19			10.21
	5	5.19		5.19	79.92
	4	**8.28**	**8.28**		
	3	8.28	12.47	8.28	
	2		48.5	17.55	
Fiber 07	10	4.98			
	9	4.98			
	8	4.98			
	7	4.98			
	6	8.16			4.74
	5				**10.18**
	4				10.18
	3		8.16	4.98	
	2		11.34	11.34	
	1		68.61	68.61	

Tabela 5.13: Setups e porcentagem de desperdícios obtidos por ILS, Crawla, Symbio e SPEA2 (5180).

Problema	Setups	ILS	Crawla	Symbio	SPEA2
Fiber 08	10	4.47			
	9	4.47			
	8	4.47			
	7	4.47			
	6	4.47			4.27
	5				4.27
	4				6.42
	3		**4.46**	**4.46**	
	2		5.67	5.67	
	1			12,69	
Fiber 09	9	9.27			
	8	9.27			
	7	11.29			10.14
	6	11.29			10.14
	5	11.29			
	4			11.29	
	3		**23.43**	**11.29**	
	2		47.71	47.71	
Fiber 10	14	4.20			
	13	4.20			
	12	4.20			
	11	4.20			
	10	4.20			
	8				4.03
	6				4.03
	5		**4.20**		
	4		5.69		
	3		7.18	5.69	
	2		22.09	22.06	

Tabela 5.14: Setups e porcentagem de desperdícios obtidos por ILS, Crawla, Symbio e SPEA2 (5180).

Problema	Setups	ILS	Crawla	Symbio	SPEA2
Fiber 11	12	6.06			
	11	6.06			
	10	4.52			
	9	4.52			4.61
	8	6.06			
	7				4.61
	6				
	5		**6.06**		
	4		7.59	6.65	
	3		10.67	9.13	
	2			52.16	
Fiber 13a	14	2.41			
	13	4.24			
	12	2.41			5.72
	11	4.24			5.72
	10	4.21			
	8				5.72
	7				
	6				
	5			6.07	
	4		**13.38**	**7.90**	
	3		28.01	13.38	
	2			51.79	

Tabela 5.15: Setups e porcentagem de desperdícios obtidos por ILS, Crawla, Symbio e SPEA2 (5180).

A seguir, mostramos os resultados para um segundo tipo dos problemas Fiber. Tais problemas são semelhantes aos anteriores onde, consideramos que dispomos de um número suficiente de peças de tamanho 9080 para realizar os cortes.

Problema	Setups	ILS	Crawla	Symbio	SPEA2
Fiber 06	6				7.79
	5	8.46			7.79
	4	8.46			
	3	**8.46**	**8.46**	**3.03**	
	2		19.30	8.46	
	1		73.53	73.53	
Fiber 07	5				5.61
	4	5.95			5.61
	3	5.95			
	2	**5.95**	**5.95**	**5.95**	
	1		17.10	17.10	
Fiber 08	5				3.03
	4	3.13			3.03
	3	**7.34**	**1.03**	**1.03**	
	2	17.87	7.34	3.13	
	1		32.60	32.60	
Fiber 09	7				2.77
	6				6.02
	5	9.95			
	4	**9.95**	**6.41**		
	3	9.95	9.95	6.41	
	2		27.69	13.50	
	1			25.46	

Tabela 5.16: Setups e porcentagem de desperdícios obtidos por ILS, Crawla, Symbio e SPEA2 (9080).

Problema	Setups	ILS	Crawla	Symbio	SPEA2
Fiber10	8				1.73
	7				1.73
	6				4.18
	5	6.98			
	4	**6.98**	**1.76**		
	3	9.59	4.37	4.37	
	2		14.81	9.59	
	1			72.21	
Fiber 11	6				3.72
	5	7.77			
	4	**5.08**	**7.77**		
	3	10.47	10.47	5.08	
	2		34.71	13.16	
	1			22.87	
Fiber 13a	9				5.46
	8				5.46
	6	5.79			
	5	8.99			
	4	12.20	8.99		
	3		15.40	8.99	
	2		**31.43**	**12.20**	
	1			40.64	

Tabela 5.17: Setups e porcentagem de desperdícios obtidos por ILS, Crawla, Symbio e SPEA2 (9080).

Problema	Setups	ILS	Crawla	Symbio	SPEA2
Fiber 13b	6				2.99
	5				2.99
	4	15.97			8.62
	3	9.53			
	2	**22.42**	**9.53**	**9.53**	
	1		13.83	13.83	
Fiber 14	9				5.32
	8				5.32
	5	5.63			
	4	**5.63**		**1.85**	
	3	43.35	5.63	5.63	
	2		24.49	20.71	
Fiber 15	7				1.29
	6				1.29
	5	10.81			
	4	7.64	4.48		
	3	32.97	7.64	4.48	
	2		**23.47**	**10.81**	
	1			20.07	

Tabela 5.18: Setups e porcentagem de desperdícios obtidos por ILS, Crawla, Symbio e SPEA2 (9080).

As Soluções obtidas nos testes comparativos com os problemas Fiber mostram bons resultados. Nas tabelas comparativas, devemos lembrar que os algoritmos ILS, Crawla e Symbio trabalham com os problemas de corte podendo ocorrer sobras nas demandas, pois, não necessariamente devem satisfazer as demandas na igualdade. Mesmo assim, exibimos nossos resultados juntamente com as soluções destes outros três métodos, para mostrar que as soluções encontradas em nossa formulação são boas, pois, obtemos resultados próximos aos dos problemas resolvidos com folgas de demanda.

Tomando como base o que foi discutido em Falkenauer [6], escolhemos a codificação em grupo para representar as soluções em nosso algoritmo genético, pois, a codificação em ordem leva em conta a ordem relativa dos itens e não os itens que estão em um grupo, como é o caso do problema de corte onde cada grupo é um padrão de corte. A evolução do tempo computacional para os problemas Fiber pode ser vista na figura (5.12). Os tempos para os problemas de 06 a 13b para as peças disponíveis para corte 5180 e 9080, mostram o aumento no tempo de execução do método SPEA2 em função do tamanho dos problemas.

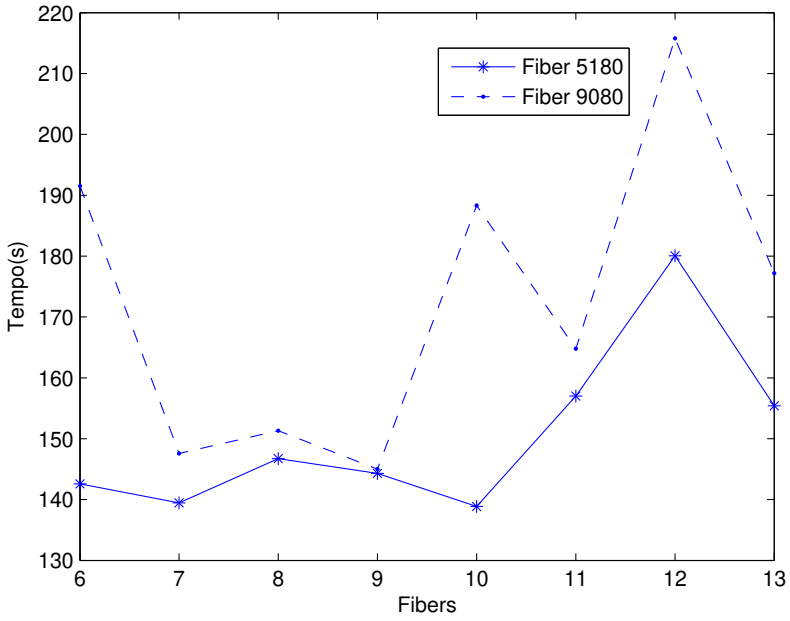


Figura 5.12: Tempos de execução

Os resultados mostram que o uso da codificação em grupo obteve boas soluções quando

comparadas aos problemas resolvidos com folgas de demanda, indicando que a codificação em grupo é vantajosa para problemas em programação matemática onde o objetivo é formar subconjuntos ou coleção de itens como é descrito em [6].

Capítulo 6

Conclusões e Perspectivas

Neste trabalho, o problema de corte unidimensional foi formulado como um problema multiobjetivo para incorporar outras funções objetivo utilizadas na literatura. As funções consideradas são o desperdício e o custo de setup relacionado aos diferentes padrões. Para resolver esta versão do problema de corte, utilizamos o algoritmo evolutivo SPEA2, desenvolvido em [29] para obter uma aproximação da Fronteira Eficiente em problemas multiobjetivo.

Os algoritmos evolutivos se destacam para estes tipos de problema, pois, não é necessária grandes adaptações para que soluções melhores sejam obtidas ao longo das iterações do método, já que a codificação de um problema pode automaticamente incorporar suas características mais importantes.

Além disso existem outros critérios que influênciam o desempenho do algoritmo como os operadores genéticos (Torneio e Mutação), tamanho da População e tamanho do Arquivo, onde são guardadas as soluções aproximadas da Fronteira Eficiente. Os parâmetros adotados neste trabalho possuem como base testes feitos em publicações e, testes prévios realizados com diversos problemas encontrados na literatura [20]. Assim acreditamos que tais parâmetros adotados contribuem de forma eficiente para a execução do algoritmo.

Analisando os resultados dos problemas gerados pelo Cutgen1 e os resultados comparativos dos problemas Fiber com os métodos ILS, Crawla e Symbio, podemos dizer

que o método implementado neste trabalho encontra boas soluções para os problemas usados como teste. Os resultados comparados com problemas resolvidos com folga confirmaram o bom desempenho, pois, mesmo atendendo os problemas na igualdade, em alguns problemas, as soluções encontradas são competitivas com as satisfeitas com folgas na demanda.

Uma vantagem da representação em grupo em relação a representação em ordem está no fato de a solução depender somente do número de diferentes tamanhos para corte e não do total de itens que devem ser cortados como ocorre na codificação em ordem. Além disso, com a representação em grupo é possível explorar melhor o espaço das soluções e computar os valores de setup e desperdício de forma mais rápida. Outro ponto atrativo seria combinar outros objetivos e estudar as soluções obtidas.

Um ponto atrativo seria implementar um sistema de alvo para priorizar determinados alvos ou metas para um determinado objetivo e assim obter um conjunto de soluções mais específica. Também seria conveniente fazer o implementação do algoritmo em outra linguagem (C, C++) para reduzir o tempo de execução do algoritmo e obter um programa fechado e próprio para este tipo de aplicação.

Referências Bibliográficas

[1] Brockhoff, N., Friedrich, T., Hebbinghaus, N., Klein, C., Neumann, F. e Zitzler, E. *Do Additional Objectives Make a Problem Harder ?*, Proceedings of the 9th annual conference on Genetic and evolutionary computation, Pages: 765-772, 2007.

[2] Cardoso, A. C. M., *Algoritmos genéticos para interpolação não linear de imagem e decodificação de códigos lineares.* FEEC, 1998.

[3] Cicirello, V. A., *Non-Wrapping Order Crossover: An Order Preserving Crossover Operator that Respects Absolute Position*, Proceedings of the 8th annual conference on Genetic and evolutionary computation, Pages: 1125-1132, 2006.

[4] Dyckhoff, H., *A typology of cutting and packing problems.* European J.Oper.Res. 44 (1990) 145-159.

[5] Erbas, C., Cerav-Erbas, S. e A. D. Pimentel, *Multiobjective optimization and evolutionary algorithms for the application mapping problem in multiprocessor system-on-chip design.* IEEE, 2005.

[6] Falkenauer, E., *A hybrid grouping genetic algorithm for bin packing.* CRIF, CP 106-P4, 1994.

[7] Fliege, J. e Svaiter, B. F. *Steepest descent methods for multicriteria Optimization*, Math Meth Oper Res (2000) 51:479-949.

[8] Gilmore, P.C e Gomory R.E, *A Liner programming approach to the cutting stock problem.*, Operation Research, Part II, 1963; Vol 11:863-88.

[9] Gong, T., *Evolutionary algorithmis for cutting stock problems*, Computers and Operations Research, Volume 29, Issue 12 , 2002.

[10] Hinterding R., Khan L., *Genetic algorithms for cutting stock problems: with and without contiguity.* Yao X, editor. Progress in evolutionary computation. Lecture notes in articial intelligence. vol.956. Brelin:Springer, 1995.p.166-86. European J.Oper.Res. 44 (1990) 145-159.

[11] Hinterding, R. *Self-adaptation using multi-chromosomes*, IEEE International Conference on Evolutionary Computation, 1997.

[12] Ishibuchi, H. e Kaige, S., *Comparison of Multiobjective Memetic Algorithms on 0/1 Knapsack Problems*, International Workshop on Memetic Algorithms, 2003.

[13] Kantorovich, L. V., *Mathematical Methods of Organizing and Planning Production.* Management Science, Vol. 6, No. 4 (Jul., 1960), pp. 366-422.

[14] Karelahti, J., *Solving the cutting stock problem in the steel industry*, Master thesis, 25.11.2002.

[15] Konak, A., Coit, D. W. e Smith, A. E. *Multi-objective optimization using genetic algorithms: A tutorial*, Reliability Engineering and System Safety 91 (2006) 992−1007.

[16] Lee, J., *In situ column generation for a cutting stock problem.* IBM Research Division, RC 23589, (apr., 2005), W0504-076.

[17] Li, H. e Zhang, Q., *Comparison Between NSGA-II and MOEA/D on a Set of Multiobjective Optimization Problems with Complicated Pareto Sets*, IEEE, Technical Report CES-476 October 2007.

[18] Liang, K., Newton, C., e Hoffman, D., *Solving cutting stock problems by evolutionary programming.* Computacional Intelligence Group, 1998.

[19] Mattson, C. A., Muller, A. A. e Messac, A., *Minimial Representation of multiobjective design space using a smart pareto filter*, , AIAA/ISSMO symmpsium on multidisciplinary analysis and Optimization, Paper No. AIAA 2002-5458, Atlanta.

[20] Michalewicz, Z., *Genetic algorithms + data structures = evolution programs.*, 3rd rev, Springer, 1996.

[21] Mitchell M., *Genetic algorithms: An Overview*, Complexity, 1 (1) 31-39, 1995.

[22] Moretti, A. C., Salles Neto, L. L. e Golfeto, R. R., *A genetic symbiotic algorithm applied to the cutting stock problem*, Simpósio Brasileiro de Pesquisa Operacional, 2007.

[23] Moretti, A. C. e Salles Neto, L. L., *Nonlinear cutting stock problem model to minimize the number of different patterns and objects.*, Comput. Appl. Math. vol.27 no.1 Petrópolis, 2008.

[24] Sardinas, R. Q., Santana, M. R. e Brindis, E. A. *Genetic algorithm-based multiobjective optimization of cutting parameters in turning processes*, Engineering Applications of Artificial Intelligence 19 (2006) 127 - 133.

[25] Sbalzarini, I., Müller, S. e Koumoutsakos, P., *Multiobjective optimization using evolutionary algorithms.*, Center of turbulence reserch, proceedings of the summer program, 2000.

[26] Saad, O. M. e Imam, O. E., *On the solution of stochastic multiobjective integer linear programming problems with a parametric study*, Optimization Online, 2007.

[27] Sokolov, A. e Whitley, D., *Unbiased tournament telection*, Proceedings of the 2005 conference on Genetic and evolutionary computation, 2005.

[28] Schmitt, K., Mehnen, J. e Michelitsch, T. *A Predator-Prey Approach for Pareto-Optimization*, Technical Report ISSN 1433-3325 November 2004.

[29] Zitzler, E., Laumanns, M. e Bleuler, S., *A tutorial on evolutionary multiobjetive optimization.* ,Gloriastrasse 35, CH-8092 Zurich, Switzerland, 2001.

[30] Washer, G. e Gau T., *Heuristics for integer one-dimensional cutting stock problem: a combinatorial study.* OR Spektrum, 18, pp 131-144, 1996.

[31] Washer, G. e Gau T., *A problem generator for the standard one-dimensional cutting stock problem.*, European Journal of Operation Research 84, 572-579, 1995.

Apêndice. Exemplos de Problemas Utilizados

Abaixo exibimos exemplos dos problemas utilizados nos testes numéricos. Maiores detalhes sobre o gerador Cutgen1 e os problemas Fibers podem ser encontrados em [31] e [23] respectivamente.

Cutgen1 - Classe 01

Problema 1	Peça:10000	Itens:25						
Tamanhos	2485	2451	2432	2334	2199	2186	2167	1874
Demanda	10	7	5	5	11	5	5	11
Tamanhos	1847	1724	1689	1631	1614	1464	1458	1147
Demanda	2	7	9	2	1	1	10	1
Tamanhos	969	956	871	621	367	243	171	85
Demanda	1	3	8	6	9	1	1	1
Tamanhos	23							
Demanda	3							

Problema 2	Peça:10000	Itens:25						
Tamanhos	2401	2218	2052	2047	1936	1856	1795	1773
Demanda	11	1	5	3	8	10	5	1
Tamanhos	1616	1477	1332	1327	1308	1254	1074	1018
Demanda	9	8	6	6	2	4	4	4
Tamanhos	864	552	459	327	311	254	167	36
Demanda	10	4	1	7	1	2	6	9
Tamanhos	21							
Demanda	1							

Problema 3	Peça:10000	Itens:25						
Tamanhos	2476	2427	2408	2214	2188	2133	1932	1825
Demanda	1	7	11	10	3	6	1	3
Tamanhos	1386	1291	1190	1000	990	860	852	670
Demanda	3	2	6	5	1	5	3	8
Tamanhos	532	465	251	197	131	61	56	29
Demanda	4	5	1	10	7	7	8	6
Tamanhos	3							
Demanda	2							

Problema 4	Peça:10000	Itens:25						
Tamanhos	2448	2425	2424	2390	2318	2272	2263	2028
Demanda	7	3	8	7	7	9	4	8
Tamanhos	1862	1806	1796	1719	1641	1193	1099	1058
Demanda	8	5	2	3	5	5	1	3
Tamanhos	1012	989	976	858	645	599	526	469
Demanda	1	3	7	2	4	8	7	3
Tamanhos	84							
Demanda	5							

Problema 5	Peça:10000	Itens:25						
Tamanhos	2483	2364	2312	2306	2126	2091	1857	1624
Demanda	6	2	3	6	8	6	2	3
Tamanhos	1555	1518	1513	1504	1393	1373	1303	1148
Demanda	2	6	5	7	7	8	7	4
Tamanhos	874	811	753	687	613	548	450	237
Demanda	4	5	4	7	2	4	8	1
Tamanhos	166							
Demanda	8							

Cutgen1 - Classe 02

Problema 1	Peça:10000	Itens:25						
Tamanhos	2347	2188	2065	1970	1960	1857	1664	1582
Demanda	10	1	14	10	9	19	4	5
Tamanhos	1576	1557	1464	1452	1384	1355	1251	920
Demana	8	5	8	7	15	18	13	5
Tamanhos	849	798	691	263	221	131	122	113
Demanda	17	17	10	14	3	4	8	8
Tamanhos	78							
Demanda	18							

Problema 2	Peça:10000	Itens:25						
Tamanhos	2442	2441	2336	2246	2117	2089	2085	2005
Demanda	8	19	17	1	15	2	3	2
Tamanhos	1996	1880	1541	1485	1474	1418	1299	1283
Demando	7	3	18	5	20	8	21	13
Tamanhos	1280	1026	917	727	717	509	376	314
Demanda	11	4	14	16	8	6	20	4
Tamanhos	293							
Demanda	5							

Problema 3	Peça:10000	Itens:25						
Tamanhos	2442	2441	2336	2246	2117	2089	2085	2005
Demanda	8	19	17	1	15	2	3	2
Tamanhos	1996	1880	1541	1485	1474	1418	1299	1283
Demanda	7	3	18	5	20	8	21	13
Tamanhos	1280	1026	917	727	717	509	376	314
Demanda	11	4	14	16	8	6	20	4
Tamanhos	293							
Demanda	5							

Problema 4	Peça:10000	Itens:25						
Tamanhos	2488	2370	2264	1530	1508	1254	1246	1197
Demanda	10	10	9	11	1	11	12	1
Tamanhos	1187	1112	1077	973	916	892	882	879
Demanda	9	13	4	12	2	14	6	12
Tamanhos	706	508	502	438	356	133	103	96
Demanda	11	6	13	17	12	9	16	9
Tamanhos	2							
Demanda	5							

Problema 5	Peça:10000	Itens:25					
Tamanhos	2329	2241	2169	2145	2096	1939	1828	1729
Demanda	3	10	13	13	9	1	11	1
Tamanhos	1564	1563	1230	1212	1194	1128	1099	1049
Demanda	16	17	10	17	2	9	8	16
Tamanhos	798	730	580	492	402	396	364	248
Demanda	11	17	9	12	5	7	13	4
Tamanhos	225							
Demanda	16							

Cutgen1 - Classe 03

Problema 1	Peça:10000	Itens:25						
Tamanhos	2485	2451	2432	2334	2199	2186	2167	1874
Demanda	41	27	22	21	44	20	18	42
Tamanho	1847	1724	1689	1631	1614	1464	1458	1147
Demanda	8	27	36	8	1	1	39	2
Tamanho	969	956	871	621	367	243	171	85
Demanda	1	12	34	22	36	2	1	3
Tamanhos	23							
Demanda	32							

Problema 1	Peça:10000	Itens:25						
Tamanhos	2485	2451	2432	2334	2199	2186	2167	1874
Demanda	41	27	22	21	44	20	18	42
Tamanho	1847	1724	1689	1631	1614	1464	1458	1147
Demanda	8	27	36	8	1	1	39	2
Tamanho	969	956	871	621	367	243	171	85
Demanda	1	12	34	22	36	2	1	3
Tamanhos	23							
Demanda	32							

Fibers

Fiber 06	Peça:5180	Itens:6					
Tamanhos	520	1000	1066	1120	1150	1250	
Demanda	91	11	18	9	64	5	

Fiber 07	Peça:5180	Itens:4					
Tamanhos	650	660	1000	1020			
Demanda	13	47	105	18			

Fiber 08	Peça:5180	Itens:4					
Tamanhos	500	610	1000	1200			
Demanda	22	40	378	15			

Fiber 09	Peça:5180	Itens:7					
Tamanhos	500	850	900	1000	1250	1300	1500
Demanda	1	16	140	100	6	3	3

Fiber 10	Peça:5180	Itens:6					
Tamanhos	750	915	920	985	1000	1250	
Demanda	10	30	5	21	264	19	

Fiber 06	Peça:9080	Itens:6				
Tamanhos	520	1000	1066	1120	1150	1250
Demanda	91	11	18	9	64	5

Fiber 07	Peça:9080	Itens:4		
Tamanhos	650	660	1000	1020
Demanda	13	47	105	18

Fiber 08	Peça:9080	Itens:4		
Tamanhos	500	610	1000	1200
Demanda	22	40	378	15

Fiber 09	Peça:9080	Itens:7					
Tamanhos	500	850	900	1000	1250	1300	1500
Demanda	1	16	140	100	6	3	3

Fiber 10	Peça:9080	Itens:6				
Tamanhos	750	915	920	985	1000	1250
Demanda	10	30	5	21	264	19

Exemplos de Solução

Nesta seção mostramos alguns exemplos de soluções encontradas pelo método desenvolvido neste trabalho. Cada linha da matriz de respostas indica um padrão de corte e os elementos representam o número de vezes que cada item é cortado neste padrão. Os valores em [ ] indicam quantas vezes o padrão é utilizado e os valores em ( ) correspondem ao desperdício do padrão de corte em questão.

- **Exemplo 01: Ilustrando a Fronteira Eficiente no Capítulo 5.**

Exemplo 01	Setups	%Trim Loss
	7	22.38
	8	19.50
	9	19.50

- **Soluções correspondentes a Fronteira Eficiente**

- Setup=7 , % Trim loss = 22.38

$1:3:0:0:0:0:0:0:$	[2]	(0)
$0:2:1:0:0:0:0:0:$	[1]	(2)
$0:0:1:1:0:0:0:0:$	[4]	(4)
$0:0:0:1:1:0:0:0:$	[3]	(2)
$0:0:0:0:1:1:0:0:$	[5]	(0)
$0:0:0:0:0:0:1:0:$	[5]	(6)
$0:0:0:0:0:0:0:1:$	[8]	(5)
Total (Setup,Trim Loss):	[28]	(94)

- Setup=8 , % Trim loss = 19.50

$2:2:0:0:0:0:0:0:$	[1]	(1)
$0:1:1:1:0:0:0:0:$	|[5]	(0)
$0:1:0:1:0:0:0:0:$	[1]	(5)
$0:0:0:1:1:0:0:0:$	[1]	(2)
$0:0:0:0:1:1:0:0:$	[5]	(0)
$0:0:0:0:2:0:0:0:$	[1]	(1)
$0:0:0:0:0:0:1:0:$	[5]	(6)
$0:0:0:0:0:0:0:1:$	[8]	(5)
Total (Setup,Trim Loss):	[27]	(79)

- Setup=9 , % Trim loss = 19.50

$1:3:0:0:0:0:0:0:$	[2]	(0)
$0:1:1:1:0:0:0:0:$	[2]	(0)
$0:0:3:0:0:0:0:0:$	[1]	(0)
$0:0:0:2:0:0:0:0:$	[2]	(3)
$0:0:0:1:1:0:0:0:$	[1]	(2)
$0:0:0:0:1:1:0:0:$	[5]	(0)
$0:0:0:0:2:0:0:0:$	[1]	(1)
$0:0:0:0:0:0:1:0:$	[5]	(6)
$0:0:0:0:0:0:0:1:$	[8]	(5)
Total (Setup,Trim Loss):	[27]	(79)

- Fiber 06 : W=5180

Fiber 06 $m = 6$	Setups	%Trim Loss
	5	79.92
	6	10.21
	7	10.21

- **Soluções correspondentes a Fronteira Eficiente**

- Setup=5 , % Trim loss = 79.92

$1:1:2:1:0:0:$	[9]	(408)
$0:2:0:0:0:0:$	[1]	(3180)
$1:0:0:0:0:0:$	[82]	(4660)
$0:0:0:0:1:0:$	[64]	(4030)
$0:0:0:0:0:1:$	[5]	(3930)
Total (Setup,Trim Loss):	[161]	(666542)

- Setup=6 , % Trim loss = 10.21

$8:1:0:0:0:0:$	[11]	(20)
$3:0:2:1:0:0:$	[1]	(368)
$0:0:4:0:0:0:$	[4]	(916)
$0:0:0:2:1:1:$	[4]	(540)
$0:0:0:0:4:0:$	[15]	(580)
$0:0:0:0:0:1:$	[1]	(3930)
Total (Setup,Trim Loss):	[36]	(19042)

- Setup=7 , % Trim loss = 10.21

$8:1:0:0:0:0:$	[11]	(20)
$3:0:3:0:0:0:$	[1]	(422)
$0:0:4:0:0:0:$	[3]	(916)
$0:0:1:3:0:0:$	[3]	(754)
$0:0:0:0:4:0:$	[16]	(580)
$0:0:0:0:0:4:$	[1]	(180)
$0:0:0:0:0:0:1:$	[1]	(3930)
Total (Setup,Trim Loss):	[36]	(19042)