UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE CIÊNCIAS EXATAS E DA TERRA
DEPARTAMENTO DE INFORMÁTICA E MATEMÁTICA APLICADA
PROGRAMA DE PÓS-GRADUAÇÃO EM SISTEMAS E COMPUTAÇÃO

Uma Abordagem por Nuvem de Partículas para Problemas de Otimização Combinatória

Givanaldo Rocha de Souza

Natal
Rio Grande do Norte – Brasil
Fevereiro de 2006

UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE CIÊNCIAS EXATAS E DA TERRA
DEPARTAMENTO DE INFORMÁTICA E MATEMÁTICA APLICADA
PROGRAMA DE PÓS-GRADUAÇÃO EM SISTEMAS E COMPUTAÇÃO

Uma Abordagem por Nuvem de Partículas para Problemas de Otimização Combinatória

Givanaldo Rocha de Souza

Orientadora: Prof[a]. Dr[a]. Elizabeth Ferreira Gouvêa Goldbarg

Co-orientador: Prof. Dr. Marco César Goldbarg

Dissertação de mestrado apresentada à Universidade Federal do Rio Grande do Norte, para obtenção do título de Mestre em Sistemas e Computação. Área de concentração: Algoritmos Experimentais.

Natal
Rio Grande do Norte – Brasil
Fevereiro de 2006

AGRADECIMENTOS

Aos meus familiares, pelo apoio e compreensão nos momentos em que precisei estar ausente e que precisei usar com exclusividade o único computador da minha casa.

A minha noiva, Liana, por ser esta pessoa linda que eu amo muito e que tenta me entender em todos os momentos, e que me acompanhou durante todo este processo de mestrado pacientemente e que sempre vai me acompanhar em todos os momentos.

À professora Elizabeth Ferreira Gouvêa Goldbarg, pela orientação fornecida desde os tempos de graduação e por toda ajuda ao longo do processo. Ao professor Marco César Goldbarg, também pela orientação fornecida e pelas "dicas otimizadas" nas mini-reuniões.

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelo apoio financeiro fornecido.

Aos amigos do Laboratório de Algoritmos Experimentais (LAE), que foram de forte contribuição para a realização deste trabalho. E também aos amigos do PPgSC, pelos momentos de descontração.

Ao Grupo de Jovens JUCI (Jovens Unidos a Cristo e à Igreja) e ao Ministério de Música Filhos de Davi, aos quais pude ter o apoio religioso que me guiou neste processo.

A DEUS, nosso Senhor, pois é nele em que me apoio nos momentos mais difíceis e também nos mais alegres. Nunca abandonarei minha fé...

Uma Abordagem por Nuvem de Partículas para Problemas de Otimização Combinatória

Autor: Givanaldo Rocha de Souza

Orientadora: Prof[a]. Dr[a]. Elizabeth Ferreira Gouvêa Goldbarg

Co-orientador: Prof. Dr. Marco César Goldbarg

RESUMO

Os problemas de otimização combinatória têm como objetivo maximizar ou minimizar uma função definida sobre um certo domínio finito. Já as metaheurísticas são procedimentos destinados a encontrar uma boa solução, eventualmente a ótima, consistindo na aplicação de uma heurística subordinada, a qual tem que ser modelada para cada problema específico. Este trabalho apresenta algoritmos baseados na técnica de otimização por nuvem de partículas (metaheurística) para dois problemas de otimização combinatória: o Problema do Caixeiro Viajante e o Problema da Árvore Geradora Mínima Restrita em Grau Multicritério. O primeiro é um problema em que apenas um objetivo é otimizado, enquanto o segundo é um problema que deve lidar com múltiplos objetivos. Os algoritmos propostos são comparados a outras abordagens para o mesmo problema em questão, em termos de qualidade de solução, a fim de verificar a eficiência desses algoritmos.

A Particle Swarm Approach for Combinatorial Optimization Problems

Author: Givanaldo Rocha de Souza

Advisor: Prof[a]. Dr[a]. Elizabeth Ferreira Gouvêa Goldbarg

Co-advisor: Prof. Dr. Marco César Goldbarg

ABSTRACT

Combinatorial optimization problems have the goal of maximize or minimize functions defined over a finite domain. Metaheuristics are methods designed to find good solutions in this finite domain, sometimes the optimum solution, using a subordinated heuristic, which is modeled for each particular problem. This work presents algorithms based on particle swarm optimization (metaheuristic) applied to combinatorial optimization problems: the Traveling Salesman Problem and the Multicriteria Degree Constrained Minimum Spanning Tree Problem. The first problem optimizes only one objective, while the other problem deals with many objectives. In order to evaluate the performance of the algorithms proposed, they are compared, in terms of the quality of the solutions found, to other approaches.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

Capítulo 1

Introdução

Os problemas de otimização, na sua forma geral, têm como objetivo maximizar ou minimizar uma função definida sobre certo domínio. A teoria clássica de otimização trata do caso em que o domínio é infinito. Já no caso dos chamados problemas de otimização combinatória, o domínio é tipicamente finito; além disso, em geral, é fácil listar os seus elementos e também testar se um dado elemento pertence a esse domínio. Ainda assim, a idéia de testar todos os elementos deste domínio na busca pelo melhor mostra-se inviável na prática, mesmo para instâncias de tamanho moderado [64].

Como exemplos clássicos de problemas de otimização combinatória, podemos citar os problemas do caixeiro viajante, da mochila, da cobertura mínima por conjuntos, da árvore geradora mínima, da árvore de Steiner, entre outros. Todos surgem naturalmente em aplicações práticas, tais como o projeto de redes de telecomunicação e de circuitos VLSI, o empacotamento de objetos em recipientes, a localização de centros distribuidores, o escalonamento e roteamento de veículos. Outras áreas de aplicação incluem, por exemplo, a estatística (análise de dados), a economia (matrizes de entrada/saída), a física (estados de energia mínima) e a biologia molecular (alinhamento de DNA e proteínas, inferência de padrões).

As metaheurísticas são procedimentos destinados a encontrar uma boa solução, eventualmente a ótima, consistindo na aplicação de uma heurística subordinada, a qual tem que ser modelada para cada problema específico. Algumas metaheurísticas bem conhecidas são: Algoritmos Genéticos [44], Busca Tabu [36] e Têmpera Simulada [53].

As metaheurísticas, assim como os modelos de busca local [1], diferenciam-se entre si pelas seguintes características:

- Critério de escolha de uma solução inicial.
- Definição da vizinhança $N(s)$ de uma solução s.
- Critério de seleção de uma solução vizinha dentro de $N(s)$.
- Critério de parada.

Várias metaheurísticas inspiradas em fenômenos biológicos foram propostas para resolver problemas de otimização combinatória, tais como Algoritmos Genéticos [44], Algoritmos Meméticos [65], Algoritmos Culturais [77] e Colônia de Formigas [24], dentre outros. A Otimização por Nuvem de Partículas (PSO) também faz parte dessa classe de metaheurísticas bio-inspiradas. Trata-se de uma técnica baseada em população introduzida por um psicólogo, James Kennedy, e um engenheiro eletricista, Russel Eberhart, com base no comportamento de um bando de pássaros em revoadas [51].

Este trabalho apresenta algoritmos baseados na técnica de otimização por nuvem de partículas para dois problemas de otimização combinatória: o Problema do Caixeiro Viajante e o Problema da Árvore Geradora Mínima Restrita em Grau Multicritério. O

primeiro é um problema em que apenas um objetivo é otimizado, enquanto o segundo é um problema que deve lidar com múltiplos objetivos.

O Capítulo 2 descreve o Problema do Caixeiro Viajante, com a definição e aplicações no mundo real. O Capítulo 3 descreve o Problema da Árvore Geradora Mínima Restrita em Grau Multicritério, mostrando sua motivação em problemas do mundo real, apresentando alguns problemas de otimização combinatória multicritério, métodos de resolução desses problemas, a otimização multicritério evolucionária e alguns algoritmos aplicados ao problema da árvore geradora mínima restrita em grau multicritério.

O capítulo 4 apresenta a técnica de otimização por nuvem de partículas. Nesse capítulo é abordada a motivação para o desenvolvimento da técnica em questão com uma contextualização sobre a vida artificial. É descrito o algoritmo básico, apresentando-se suas características, fórmulas utilizadas, variações e parametrização. Algumas aplicações práticas dessa técnica são mostradas, além da sua caracterização em problemas discretos. Por fim, são descritos alguns algoritmos PSO para o problema do caixeiro viajante. Neste capítulo não são citados algoritmos PSO aplicados ao problema da árvore geradora mínima restrita em grau multicritério, pois não foram encontrados na literatura.

O capítulo 5 apresenta os algoritmos propostos, explicando a construção de cada um. No capítulo 6 são apresentados os resultados computacionais, comparando-os com resultados de outros algoritmos existentes na literatura para o problema abordado. No capítulo 7 são mostrados os trabalhos futuros. Finalmente, o capítulo 8 apresenta as considerações finais sobre o trabalho.

Capítulo 2

O Problema do Caixeiro Viajante

O Problema do Caixeiro Viajante (PCV) é um dos mais tradicionais e conhecidos problemas de otimização combinatória [11, 39]. Os problemas de roteamento lidam em sua maior parte com passeios (ou *tours*) sobre pontos de demanda ou oferta. Dentre os tipos de passeio, um dos mais importantes é o denominado hamiltoniano. Seu nome é devido a William Rowan Hamilton que em 1857 propôs um jogo chamado *Around the World* (figura 1). O jogo era feito sobre um dodecaedro em que cada vértice estava associado a uma cidade importante da época. O desafio consistia em encontrar uma rota através dos vértices do dodecaedro que iniciasse e terminasse em uma mesma cidade passando uma única vez em cada cidade. Em homenagem a Hamilton, uma solução do seu jogo passou a ser chamada de ciclo hamiltoniano (figura 2).

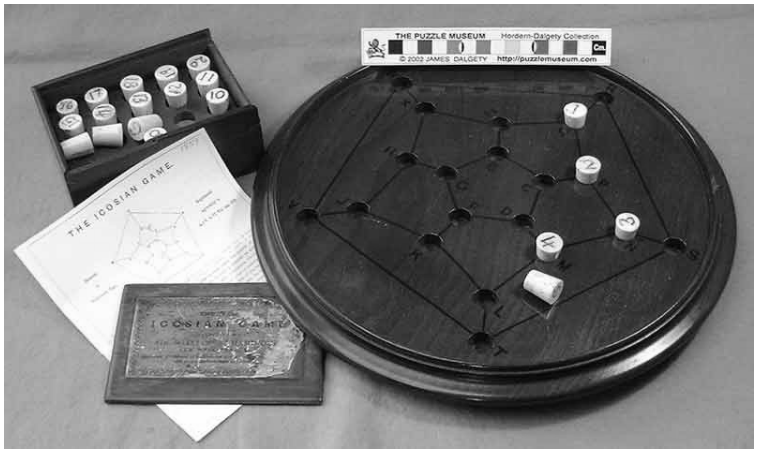


Figura 1: Jogo *Around the World*, proposto por Hamilton

Um grafo, numa definição bem simples, é um conjunto de vértices e arestas. Os vértices são pontos que podem representar cidades, depósitos, postos de trabalho ou atendimento, por exemplo. As arestas são linhas que conectam os vértices, representando relações entre os vértices. Por exemplo, as arestas podem representar ruas ou estradas entre duas cidades. Um circuito hamiltoniano é um passeio por todos os vértices de um grafo retornando ao vértice origem, passando por cada um dos outros vértices apenas uma vez.

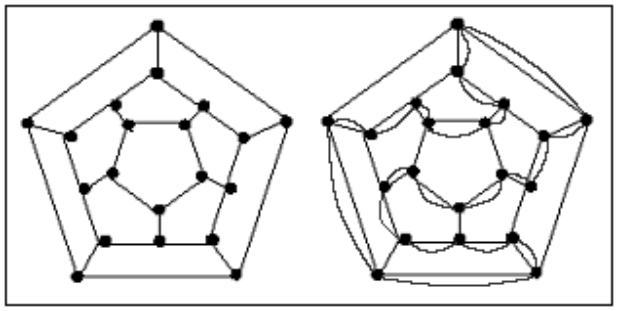


Figura 2: Solução do jogo proposto por Hamilton

O PCV é um problema de otimização associado ao da determinação dos caminhos hamiltonianos em um grafo qualquer, cujo objetivo é encontrar, em um grafo $G = (V, E)$, em que V é o conjunto de vértices do grafo e E consiste nas arestas que ligam esses vértices, com $|V| = n \geq 3$ e $c_{ij} \geq 0$ associado a cada uma das arestas (i, j), o caminho hamiltoniano de menor custo. Este problema de otimização, formalmente, é considerado intratável, pertencendo à classe NP-Árdua [50].

O PCV é importante devido a pelo menos três de suas características: grande aplicação prática, grande relação com outros modelos e grande dificuldade de solução exata. Em suas diversas versões, o PCV está presente em inúmeros problemas práticos, como por exemplo:

- Programação de operações de máquinas em manufatura [30].
- Programação de transporte entre células de manufatura [29].
- Otimização do movimento de ferramentas de corte [13].
- Otimização de perfurações em placas de circuitos impressos [75].
- Na maioria dos problemas de roteamento de veículos [7].
- Na solução de problemas de sequenciamento [91].
- Na solução de problemas de programação e distribuição de tarefas em plantas [81].
- Trabalhos administrativos [60].

No problema da perfuração em placas de circuitos impressos (figura 3), quando as diversas camadas que compõem uma placa estão assentadas, perfurações são feitas utilizando máquinas automáticas de controle numérico. As perfurações são necessárias para permitir a fixação de componentes na placa (transistores, resistores, CIs e outros componentes) ou para possibilitar os contatos entre as diferentes camadas. O problema do caixeiro viajante aparecerá na execução eficiente das perfurações.

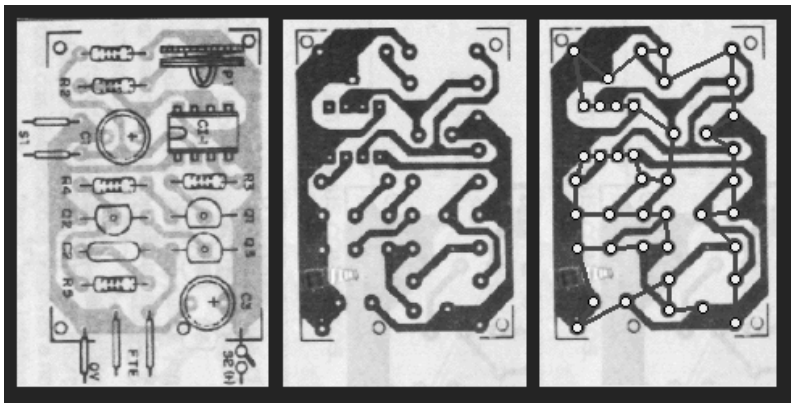


Figura 3: Otimização de perfurações em placas de circuitos impressos

Quando $c_{ij} = c_{ji}$, $\forall i, j \in V$, tem-se o Problema Simétrico do Caixeiro Viajante (PCVS), caso contrário será denominado Assimétrico (PCVA). Problemas em que a desigualdade triangular se verifica ($c_{ij} + c_{jk} \geq c_{ik}$, $\forall i, j, k \in V$) são chamados de Euclidianos (PCVE).

Quando os vértices do grafo são pontos do plano com coordenadas (x_i, y_i), $i = 1, 2..., n$, as métricas mais utilizadas são:

$$c_{ij} = \begin{cases} \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} & \text{(Euclidiana)} \\ |x_i - x_j| + |y_i - y_j| & \text{(Manhattan)} \\ \max \{ |x_i - x_j|, \ |y_i - y_j| \} & \text{(Chebyshev)} \\ \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + 0.5(x_i - x_j)(y_i - y_j)} & \text{(Afim)} \end{cases}$$

O Problema de Programação Inteira Bivalente correspondente é formulado em um grafo completo com n vértices e $c_{ij} > 0$, $\forall (i, j) \in E$ e com $c_{ii} = +\infty$, utilizando variáveis x_{ij}, tais que:

$$x_{ij} = \begin{cases} 1, \text{ se } (i, j) \in E \text{ está na solução} \\ 0, \text{ caso contrário} \end{cases}$$

Tem-se então a modelagem clássica para o PCV em variáveis inteira 0-1 de Dantzig, Fulkerson e Johnson [21]:

$$Minimizar \quad x_0 = \sum_{j=1}^{n} \sum_{i=1}^{n} c_{ij} . x_{ij}$$

sujeito a :

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad \forall j \in N \qquad (i)$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad \forall i \in N \qquad (ii)$$

$$\sum_{i,j \in S} x_{ij} \leq |S| - 1 \qquad \forall S \subset N \qquad (iii)$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j \in N \qquad (iv)$$

em que a variável binária x_{ij} assume valor igual a 1 se o arco $(i, j) \in E$ for escolhido para integrar a solução, e 0 caso contrário. S é um subgrafo de G em que $|S|$ representa o número de vértices desse grafo. As restrições *i* e *ii* garantem que cada vértice é visitado apenas uma vez, no entanto não eliminam os chamados *subtours* ou ciclos isolados. A eliminação dos *subtours* é garantida pela inclusão da restrição *iii*.

Essa formulação destaca um importante aspecto do PCV que é sua natureza combinatória. Pela formulação, pode-se concluir que solucionar um PCV é determinar uma permutação legal de custo mínimo.

Por ser um problema de grande importância, há inúmeras abordagens aplicadas ao PCV na literatura. A listagem a seguir mostra algumas dessas abordagens:

- **Algoritmos Exatos:** Balas e Toth [3] (branch and bound), Smith e Thompson [84] (enumeração implícita).
- **Algoritmos Gulosos:** Bellmore e Nemhauser [6] (vizinho mais próximo), Rosenkrantz *et al.* [80] (inserção), Norback e Love [67] (inserção pelo maior ângulo).
- **Busca Local:** Lin e Kernighan [61] (heurística *k*-opt Lin-Kernighan), Gendrau *et al.* [33] (GENI e GENIUS), Johnson *et al*. [49] (2-opt e 3-opt).
- **Multistart:** Boese *et al.* [8], Hagen e Kahng [43].
- **Têmpera Simulada:** Kirkpatrick *et al.* [53], Cerny [12], Bonomi e Lutton [9], Johnson *et al.* [48].
- **Busca Tabu:** Glover [37, 38], Fiechter [31], Misevičius [63].
- **Algoritmos Genéticos:** Brady [10], Valenzuela e Jones [87], Mühlenbein *et al.* [66].
- **Colônia de Formigas:** Dorigo e Gambardella [24].

Existe ainda a heurística de Christofides [14], que usa a transformação de uma árvore geradora mínima em um *tour*.

Capítulo 3

O Problema da Árvore Geradora Mínima Restrita em Grau Multicritério

O problema da Árvore Geradora Mínima (AGM) consiste em encontrar uma árvore geradora, de custo mínimo, em um grafo com arestas valoradas, tendo como aplicação muito comum o **projeto de redes** (*network design*). Este problema já foi bastante estudado, tendo muitos algoritmos polinomiais eficientes desenvolvidos. Dentre esses algoritmos, podem ser destacados os algoritmos de Dijkstra [23], Kruskal [59] e Prim [71] como clássicos neste problema. Bazlamaçci e Hindi [5] apresentam uma revisão sobre o problema da árvore geradora mínima onde são descritos outros algoritmos para sua resolução.

Se houver, em adição, uma constante d indicando o máximo grau permitido para cada vértice, então o problema é definido como **Árvore Geradora Mínima Restrita em Grau** (AGM-d) [73, 55]. Diferente da AGM, a AGM-d não pode ser encontrada usando um algoritmo polinomial. Encontrar uma AGM-d é geralmente muito mais difícil. Pode ser constatado que encontrar uma AGM-2 é NP-Árduo [50], a partir da sua equivalência ao problema do caminho hamiltoniano de custo mínimo, e ainda que, para $d \geq 3$, não há algoritmos polinomiais conhecidos para obter a AGM-d. De fato, decidir se uma árvore geradora mínima restrita em grau existe em um grafo G, para algum $d \geq 2$, é um problema NP-Completo [32].

No mundo real, há casos em que se devem considerar simultaneamente **múltiplos critérios** na determinação de uma AGM, devido aos múltiplos atributos definidos em cada aresta. Por exemplo, no projeto de *layout* de sistemas de telecomunicação, além do custo para conexão entre cidades ou terminais, outros fatores tais como o tempo para comunicação ou construção, a complexidade para construção e a confiabilidade são importantes e têm que ser levados em consideração. Em todos estes casos, a AGM com multicritério é uma representação mais realística de um problema prático.

O problema da **Árvore Geradora Mínima Restrita em Grau Multicritério** (AGM-mc$d$) não consiste apenas em estender o problema AGM-d de um único critério para AGM-d com múltiplos critérios. Em geral, não se pode obter a solução ótima do problema porque, na prática, estes critérios geralmente são conflitantes entre si. As soluções (chamadas de soluções não-dominadas, conceito que será visto mais a frente) para este problema formam um conjunto denominado **conjunto de soluções ótimas de Pareto**, conceito formulado por Vilfredo Pareto no século XIX [69]. Porém, os cálculos destas soluções é uma tarefa difícil por se tratar de um problema de otimização NP-Árduo [50]. Uma solução comum é simplificar os múltiplos critérios para um único critério, justamente porque existem bons algoritmos polinomiais que resolvem os problemas com único critério (estes mesmos se tornando heurísticos quando se trata da AGM-d). Entretanto, apenas um único ponto do conjunto de soluções ótimas de Pareto pode ser obtido, além da dificuldade em transformar múltiplos critérios em um único critério. O **agente de decisão**, na prática, pode preferir

apenas um ponto do conjunto de soluções ótimas de Pareto, mas pode ser útil ter todas as outras possíveis soluções.

3.1. Problemas Multicritério

A resolução de muitos dos problemas do dia-a-dia consiste em escolher dentre várias alternativas viáveis, ou seja, tomar decisões. A maioria destes problemas é de difícil resolução, uma vez que envolvem múltiplos critérios (objetivos), geralmente conflitantes entre si. Ehrgott e Gandibleux [26] apresentam uma revisão a respeito dos problemas multicritério.

Alguns exemplos de problemas práticos multicritério são:

- Escolher o melhor local para a construção de uma ponte, em que os critérios poderiam ser: o custo, o impacto ambiental sobre o rio e o volume de tráfego.
- A construção de uma rede de dutos (água ou gás natural, por exemplo) sobre uma determinada região, considerando como critérios a otimizar o comprimento dos dutos, o impacto ambiental na região e o custo de escavação.
- Determinar o trajeto mais econômico para efetuar a entrega de produtos a clientes, em que os critérios poderiam ser: tempo, distância e tráfego.
- Decidir qual carro novo comprar, considerando Golf, Astra, Mondeo e Corolla. A decisão será tomada de acordo com o preço (o mais barato), a eficiência do motor (a menor razão *l*/100km, em que *l* é a quantidade de litros de combustível consumida) e a potência (a maior). Então, têm-se quatro alternativas e três critérios:

		Alternativas			
		Golf	Astra	Mondeo	Corolla
Critérios	Preço	31	29	30	27
	Eficiência	7.2	7.0	7.5	7.8
	Potência	90	75	80	75

Qual será a melhor alternativa?

Note que, para cada critério, a tomada de decisão é fácil.

Justamente por existirem conflitos entre os critérios, o agente de decisão, que tem a responsabilidade de decidir, tem que ponderar os critérios a efetuar, visando encontrar uma solução que seja mais satisfatória, ou seja, que melhor reflita as preferências dele.

Durante a década de 50, com o desenvolvimento da Pesquisa Operacional como disciplina científica, pensou-se que esta iria desenvolver métodos e técnicas capazes de resolver a maioria dos problemas de decisão tanto nas indústrias quanto nos setores de serviços. Muitos autores valorizaram a utilização de modelos matemáticos de otimização, pois consideravam que todos os modelos se baseavam na otimização de apenas uma função-objetivo. Mas, na década de 70, com a complexidade crescente do ambiente sócio-econômico que caracteriza as sociedades tecnológicas modernas, verificou-se que quase todos os problemas mais importantes envolviam vários critérios, geralmente conflitantes entre si, tornando-se difícil a formulação, modelagem e solução do problema. Neste sentido, as preocupações dos agentes de decisão não se limitam apenas ao fator econômico, mas também em fatores ambientais, políticos, sociais, estéticos, entre outros [4].

3.2. Otimização Combinatória Multicritério

Otimização multicritério (ou multiobjetivo) pode ser definida como o problema de encontrar um vetor de variáveis de decisão as quais satisfazem certas restrições e otimizam um vetor-função cujos elementos representam as funções-objetivo. Estas funções formam uma descrição matemática de critérios os quais geralmente são conflitantes entre si. Neste caso, o termo "otimizar" significa encontrar uma solução com os valores aceitáveis pelo AD de todas as funções-objetivo.

De modo geral, um problema multicritério pode ser formulado matematicamente da seguinte forma:

Maximizar $f_i(\mathbf{x}) = c_i.\mathbf{x} \therefore i = 1, 2, ..., h$

sujeito a $\mathbf{x} \in X = \{\mathbf{x} \in \Re^n : \mathbf{x} \geq 0, A.\mathbf{x} = b, b \in \Re^m\}$

em que:

$h \rightarrow$ número de funções-objetivo (critérios);

$n \rightarrow$ número de variáveis do problema (de decisão e folga);

$m \rightarrow$ número de restrições do problema;

$X \rightarrow$ região admissível no espaço das decisões (ou das variáveis);

$\mathbf{x} \rightarrow$ vetor das variáveis do problema ($\mathbf{x} = [x_1\ x_2\ ...\ x_n]^T$ é a solução admissível);

c_j ($j = 1, 2, ..., h$) $\rightarrow$ vetor dos coeficientes da função-objetivo f_j;

$A \rightarrow$ matriz dos coeficientes ($m \times n$);

$b \rightarrow$ vetor dos termos independentes (recursos disponíveis).

A região admissível no espaço das funções-objetivo (o conjunto de todas as imagens dos pontos em X) pode ser definida da seguinte forma:

$$Z = \{\mathbf{z} \in \Re^h : \mathbf{z} = (f_1(\mathbf{x}), f_2(\mathbf{x}), ..., f_h(\mathbf{x})), \mathbf{x} \in X\}$$

Na resolução de problemas com apenas um critério, procura-se encontrar a solução ótima, ou seja, a solução admissível que otimize a função-objetivo, cujo valor é único, mesmo que existam soluções ótimas alternativas. Já em problemas com múltiplos critérios, esse conceito não é aplicável, uma vez que uma solução admissível que otimize um dos critérios não otimiza, em geral, os outros critérios, quando estes estão em conflito.

Portanto, na resolução de problemas multicritério procura-se encontrar uma *melhor solução de compromisso* para o AD, que possa constituir uma solução final do problema de decisão. Desta forma, a noção de **solução ótima** é substituída pela noção de **solução não-dominada** (chamada também de eficiente, ótima de Pareto ou não-inferior).

Uma solução admissível diz-se **dominada** por outra se, ao passar-se da primeira para a segunda, existir melhoria de pelo menos um dos critérios, permanecendo inalterados os restantes. Por outro lado, uma solução **não-dominada** caracteriza-se por não existir uma outra solução admissível que melhore simultaneamente todos os critérios, isto é, a melhoria em um critério é alcançada à custa de piorar pelo menos um dos outros [4]. Esta relação é comumente representada pelos símbolos $\succ$ e $\prec$, em que $\mathbf{x} \succ \mathbf{y}$ indica que **y** é dominada por **x**, ou **x** domina **y**.

Em alguns trabalhos, como o de Coello [17], usa-se o conceito de *dominância fraca* e *dominância forte*, onde a forte seria semelhante ao descrito no parágrafo anterior, e a fraca não considera a condição na qual pelo menos um critério tem que ser melhorado. Pode-se notar que uma solução fortemente dominada é também fracamente dominada, sendo a recíproca falsa.

Matematicamente, tem-se:

- Sejam $\mathbf{x} = [x_1\ x_2\ ...\ x_n\ ]^T$ e $\mathbf{y} = [y_1\ y_2\ ...\ y_n\ ]^T$ duas soluções admissíveis de um problema multicritério ($\mathbf{x}, \mathbf{y} \in X$). A solução **x** domina a solução **y** se, e somente se (para um problema de minimização):

  $f_j(x_1, x_2, ..., x_n) \leq f_j(y_1, y_2, ..., y_n)$ para qualquer j e,

  $f_j(x_1, x_2, ..., x_n) < f_j(y_1, y_2, ..., y_n)$ para algum j, com $j = 1, 2, ..., h$.

- Seja **x** uma solução admissível de um problema multicritério ($\mathbf{x} \in X$). A solução **x** diz-se não-dominada se, e somente se, não existir outra solução admissível **y** ($\mathbf{y} \in X$) que domine **x**.

Há ainda o conceito de **eficiência**, que é geralmente utilizado para pontos no espaço de decisão, enquanto o conceito de não-dominância é utilizado para a respectiva imagem no espaço das funções-objetivo, ou seja, uma solução não-dominada é a imagem de uma solução eficiente [4].

Matematicamente, tem-se:

- Uma solução admissível $\mathbf{x}$ ($\mathbf{x} \in X$) é eficiente se, e somente se, não existe outra solução admissível $\mathbf{y}$ ($\mathbf{y} \in X$) tal que $f(\mathbf{y}) \leq f(\mathbf{x})$ e $f(\mathbf{y}) \neq f(\mathbf{x})$, em que $f = (f_1, f_2, ..., f_h)$. Caso contrário, $\mathbf{x}$ é ineficiente. Ao conjunto das soluções eficientes, dá-se o nome de **fronteira eficiente** ou **conjunto de soluções não-inferiores**.
- Seja $\mathbf{x} \in Z$. Então $\mathbf{x}$ é não-dominado se, e somente se, não existe outro $\mathbf{y} \in Z$, tal que $\mathbf{y} \leq \mathbf{x}$ e $\mathbf{y} \neq \mathbf{x}$. Caso contrário, $\mathbf{x}$ é um vetor de critérios dominado.

Ao achar as soluções não-dominadas, pode-se traçar uma curva pertencente ao $\Re^h$ chamada de **fronteira de Pareto** (*Pareto-front*) ou curva minimal. Geralmente, o objetivo principal da otimização de um problema multicritério é conseguir descrever uma função, ou um conjunto de pontos que formem uma curva, que traça da forma mais aproximada possível o *Pareto-front*. A figura 4 mostra um exemplo de um *Pareto-front* para o problema da árvore geradora mínima biobjetivo.

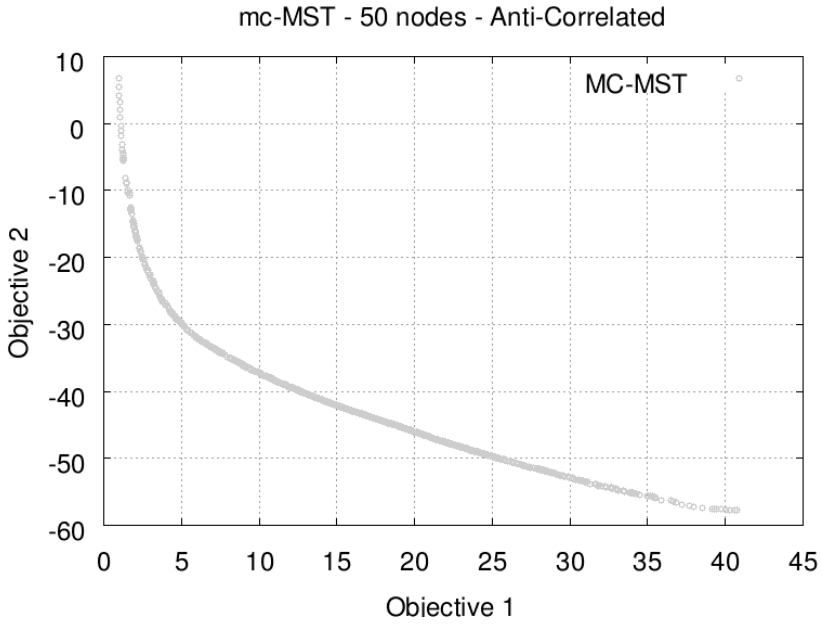


Figura 4: Fronteira de Pareto para um problema biobjetivo

Podem existir soluções não-dominadas que não estejam situadas na fronteira do invólucro convexo (contorno convexo), mas sim no seu interior. Às soluções não-dominadas que pertencem ao contorno convexo, dá-se o nome de **soluções suportadas**, as quais ainda podem ser extremas ou não-extremas. Às soluções que se encontram no interior do invólucro convexo, dá-se o nome de **soluções não-suportadas**. A figura 5 exemplifica os conceitos de solução não-dominada suportada (extrema e não-extrema) e não-suportada. As soluções 1, 2 e 4 são não-dominadas suportadas extremas (vértices); a solução 3 é não-dominada suportada não-extrema (é uma combinação convexa das soluções 2 e 4, pois encontra-se representada sobre a aresta $\overline{24}$); as soluções 5 e 6 são não-dominadas não-suportadas (dominadas pelas combinações convexas de 2 com 3 e de 3 com 4, respectivamente).

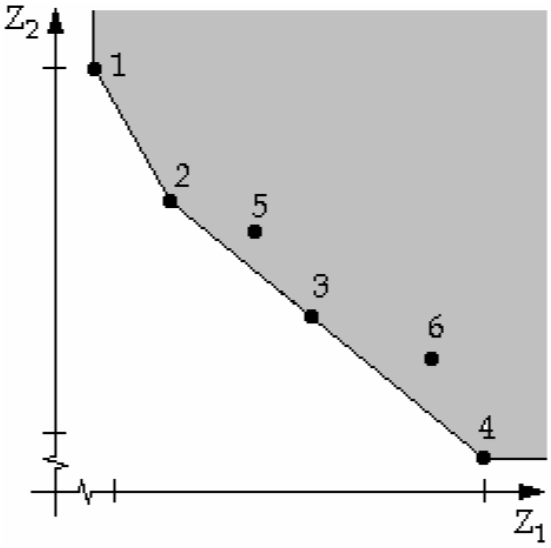


Figura 5: Tipos de soluções não-dominadas – suportadas e não-suportadas

Uma solução de compromisso satisfatória para o problema multicritério deverá ser eficiente, cujos valores das funções-objetivo associados àquela solução sejam satisfatórios para o AD e de tal modo que seja aceitável como solução final do processo de decisão. Desta forma, é apenas sobre o conjunto das soluções eficientes que deve recair a atenção do AD.

Retomando o exemplo da compra de um carro descrito no início da seção 2, considere apenas o preço e a eficiência do motor como critérios. A figura 6 mostra o gráfico bidimensional que confronta esses dois critérios dentre as alternativas existentes.

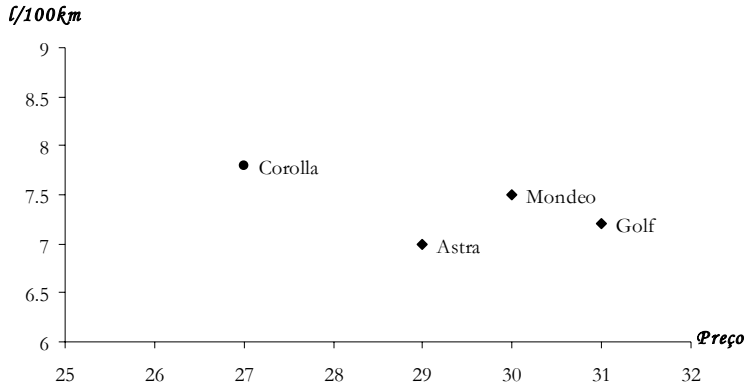


Figura 6: Espaço de critério para o problema da escolha do carro

Pode-se perceber que o Astra e o Corolla são soluções ótimas de Pareto (Ambos o Mondeo e o Golf são mais caros e menos eficientes que o Astra).

Considere X = {Golf, Astra, Mondeo, Corolla} o **conjunto admissível** (conjunto de alternativas) do problema de otimização. Denote o **preço** como sendo uma função f_1 e a

eficiência como sendo f_2, então f_i: X $\rightarrow$ $\Re$ são funções-objetivo (ou funções-critério) e o problema de otimização é:

$$\min_{x \in X}(f_1(x), f_2(x))$$

A imagem de X sobre $f = (f_1, f_2)$ é f(X). Entretanto, fica a pergunta: o que minimizar realmente significa nesse caso? Poderia-se classificar cada um dos critérios. Por exemplo, o preço poderia ser mais importante que a eficiência e esta mais importante que a potência. Então o vetor de critérios ($f_1(\mathbf{x})$, $f_2(\mathbf{x})$, $f_3(\mathbf{x})$) são comparados lexicograficamente. Teria-se então o Corolla como a única solução ótima ($\mathbf{x}^*$).

3.3. Métodos de Resolução de Problemas Multicritério

Ultimamente, tem-se realizado um esforço considerável no sentido de desenvolver métodos para o cálculo de soluções não-dominadas, tendo sempre em vista a contribuição que o AD pode fornecer na procura de tais soluções, através de seu sistema de preferências. Assim, com exceção do caso trivial (em que existe uma solução que otimiza simultaneamente todos os critérios), podem-se destacar três abordagens para resolver um problema multicritério: *modo a priori*, *modo a posteriori* e *modo interativo* [26].

No *modo a priori*, o AD começa indicando suas preferências, a partir das quais é possível transformar o problema inicial em um problema monocritério, por exemplo, através de uma função-utilidade (expressão analítica construída a partir da parametrização dos vários critérios). Desta forma, apesar de existir modelagem multicritério do problema, pretende-se otimizar a função-utilidade, cuja solução ótima será a solução final. A maior dificuldade deste processo está no fato de não ser possível, na maioria dos casos, obter uma representação matemática da função utilidade por parte do AD, ou seja, é difícil obter os parâmetros para construir uma função-utilidade que agregue, numa única dimensão, todos os critérios em análise.

No *modo a posteriori*, são calculadas todas as soluções eficientes do problema (ou parte delas), que depois serão colocadas à disposição do AD para serem avaliadas. Uma desvantagem desse modo é que há um elevado esforço computacional necessário para o cálculo das soluções eficientes.

No *modo interativo*, o AD expressa suas preferências através de um processo de diálogo, de forma a conduzir a pesquisa para a zona da região admissível onde se localizam as soluções que melhor correspondem ao seu sistema de preferências. Na prática, este modo tem mostrado ser o mais eficaz na pesquisa de uma solução final, através da utilização de processos de iteração homem-máquina, alternando fases de cálculo com fases de diálogo. A intervenção humana no processo de pesquisa da solução é uma das características que distingue os métodos de programação multicritério dos tradicionais métodos de programação matemática (um único critério).

3.4. Sistemas de Apoio à Decisão

A partir do final da década de 80, tem-se observado um desenvolvimento considerável de métodos interativos, os quais têm vindo a constituir meios de apoio à tomada de decisão, em inúmeros problemas em áreas como engenharia, gestão e planejamento. Verificou-se também que os cientistas de diferentes áreas passaram a participar cada vez mais em programas de investigação multidisciplinar [4].

A característica principal dos métodos interativos é a existência de fases de cálculo, que alternam com fases de diálogo. Em cada interação, o AD analisa uma ou mais soluções, que ao serem confrontadas com suas preferências, contribuem para a orientação do processo de pesquisa de novas soluções durante o processo de decisão. Assim, podem ser tomados diferentes caminhos, conduzindo o processo de cálculo para a zona da região admissível onde se encontram as melhores soluções. A condição de parada do processo interativo é a satisfação do AD, e não um teste de convergência de alguma função-utilidade.

Um dos princípios dos sistemas de apoio à decisão é ajudar interativamente o AD em todo o processo de busca da solução, reduzindo progressivamente o âmbito da pesquisa e minimizando tanto o esforço computacional quanto o esforço mental exigido do AD. A interação homem-máquina é uma componente fundamental nos sistemas de apoio à decisão, tendo como objetivos contribuir para a criação do sistema de preferências do AD e ampliar as suas capacidades de processamento de informação e decisão, tornando-se bastante importante na resolução de problemas multicritério.

3.5. Otimização Multicritério Evolucionária

O uso do termo **otimização multicritério evolucionária** consiste no uso de algoritmos evolucionários para manusear mais de uma função-objetivo ao mesmo tempo [16].

Ao longo dos anos, foram desenvolvidas inúmeras técnicas para lidar com problemas de otimização multicritério. Contudo, apenas nos últimos anos é que os pesquisadores puderam perceber a potencialidade dos algoritmos evolucionários na resolução destes problemas. O uso dos algoritmos evolucionários na resolução dos problemas de otimização multicritério foi sugerido por Rosenberg na década de sessenta [79], mas esta área de pesquisa permaneceu inexplorada cerca de vinte e cinco anos. Recentemente é que alguns pesquisadores mostraram interesse em pesquisas relacionadas a essa área. A primeira implementação de uma abordagem evolucionária multicritério foi do pesquisador Schaffer (*Vector Evaluation Genetic Algorithm* – VEGA [82]), introduzida em meados da década de oitenta e apresentada na *Primeira Conferência Internacional em Algoritmos Genéticos* (1985).

Coello revisa alguns conceitos básicos relacionados ao uso dos algoritmos evolucionários multiobjetivo [16]. Ele também apresenta uma revisão da literatura com

algumas técnicas que usam o conceito de otimização multiobjetivo evolucionária [17], indicando algumas das principais aplicações, suas vantagens, desvantagens e grau de aplicabilidade.

Os algoritmos evolucionários são particularmente apropriados para resolver problemas de otimização multicritério, justamente porque eles tratam simultaneamente com um conjunto de possíveis soluções (população). Isto permite encontrar vários membros do conjunto de soluções ótimas de Pareto em uma única execução do algoritmo, ao invés de ter que realizar uma série de execuções em separado, como é o caso das técnicas tradicionais de programação matemática.

3.6. Árvore Geradora Mínima Multicritério

Considere um grafo conectado não-direcionado $G = (V, E)$, onde $V = \{v_1, v_2, ..., v_n\}$ é um conjunto finito de vértices, e $E = \{e_1, e_2, ..., e_m\}$ é um conjunto finito de arestas representando conexões entre os vértices. Cada aresta tem p números reais positivos associados, representando p atributos definidos nela e denotado com $\mathbf{w}_i = (w_{1i}, w_{2i}, ..., w_{pi})$ com $(i = 1, 2, ..., m)$. Na prática, w_{ki} $(k = 1, 2, ..., p)$ pode representar distância, custo ou outros atributos.

Considere $\mathbf{x} = (x_1 x_2 ... x_m)$ conforme definido a seguir:

$$x_i = \begin{cases} 1 & \text{se a aresta } e_i \text{ é selecionada,} \\ 0 & \text{caso contrário.} \end{cases}$$

Então, uma árvore geradora do grafo G pode ser expressa pelo vetor $\mathbf{x}$. Considere X como sendo o conjunto de todos os vetores correspondentes às árvores geradoras no grafo G, o problema da árvore geradora mínima multicritério (AGM-mc) pode ser formulado da seguinte forma:

$$\min \quad z_1(x) = \sum_{i=1}^{m} w_{1i} x_i$$

$$\min \quad z_2(x) = \sum_{i=1}^{m} w_{2i} x_i$$

$$\cdots$$

$$\min \quad z_p(x) = \sum_{i=1}^{m} w_{pi} x_i$$

sendo $\mathbf{x} \in X$, onde $z_i(\mathbf{x})$ corresponde ao i-ésimo objetivo a ser minimizado para o problema. Como já foi dito no início desta seção, se houver, em adição, uma constante d indicando o máximo grau de cada vértice, então o problema é definido como Árvore Geradora Mínima Restrita em Grau Multicritério (AGM-mc$d$).

Se comparado ao problema da árvore geradora mínima tradicional, a única diferença está no número de objetivos. Contudo, é justamente por causa desses múltiplos objetivos existem e serem conflitantes entre si que não se pode determinar qual aresta tem o peso mínimo e formar a árvore geradora mínima por algum algoritmo tradicional (por exemplo, o algoritmo de Prim). Se esses múltiplos objetivos forem transformados em um único objetivo, de acordo com algum critério, então seria fácil resolver o problema usando algum algoritmo tradicional para AGM. A desvantagem é que apenas uma única solução do conjunto de soluções ótimas de Pareto seria obtida. Entretanto, esta transformação não é trivial na prática. Portanto, é necessário desenvolver técnicas que lidem com este problema multicritério e que forneça bastantes alternativas para o AD fazer a escolha pela solução [94].

3.7. Algoritmos Aplicados ao Problema da AGM-mc*d*

Esta seção apresenta alguns algoritmos encontrados na literatura para resolver o problema da Árvore Geradora Mínima Multicritério, alguns adaptados à restrição de grau.

3.7.1. Algoritmo Prim-mc*d*

Definido por Knowles e Corne para AGM-mc [54] e para AGM-mc*d* [57], este algoritmo encontra no mínimo um subconjunto de soluções ótimas de Pareto para alguma instância do problema AGM-mc*d*. A seguir, será mostrado como o algoritmo clássico de Prim [71] para AGM funciona e, em seguida, é mostrada a sua adaptação para o problema da AGM-mc*d*.

O algoritmo de Prim usa dois conjuntos de vértices, C (conectados) e U (não conectados). Inicialmente, C contém apenas um vértice escolhido aleatoriamente, enquanto U contém todos os outros vértices restantes. O algoritmo procede movendo vértices de U para C, sendo um por vez, até que o conjunto U esteja vazio. Cada movimento é associado com a adição de uma aresta específica à árvore em crescimento. Quando U estiver vazio, a árvore estará completa. A aresta escolhida a cada passo é a de menor peso dentre aquelas que seguem a seguinte regra: conectar o vértice $u \in U$ ao vértice $c \in C$ e sua adição na árvore não formar ciclo. Ao adicionar a aresta (u, c), o vértice u é movido de U para C.

Este procedimento resolve rapidamente e eficientemente o problema da AGM com um único critério. Para resolver o problema da AGM-*d*, basta que, a cada passo na construção da árvore, seja verificada a violação da restrição de grau antes de adicionar a próxima aresta. Uma simples modificação, na qual se adiciona um teste extra de possibilidade ao mover um vértice entre os dois conjuntos, permite construir soluções para o problema da AGM-mc*d*, só que neste caso trata-se de uma heurística aproximativa, ao invés de um método exato.

Ao preparar a versão multicritério do algoritmo de Prim, deve-se primeiro observar que as soluções ótimas de Pareto para o problema da AGM-mc*d* podem ser encontradas simplesmente usando o algoritmo básico de Prim (ou qualquer outro algoritmo que resolva a AGM com um único objetivo), substituindo o vetor de pesos por uma **soma ponderada**. Isto pode ser conseguido desde que, para algum vetor de escalarização $\boldsymbol{\lambda} = (\lambda^1, \lambda^2, ..., \lambda^K)$ com $\sum_{k=1}^{K} \lambda^k = 1$, e $\lambda^k \geq 0$, $k = 1...K$, tenha-se o seguinte:

$$\sum_{(i,j)\in E_T} \lambda.w_{i,j} = \sum_{k=1}^{K} \lambda^k \left(\sum_{(i,j)\in E_T} w_{i,j}^k \right)$$

em que $E_T \subset E$ é o conjunto das arestas pertencentes à árvore geradora T.

Uma árvore que otimiza o termo da esquerda será uma solução para o problema da AGM com um único objetivo definido pela escalarização, facilmente encontrada pelo algoritmo de Prim. Mas a reescrita deste termo no lado direito revela que esta árvore deve ser também uma solução de Pareto (*Pareto-front*) da AGM multicritério correspondente. Caso contrário, uma árvore existe a qual deve melhorar um ou mais termos dos pesos somados, contradizendo a suposição que esta soma já seria mínima.

Portanto, pode-se encontrar uma solução ótima para o problema da AGM-mc*d* trocando o vetor de pesos definido em cada aresta pertencente a *G* por um peso escalar *b* formado pelo produto interno de $\boldsymbol{\lambda}$ e **w**, $b = \boldsymbol{\lambda}.\mathbf{w}$, e encontrando uma árvore geradora que minimize a soma dos pesos das arestas escalarizadas.

O pseudocódigo do algoritmo Prim-mc*d* é descrito a seguir:

Algoritmo: Prim-mc*d*

$U \Rightarrow$ conjunto dos vértices não conectados
$C \Rightarrow$ conjunto dos vértices conectados
$E_T \Rightarrow$ conjunto de arestas da árvore geradora T
$S \Rightarrow$ conjunto das árvores geradoras não-dominadas

$S \leftarrow \varnothing$
$i \leftarrow 0$
enquanto $\left(i < \binom{s+K-1}{K-1} \right)$ /* Laço principal */
  $\lambda \leftarrow$ proxVetor(λ)
  $E_T \leftarrow \varnothing$
  $U \leftarrow V$
  $C \leftarrow \varnothing$
  $v \leftarrow$ aleatório(V) /* Início do algoritmo de Prim */
  $C \leftarrow C \cup v$
  $U \leftarrow U \setminus v$
  enquanto (|C| < |V|)
    Selecione uma aresta (u, v) com $u \in C$, $v \in U$ tal que
    $\forall i \in C, j \in U \ \sum_{k=1}^{K} \lambda^k . w_{u,v}^k \leq \sum_{k=1}^{K} \lambda^k . w_{i,j}^k$
    $E_T \leftarrow E_T \cup (u, v)$ – *adiciona se não violar restrição de grau*
    $C \leftarrow C \cup v$
    $U \leftarrow U \setminus v$
  fim_enquanto
  $S \leftarrow S \cup T \leftarrow (V, E_T)$
  $i \leftarrow i + 1$
fim_enquanto
retorne (S)

O algoritmo iterativamente muda o vetor de escalarização λ e usa o algoritmo de Prim para encontrar um conjunto de soluções ótimas para uma AGM-mc*d*. A cada iteração do laço principal, um novo vetor de escalarização λ é gerado pela função *proxVetor*(λ), que gera sucessivamente cada vetor de escalarização normalizado, λ, com componentes igual a l/s, $l = 0..s$, onde s é um parâmetro que controla o número de diferentes vetores que serão gerados, resultando em $\binom{s+K-1}{K-1}$ diferentes vetores de escalarização e, para cada vetor λ gerado, o algoritmo de Prim é aplicado.

Para um grande número de diferentes vetores de escalarização, o algoritmo Prim-mc*d* pode gerar uma aproximação para S^*, o conjunto de árvores geradoras ótimas de Pareto, sendo satisfatório em muitos casos. Contudo, ele geralmente não consegue encontrar o conjunto S^* completo, porque só encontra soluções nas extremidades do espaço de soluções de Pareto – as chamadas soluções eficientes *suportadas* [54].

3.7.2. Algoritmo de Zhou & Gen

Trata-se de um algoritmo genético, descrito em [94]. Como já descrito na seção 3.5, por ser baseado em população, um algoritmo genético pode fornecer um conjunto de soluções candidatas (ou soluções ótimas de Pareto) de uma só vez, ao invés de apenas uma por execução.

Quanto à representação dos cromossomos, é usado um método de permutação conhecido como **numeração de Prüfer** [72], o que torna possível permutar $(n - 2)$ dígitos, ao invés de n dígitos, a fim de representar uma árvore com n vértices onde cada dígito é um inteiro entre 1 e n (inclusive). Para qualquer árvore em um grafo completo, sempre há no mínimo dois vértices que são folhas (há apenas uma aresta conectada a eles). Baseado neste fato, pode-se facilmente construir o número de Prüfer de acordo com o procedimento descrito a seguir.

Procedimento de Codificação

Passo 1: Considere j o menor vértice-folha rotulado numa árvore rotulada T.
Passo 2: Escolha k como sendo o dígito a compor o número de Prüfer na permutação se o vértice k é incidente ao vértice j.
Passo 3: Remova o vértice j e a aresta de j até k. Tem-se agora uma árvore com $(n - 1)$ vértices.
Passo 4: Repita os passos anteriores até que reste apenas uma aresta. O resultado será o número de Prüfer ou permutação com $(n - 2)$ dígitos.

A figura 7 ilustra uma árvore e seu respectivo número de Prüfer. Também é possível gerar uma única árvore a partir do número de Prüfer seguindo o procedimento a seguir.

Procedimento de Decodificação

Passo 1: Considere P como sendo o número de Prüfer, e $\overline{P}$ como sendo o conjunto de todos os vértices não incluídos em P.
Passo 2: Considere j o vértice com o menor rótulo em $\overline{P}$, e k o dígito mais a esquerda de P. Adicione a aresta de j até k na árvore. Remova j do conjunto $\overline{P}$ e k do número P. Se k não tiver mais ocorrências em P, coloque-o em $\overline{P}$. Repita o processo até não haja mais dígitos em P.
Passo 3: Se não houver mais dígitos em P, então há exatamente 2 vértices, r e s, no conjunto $\overline{P}$. Adicione uma aresta de r para s na árvore e então é formada uma árvore com $(n - 1)$ arestas.

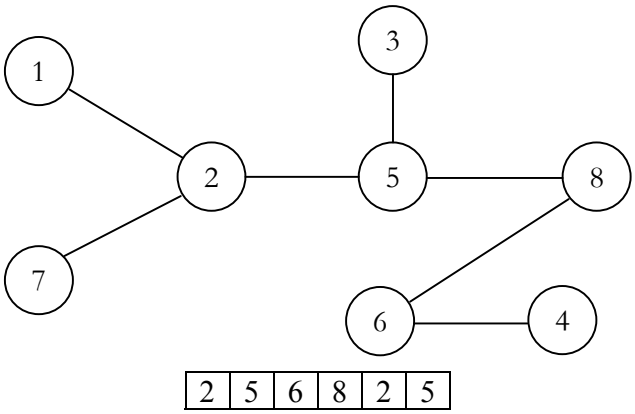


Figura 7: Uma árvore e seu número de Prüfer

O tamanho do cromossomo é $(n - 2)$, o tamanho do espaço de busca é n^{n-2} e a relação entre o número de Prüfer e a árvore geradora é de um-para-um, ou seja, um número de Prüfer corresponde a apenas uma árvore geradora e vice-versa. Por essa razão, a probabilidade de produzir aleatoriamente uma árvore geradora é definitivamente igual a 1, o que significa que esta codificação é capaz de representar todas as possíveis árvores e qualquer população inicial ou descendentes de operações de cruzamento e mutação ainda formam uma árvore.

Para este algoritmo é adotado o operador de cruzamento uniforme [86]. O cruzamento uniforme primeiramente gera uma máscara de cruzamento aleatória e então ocorre a troca relativa de genes entre os pais de acordo com essa máscara. Uma máscara de cruzamento é simplesmente uma cadeia de caracteres binária com o mesmo tamanho do cromossomo. A paridade de cada bit na máscara determina, para cada bit correspondente em um descendente, de qual pai ele receberá o bit. A figura 8 ilustra esta operação.

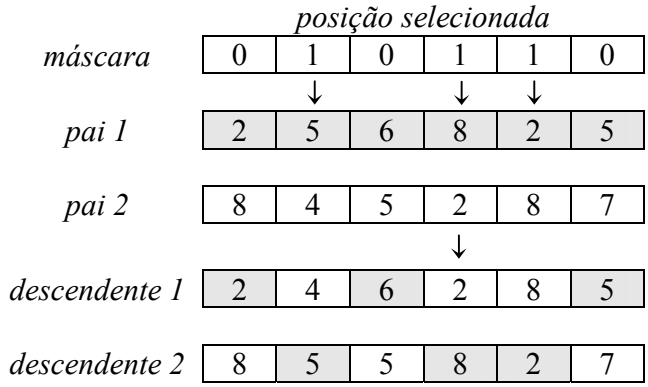


Figura 8: Operação de cruzamento

A mutação é executada como uma perturbação aleatória dentro de uma faixa admissível de números inteiros entre 1 e n. A figura 9 ilustra este caso.

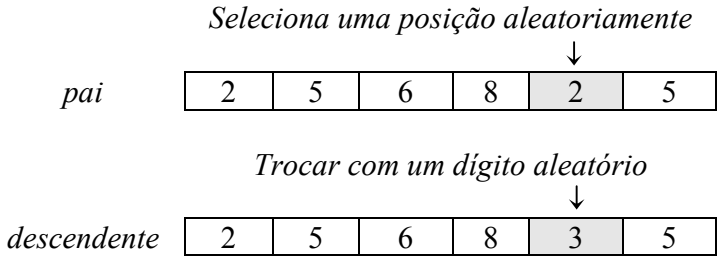


Figura 9: Operação de mutação

No processo de busca, a solução ótima para o problema AGM-mc*d* é um conjunto de soluções – as soluções ótimas de Pareto. Com isso, diferentes estratégias de avaliação e seleção resultarão em diferentes conjuntos de soluções. Para este algoritmo, há duas estratégias de avaliação e seleção as quais refletem as preferências do agente de decisão na abordagem multicritério.

Estratégia 1: A fim de avaliar a adequação de cada indivíduo no algoritmo, para o problema AGM-mc*d*, pode-se desenvolver uma função de avaliação adaptativa baseada no método de ponderação dos objetivos [94]. Os valores de adequação de todos os indivíduos são calculados de acordo com essa função. A cada geração, o conjunto de soluções de Pareto candidatas é atualizado e a função de avaliação adaptativa é sempre recalculada no decorrer do processo de busca. Em termos de espaço (geométrico) de critérios, essa função age como se fosse um hiperplano. Sobre a ação da seleção, o hiperplano força todas as soluções de Pareto candidatas a moverem na direção do seu gradiente negativo, se o problema é minimizado, e as expande em direção ao ponto ideal o mais próximo possível (figura 10).

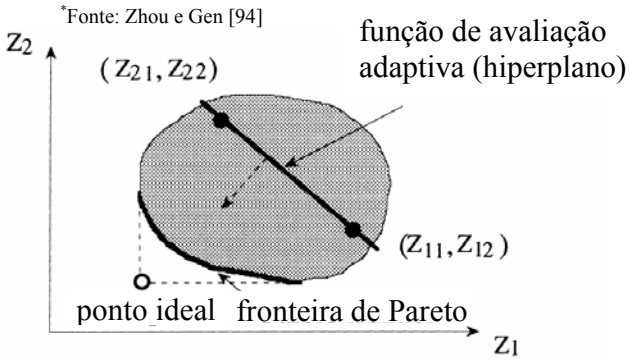


Figura 10: Processo evolucionário da Estratégia 1

O procedimento de avaliação pode ser operado como mostrado a seguir:

Procedimento de Avaliação para a Estratégia 1

Passo 1: Decodifique todos os cromossomos e calcule seus valores de objetivo em cada função-objetivo.

Passo 2: Determine λ_k (k = 1, 2, ..., p), de acordo com o método de ponderação de objetivos.

Passo 3: Determine o valor da adequação de todos os indivíduos de acordo com a fórmula: $avalie(T) = \sum_{k=1}^{p} \lambda_k . z_k$.

Estratégia 2: Na estratégia 1, por causa da ação seletiva em direção ao ponto ideal, é possível fazer com que todas as soluções ótimas de Pareto fiquem na região próxima a este ponto, o que muitas vezes pode não ser desejável na prática. Em alguns casos, pode-se desejar obter as soluções distribuídas ao longo da fronteira de Pareto, a qual pode fornecer bastantes alternativas para o agente de decisão escolher. Nesta estratégia foi adotada a *técnica de classificação não-dominada* sugerida por Srinivas e Deb [85] que idealiza esta situação.

Nesta técnica, antes da seleção ser executada a população é classificada com base em uma não-dominação do indivíduo. Os indivíduos não-dominados presentes na população são primeiro identificados da população atual. Então, todos estes indivíduos irão constituir a primeira fronteira não-dominada na população e são associados a um alto valor fictício de adequação, que serve para dar um potencial produtivo igual a todos estes indivíduos não-dominados. Para manter diversidade na população, estes indivíduos classificados são então *compartilhados* com seus valores fictícios de adequação. O compartilhamento significa realizar a operação de seleção usando os valores de adequação degradados, que são obtidos dividindo o valor original da adequação de um indivíduo por um valor proporcional ao número de indivíduos ao redor dele. Isto causa múltiplos pontos ótimos de Pareto co-existentes na população.

Após o compartilhamento, estes indivíduos não-dominados são desconsiderados temporariamente, a fim de que seja processado o resto da população do mesmo modo para que seja identificada a segunda fronteira não-dominada. A estes novos pontos não-dominados são associados novos valores fictícios de adequação menores do que a menor adequação *compartilhada* da fronteira anterior. Este processo é realizado até que toda a população seja classificada em várias fronteiras. A população é então reproduzida de acordo com os valores fictícios de adequação. O procedimento de avaliação pode ser resumido como mostrado a seguir:

Procedimento de Avaliação para a Estratégia 2

Passo 1: Determine todos os indivíduos não-dominados P_c da população atual e associe um alto valor fictício para a adequação deles.

Passo 2: Calcule o *contador de nicho* de cada indivíduo, m_j:

$$m_j = \sum_{k \in P_c} sh(d_{jk}), \text{ onde } sh(d_{jk}) = \begin{cases} 1 - \left(\dfrac{d_{jk}}{\delta_{share}} \right)^2 & se\ d_{jk} < \delta_{share}, \\ 0 & caso\ contrário, \end{cases}$$

e d_{jk} é a distância fenotípica entre dois indivíduos j e k no conjunto de soluções não-dominadas atual, e δ_{share} é a distância fenotípica máxima entre quaisquer dois indivíduos que se tornem membros de um nicho.

Passo 3: Calcule a adequação *compartilhada* de cada indivíduo dividindo os seus valores fictícios de adequação pelo seu respectivo *contador de nicho*.

Passo 4: Desconsidere todos os indivíduos classificados como não-dominados, retorne ao passo 1 e continue o processo até que toda a população seja classificada.

Com relação à seleção, apenas o método da roleta é adotado. Os indivíduos do primeiro conjunto não-dominado têm mais chance de serem selecionados na próxima geração do que aqueles no segundo conjunto, e assim por diante.

Por causa da existência da restrição de grau em cada vértice, os cromossomos gerados aleatoriamente na população inicial e os descendentes produzidos pelos operadores de mutação e cruzamento podem estar inválidos. É necessário modificar o grau de cada vértice ilegal no cromossomo.

Considere V_{dc} como sendo o conjunto de vértices cujo grau ainda não foi verificado ou modificado no cromossomo. Se um vértice viola a restrição de grau, isto quer dizer que o número de vezes que este vértice aparece no cromossomo é maior do que (d – 1). Então decremente esse número de entradas do vértice no cromossomo verificando a entrada extra e aleatoriamente trocando com outro vértice pertencente a V_{dc}. A figura 11 ilustra um caso para restrição de grau igual a três (d = 3).

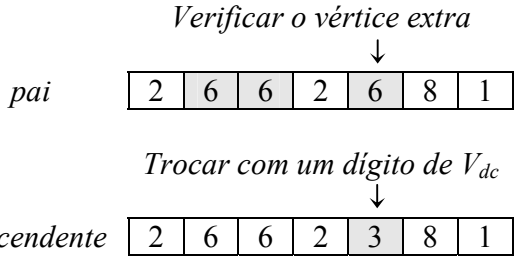


Figura 11: Modificando o grau de cromossomo (*d* = 3)

O pseudocódigo do algoritmo genético de Zhou & Gen é mostrado abaixo:

Procedimento: algoritmo genético para AGM-mc

t ← 0
inicie a população inicial *P*(0)
modificar o grau dos vértices em *P*(0), se necessário
determine o conjunto de soluções não-dominadas *E*(0)
enquanto (**não** condição-de-parada)
    recombine *P*(t) para gerar a população de descendentes *C*(t)
    modificar o grau dos vértices em *P*(t), se necessário
    atualize *E*(t)
    se (preferência na Estratégia 1)
        avalie *P*(t) e *C*(t) pela Estratégia 1
        selecione *P*(t+1) a partir de *P*(t) e *C*(t) pela Estratégia 1
    senão
        avalie *P*(t) e *C*(t) pela Estratégia 2
        selecione *P*(t+1) a partir de *P*(t) e *C*(t) pela Estratégia 1
    fim_se
    t ← t + 1
fim_enquanto

3.7.3. Algoritmo AESSEA

Algoritmo genético, descrito por Knowles e Corne para AGM-mc [54] e para AGM-mc*d* [57], baseado em procedimentos já implementados em outro algoritmo chamado PAES (*Pareto Archived Evolutionary Strategy*) [56]. O AESSEA (*Archived Elitist Steady-State Evolutionary Algorithm*) mantém um conjunto de soluções não-dominadas arquivadas e usa este conjunto para estimar a qualidade das novas soluções geradas. Ele usa o método RPM (*Randomized Primal Method*) [55] para gerar a população inicial, e depois utiliza apenas a codificação direta (*Edge-set*) [73] e seus respectivos operadores de mutação e cruzamento. A seguir serão descritos, em resumo, o método RPM e a codificação direta, considerando primeiro apenas um objetivo e depois mostrando a adaptação para multicritério.

Método RPM

Conforme descrito na seção 3.7.1, o algoritmo de Prim (após as condições de início, na qual a árvore é configurada para conter um único vértice arbitrário) itera por ($|V| - 1$) passos. Em cada um destes passos a aresta com custo mínimo é encontrada, ligando um vértice desconectado à árvore. No método RPM, cada um destes passos agora envolve a escolha de qualquer aresta possível, ao invés de uma com custo mínimo, embora haja uma tendência às arestas de baixo custo.

O RPM usa uma estrutura de dados dinâmica que consiste numa tabela T_b, que associa uma lista ordenada de arestas a cada vértice. Na inicialização, a entrada em T_b para algum vértice i é uma lista de todas as arestas incidentes a i, em ordem ascendente de pesos. A figura 12 mostra a matriz de adjacências do grafo tomado como exemplo, enquanto a figura 13 mostra a construção inicial da tabela T_b a partir desse grafo.

j	1	2	3	4	5	6	7	8	9
1	–	224	224	361	671	300	539	800	943
2		–	200	200	447	283	400	728	762
3			–	400	566	447	600	992	949
4				–	400	200	200	539	583
5					–	600	447	781	510
6						–	283	500	707
7							–	361	424
8								–	500
9									–

Figura 12: Matriz de adjacências do grafo tomado como exemplo para o RPM

Enquanto o RPM executa, a árvore geradora é construída iterativamente, e T_b é alterada dinamicamente sempre que uma nova aresta é adicionada à árvore. O objetivo é que essa lista de arestas ordenadas contenha apenas arestas válidas e que não tenham sido incluídas na árvore.

escolha da aresta	vértice *i*								
	1	2	3	4	5	6	7	8	9
1 (peso)	3 (224)	3 (200)	2 (200)	2 (200)	4 (400)	4 (200)	4 (200)	7 (361)	7 (424)
2 (peso)	2 (224)	4 (200)	1 (224)	7 (200)	7 (447)	7 (283)	6 (283)	9 (500)	8 (500)
3 (peso)	6 (300)	1 (224)	4 (400)	6 (200)	2 (447)	2 (283)	8 (361)	6 (500)	5 (510)
4 (peso)	4 (361)	6 (283)	6 (447)	1 (361)	9 (510)	1 (300)	2 (400)	4 (539)	4 (583)
5 (peso)	7 (539)	7 (400)	5 (566)	3 (400)	3 (566)	3 (447)	9 (424)	2 (728)	6 (707)
6 (peso)	5 (671)	5 (447)	7 (600)	5 (400)	6 (600)	8 (500)	5 (447)	5 (781)	2 (762)
7 (peso)	8 (800)	8 (728)	9 (949)	8 (539)	1 (671)	5 (600)	1 (539)	1 (800)	1 (943)
8 (peso)	9 (943)	9 (762)	8 (992)	9 (583)	8 (781)	9 (707)	3 (600)	3 (992)	3 (949)

Figura 13: Construção inicial da tabela T_b

Essa tabela T_b pode ser usada no algoritmo de Prim-*d* (AGM com restrição de grau), no passo onde é necessário selecionar a aresta de menor peso que conecta um vértice já adicionado na árvore a outro ainda não adicionado. Para executar este passo, verificam-se todos os vértices em T_b que já estão na árvore e retorna a aresta do topo (menor peso) de cada uma das listas associadas ao vértice em questão, originando o conjunto L de arestas de baixo custo, de onde a menor aresta é então selecionada.

O RPM trabalha semelhante ao algoritmo de Prim-*d*, com a exceção de que nem sempre se escolhe a primeira aresta (topo) em cada lista para fazer parte de L. Ao invés

disso, constrói-se o conjunto L escolhendo a $a(i, d_i)$-ésima aresta de cada entrada apropriada em T_b, isto é, para o vértice i contendo o grau corrente d_i escolhe-se a $a(i, d_i)$-ésima aresta de menor peso da lista correspondente. Os valores do alelo a, $1 \leq a \leq n$, para cada vértice i em cada grau $d_i < d$ são dados como entrada para o RPM sob a forma de um cromossomo tabular ou vetor-solução (figura 14).

grau atual do vértice i: d_i	vértice i								
	1	2	3	4	5	6	7	8	9
1	**2**	1	4	2	3	2	1	**5**	**1**
2	3	1	**2**	**1**	1	**2**	1	3	2

Figura 14: Exemplo de um cromossomo usando RPM

O RPM usa um cromossomo tabular (duas dimensões) de tamanho n x d-1, onde n é o número de vértices do grafo e d é o grau de restrição. Os cromossomos são codificados para construir uma árvore geradora mínima restrita em grau S iterativamente da seguinte forma:

Primeiro passo: coloque aleatoriamente um vértice em S.

Passo geral: adicione uma aresta que uma um vértice i presente em S e cujo grau d_i seja menor do que d, a um vértice não presente em S.

A escolha da aresta no passo geral depende do peso dela no grafo em questão e dos valores armazenados no cromossomo.

O *passo geral* é implementado conforme abaixo:

- Para cada vértice i em S cujo grau atual d_i é menor do que d construa uma lista ordenada l_i contendo cada aresta incidente a i que una aos vértices não contidos em S, em ordem crescente de peso.
- Para cada lista l_i, procure o valor do alelo $a(i, d_i)$ armazenado no cromossomo na posição (i, d_i). Coloque a (i, d_i)-ésima aresta da lista l_i no conjunto de arestas L.
- Selecione a aresta de menor peso contida em L.

Para ilustrar o RPM, considere o grafo com restrição de grau 3 parcialmente construído da figura 15 e seu respectivo cromossomo mostrado na figura 14. Após algumas execuções do RPM, há seis arestas (sete vértices) inclusas na árvore. Para adicionar a próxima aresta, a lista de arestas para cada um dos vértices será verifica na tabela T_b para a situação corrente (Figura 16).

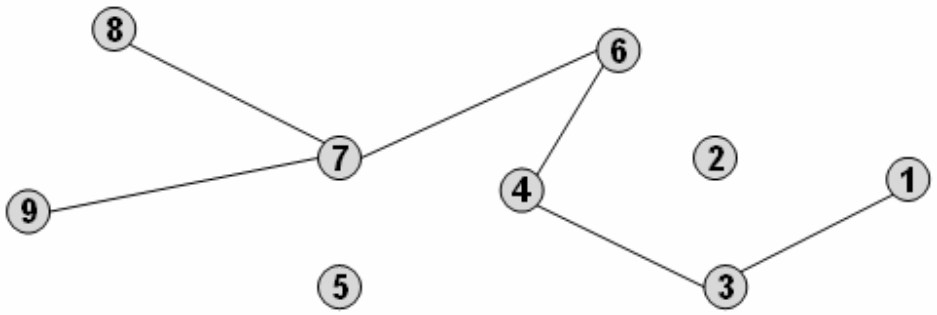


Figura 15: Grafo parcialmente construído (*d* = 3)

De acordo com a figura 16, todos os vértices contêm apenas duas arestas válidas: as arestas que conectam os vértices 2 e 5. Na figura 14, os alelos que serão usados na escolha da próxima aresta estão destacados. Estes alelos são então usados para selecionar as arestas que irão fazer parte do conjunto *L*. As arestas escolhidas de acordo com o cromossomo estão destacadas na figura 16, sendo que para o vértice 8, onde a escolha foi 5, seleciona a maior escolha aceitável. Os vértices 2 e 5 são ignorados, pois ainda não estão na árvore, e o vértice 7 também é ignorado, pois já atingiu a restrição de grau. Nesse caso, $L = \{(1, 5), (3, 5), (4, 2), (6, 5), (8, 5), (9, 5)\}$. A aresta (4, 2) é então escolhida para ser adicionada na árvore, pois possui o menor custo do conjunto *L*.

escolha da aresta	vértice *i*								
	1	2	3	4	5	6	7	8	9
1 (peso)	2 (224)	–	2 (200)	**2 (200)**	–	2 (283)	–	2 (728)	**5 (510)**
2 (peso)	**5 (611)**	–	**5 (566)**	5 (400)	–	**5 (600)**	–	**5 (781)**	2 (762)

Figura 16: Tabela T_b para o grafo parcialmente construído com *d* = 3

No AESSEA, os alelos dos cromossomos do RPM (usado apenas na inicialização) são definidos inicialmente usando uma função exponencial definida por:

$$q = \left\lfloor e^{x^p \log(|V|)} \right\rfloor$$

em que *x* é número aleatório distribuído uniformemente entre [0, 1[e *p* é um parâmetro que controla quão forte é a orientação a valores baixos. ($p = 2$, neste algoritmo).

É necessário adaptar o RPM para o problema multicritério, Prim-mc*d*. Em adição ao cromossomo, um vetor de escalarização $\lambda = (\lambda_1, \lambda_2, ..., \lambda_K)$ é inicializado a partir de um distribuição aleatória uniforme tal que $\sum_{k=1}^{K} \lambda_k = 1$ e $\forall k \in 1..K, \lambda_k \geq 0$.

Cada cromossomo é então codificado usando o RPM e com o vetor de aresta substituído por um conjunto de escalares $b_{i,j}$, produzidos pelo produto interno do vetor de escalarização λ e *w*: $\forall (i, j) \in E, b_{i,j} = \lambda \cdot w_{i,j}$.

Codificação Direta

Uma árvore geradora T é explicitamente representada pela lista de arestas as quais abrangem esta árvore. O operador de mutação simplesmente troca uma das arestas de E_T com outra aresta que não esteja presente em T, respeitando as condições de restrição. Segue abaixo a descrição da mutação para a codificação direta (a figura 17 ilustra um exemplo de mutação):

- Uma aresta que atualmente não está na árvore T é escolhida para inserção; esta escolha é baseada nas arestas de menor peso.
- Esta inserção irá gerar um ciclo em T; é feita então uma escolha aleatória entre as arestas que formam o ciclo (com exceção da aresta inserida) e a aresta escolhida é então removida de E_T.

Deve-se tomar cuidado para que nenhum dos vértices da aresta inserida tenha grau d, já que sua inserção poderia causar uma violação na condição de restrição de grau.

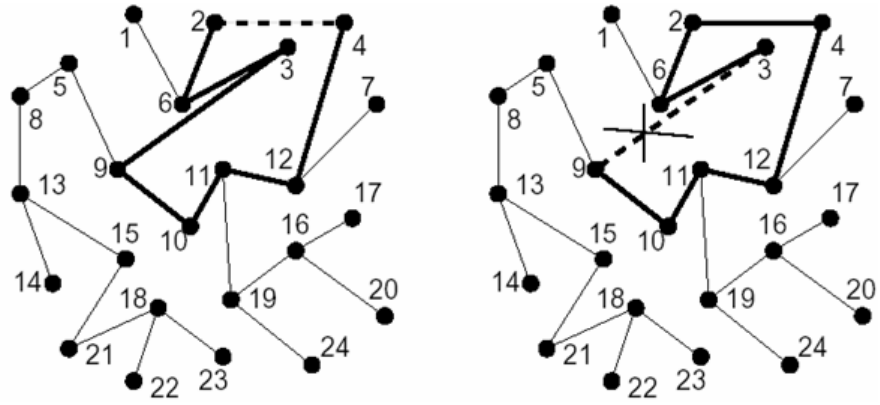


Figura 17: Exemplo de mutação, inserindo aresta (2, 4) e removendo aresta (3, 9)

O operador de cruzamento é construído baseado no princípio de herança de tantas arestas quanto possíveis de ambos os pais, A figura 18 ilustra um exemplo de cruzamento. Inicialmente, um filho é formado pela interseção dos conjuntos de arestas dos pais, $E_T = E_T^1 \cap E_T^2$, e então quase sempre forma uma floresta de árvores não-geradoras. As arestas restantes (que pertencem a apenas um dos pais, $E_T^1 \cup E_T^2 \setminus E_T$) são sucessivamente selecionadas e incluídas na árvore, se possível.

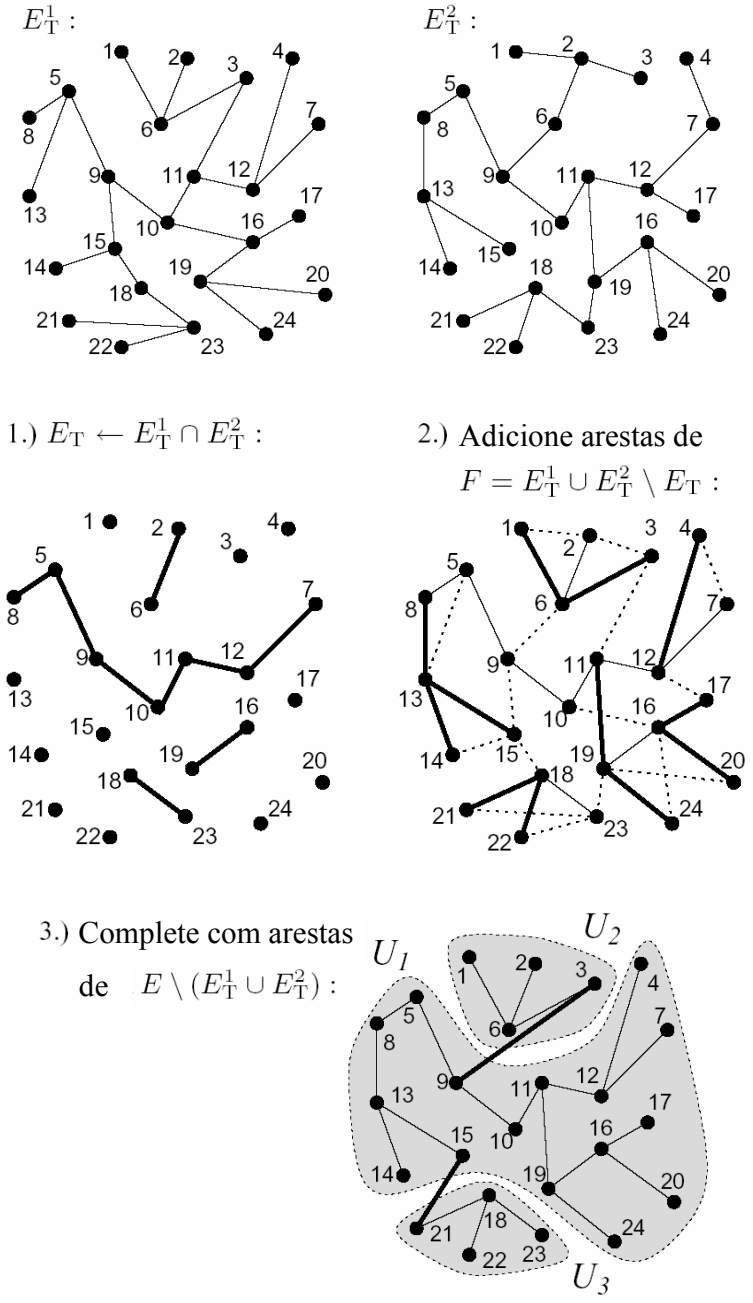


Figura 18: Exemplo de cruzamento

O método de seleção das arestas é por torneio, visando a escolha da aresta de menor peso, sendo somente inseridas as arestas que não violam a condição de grau nem formam ciclo em E_T. Se uma árvore geradora é formada (com |V|-1 arestas), o procedimento termina. Porém, devido à restrição de grau, esse é um caso difícil de ocorrer, formando uma árvore geradora parcial. Ela deve ser completada incluindo também arestas que não estão presentes nos pais, $E \setminus (E_T^1 \cup E_T^2)$. Com o intuito de completar a árvore geradora parcial, todos os componentes não-conectados são determinados em um primeiro passo, ou seja, o

conjunto V é particionado em conjuntos separados U_k contendo somente vértices conectados uns aos outros. Então, estes componentes são conectados à árvore geradora final escolhendo repetidamente dois vértices aleatórios com grau menor que d, em dois componentes não-conectados e incluindo uma aresta entre eles em E_T.

Após o processo de inicialização do AESSEA, usando o RPM, apenas a codificação direta é utilizada. Os operadores de mutação e cruzamento são adaptados para trabalhar com vetor de pesos das arestas.

No operador de cruzamento adaptado, o descendente herda o vetor de escalarização $\boldsymbol{\lambda}$ do pai a. Então os pesos escalarizados $b_{i,j}$ são usados na seleção por torneio das arestas usadas na operação de união do cruzamento descrito acima.

Para adaptar o operador de mutação é um pouco mais complexo. Na mutação definida por Raidl [73], as arestas são ordenadas pelos pesos (uma vez, no início do algoritmo) e o operador de mutação usa esta lista ordenada para direcionar a seleção da aresta a ser inserida, com tendência a arestas de menor peso. No problema multicritério não é possível ordenar as arestas por peso. Contudo, se o vetor de pesos for substituído por um escalar $c_{i,j}$, produzido pelo produto interno do vetor de pesos com um vetor de escalarização μ, com $\sum_{k=1}^{K} \mu_k = 1$ e $\forall k \in 1..K$, $\mu_k \geq 0 : \forall (i, j) \in E$, $c_{i,j} = \mu.w_{i,j}$. Então as arestas podem ser ordenadas em ordem crescente de c. Poderia usar o vetor de escalarização λ associado com cada vetor-solução, fazendo $\mu = \lambda$, para gerar todos os $c_{i,j}$, mas não é desejável reordenar as arestas a cada λ diferente, comprometendo o desempenho computacional. Para contornar este problema, faz-se um pré-ordenamento das arestas para um pequeno conjunto de vetores μ igualmente distribuídos no início do algoritmo. Por exemplo, cinco vetores para μ seriam: (0, 1), (0.25, 0.75), (0.5, 0.5), (0.75, 0.25), (1, 0).

O pseudocódigo do AESSEA é mostrado abaixo:

Algoritmo: AESSEA

P $\Rightarrow$ população
N $\Rightarrow$ arquivo de árvores geradoras não-dominadas
a, b, c e x $\Rightarrow$ vetores-solução

```
N ← ∅
para cada (x ∈ P)
    inicie(x)
    avalie(x)
    arquive(x)
i ← 0
enquanto (i < num_avaliações)
    se rand() < p_c
        a ← rand_mem(P)
        b ← rand_mem(P)
        c ← cruzamento(a, b)
    senão
        a ← rand_mem(P)
        c ← a
    c ← mutação(c)
    avalie(c)
    se (c ≺ a)
        arquive(c)
        P ← P ∪ c \ a
    senão se ((∃j ∈ N | j ≺ a) ∧ (∃k ∈ N | k ≺ c))
        P ← P ∪ c \ a
    senão se (¬∃j ∈ N | j ≺ c)
        se ((g_pop(c) < g_pop(a)) ∨ (∃j ∈ N | c ≺ j))
          arquive(c)
          P ← P ∪ c \ a
    i ← i + 1
retorne (N)
```

A função *rand*() retorna um número dentro do intervalo [0, 1[com probabilidade uniforme, e a função *rand_mem*(*P*) retorna, com probabilidade uniforme, um membro da população atual, *P*. A função *arquive*(*c*) atualiza o arquivo de soluções não dominadas *N* com *c*, isto é, *c* é adicionada a *N* caso seja uma solução não-dominada em relação às outras soluções contidas em *N*, e se *c* domina quaisquer membros de *N* os membros dominados são removidos. O arquivo *N* tem uma capacidade finita *tam_arq* e no caso de adicionar *c* a *N* causa $|N| > tam\_arq$, uma solução na região mais cheia em *N* é removida. Esta estratégia de arquivamento é exatamente a mesma do algoritmo PAES [56]. A função *g_pop*(*x*)

retorna o número de soluções na mesma região no espaço de objetivos da solução *x*. As funções *inicie*(*x*), *avalie*(*x*), *cruzamento*(*a*, *b*) e *mutação*(*c*) são dependentes da codificação usada no algoritmo, nesse caso, a codificação direta (com RPM na inicialização).

3.7.4. Avaliação de Algoritmos para Problemas Multicritério

Em otimização com um único objetivo, a qualidade de uma solução aproximada é avaliada de uma maneira simples, fazendo-se a diferença relativa entre os valores da solução heurística e da solução ótima. No entanto, em otimização multicritério, a avaliação de aproximações do conjunto Pareto-ótimo não é tão trivial. Não existe uma medida simples e natural que seja capaz de capturar informação sobre a qualidade de um conjunto aproximado em relação ao conjunto Pareto-ótimo (conjunto de referência) [2].

Devido a esta dificuldade em avaliar os resultados, tem sido comum indicar o desempenho dessas heurísticas simplesmente plotando as soluções não-dominadas [88] e analisando os gráficos. Contudo, esta forma não é considerada aceitável, pois ela não diz nada sobre a robustez do algoritmo para múltiplas execuções independentes, e além do mais, ela não relaciona a qualidade de solução ao tempo computacional, ou a convergência do algoritmo.

Várias técnicas desenvolvidas recentemente procuram prover resultados os quais podem resumir e fazer inferências a partir dos dados coletados de várias execuções independentes de um algoritmo, e que podem ser usadas para medir a taxa de convergência de um algoritmo multicritério. Zitzler *et al.* [95] sugeriram três objetivos da busca multicritério de Pareto que podem ser identificados e medidos:

- A distância do conjunto não-dominado resultante para o conjunto Pareto-ótimo deveria ser minimizada.
- Uma boa (na maioria dos casos, uniforme) distribuição das soluções encontradas é desejável. A avaliação deste critério poderia ser baseada em uma métrica de distância.
- O tamanho da fronteira não-dominada obtida deveria ser maximizado, isto é, para cada objetivo, uma larga escala de valores deveria ser coberta pelas soluções não-dominadas.

Embora a lista acima sirva como uma diretriz para os objetivos da busca de Pareto, ela não é completamente geral. Por exemplo, se a fronteira de Pareto real for constituída de apenas um ponto, então o terceiro item não é apropriado. Se, por outro lado, os pontos da fronteira de Pareto real não são uniformemente distribuídos, então uma aproximação que contém quase todos os pontos dessa fronteira não obedecerá ao segundo item. Um objetivo mais geral na otimização de Pareto seria expandir somente o primeiro item, definindo o que é significativo pela distância de um conjunto de soluções não-dominadas para outro.

Uma métrica pode comparar duas aproximações diretamente usando uma medida escalar $R(A, B)$ que descreve o quanto A é melhor que B, e $R(B, A)$ que descreve o quanto B

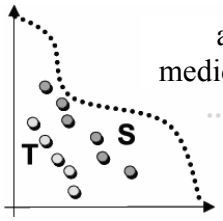
é melhor que A. Se $R(A, B) = c - R(B, A)$ para alguma constante c para todos os pares de conjuntos não-dominados A, B então R é "simétrica". Métricas deste tipo são chamadas de **comparativas diretas**.

Uma abordagem alternativa para comparar duas aproximações é usar um conjunto de referência, talvez o conjunto eficiente, e comparar ambas as aproximações com este conjunto, e então comparar os resultados. A este tipo de métrica dá-se o nome de **métrica de referência**.

Uma terceira alternativa é medir alguma propriedade de cada conjunto que não dependem uma da outra, ou qualquer outro conjunto-referência de pontos, e comparar os resultados destas duas medições. Estas métricas são chamadas de **independentes**.

Freqüentemente, diferentes medidas de qualidade unárias são combinadas e, como resultado, dois conjuntos de aproximações A e B são analisados, comparando a medida de qualidade correspondente, conforme ilustra a figura 19.

*Fonte: Zitzler, E.; Laumanns, M; Bleuler, S. (2002). A Tutorial on Evolutionary Multiobjecive Optimization.



aplicação de medidas de qualidade

	S	T
hipervolume	432.34	420.13
distância	0.3308	0.4532
diversidade	0.3637	0.3463
espalhamento	0.3622	0.3601
cardinalidade	6	5

Figura 19: Transformação das aproximações em valores reais

A seguir serão descritos algumas técnicas de avaliação para problemas multicritério citadas no trabalho de Arroyo [2]:

As duas primeiras técnicas são baseadas nas Medidas de Cardinalidade. A primeira é a porcentagem de pontos do conjunto de referência R (Pareto-ótimo) encontradas nos conjunto de soluções aproximadas A:

$$C_1 = \frac{|A \cap R|}{|A|} \times 100\%$$

Uma primeira dificuldade é a obtenção do conjunto de referência, pois na maioria dos problemas reais, pode ser impraticável o mapeamento de todas as soluções deste conjunto. Neste caso, a medida C_1 não é apropriada para medir a qualidade de um conjunto aproximado A.

Outra técnica envolvendo cardinalidade é a porcentagem de pontos aproximados não-dominados por pontos de referência:

$$C_2 = \frac{\left|\{z \in A : \neg \exists\, z' \in R \,/\, z' \succ z\}\right|}{|A|} \times 100\%$$

Esta medida também não é apropriada para medir a qualidade de um conjunto aproximado A, pois ela não considera a distribuição dos pontos aproximados e a distância em relação aos pontos de referência.

O problema da distância foi abordado no método baseado nas Medidas das Distâncias de Czyzak e Jaszkiewicz [20], onde são definidos os parâmetros de avaliação D_{med} (médias das distâncias de um ponto $z \in R$ ao ponto mais próximo da aproximação A) e D_{max} (máximo das distâncias mínimas de um ponto $z \in R$ a algum ponto em A). Essa técnica visa evitar as desvantagens das medidas de cardinalidade, medindo a proximidade de um conjunto aproximado A em relação a um conjunto de referência R.

Daniels [22] apresenta uma técnica, chamada de Avaliação Analítica, para determinar a qualidade da aproximação de um conjunto de pontos aproximados em relação ao conjunto Pareto-ótimo. Esta técnica é baseada em um algoritmo polinomial, explorando o erro relativo de aproximação através da variação de pesos da função polinomial. O mapeamento das soluções aproximadas pela varredura dos valores dos pesos da função polinomial gera um conjunto de pontos eficientes e aproximados, que são comparados através do erro de aproximação relativo. O critério de avaliação consiste na compilação do erro médio E_{med} e do erro máximo E_{max}.

As funções de utilidade $R1$, $R2$ e $R3$ de Hansen e Jaszkiewicz [41] são técnicas que correspondem a funções mais genéricas e podem exprimir a preferência do tomador de decisões em função de uma aproximação do ponto ideal identificado por uma heurística, pois as medidas de distância não expressam a preferência do tomador de decisão.

Van Veldhuizen e Lamont [88] propõem uma função métrica para calcular a proximidade de um conjunto de pontos aproximados A ao conjunto de referência R, entretanto, a distância envolve apenas algumas regiões de R. Em complemento, é proposta uma outra métrica que mede a distribuição dos pontos no conjunto A [88].

A subseção 3.7.4.1 descreve a métrica $\boldsymbol{S}$, proposta por Zitzler [96], recomendada por Knowles, a partir de uma análise feita com 14 métricas [58]. Outras métricas também recomendadas por Knowles são as funções de utilidade de Hansen e Jaszkiewicz [41], $R1$, $R2$ e $R3$. A seção 3.7.4.1 descreve os indicadores de qualidade I_{ε} e $I_{\varepsilon+}$, definidos por Zitzler *et al.* [97], que são indicadores binários usados para definir se um conjunto não-dominado A é melhor ou não que outro conjunto B.

3.7.4.1. Métrica *S*

Considere um conjunto aproximado A. A métrica $\boldsymbol{S}$ calcula o hiper-volume (no caso de dois objetivos, a hiper-área) de uma região multidimensional compreendida entre A e um ponto de referência Z_{ref}, conforme ilustra a figura 20. Considere agora dois conjuntos aproximados A e B. Para avaliar a qualidade de A com relação a B, computa-se a relação:

$$\frac{S(A) - S(B)}{S(B)}$$

Pode-se tomar como ponto de referência, Z_{ref}, o pior valor de cada objetivo dentre todas as soluções de Pareto encontradas pelas execuções. Se for um problema de minimização, então são os pontos com valores máximos em cada objetivo, e mínimo para maximização.

Para calcular a métrica *S* em problemas bi-objetivo, os pontos são ordenados em ordem decrescente dos valores do primeiro objetivo, e então a seguinte expressão é avaliada:

$$\sum_{i \in 1..|A|} | z_1^i - z_1^{ref} | \cdot | z_2^i - z_2^{i-1} | \quad \therefore \text{ onde } z_2^0 \text{ é inicialmente definido como } z_2^{ref} .$$

O propósito dessa métrica é avaliar a qualidade geral de um conjunto não-dominado. Trata-se de uma métrica independente (embora um único ponto de referência tenha que ser escolhido para limitar a região dominada). Essa métrica também é classificada como não-cardinal[1].

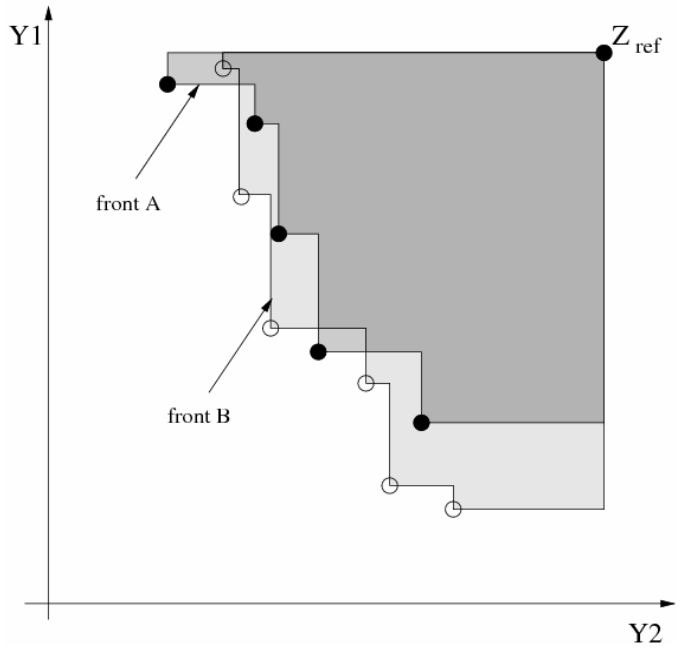


Figura 20: Exemplo da métrica *S* para dois conjuntos aproximados *A* e *B*

Há muitas vantagens nessa métrica, dentre elas, não há necessidade de conhecer o conjunto Pareto-ótimo ou outros pontos de referência para usá-la. Por ouro lado, essa métrica tem elevado processamento computacional, $O(n^{k+1})$, sendo *k* o número de objetivos, o que a torna inviável para o uso com muitos objetivos ou muitos conjuntos não-dominados.

[1] Uma métrica é cardinal se é baseada na contagem do número de vetores (pontos) do conjunto. Caso contrário será não-cardinal.

3.7.4.2. ε-Indicador Binário

Suponha, sem perda de generalidade, um problema de minimização com *n* objetivos positivos, isto é, $Z \subseteq \Re^{+^n}$. Um vetor de objetivos $z^1 = \{ z_1^1, z_2^1, ..., z_n^1 \} \in Z$ ε-domina outro vetor de objetivos $z^2 = \{ z_1^2, z_2^2, ..., z_n^2 \} \in Z$, escrito como $z^1 \succeq_\varepsilon z^2$, se e somente se:

$$\forall 1 \leq i \leq n / \; z_i^1 \leq \varepsilon \cdot z_i^2$$

para um dado $\varepsilon > 0$. Define-se o *ε-indicador binário* I_ε como:

$$I_\varepsilon (A, B) = \inf_{\varepsilon \in \Re} \{ \forall z^2 \in B \;\; \exists z^1 \in A / z^1 \succeq_\varepsilon z^2 \}$$

para quaisquer dois conjuntos aproximados *A*, *B*.

Informalmente falando, um vetor z^1 ε-domina outro vetor z^2, se puder multiplicar cada objetivo em z^2 por um fator ε e o vetor objetivo resultante ainda seja fracamente dominado por z^1. Consequentemente, o ε-indicador fornece o fator pelo qual um conjunto aproximado é pior do que outro com relação a todos os objetivos.

Da mesma maneira, um ε-indicador aditivo $I_{\varepsilon+}$ pode ser definido como:

$$I_{\varepsilon+} (A, B) = \inf_{\varepsilon \in \Re} \{ \forall z^2 \in B \;\; \exists z^1 \in A / z^1 \succeq_{\varepsilon+} z^2 \}$$

em que $z^1 \succeq_{\varepsilon+} z^2$ se e somente se:

$$\forall 1 \leq i \leq n / \; z_i^1 \leq \varepsilon + z_i^2$$

A tabela 1 mostra o significado do resultado da aplicação dos indicadores I_ε e $I_{\varepsilon+}$ em dois conjuntos aproximados *A* e *B*; e a tabela 2 mostra o significado da simbologia usada nas relações de dominância indicadas na tabela 1.

	Compatível com respeito à relação:					
	$\succ\succ$	$\succ$	$\triangleright$	$\succeq$	=	$\|\|$
I_ε	$I_\varepsilon(A, B) < 1$	-	$I_\varepsilon(A, B) \leq 1$ $I_\varepsilon(B, A) > 1$	$I_\varepsilon(A, B) \leq 1$	$I_\varepsilon(A, B) = 1$ $I_\varepsilon(B, A) = 1$	$I_\varepsilon(A, B) > 1$ $I_\varepsilon(B, A) > 1$
$I_{\varepsilon+}$	$I_{\varepsilon+}(A, B) < 0$	-	$I_{\varepsilon+}(A, B) \leq 0$ $I_{\varepsilon+}(B, A) > 0$	$I_{\varepsilon+}(A, B) \leq 0$	$I_{\varepsilon+}(A, B) = 0$ $I_{\varepsilon+}(B, A) = 0$	$I_{\varepsilon+}(A, B) > 0$ $I_{\varepsilon+}(B, A) > 0$

Tabela 1: Análise do resultado fornecido pelos indicadores binários

Relação	Simbologia
Domina fortemente	$A \succ\succ B$
Domina	$A \succ B$
Melhor	$A \triangleright B$
Domina fracamente	$A \succeq B$
Incomparável	$A \,\|\|\, B$

Tabela 2: Descrição da simbologia usada nas relações de dominância

Capítulo 4

Otimização por Nuvem de Partículas (PSO)

A Otimização por Nuvem de Partículas (PSO, do inglês, *Particle Swarm Optimization*) é uma técnica baseada em população, desenvolvida por James Kennedy (Psicólogo) e Russell Eberhart (Engenheiro Eletricista) [51], com base no comportamento de pássaros em revoadas, modelado pelo biólogo Frank Heppner [42].

PSO possui muitas semelhanças com os algoritmos evolucionários, tais como os Algoritmos Genéticos (AG). O sistema é iniciado com uma população de soluções aleatórias e procura por soluções ótimas atualizando essa população. Contudo, diferente do AG, PSO não possui operadores de evolução, tais como cruzamento e mutação. Na PSO, as soluções potenciais, chamadas de partículas, "voam" sobre o espaço de soluções seguindo as partículas ótimas atuais, ou seja, as melhores partículas [25].

Comparado ao AG, as vantagens da PSO são: fácil de implementar e há poucos parâmetros a serem ajustados. A PSO tem sido aplicada com sucesso em muitas áreas: otimização de funções, treinamento em redes neurais artificiais, controle de sistemas fuzzy, além de outras áreas as quais os algoritmos genéticos podem ser aplicados [45].

4.1. Simulando o Comportamento Social

Vários cientistas têm criado simulações computacionais de várias interpretações do movimento de organismos em bando de pássaros, em cardume de peixes ou em enxame de abelhas. Reynolds [76] e Heppner [42] apresentaram modelos de simulações de bando de pássaros. Reynolds estava curioso por causa da estética da coreografia do vôo dos pássaros, e Heppner estava interessado em descobrir a base das regras que permitiam um grande número de pássaros juntarem-se simultaneamente, sempre mudando a direção rapidamente, separando e reagrupando. Estes cientistas tinham a percepção que processos locais, tais como os modelados por autômatos celulares, poderiam fundamentar a dinâmica de grupo imprevisível do comportamento social dos pássaros. Ambos os modelos se concentravam na manipulação das distâncias inter-individuais, ou seja, a sincronia de comportamento do bando foi pensada como uma função de esforços dos pássaros em manter uma distância ótima entre eles e seus vizinhos.

Não parece ser um salto muito grande de lógica supor que algumas mesmas regras fundamentam o comportamento social animal (incluindo rebanhos, cardumes e bandos) e dos humanos. Wilson [92] escreveu, com relação ao cardume de peixes, "Pelo menos em teoria, membros do cardume podem tirar proveito de descobertas e experiências anteriores dos outros membros durante a procura por comida. Esta vantagem pode se tornar decisiva,

valendo mais que as desvantagens de competição por comida, sempre que o recurso esteja distribuído imprevisivelmente em pedaços" (p. 209). Esta declaração sugere que o compartilhamento social de informação entre indivíduos oferece uma vantagem evolucionária: esta hipótese foi fundamental ao desenvolvimento da otimização por nuvem de partículas [51].

4.2. Contexto: Vida Artificial

O termo "Vida Artificial" (*ALife*, como é conhecido em inglês) é usado para descrever pesquisas em sistemas feitos pelo homem que possuem algumas das propriedades essenciais da vida real (http://www.alife.org). *ALife* engloba dois tópicos de pesquisa:

- Como as técnicas computacionais podem ajudar quando se estudam os fenômenos biológicos.
- Como as técnicas biológicas podem ajudar em problemas computacionais.

Como se pode notar, a PSO se inclui no segundo tópico. Há inúmeras técnicas computacionais inspiradas em sistemas biológicos, por exemplo, redes neurais artificiais é um modelo simplificado do cérebro humano; algoritmo genético é inspirado na teoria da evolução; colônia de formigas etc. O sistema biológico ao qual a PSO está incluída é o *sistema social*, mais especificamente, o comportamento coletivo de indivíduos interagindo com seu ambiente e entre si (do inglês, *swarm intelligence*).

O conceito de nuvem de partículas originou-se a partir da simulação de um sistema social simplificado. A idéia original foi simular graficamente a "coreografia" de um bando de pássaro ou cardume de peixes. Contudo, descobriu-se que esse modelo poderia ser usado na otimização de problemas.

4.3. Algoritmo de Otimização por Nuvem de Partículas

Suponha o seguinte cenário: um grupo de pássaros está aleatoriamente procurando por comida em uma região qualquer, sendo que há somente um pedaço de comida nesta região e nenhum pássaro sabe onde a comida está. Mas eles sabem o quão distante a comida está em cada iteração. Qual será a melhor estratégia para encontrar a comida? Uma boa escolha é seguir o pássaro que está mais próximo da comida.

A PSO tomou como base este cenário e usou-o para resolver os problemas de otimização. Na PSO, cada solução potencial é um pássaro no espaço de soluções, o qual é chamado de partícula. Todas as partículas possuem valores de adequação os quais são avaliados pela função-objetivo a ser otimizada, e têm velocidades as quais direcionam o

vôo das partículas. As partículas voam sobre o espaço de soluções seguindo as partículas ótimas atuais.

Cada partícula tem as seguintes características:

- Possui uma posição e uma velocidade.
- Tem conhecimento da sua posição, e também do valor da função-objetivo para esta posição.
- Tem conhecimento dos seus vizinhos, assim como da melhor posição e do melhor valor da função-objetivo dentre eles.
- Guarda sua melhor posição já atingida.

PSO é iniciado com um grupo de partículas aleatórias (que são as soluções potenciais) e então procura por soluções ótimas atualizando as partículas a cada iteração. Em cada iteração, cada partícula é atualizada seguindo dois valores. O primeiro é a melhor solução (adequação) encontrada pela partícula ao longo da busca, chamado de **pbest**. O outro valor é a melhor solução encontrada por todas as partículas, chamado de **gbest**. A cada passo, o comportamento de uma determinada partícula depende de três possíveis escolhas:

- Seguir seu próprio caminho.
- Seguir em direção à sua melhor posição já encontrada (**pbest**).
- Seguir em direção à melhor posição do melhor vizinho (**gbest**).

Após encontrar estes dois valores "ótimos", a partícula atualiza sua velocidade e posição com as seguintes equações:

$$v_{i+1} = w.v_i + c_1.rand().(pbest_i - p_i) + c_2.rand().(gbest - p_i) \quad (1)$$

$$p_{i+1} = p_i + v_{i+1} \quad (2)$$

em que v é a velocidade da partícula, p se refere à partícula atual (solução), rand() corresponde a um número aleatório entre (0,1), w é chamado de fator de inércia e c_1, c_2 são fatores de aprendizagem, usualmente $c_1 = c_2 = 2$. Os três coeficientes (que representam papéis social/cognitivo) w, c_1 e c_2 respectivamente significam:

- O quanto a partícula confia em si mesma.
- O quanto ela confia na sua experiência.
- O quanto ela confia nos seus vizinhos.

O pseudocódigo para PSO é mostrado abaixo:

```
Para cada partícula p
  Inicie p
Fim_Para

Faça
  Para cada partícula p faça
    Calcule o valor da função de adequação de p
    Se a adequação de p é melhor que a adequação de pbest então
      pbest ← p
  Fim_Para

  Defina a partícula de melhor adequação de todas como gbest
  Para cada partícula p faça
    Calcule a velocidade de p, de acordo com a equação (1)
    Atualize a posição de p, de acordo com a equação (2)
  Fim_Para
Enquanto (critério de parada) não é satisfeito
```

As velocidades das partículas em cada dimensão são limitadas por uma máxima velocidade, V_{max}.

Há muitas maneiras de se definir uma vizinhança [52], mas duas classes podem ser distinguidas:

- Vizinhança "física" (ou geográfica), que leva em conta as distâncias. Na prática, as distâncias são computadas a cada passo e tomadas as *k* arestas mais próximas como vizinhas.
- Vizinhança "social", que leva em conta os relacionamentos. Na prática, para cada partícula, sua vizinhança é definida com uma lista de partículas. Portanto, não é necessário calcular distâncias, sendo uma grande vantagem para alguns casos, particularmente para espaços discretos. Pode notar também que, se o processo converge, uma vizinhança social tende a se tornar uma vizinhança física.

4.4. Controle de Parâmetros

Há dois passos importantes quando se aplica PSO em problemas de otimização: a representação da solução e a função de cálculo da adequação. Não há muitos parâmetros a serem configurados. Abaixo segue uma lista dos parâmetros, assim como valores típicos para eles:

- *Número de partículas*: o intervalo típico é 20-40. Para a maioria dos problemas, 10 partículas são suficientes para alcançar bons resultados. Para alguns problemas mais difíceis, pode-se tentar 100-200 partículas.

- *Dimensão das partículas*: determinada pelo problema a ser otimizado.
- V_{max}: determine a máxima mudança que uma partícula pode tomar durante uma iteração.
- *Fatores de aprendizagem*: c_1 e c_2 usualmente são iguais a 2. Contudo, podem variar no intervalo [0,4].
- *Condição de parada*: o número máximo de iterações que o algoritmo executa e o requisito de erro mínimo. Estas condições de parada dependem do problema.

4.5. Aplicações

A PSO pode ser aplicada aos problemas de otimização, desde que se defina como irão funcionar os parâmetros básicos, por exemplo, a interação entre a posição e a velocidade.

A PSO é usada em redes neurais artificiais, em lugar do método *back-propagation*, para treinar a rede com a mesma eficiência em menor tempo [51]. Em virtude disso, algumas aplicações reais nas áreas de diagnóstico médico, indústria, etc. usam uma rede neural guiada pelo algoritmo PSO.

Segue abaixo algumas aplicações práticas relatadas na literatura que usam a otimização por nuvem de partículas:

- Missões de inspeção/vigilância com base no problema do caixeiro viajante (Instituto de Tecnologia da Força Aérea Americana) [83].
- Otimização de funções de controle nebulosas (lógica *Fuzzy*) [27].
- Identificação de partículas por dispersão de luz [70].
- Programação de sistemas de manufatura [46].
- Controle reativo de tensão elétrica e potência [93].

4.6. PSO em Problemas Discretos

Abaixo, tem-se uma lista de requisitos necessários à PSO:

- Um espaço de soluções (partículas) $S = \{s_i\}$.
- Uma função-objetivo *f* em S, dentro de um conjunto de valores $S \xrightarrow{f} C = \{c_i\}$, cujos valores mínimos estão nas partículas-solução.
- Caso se queira uma vizinhança física, será necessário calcular uma distância *d* no espaço de soluções.

Geralmente, S é um espaço real $\Re^n$, e *f* uma função numérica contínua. Mas, nada nas equações (1) e (2) da subseção 4.3 dizem que deve ser dessa forma. Em particular, S pode ser um conjunto finito de partículas e *f* uma função discreta, e tão logo se possam definir as operações e objetos matemáticos básicos mostrados a seguir, pode-se usar a PSO:

- Posição e velocidade de uma partícula.
- Subtração (*posição*, *posição*) $\longrightarrow$ *velocidade*.
- Multiplicação externa (*número real*, *velocidade*) $\longrightarrow$ *velocidade*.
- Adição (*velocidade*, *velocidade*) $\longrightarrow$ *velocidade*.
- Mover (*posição*, *velocidade*) $\longrightarrow$ *posição*.

4.7. Algoritmos aplicados ao caixeiro viajante usando PSO

Alguns algoritmos aplicados ao PCV usando a otimização por nuvem de partículas foram implementados anteriormente ao algoritmo proposto. Serão descritas abaixo quatro implementações: o algoritmo de Clerc [15] e o algoritmo de Wang *et al*. [90], que são bem semelhantes; o algoritmo de Machado e Lopes [62] e o algoritmo de Pang *et al*. [68], que mostram abordagens híbridas.

O **algoritmo básico de Clerc** [15] define alguns elementos do PSO para que possa ser aplicado ao PCV. Estes elementos são descritos a seguir.

Posição

Considere $G = (V, E)$ o grafo a ser avaliado. V é o conjunto de vértices (nós) e E o conjunto de arestas valoradas (arcos). Os nós do grafo são numerados de 1 até N (número de nós). Cada elemento de E consiste em uma tripla $(i, j, w_{i,j})$ com $i, j \in \{1, ..., N\}$ e $w_{i,j} \in \Re$. Como se está procurando por ciclos, consideram-se seqüências com N+1 nós, todos diferentes, exceto o último nó que é igual ao primeiro. Esta seqüência é vista como uma partícula no espaço de soluções. Portanto, o espaço de soluções é definido como o conjunto finito de todas as partículas.

Função-objetivo

Considere uma partícula $x = (n_1, n_2, ..., n_N, n_{N+1})$, $n_1 = n_{N+1}$. Esta partícula somente é aceita se existirem todos os arcos (n_i, n_{i+1}). Para cada partícula, uma possível função-objetivo pode ser definida como (sendo N o número de cidades):

$$f(x) = \sum_{i=1}^{N} w_{n_i, n_{i+1}}$$

Este função-objetivo tem um número finito de valores e seu mínimo global consiste na melhor solução.

Velocidade

O objetivo é definir o operador v o qual, quando aplicado à partícula, retorna outra posição. Define-se então uma permutação de N elementos, onde há uma lista de transposições. O comprimento dessa lista é $||v||$. A velocidade é então definida por uma lista de pares de vértices, que indicam uma seqüência de operações 2-troca a ser realizada em uma partícula p, para realizar a movimentação da partícula p:

$$v = \mathbf{(}(i_k, j_k)\mathbf{)} \therefore i_k, j_k \in \{1, .., N\}, k\uparrow_1^{||v||}$$

O que significa "troque os números (i_1, j_1), depois os números (i_2, j_2), etc. e, por fim, os números $(i_{||v||}, j_{||v||})$".

Mover(posição, velocidade)

Considere x uma partícula e v uma velocidade. A partícula $x' = x + v$ é definida aplicando a primeira transposição de v em x, a segunda ao resultado etc.

Exemplo: $x = (1, 2, 3, 4, 5, 1) \therefore v = \mathbf{(}(1, 2), (2, 3)\mathbf{)}$

Aplicando v em x, tem-se: (**2**, **1**, 3, 4, 5, **2**) $\Rightarrow$ (**3**, 1, **2**, 4, 5, **3**)

Subtração(posição, posição)

Considere x_1 e x_2 duas partículas. A diferença $x_2 - x_1$ é definida como a velocidade v, encontrada por um algoritmo, de forma que aplicando v em x_1 resulta em x_2. Em particular, o algoritmo é escolhido tal que se tenha:

$$x_2 - x_1 = \neg(x_1 - x_2) \quad \text{e} \quad x_1 = x_2 \Rightarrow v = x_2 - x_1 = \varnothing$$

Adição(velocidade, velocidade)

Considere v_1 e v_2 duas velocidades. A fim de computar $v_1 \oplus v_2$ considera-se uma lista de transposições que contém primeiro as transposições de v_1, seguidas pelas de v_2. Em particular, esta operação é definida tal que $v \oplus \neg v = \varnothing$. Em relação ao tamanho, tem-se que $||v_1 \oplus v_2|| \leq ||v_1|| + ||v_2||$.

Multiplicação(número real, velocidade)

Considere c um coeficiente real e v uma velocidade. Há diferentes casos, dependendo do valor de c.

- Caso em que $c = 0 \Rightarrow c.v = \varnothing$.
- Caso em que $c \in]0,1] \Rightarrow$ trunca-se v. Considere $||c.v||$ como o maior inteiro menor ou igual do que c.$||v||$. Então, define-se $c.v = ((i_k, j_k)), k\uparrow_1^{||cv||}$.
- Caso em que $c > 1 \Rightarrow$ significa ter $c = k + c'$, $k \in \mathrm{N}^*$, $c' \in [0,1[$. Então, define-se $c.v = \underbrace{v \oplus v \oplus \ldots v}_{k \text{ vezes}} \oplus c'.v$.

- Caso em que $c < 0 \Rightarrow$ escrevendo $c.v = (-c).\neg v$, basta considerar o caso anterior.

Distância entre duas partículas

Considere x_1 e x_2 duas partículas. A distância é definida por $d(x_1, x_2) = ||x_2 - x_1||$. Para essa distância, tem-se as seguintes propriedades (sendo x_3 uma terceira partícula):

$||x_2 - x_1|| = ||x_1 - x_2||$

$||x_2 - x_1|| = 0 \Leftrightarrow x_1 = x_2$

$||x_2 - x_1|| \leq ||x_2 - x_3|| + ||x_3 - x_1||$

Definindo-se esses elementos, podem-se reescrever as equações (1) e (2) da seção 4.3 como se segue (neste algoritmo, considera-se $c_1 = c_2$.):

$v[] = w.v[] \oplus c_1.rand().(pbest[] - p[]) \oplus c_2.rand().(gbest[] - p[])$

$p[] = p[] + v[]$

O **algoritmo de Wang *et al.*** [90] apresenta a mesma estrutura do algoritmo de Clerc [15], sendo que no algoritmo de Wang *et al.* a partícula é representada por uma seqüência de N nós distintos, ao invés de N+1 nós: $x = (n_1, n_2, ..., n_N)$, de tal forma que o custo da função-objetivo é calculado por (sendo N o número de cidades):

$$f(x) = \sum_{i=1}^{N-1} w_{n_i, n_{i+1}} + w_{N,1}$$

Outra diferença é na definição da velocidade. No algoritmo de Clerc a troca ocorre entre os números definidos no vetor, enquanto no algoritmo de Wang *et al.* a troca é feita entre as posições. Tome o mesmo exemplo usado para o algoritmo de Clerc: x = (1, 2, 3, 4, 5, 1) $\therefore$ v = **(**(1, 2), (2, 3)**)**. No algoritmo de Clerc o resultado é:

$x' = x + v =$ (**2**, **1**, 3, 4, 5, **2**) $\Rightarrow$ (**3**, 1, **2**, 4, 5, **3**)

Para o algoritmo de Wang *et al.* fica da seguinte maneira, lembrando que x = (1, 2, 3, 4, 5):

$x' = x + v =$ (**2**, **1**, 3, 4, 5) $\Rightarrow$ (2, **3**, **1**, 4, 5)

O **algoritmo de Machado e Lopes** [62] consiste em um modelo híbrido baseado no PSO usando conceitos de algoritmos genéticos (AG) [40] e busca local rápida (FLS) [89]. Do PSO, este modelo usa o conceito de máximo local, máximo global e movimento das partículas. Do AG, usa-se a representação da solução como um vetor numérico e o cruzamento para mover as partículas pelo espaço discreto. A FLS é usada para melhorar soluções encontradas durante a busca, avaliando próximos a cada partícula.

Cada partícula representa uma possível solução para o PCV, sendo composta pelos seguintes componentes: posição atual, melhor solução local, velocidade atual, adequação atual da partícula e adequação da melhor solução local. A adequação é calculada dividindo a *mínima distância pra cálculo da adequação* (D_{min}) pelo custo do *tour* representado pela

solução atual. D_{min} é configurado como sendo o valor ótimo do *tour*, e se esse valor ótimo não é conhecido, D_{min} pode ser configurado igual a 1.

Para cada partícula i, V_i é iniciado aleatoriamente e a posição atual é iniciada com um *tour* válido aleatório. Para a primeira iteração, a melhor solução local recebe o valor da posição atual. Após estes passos, a adequação de cada partícula é calculada. A velocidade (que usa a distância da posição atual para a melhor solução local e para a melhor solução global) é calculada baseada na distância de Hamming: dadas duas partículas A e B representando *tours*, a distância entre elas é calculada comparando-se os dois vetores, partindo de uma cidade comum aos dois. Inicialmente, esta distância é nula e, para cada posição comparada dos vetores, sempre que elas são diferentes a distância é incrementada em uma unidade.

Em todas as iterações, a distância média entre as partículas e a melhor solução global é calculada. Se este valor é 5% menor do que o número de cidades, uma aglutinação é caracterizada e as partículas são espalhadas. No caso de aglutinação ao redor da melhor solução local, todas as partículas próximas terão seu valor alterado para um valor aleatório. O movimento das partículas é baseado nas equações básicas do PSO e o no operador OX que recombina duas possíveis soluções. O refinamento (melhoria) das partículas é feito usando a estratégia FLS (busca local).

O **algoritmo de Pang *et al.*** [68] usa o espaço contínuo para busca de soluções. Ele constrói um mapeamento deste espaço contínuo para o espaço discreto de permutações do PCV. Um número real positivo P_MAX é usado para definir a abrangência para cada partícula no espaço contínuo. Se X_i denota a posição de alguma partícula i, então $X_i = (x_{i1}, x_{i2}, ..., x_{id})$ onde $x_{id} \in$ [-P_MAX, +P_MAX]. Um número real positivo V_MAX é usado como limitante para a velocidade máxima, V_MAX = a*P_MAX, $a \in (0, 1)$. Se V_i é a velocidade para alguma partícula i, então $V_i = (v_{i1}, v_{i2}, ..., v_{id})$ onde $v_{id} \in$ [-V_MAX, +V_MAX].

O mapeamento do espaço contínuo (Ω^n) para o espaço discreto (Φ^n), de tamanho n, deve satisfazer as seguintes condições: (i) para qualquer vetor X em Ω^n, deve haver uma, e somente uma, permutação π correspondente a X, $f(X) = \pi$; e (ii) o mapeamento f deve utilizar a relação entre os componentes de X para indicar a relação de ordem de π. Se $i > j$, no mapeamento f, $X\pi_i$ está antes de $X\pi_j$. O método chamado *Great Value Priority* (GVP) é usado para fazer este mapeamento.

Para alguma posição no espaço contínuo, o vetor correspondente $X = (x_1, x_2, ..., x_n)$, onde $x_i \in$ [-P_MAX, +P_MAX], $i = \{1, 2\ldots, n\}$, de acordo com o GVP, no espaço discreto a única permutação $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ que corresponde a X deve ter as seguintes características:

- Para cada x_i, se $\left[k = 1 + \sum_{j=1}^{n} if\,(x_j > x_i)\ 1\ \ senão\ \ 0 \right]$ então $\pi_k = i$.

Por exemplo, suponha X = (0.2, 0.3, 0.1, 0.8), após uma operação de ordenação descendente, tem-se (0.8, 0.3, 0.2, 0.1), resultando na permutação π = (4, 2, 1, 3). A adequação de cada partícula é obtida calculando o comprimento do *tour* da solução no espaço discreto, sendo este o principal objetivo do mapeamento de espaços.

São aplicadas técnicas de busca local (2-opt) às partículas no espaço discreto e então é feito o mapeamento reverso para o espaço contínuo. Para isso, é proposto um método chamado *Loseless Swap Restore* (LSR). Suponha $X_p = (x_{p1}, x_{p2}, \ldots, x_{pn})$ como sendo o vetor de posição de uma partícula P, e a permutação correspondente é $\pi_p = (\pi_{p1}, \pi_{p2}, \ldots, \pi_{pn})$. Após pesquisar a vizinhança $\rho(\pi_p)$ de π_p, é encontrada uma solução melhor π_p'. Considere σ como sendo uma permutação, e $\pi_p' = \sigma\pi_p$, suponha $\pi_p' = (\pi_{p1}', \pi_{p2}', \ldots, \pi_{pn}')$. No método LSR, considera-se o vetor X_p como uma permutação, a mesma permutação σ age em X_p, então tem-se o vetor restaurado $X_p' = \sigma X_p$.

Por exemplo, X_p = (-0.1, 0.2, -0.3, 0.8), após o método GVP, π_p = (4, 2, 1, 3). Após aplicada a busca local, consegue-se um solução melhor π_p‘ = (2, 4, 3, 1), então a permutação σ = {(2, 4), (1, 3)} agem em X_p, resultando em X_p‘ = (-0.3, 0.8, -0.1, 0.2).

Operações caóticas são introduzidas para prevenir que as partículas cheguem logo a um estado de equilíbrio. A posição e a velocidade são definidas com valores aleatórios:

- $x_{id} = rand()$*P_MAX e $v_{id} = rand()$*V_MAX.

A população é iniciada com posições e velocidades aleatórias de acordo com P_MAX e V_MAX. O algoritmo executa em um número de iterações definidos por MAX_GEN, sendo que, para cada iteração, ocorre o seguinte: (*i*) para cada partícula, execute a operação caótica com probabilidade *DispitiveProb*; (*ii*) calcule a permutação π_i para cada posição X_i e calcule a adequação da partícula P_i; (*iii*) realize a busca local para π_i com probabilidade *TwoOptProb*, e restaure a melhor solução encontrada para o espaço contínuo; (*iv*) atualize a melhor solução local da partícula P_i, caso necessário. Ao final de uma iteração, atualize a melhor solução global, se for o caso, e o número de iterações. Finalmente, exiba a melhor solução.

A seção 6.1 apresenta os resultados computacionais para os algoritmos propostos aplicados ao PCV, onde os algoritmos de Wang *et al.* [90], Pang *et al.* [68] e Machado e Lopes [62] são comparados com esses algoritmos propostos. O que se pode notar, analisando essas comparações (tabelas 2 e 3), é que a abordagem híbrida de Machado e Lopes [62] apresenta melhores resultados do que as abordagens de Wang *et al.* [90] e Pang *et al.* [68].

Capítulo 5

Algoritmos Propostos

Esta seção apresenta os algoritmos propostos neste trabalho:

- Dois algoritmos baseados na Otimização por Nuvem de Partículas (PSO), com procedimentos de busca local e *path-relinking* propostos como operadores de velocidade, aplicados ao Problema do Caixeiro Viajante (PCV).
- Um algoritmo também baseado em PSO aplicado ao Problema da Árvore Geradora Mínima Restrita em Grau Multicritério (AGM-mc*d*), nesse caso, com dois objetivos. Ele também possui a mesma estrutura dos algoritmos aplicados ao PCV, usa procedimentos de busca local e *path-relinking* como operadores de velocidade.

A busca local é um método tradicional de otimização que inicia com uma solução qualquer e procede procurando por uma melhor solução na vizinhança definida para a solução inicial. Se uma solução melhor é encontrada, então ela se torna a nova solução do problema e novamente o processo de busca na vizinhança é executado. Este processo continua até que nenhuma solução melhor seja encontrada [1].

Path-relinking é uma técnica de intensificação cuja idéia foi originalmente proposta por Glover [34] no contexto dos métodos de agendamento para obter melhores regras de decisão para problemas de agendamento de tarefas (*job shop scheduling*) [35]. A estratégia consiste em gerar um caminho entre duas soluções, criando novas soluções. Sendo x_s a solução de origem e x_t a solução de destino, um caminho entre x_s e x_t é uma seqüência x_s, $x_s(1)$, $x_s(2)$, ..., $x_s(r) = x_t$, onde $x_s(i+1)$ é obtida a partir de $x_s(i)$ por meio de movimentos introduzidos em $x_s(i+1)$ que reduzem a distância entre as soluções de origem e destino.

As condições de origem e destino podem ser alternáveis. Algumas estratégias para considerar tais situações são:

- *Forward*: a solução pior entre x_s e x_t é a origem e a outra é a destino.
- *Backward*: a solução melhor entre x_s e x_t é a origem e a outra é a destino.
- *Back and forward*: duas trajetórias distintas são exploradas, a primeira usando a solução melhor entre x_s e x_t como origem e a segunda usando a solução pior como origem.
- *Mixed*: dois caminhos são simultaneamente explorados, o primeiro iniciando com a melhor solução e o segundo iniciando com a pior solução entre x_s e x_t, até que eles encontrem uma solução intermediária eqüidistante de x_s e x_t.

5.1. Algoritmos Aplicados ao PCV

Para os dois algoritmos aplicados ao PCV, a posição da partícula é representada por uma seqüência de N nós distintos, $x = (n_1, n_2, ..., n_N)$, onde N é o número de cidades. O valor da adequação é calculado por:

$$f(x) = \sum_{i=1}^{N-1} w_{n_i, n_{i+1}} + w_{N,1}$$

As partículas são iniciadas com uma versão aleatória adaptada da heurística do vizinho mais próximo [6]. O procedimento é similar à fase construtiva de um algoritmo GRASP [28]. Primeiramente, uma cidade é escolhida de forma aleatória, então outras cidades são adicionadas à solução em cada passo. Uma lista de candidatas é construída com 5% das cidades mais próximas à última inserida na solução. Uma cidade é escolhida aleatoriamente dessa lista e então adicionada à solução. Este passo se repete até que uma solução para o PCV esteja completa.

O primeiro algoritmo, PSO aplicado ao PCV, usa como procedimento de busca local um operador de inversão para construir as soluções vizinhas. Este operador inverte os elementos de uma partícula compreendidos entre dois índices, a e b. A diferença b–a varia entre 1 (simulando um movimento 2-troca) e N–1. Dessa forma, quando v_1 é aplicada à particular p, o procedimento de busca local começa invertendo seqüências de dois elementos, depois de três elementos, e assim por diante. A figura 21 ilustra a inversão de uma seqüência de elementos definida pelo intervalo $a = 2$ e $b = 5$ (uma seqüência com quatro elementos) da partícula p, resultando na partícula p'.

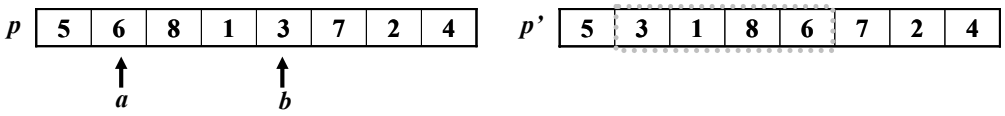


Figura 21: Inversão de elementos entre os índices *a* e *b*

O segundo algoritmo, também aplicado ao PCV, usa como procedimento de busca local a vizinhança de Lin-Kernighan, LK [61]. Ela é reconhecida como um método de melhoria eficiente para o PCV. O algoritmo básico de LK tem um certo número de decisões a serem feitas e dependendo das estratégias adotadas pelas implementações distintas de cada programador, ele pode resultar em diferentes desempenhos. A literatura contém relatos de muitas implementações do LK com características variadas [47]. Neste algoritmo, é usada a implementação do LK do *Concorde TSP Solver* [18].

Um outro operador de velocidade, usado nos dois algoritmos, é considerado quando uma partícula tem que mover de sua posição atual para uma outra posição (a melhor posição já encontrada por ela – *pbest* – ou a melhor posição de todas – *gbest*). Nestes casos, o operador *path-relinking* é usado entre as duas soluções.

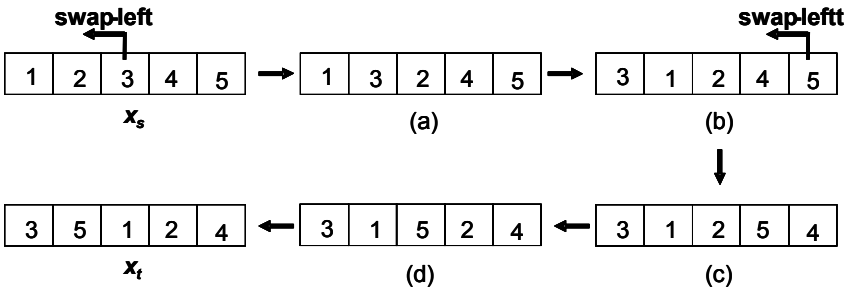


Figura 22: Operador de *path-relinking*

A velocidade utilizada para mover uma partícula a partir da origem até a posição de destino é definida como uma operação de *path-relinking*. O operador de *relinking* usado nestes dois algoritmos é ilustrado na figura 22. O operador troca um elemento com o vizinho da direita (esquerda). São mostrados os passos do procedimento de *path-relinking* que exploram soluções no caminho entre a solução de origem (1, 2, 3, 4, 5) e a solução de destino (3, 5, 1, 2, 4). Primeiro, o elemento 3 é movido para a primeira posição trocando-o com os elementos 2 (figura 22(a)) e 1 (figura 22(b)). Agora, o elemento 5 deve ser movido para a segunda posição. Ele é trocado como o elemento 4 (figura 22(c)), elemento 2 (figura 22(d)) e, finalmente, como o elemento 1, quando a solução destino é então alcançada. Os operadores de troca seguem procedimentos $O(n^2)$. O *path-relinking* é aplicado simultaneamente da solução origem para o destino e vice-versa (*back and forward*).

Combinações de movimentos são possíveis. Por exemplo, para combinar a primeira e terceira opção de movimento, pode-se parar o procedimento de busca local após um dado número de iterações e então aplicar o *path-relinking* entre a solução obtida pela busca local e a melhor solução global. Embora essas combinações pudessem ser feitas, nos dois algoritmos propostos, nenhuma combinação de movimentos foi implementada. Probabilidades iniciais são associadas a cada um dos três movimentos: $pr_1 = 90\%$, $pr_2 = 5\%$ e $pr_3 = 5\%$. Estas probabilidades vão sendo alteradas à medida que o algoritmo avança na execução. Um modelo geral para os dois algoritmos implementados é mostrado a seguir.

```
procedimento PSO_TSP
{
  defina probabilidades iniciais para os movimentos:
    pr1 = x /* seguir o próprio caminho, v1 */
    pr2 = y /* seguir pbest, v2 */
    pr3 = z /* seguir gbest, v3 */
    /* x + y + z = 1 (100%) */

  para i = 1 até #partículas   /* iniciar a população */
    pi = vizinho_mais_próximo_adaptado()
  fim para

  faça
    para i = 1 até #partículas faça
      adequação(pi) = avalie(pi)
      se adequação(pi) é melhor que adequação(pbesti) então
        pbesti ← pi
    fim_Para
    defina a partícula de melhor adequação como gbest
    para i = 1 até #partículas faça
      escolha a velocidade de pi com base em pr1, pr2 e pr3:
        v1: pi = busca local(pi)
        v2: pi = path_relinking(pi, pbesti)
        v3: pi = path_relinking(pi, gbest)
      atualize a posição de pi
    fim_Para
    atualize as probabilidades:
      pr1 = pr1×0.95; pr2 = pr2×1.01; pr3 = 100%-(pr1+pr2)
  enquanto (critério de parada) não é satisfeito
}
```

No início, os algoritmos definem as probabilidades associadas com cada velocidade para as partículas, onde pr_1, pr_2 e pr_3 correspondem, respectivamente, à probabilidade de que a partícula siga seu próprio caminho, siga em direção a sua melhor posição já atingida (*pbest*) e siga em direção a melhor posição encontrada até o momento (*gbest*). Então, os algoritmos procedem modificando a posição das partículas de acordo com o operador de velocidade aleatoriamente escolhido. No fim, as probabilidades são atualizadas.

A representação de uma posição para ambos os algoritmos é uma seqüência de permutações, $p = (x_1, x_2, ..., x_N)$, onde N é o número de cidades. Inicialmente, é atribuído um valor alto para pr_1 e valores baixos para pr_2 e pr_3. O objetivo é que a partícula siga movimentos próprios mais frequentemente nas primeiras iterações. Durante a execução, esta situação vai sendo alterada e, nas iterações finais, pr_3 terá o maior valor. A idéia é intensificar a busca em boas regiões do espaço de busca nas iterações finais.

5.2. Algoritmo Aplicado ao Problema da AGM-mc*d*

O algoritmo aplicado ao problema da AGM-mc*d* usa o mesmo modelo descrito para os dois algoritmos aplicados ao PCV: uma busca local como operador de velocidade para seguir o seu próprio caminho, e o *path-relinking* como operador de velocidade para seguir da posição atual em direção a uma outra posição (pbest ou gbest) usando a estratégia *forward*. Contudo, como será aplicado a outro problema de otimização, mudam-se as representações e os operadores.

A representação usada para cada posição consiste numa lista de arestas, as quais irão formar a árvore geradora mínima, $p = \{(x_1, x_2), (y_1, y_2), ..., (z_1, z_2)\}$. Esta representação é conhecida como codificação direta [73]. Para inicialização das partículas é usado um algoritmo de busca em profundidade, sendo uma execução para cada partícula. Primeiramente são definidos aleatoriamente os valores para o vetor de escalarização λ, usado para ponderar os pesos das arestas, nesse caso, com dois objetivos. O custo de uma aresta é o valor resultante da equação

$$b_{i,j} = \sum_{k=1}^{K} \lambda^k . w_{i,j}^k$$

em que $K = 2$. Também é definido aleatoriamente o vértice inicial para começar a busca em profundidade. É chamado então o procedimento da busca em profundidade, seguindo a estrutura a seguir:

```
procedimento busca_profundidade(v, λ1, λ2, d)
{
  marcar(v);
  push(v);  // inserir na Pilha

  w = menor_vértice_adjacente_não_marcado(v, λ1, λ2, d);

  se (w != vazio) então  /* retornou algum vértice */
  {
    k = menor vértice adjacente marcado(w, v, λ1, λ2, d);
    se (k != vazio) então  /* retornou algum vértice */
      visita(w, k);
    senão  /* visita aresta (v,w) */
      visita(v, w);
    busca_profundidade(w, λ1, λ2, d);
  }

  v = pop(); // retirar o elemento do topo da Pilha
  /* rotina de retrocesso, procurando por vértices não-marcados */
  if (todos_marcados() == falso) {
    desmarcar(v);
    busca_profundidade(v, λ1, λ2, d);
  }
}
```

O procedimento `busca_profundidade(v, λ1, λ2, d)` é chamado com o vértice *v* escolhido como parâmetro e os elementos do vetor de escalarização λ_1 e λ_2 e a restrição de grau *d*. Esse procedimento segue visitando os vértices, traçando um caminho em profundidade. Quando termina esse caminho, ele retrocede procurando por arestas ainda não marcadas e traça um novo caminho, até que todos os vértices estejam marcados, ou seja, até que uma árvore seja construída. O procedimento `visitar(v, w)` insere a aresta (*v*, *w*) na árvore-solução e incrementa os graus dos vértices *v* e *w*. O procedimento `menor_vértice_adjacente_não_marcado(v, λ1, λ2, d)` retorna o vértice *w* (não-marcado) cuja aresta (*v*, *w*) é a de menor custo e que não viole a restrição de grau *d*. Já o procedimento `menor_vértice_adjacente_marcado(w, v, λ1, λ2, d)` retorna o vértice *k* (marcado) cuja aresta (*w*, *k*) é menor que a aresta (*v*, *w*) e que não viole a restrição de grau *d*. Isso serve para incluir a aresta (*w*, *k*) de custo ainda menor em lugar da aresta (*v*, *w*), aumentando as chances de se ter uma árvore de menor custo.

Há uma boa variação nas soluções iniciais, pois, além do vértice inicial ser definido aleatoriamente, o vetor de escalarização λ também é definido de forma aleatória, garantindo assim que mesmo que seja escolhido o mesmo vértice inicial para formar outra partícula, não se pode afirmar que a árvore gerada será a mesma. No caso da inicialização, não é verificado se, a cada partícula gerada, a solução é não-dominada com relação à outra partícula já existente.

Após o procedimento de inicialização, ocorre a definição dos valores do vetor de escalarização μ, semelhante ao processo de escolha do vetor λ, que será usado em cada iteração do algoritmo deste ponto em diante, a fim de avaliar as arestas.

Quanto ao procedimento de busca local, adiciona-se uma aresta que ainda não está na árvore, verificando se esta não viola a restrição de grau ao ser inserida. Esta inserção é baseada nas arestas de menor peso, calculadas por meio do vetor de escalarização μ. Deve-se então remover uma das arestas do conjunto de arestas do ciclo formado, com exceção da aresta que foi inserida. Quanto ao critério de escolha de inserção de uma aresta, insere-se a melhor aresta não pertencente à árvore tendo como base o escalar resultante da soma ponderada dos objetivos com o vetor de escalarização μ. Com relação à remoção, retira-se a pior aresta, diferente da inserida, tendo como base também o escalar resultante da soma ponderada dos objetivos com o vetor de escalarização μ para cada aresta. A figura 23 mostra um exemplo para dois casos que podem ocorrer no processo de busca local, exemplificando para uma restrição de grau igual a três (d = 3). A figura 23(a) ilustra quando a aresta (2, 3) é inserida e a aresta (1, 3) é escolhida pra ser retirada da árvore. A figura 23(b) ilustra quando se tenta inserir a aresta (2, 4), porém ela viola a restrição de grau no vértice 2. É então inserida a aresta (4, 5), sendo escolhida para ser removida a aresta (3, 4).

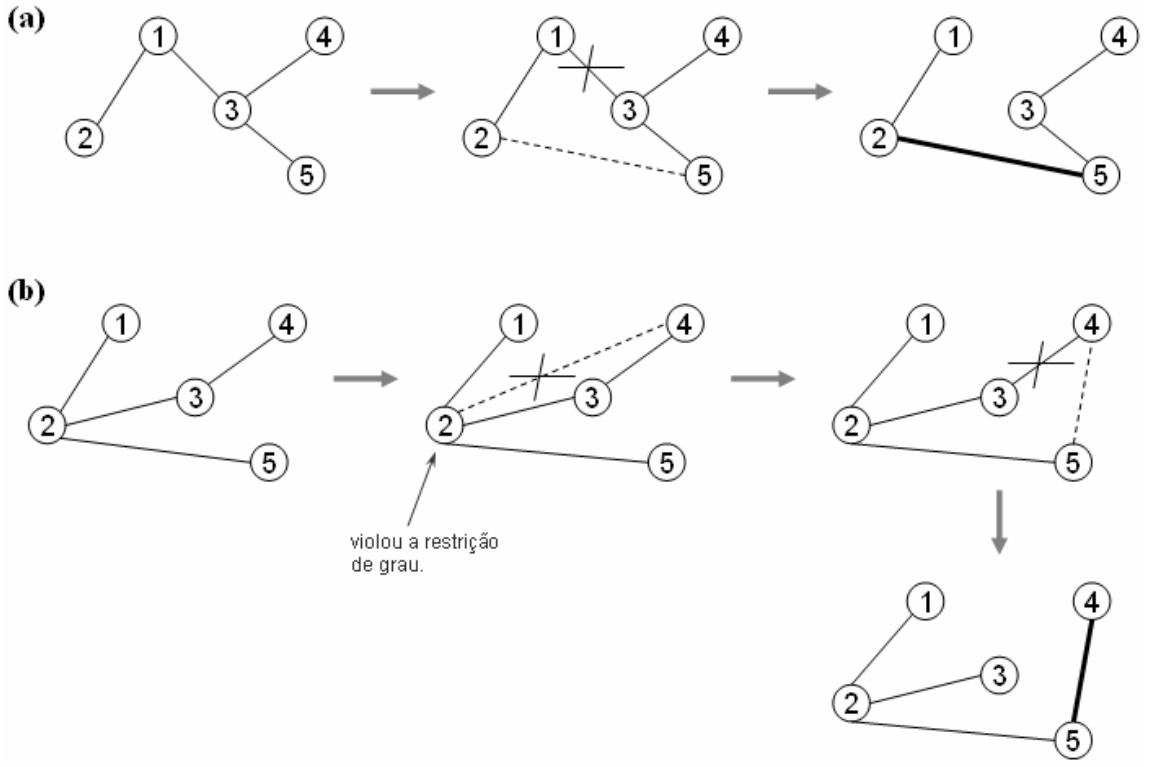


Figura 23: Exemplo do operador de busca local para o problema da AGM-mc*d* (d = 3)

Para o operador *path-relinking*, a estratégia é inserir na posição de origem arestas existentes na posição de destino e que não estejam contidas na posição de origem, até ficar igual à posição de destino. O passo seguinte é retirar uma aresta do ciclo formado que não esteja presente na posição de destino. Os critérios para inserção e remoção de arestas da posição de origem seguem os mesmos critérios descritos na busca local. Para inserção, escolhe-se a melhor aresta (com base no escalar resultante da soma ponderada dos objetivos com o vetor de escalarização μ) que esteja presente na posição de destino e ausente na solução de origem e que não viole a restrição de grau quando for inserida. Na remoção, retira-se a pior aresta que não esteja presente na posição de destino, tendo como base também o escalar resultante da soma ponderada dos objetivos com o vetor de escalarização μ. As arestas existentes em ambas as posições não serão manipuladas, ou seja, não serão removidas da árvore. Cada passo desse processo consistirá em uma posição intermediária que poderá ser uma solução não-dominada.

A figura 24 mostra um exemplo gráfico dessa operação, partindo da posição x_s = {(1,2), (1,3), (3,4), (3,5)} para x_t = {(1,4), (2,3), (3,5), (4,5)}. Note que elas possuem apenas uma aresta em comum, (3,5), a qual não será manipulada no processo. Na mudança do passo (a) para o passo (b), foi inserida a aresta (4, 5) formando o ciclo (3, 4, 5, 3), tendo agora que remover uma aresta desse ciclo. Foi então removida a aresta (3, 4), formando a solução intermediária no passo (c). Em seguida, no passo (d), foi inserida a aresta (2, 3) formando o ciclo (1, 2, 3, 1) e então foi removida a aresta (1, 3), resultando na solução intermediária do passo (e). No passo (f), foi inserida a aresta (1, 4) formando o ciclo (1, 2,

3, 5, 4, 1), sendo removida a aresta (1, 2) e resultando na solução que corresponde à posição de destino, no passo (g).

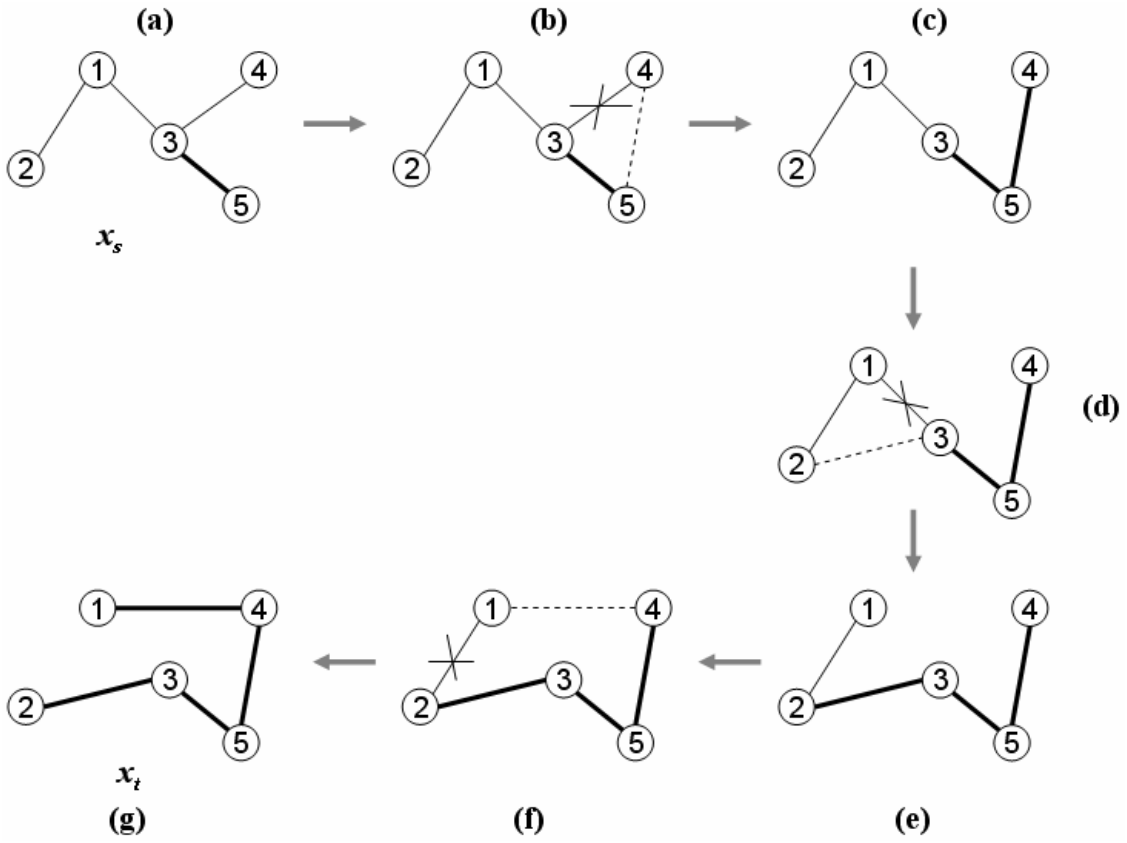


Figura 24: Exemplo do operador *path-relinking* para o problema da AGM-m*c*d (d = 3)

Outro fator a ser considerado é como definir a noção de melhor local (pbest) e melhor global (gbest), já que não se terá apenas uma solução, mas sim um conjunto de soluções não-dominadas (repositório). Para este algoritmo, foram definidos repositórios locais de soluções não-dominadas para cada partícula, substituindo o conceito de melhor local. No caso do melhor global, existe um repositório global que é compartilhado por todas as partículas.

Quando se vai usar o operador de *path-relinking* entre a posição atual e pbest, é selecionada aleatoriamente do repositório local uma solução não-dominada para fazer o papel de pbest. As soluções intermediárias que forem não-dominadas serão adicionadas ao repositório local e global e uma é escolhida para ser a posição corrente. Ao usar o operador de *path-relinking* entre a posição atual e gbest, é escolhida uma solução não-dominada do repositório global e as soluções não-dominadas intermediárias serão adicionadas ao repositório global e local e uma é escolhida para ser a posição corrente da partícula em questão.

Nesse processo, foram usadas 20 partículas, em que o tamanho do repositório local possui o tamanho máximo de 10 soluções e o repositório global possui o tamanho máximo de 10000 soluções. Quando os repositórios estão cheios e tem-se uma nova solução não-dominada para ser adicionada, escolhe-se aleatoriamente uma solução do repositório em questão e sorteia pelo método da roleta se a solução escolhida deve ser substituída pela nova solução ou não. No repositório local, a nova solução tem 50% de chances de entrar, enquanto a solução que está para ser substituída tem 50% de chances de permanecer no repositório. Quando ao repositório global, a nova solução tem 60% de chances de entrar e a solução a ser substituída tem 40% de chances de permanecer. Isso faz com que o repositório global seja mais susceptível a mudanças que o repositório local. O algoritmo executa por 100 iterações.

Tanto para a busca local quanto para o operador de *path-relinking*, o procedimento principal é a identificação do ciclo formado, do qual será retirada uma aresta. Esse procedimento tem complexidade $O(n)$. A verificação da satisfação da restrição de grau é feita em tempo constante. No máximo, a arestas serão analisadas. Portanto, uma iteração de um dos dois procedimentos terá complexidade $O(m+n)$.

A partir da aresta (v, w) a ser inserida, o vértice w é marcado (e será o vértice raiz) e um procedimento de busca em profundidade adaptado é chamado com o vértice w como parâmetro. Esse procedimento realiza a busca em profundidade sendo que, quando há mais de dois vértices marcados, ele verifica se o vértice raiz é adjacente ao vértice que se está analisando. Se essa condição for verdadeira, é porque formou o ciclo. O procedimento então pára e retorna o conjunto de arestas que forma o ciclo.

Probabilidades iniciais são associadas a cada um dos três movimentos: $pr_1 = 90\%$, $pr_2 = 5\%$ e $pr_3 = 5\%$. Estas probabilidades vão sendo alteradas à medida que o algoritmo avança na execução. O algoritmo proposto será comparado com o algoritmo AESSEA, definido por Knowles e Corne [54, 57], usando gráficos e o indicador de qualidade I-epsilon aditivo ($I_{\varepsilon+}$) definido por Zitzler [97].

Um modelo geral para o algoritmo proposto é mostrado a seguir:

```
procedimento PSO_AGM-mcd
{
  defina probabilidades iniciais para os movimentos:
    pr1 = x /* seguir o próprio caminho, v1 */
    pr2 = y /* seguir pbest (repositório local: arc_local), v2 */
    pr3 = z /* seguir gbest (repositório global: arc_global), v3 */
    /* x + y + z = 1 (100%) */

  para i = 1 até #partículas faça  /* iniciar população */
    λ1 = rand(0, 1)
    λ2 = 1 - λ1
    raiz = rand(0, num_vertices-1)
    pi = busca_profundidade(raiz, λ1, λ2)
  fim_para

  construa arc_global com as soluções não dominadas da população
  para cada partícula gere arc_local(pi) com pi

  faça
    μ1 = rand(0, 1)
    μ2 = 1 - μ1

    para i = 1 até #partículas faça
      Escolha uma velocidade para pi com base em pr1, pr2 e pr3
        v1: pi = busca local(pi)
        v2: selecione pbest em arc_local(pi)
            q = path_relinking(pi, pbest)
            se q é solução não-dominada em relação a arc local(pi)
            então atualiza arc_local(pi) com q
            pi = q
        v3: selecione gbest em arc_global
            q = path_relinking(pi, gbest)
            pi = q
      Se pi é solução não-dominada em relação a arc_global então
        Atualize arc_global com pi
    fim_Para

    atualize as probabilidades:
      pr1 = pr1×0.95; pr2 = pr2×1.01; pr3 = 100%-(pr1+pr2)
  enquanto (critério de parada) não é satisfeito
}
```

Capítulo 6

Experimentos Computacionais

Os três algoritmos propostos foram implementados em C++ em um computador Pentium IV (3.0 GHz e 512 Mb de RAM) com sistema operacional Linux. A seguir são mostrados os experimentos computacionais realizados.

6.1. Algoritmos Aplicados ao PCV

Os dois algoritmos aplicados ao PCV foram aplicados a instâncias simétricas do repositório TSPLIB [74] com tamanhos variando de 51 até 7397. Os critérios de parada são:

- Encontrar a solução ótima, caso seja conhecida.
- Atingir um número de iterações definido.

Quanto ao número de iterações, foi feita uma análise com 20 e 50 iterações, fixando o tamanho da população em 20. A tabela 3 mostra resultados para um conjunto de instâncias da TSPLIB [74], para 20 execuções diferentes, em que esses resultados são equivalentes para ambas as iterações. As colunas da tabela 3 mostram o nome das instâncias, a melhor solução alcançada (Min) e a solução média, em termos de desvio percentual, e a média dos tempos de execução, em segundos.

O desvio percentual é calculado com a equação (3), onde *Sol* e *Opt* são, respectivamente, a melhor (ou média) solução obtida pelos algoritmos e a solução ótima.

$$(Sol - Opt) \times 100 / Opt \qquad (3)$$

Instância	PSO-LK (20 Iterações)			PSO-LK (50 Iterações)		
	Min	Média	T_{med} (s)	Min	Média	T_{med} (s)
pr439	0	0	0,779	0	0	0,771
pcb442	0	0	0,794	0	0	0,796
d493	0	0	19,378	0	0	18,722
rat575	0	0	6,474	0	0	4,945
p654	0	0	1,902	0	0	1,971
d657	0	0	12,416	0	0	9,814
rat783	0	0	5,246	0	0	5,226
dsj1000	0,0027	0,0031	178,482	0,0027	0,0030	185,812
pr1002	0	0	9,499	0	0	9,378
u1060	0	0	38,181	0	0	36,830
vm1084	0	0,0010	34,740	0	0,0016	46,733
pcb1173	0	0,0001	48,181	0	0	16,879
d1291	0	0	29,862	0	0	67,156
rl1304	0	0	21,618	0	0	20,092
rl1323	0	0,0092	225,32	0	0,0050	212,936
nrw1379	0,0017	0,0085	417,798	0,0017	0,0110	423,701
fl1400	0	0	15,423	0	0	15,071
fl1577	0	0,0135	461,995	0	0,0146	459,496
vm1748	0	0,0018	854,173	0	0,0009	875,8995
u1817	0	0,0863	789,18	0	0,0905	877,0525
rl1889	0	0,0073	894,432	0	0,0053	914,1235
d2103	0	0,0043	1137,55	0	0,0111	1133,561
u2152	0	0,0717	1415,32	0	0,0630	1371,609
pr2392	0	0,0021	577,782	0	0,0028	838,784
pcb3038	0,0101	0,0396	323,93	0,0036	0,0456	331,014
fl3795	0	0,0142	621,62	0	0,0231	870,878
fnl4461	0,0296	0,0462	583,76	0,0246	0,0495	552,653
MÉDIA	0,0016	0,0114	323,179	0,0012	0,0121	344,515

Tabela 3: Ajuste de parâmetro em relação ao número de iterações

Observando as médias calculadas para os resultados mostrados, na parte inferior da tabela 3, em termos percentuais, tem-se que a média dos mínimos com a configuração de 50 iterações é 33,3% melhor do que a configuração de 20 iterações. Contudo, na média dos resultados médios e na média dos tempos de execução, a configuração com 20 iterações é melhor 6,14% e 6,6%, respectivamente, do que a configuração de 50 iterações. Nota-se uma leve melhoria para 20 iterações, sendo esta a configuração utilizada.

Quanto ao tamanho da população, foram testadas configurações com 10, 15, 20 e 25 partículas, fixando o número de iterações em 20. A tabela 4 mostra resultados para um conjunto de instâncias da TSPLIB [74], para 20 execuções diferentes, em que esses resultados são melhores quando o tamanho das partículas é igual a 20 e 25. As colunas da tabela 4 mostram o nome das instâncias e, para cada tamanho de população, apresenta a

solução média (Méd), em termos de desvio percentual, e a média dos tempos de execução, em segundos.

Instância	Tamanho da População (partículas)							
	10		15		20		25	
	Média	T_{med}(s)	Média	T_{med}(s)	Média	T_{med}(s)	Média	T_{med}(s)
pr439	0	0,379	0	0,583	0	0,7795	0	0,955
pcb442	0	0,395	0	0,604	0	0,7945	0	0,982
d493	0	9,222	0	13,863	0	19,3785	0	24,44
rat575	0	5,945	0,0015	6,622	0	6,474	0	6,229
p654	0	0,946	0	1,413	0	1,9025	0	2,349
d657	0	9,84	0	10,029	0	12,4165	0	11,417
rat783	0	2,651	0	3,973	0	5,2465	0	6,705
pr1002	0	6,788	0	7,44	0	9,4995	0	11,638
u1060	0,0031	50,758	0,0035	69,511	0	38,1815	0	60,587
vm1084	0	18,133	0,0032	35,635	0,0010	34,7405	0	20,239
pcb1173	0,0002	31,632	0,0002	35,108	0,0001	48,181	0	24,92
d1291	0,0193	55,802	0	31,979	0	29,862	0,0003	84,374
rl1304	0	31,745	0	20,788	0	21,618	0	39,041
rl1323	0,0187	123,62	0,0117	173,33	0,0092	225,32	0,0030	181,53
nrw1379	0,0099	210,827	0,0145	321,165	0,0085	417,7985	0,0072	569,463
fl1400	0	8,515	0	11,607	0	15,423	0	19,267
fl1577	0,0800	259,319	0,0229	352,003	0,0135	461,995	0,0175	579,052
vm1748	0,0117	483,823	0	644,796	0,0018	854,173	0	1147,853
u1817	0,1288	460,225	0,1003	652,553	0,0863	789,18	0,0919	1158,719
rl1889	0,0197	459,194	0,0173	471,295	0,0073	894,432	0,0062	1205,756
MÉDIA	0,0146	111,488	0,00875	143,2149	0,00638	194,3698	0,00630	257,7758

Tabela 4: Ajuste de parâmetro em relação ao tamanho da população

Observando as médias calculadas para os resultados mostrados, na parte inferior da tabela 4, em termos percentuais, tem-se que para a população com tamanho 10, a média dos resultados médios é 66,86% pior do que para a população com tamanho 15, entretanto, na média do tempo de execução, a população com tamanho 10 é melhor 28,46% do que a população com tamanho 15. Já comparando as populações de tamanho 15 e 20, tem-se que a população com tamanho 15 é pior 37,15% do que a de tamanho 20 com relação à média dos resultados médios e melhor 35,72% com relação à média dos tempos de execução. Na comparação das populações de tamanho 20 e 25, a população com tamanho 20 é pior apenas 1,27% do que a de tamanho 25 com relação à média dos resultados médios e melhor 32,62% com relação à média dos tempos de execução. Sendo assim, a configuração com o tamanho da população igual a 20 foi então escolhida.

Um primeiro experimento comparou os dois algoritmos propostos aplicados ao PCV em 11 instâncias simétricas com tamanhos variando de 51 até 2103. Foram realizadas 20 execuções independentes para cada algoritmo em cada instância. Os resultados são mostrados na tabela 5 em termos de desvio percentual da solução ótima.

As colunas mostram o nome das instâncias do TSPLIB, a melhor solução (Min) e a solução média para os dois algoritmos denotados por PSO-INV (com a busca local de inversão) e PSO-LK (com a busca local de Lin-Kernighan).

Instância	PSO-INV		PSO-LK	
	Min	Média	Min	Média
eil51	0,2347	1,9836	0	0
berlin52	0	2,0041	0	0
eil76	2,4164	4,5167	0	0
rat195	5,8114	8,7581	0	0
pr299	5,8476	7,9952	0	0
pr439	4,4200	8,0111	0	0
d657	6,9656	9,6157	0	0
pr1002	9,8574	11,1900	0	0
d1291	13,2104	15,5505	0	0
rl1304	10,4432	11,9942	0	0
d2103	16,7383	18,4180	0	0,0043

Tabela 5: Comparação dos dois algoritmos PSO aplicados ao PCV

O PSO-LK apresentou soluções bem melhores do que o PSO-INV, já que a busca local LK é mais poderosa do que a busca local de inversão.

Fazendo uma comparação dos resultados do PSO-INV com os resultados dos algoritmos relatados nos trabalhos de Pang *et al*. [68] e Machado e Lopes [62], percebe-se que o PSO-INV apresenta melhores resultados que o algoritmo de Pang *et al*., porém apresenta piores resultados se comparado com os resultados do algoritmo de Machado e Lopes. Este último, entretanto, não supera o PSO-LK em termos de qualidade de solução.

Como no trabalho de Wang *et al*. [90] é mostrado o resultado apenas para uma instância assimétrica do PCV, o algoritmo relatado neste trabalho foi implementado a fim de comparar seu desempenho com o PSO-INV. A tabela 6 mostra os resultados médios (em desvio percentual) do PSO-INV e dos algoritmos PSO-P [68] e PSO-W [90] para três instâncias simétricas do PCV com $N > 50$ do experimento computacional relatado por Pang *et al*. [68]. A tabela 7 mostra os resultados médios (em desvio percentual) do PSO-INV, do PSO-LK e do algoritmo PSO-M [62] para sete instâncias simétricas do PCV com $N > 75$ do experimento computacional relatado por Machado e Lopes [62].

Instância	PSO-W	PSO-P	PSO-INV
eil51	114	2,54	1,98
berlin52	115	2,12	2,00
eil76	162	4,75	4,52

Tabela 6: Comparação de duas heurísticas PSO com o PSO-INV

Instância	PSO-M	PSO-INV	PSO-LK
rat195	0,9834	8,7581	0
pr299	0,5900	7,9952	0
pr439	2,9561	8,0111	0
d657	3,8490	9,6157	0
d1291	4,5811	15,5505	0
rl1304	3,2456	11,9942	0
d2103	7,7493	18,4180	0,0043

Tabela 7: Comparação de uma heurística PSO com o PSO-INV e o PSO-LK

Os resultados mostrados na tabela 6 e na tabela 7 correspondem à média das soluções encontradas em vinte execuções independentes. O algoritmo PSO-INV também foi aplicado à instância assimétrica br17 relatada por Wang *et al*. [90]. Os resultados são mostrados na tabela 8, onde as colunas mostram o algoritmo, a melhor solução e a média, em termos de desvio percentual da solução ótima, e o tempo médio de processamento em segundos.

Algoritmo	Min	Média	T (s)
PSO-INV	0	0	< 0,01
PSO-W	0	16,66	60,04

Tabela 8: Comparação do PSO-INV e PSO-W para a instância br17 do TSPLIB

Um experimento computacional investigou as diferenças entre os resultados obtidos pelo procedimento LK [6] e o algoritmo PSO-LK, onde o objetivo foi descobrir se o PSO-LK era capaz de melhorar os resultados do LK. A tabela 9 mostra o desvio percentual do ótimo, para trinta instâncias assimétricas do TSPLIB com N variando de 439 até 7397. Os elementos marcados em negrito mostram onde houve melhoria. Foram realizadas vinte execuções independentes para cada algoritmo. Das trinta instâncias do experimento, o PSO-LK foi capaz de melhorar vinte e três resultados mínimos e todos os resultados médios. Uma análise estatística mostra que, na média, melhorias de 88% e 89% foram alcançadas nos resultados mínimos e nos médios, respectivamente.

Instância	LK			PSO-LK		
	Min	**Média**	**T_{med}(s)**	**Min**	**Média**	**T_{med}(s)**
pr439	0	0,0463	0,88	0	**0**	0,78
pcb442	0	0,1119	0,67	0	**0**	0,80
d493	0,0029	0,1216	2,21	**0**	**0**	19,38
rat575	0,0295	0,1277	0,62	**0**	**0**	6,47
p654	0	0,0078	3,72	0	**0**	1,90
d657	0,0020	0,1500	1,70	**0**	**0**	12,42
rat783	0	0,0704	0,90	0	**0**	5,25
dsj1000	0,0731	0,2973	5,89	**0,0027**	**0,0031**	178,48
pr1002	0	0,1318	1,96	0	**0**	9,50
u1060	0,0085	0,1786	3,12	**0**	**0**	38,18
vm1084	0,0017	0,0669	2,59	**0**	**0,0010**	34,74
pcb1173	0	0,1814	1,78	0	**0,0001**	48,18
d1291	0,0039	0,4333	1,79	**0**	**0**	29,86
rl1304	0,0202	0,3984	2,67	**0**	**0**	21,62
rl1323	0,0463	0,2300	2,34	**0**	**0,0092**	225,32
nrw1379	0,0547	0,1354	2,45	**0,0017**	**0,0085**	417,80
fl1400	0	0,1215	14,87	0	**0**	15,42
fl1577	0,7371	2,2974	6,30	**0**	**0,0135**	461,99
vm1748	0,0903	0,1311	4,66	**0**	**0,0018**	854,17
u1817	0,1976	0,5938	1,88	**0**	**0,0863**	789,18
rl1889	0,1836	0,3844	4,58	**0**	**0,0073**	894,43
d2103	0,0597	0,3085	2,82	**0**	**0,0043**	1137,53
u2152	0,2381	0,5548	2,16	**0**	**0,0717**	1415,32
pr2392	0,0775	0,3904	3,78	**0**	**0,0021**	577,78
pcb3038	0,1598	0,2568	5,42	**0,0101**	**0,0396**	323,94
fl3795	0,5665	1,0920	15,60	**0**	**0,0142**	621,63
fnl4461	0,0882	0,1717	9,08	**0,0296**	**0,0462**	583,78
rl5915	0,3528	0,5343	10,08	**0,0122**	**0,0633**	1359,25
rl5934	0,2221	0,4761	10,65	**0,0012**	**0,0650**	983,04
pla7397	0,1278	0,2912	21,85	**0,0075**	**0,0253**	1563,22

Tabela 9: Comparação dos resultados do LK com PSO-LK

A tabela 10 mostra uma comparação dos resultados obtidos pelo PSO-LK e de três heurísticas efetivas: Tourmerge [19], uma variante iterativa do LK, NYYY (descrita em http://www.research.att.com/~dsj/chtsp/) e outra variante iterativa do LK, JM. Os resultados dessas heurísticas foram obtidos no *site* do DIMACS *Challenge*: http://www.research.att.com/~dsj/chtsp/results.html.

As colunas relacionadas ao PSO-LK mostram os resultados mínimos e médios encontrados em vinte execuções diferentes. As colunas relacionadas ao algoritmo Tourmerge mostram os resultados mínimos e médios em cinco execuções independentes. Os resultados para as instâncias fnl4461 e pla7397 não relatados. As colunas relacionadas com os algoritmos NYYY e JM mostram os resultados mínimos obtidos em dez execuções com *N* iterações.

Instância	PSO-LK		Tourmerge		ILKNYYY	ILKJM
	Min	Média	Min	Média	Nb10	Nb10
dsj1000	0,0027	0,0031	0,0027	0,0478	0	0,0063
pr1002	0	0	0	0,0197	0	0,1482
u1060	0	0	0	0,0049	0,0085	0,0210
vm1084	0	0	0	0,0013	0,0217	0,0217
pcb1173	0	0	0	0,0018	0	0,0088
d1291	0	0	0	0,0492	0	0
rl1304	0	0	0	0,1150	0	0
rl1323	0	0,0092	0,01	0,0411	0,01	0
nrw1379	0,0017	0,0085	0	0,0071	0,0247	0,0018
fl1400	0	0	0	0	0	0
fl1577	0	0,0135	0	0,0225	0	0
vm1748	0	0,0018	0	0	0	0
u1817	0	0,0863	0,0332	0,0804	0,1643	0,2657
rl1889	0	0,0073	0,0082	0,0682	0,0082	0,0041
d2103	0	0,0043	0,0199	0,3170	0,0559	0
u2152	0	0,0717	0	0,0794	0	0,1743
pr2392	0	0,0021	0	0,0019	0,0050	0,1495
pcb3038	0,0101	0,0396	0,0036	0,0327	0,0247	0,1213
fl3795	0	0,0142	0	0,0556	0	0,0104
fnl4461	0,0296	0,0462	---	---	0,0449	0,1358
rl5915	0,0122	0,0633	0,0057	0,0237	0,0580	0,0168
rl5934	0,0012	0,0650	0,0023	0,0104	0,0115	0,1723
pla7397	0,0075	0,0253	---	---	0,0209	0,0497

Tabela 10: Comparação de heurísticas para instâncias simétricas do PCV

Das vinte e uma instâncias as quais o Tourmerge apresenta resultados, PSO-LK encontrou cinco resultados mínimos melhores, enquanto o Tourmerge encontrou três mínimos resultados melhores e ambos encontraram resultados com a mesma qualidade para treze instâncias. Com relação aos resultados médios, o Tourmerge encontrou valores melhores em sete instâncias, enquanto o PSO-LK encontrou resultados melhores em treze instâncias e ambos encontraram resultados com a mesma qualidade para apenas uma instância. As médias das colunas "Min" e "Média" para as vinte e uma instâncias são 0,0013 e 0,0186 para o PSO-LK e 0,0041 e 0,0467 para o Tourmerge. Estes resultados mostram que, em média, o PSO-LK apresenta melhor desempenho do que o Tourmerge tantos nos resultados mínimos quanto nos médios.

Comparando o PSO-LK com o ILK-NYYY, pode-se observar que das vinte e três instâncias do experimento, o PSO-LK encontrou treze melhores resultados, enquanto o ILK-NYYY encontrou melhor resultado para apenas uma instância e o mesmo resultado foi encontrado para nove instâncias. As médias dos melhores resultados, para as vinte e três instâncias, foram 0,0028 para o PSO-LK e 0,0199 para ILK-NYYY.

A comparação com o ILK-JM mostra que o PSO-LK encontra melhores resultados em dezesseis instâncias e o mesmo resultado foi encontrado para sete instâncias. As médias

dos melhores resultados, para as vinte e três instâncias, foram 0.0028 para o PSO-LK e 0.0569 para o ILK-JM.

6.2. Algoritmo Aplicado ao Problema da AGM-mc*d*

O terceiro algoritmo, aplicado ao problema da AGM-mc*d*, foi testado com as instâncias definidas por Knowles e Corne [57], sendo grafos completos com 10, 25, 50, 100 e 200 vértices e 2 objetivos. Os tipos das instâncias usadas são listados abaixo:

- **Côncavas:** grafos valorados com números reais com uma larga região côncava na fronteira de Pareto.
- **Correlacionadas:** grafos valorados com números reais aleatoriamente combinados.

As instâncias com 10, 25 e 50 vértices foram fornecidas por J. D. Knowles; e as instâncias com 100 e 200 vértices foram geradas e fornecidas por Rocha [78], seguindo a metodologia descrita por Knowles e Corne [57].

Primeiramente, fez-se uma comparação gráfica do algoritmo proposto com o AESSEA, definido por Knowles e Corne [54, 57]. Essa comparação gráfica na maioria das vezes é ineficiente para definir qual algoritmo é melhor. A seção 3.7.4 apresenta algumas técnicas de avaliação para problemas multicritério que permitem definir qual algoritmo é mais eficiente. Neste trabalho, utilizou-se como técnica de avaliação o indicador de qualidade I-epsilon aditivo ($I_{\varepsilon+}$), apresentado na seção 3.7.4.2.

As instâncias testadas para os dois algoritmos (AESSEA e PSO-AGMmcd) foram as seguintes: 10vconc1, 10vcorr1, 10vcorr2, 25vconc1, 25vcorr1, 25vcorr2, 50vconc1, 50vcorr1, 50vcorr2, 100vconc1. 100vconc2, 100vcorr1, 200vcorr1 e 300vcorr1. O algoritmo AESSEA, fornecido por J. D. Knowles, para o grau de restrição 3, gerou resultados apenas para as instâncias 10vconc1, 10vcorr1 e 10vcorr2 e, para grau 4, gerou resultados para as instâncias listadas no início deste parágrafo. O algoritmo AESSEA não conseguiu gerar resultados para as instâncias 100vcorr2, 200vconc1, 200vconc2, 200vcorr2, 300vconc1, 300vconc2, 300vcorr2, 400vconc1, 400vconc2, 400vcorr1, 400vcorr2, 500vconc1, 500vconc2, 500vcorr1, 500vcorr2, sendo estas não relatadas para este algoritmo. A razão pelo qual o AESSEA não gerou resultados para estas instâncias é desconhecida pelo autor deste trabalho.

As figuras 25 e 26 mostram os gráficos para um grau de restrição 3 com 10 vértices para os dois tipos de instâncias (côncava e correlacionada). Pode-se notar que o algoritmo proposto, definido como PSO-AGMmcd, possui uma fronteira de Pareto bem semelhante à fronteira gerada pelo AESSEA, sendo difícil de determinar qual é o melhor. Quanto ao espalhamento, o AESSEA apresenta-se melhor na instância 10vConc1 (figura 25).

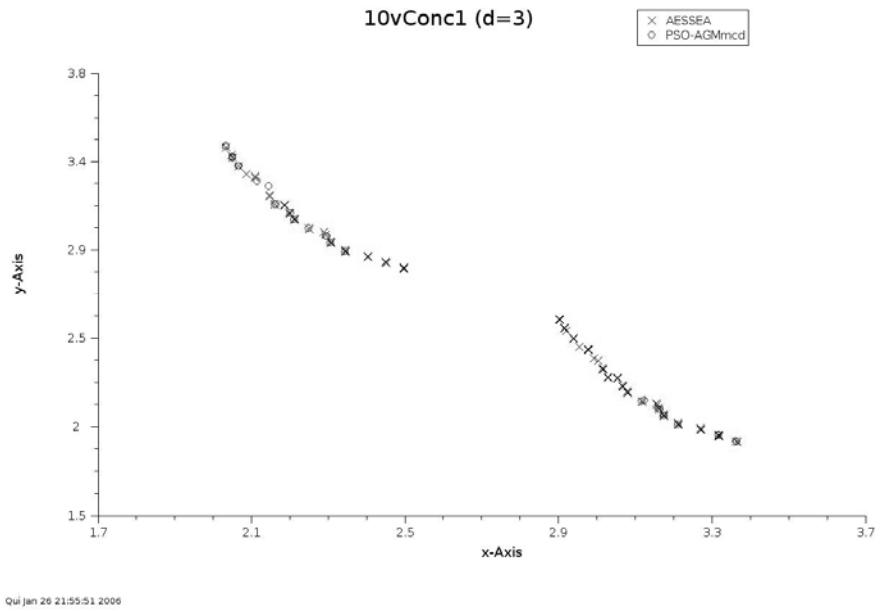


Figura 25: Instância do tipo côncava com grau de restrição 3 (10 vértices)

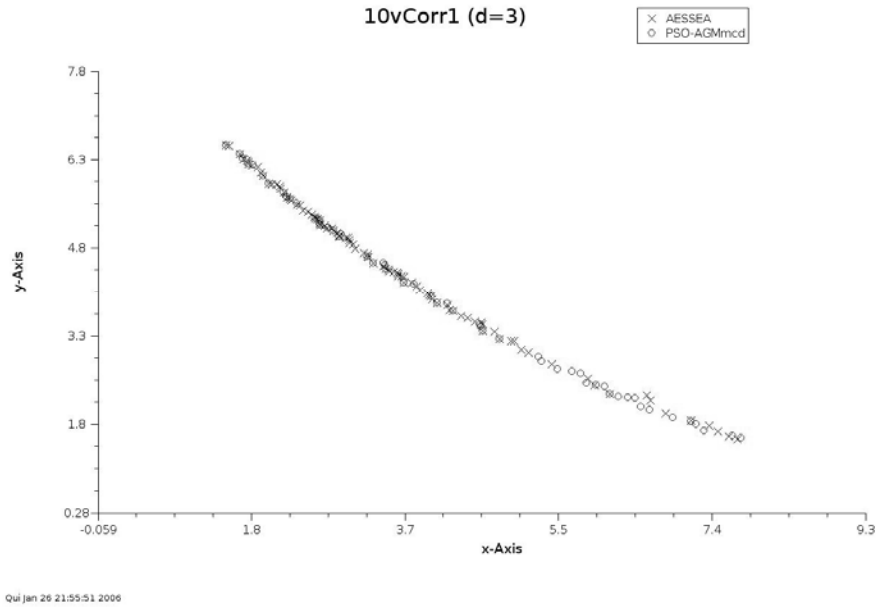


Figura 26: Instância do tipo correlacionada com grau de restrição 3 (10 vértices)

As figuras 27 e 28 mostram os gráficos para um grau de restrição 4 com 10 vértices para os dois tipos de instâncias (côncava e correlacionada). Nestes dois casos, também se pode notar que o algoritmo proposto possui uma fronteira de Pareto bem semelhante à fronteira gerada pelo AESSEA, sendo também difícil de determinar qual é o melhor. Quanto ao espalhamento, o AESSEA apresenta-se melhor na instância 10vConc1 (figura 27).

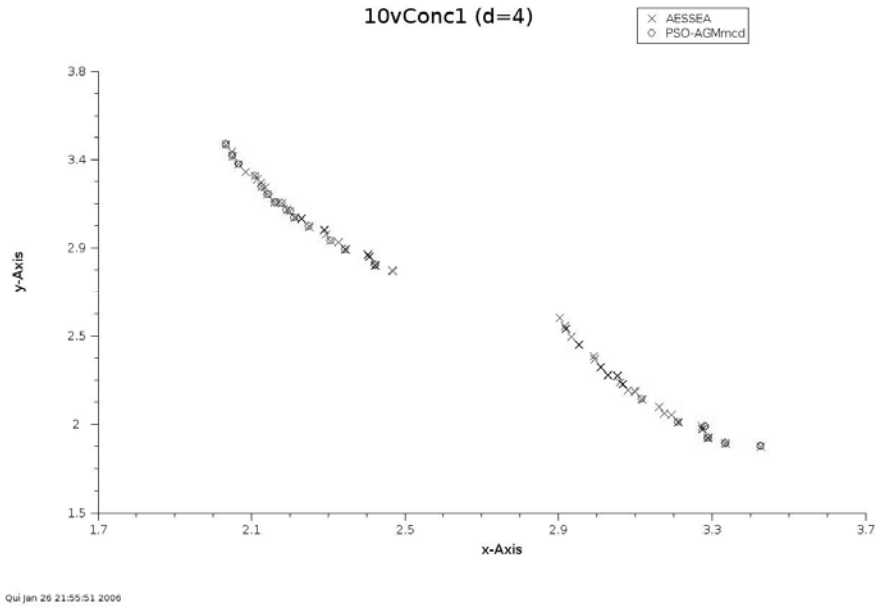


Figura 27: Instância do tipo côncava com grau de restrição 4 (10 vértices)

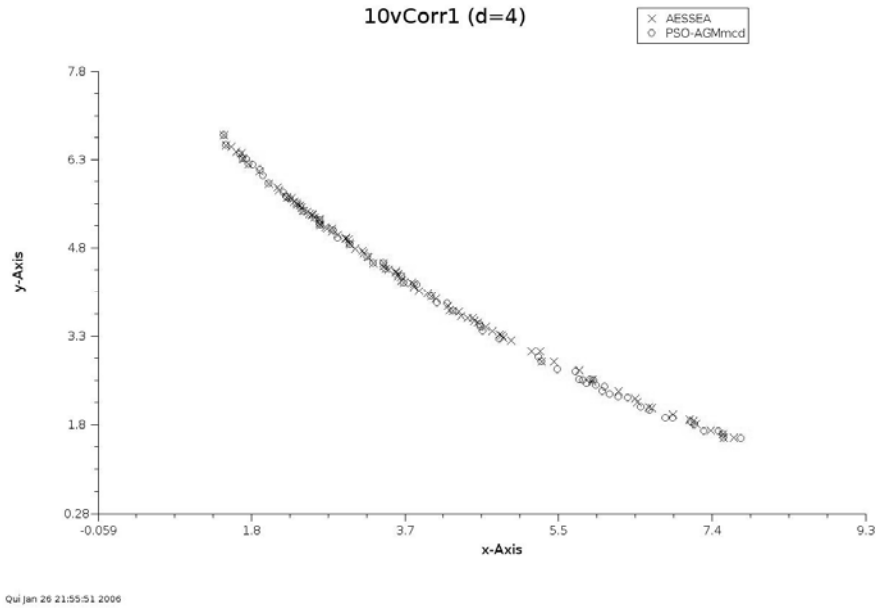


Figura 28: Instância do tipo correlacionada com grau de restrição 4 (10 vértices)

As figuras 29 e 30 mostram os gráficos para um grau de restrição 4 com 25 vértices nos dois tipos de instâncias (côncava e correlacionada). Já é possível notar que o algoritmo proposto apresenta uma leve melhoria com relação ao AESSEA, mas isso não é trivial de se

afirmar. Quanto ao espalhamento, o AESSEA apresenta-se melhor na instância 25vConc1 (figura 29), na região côncava.

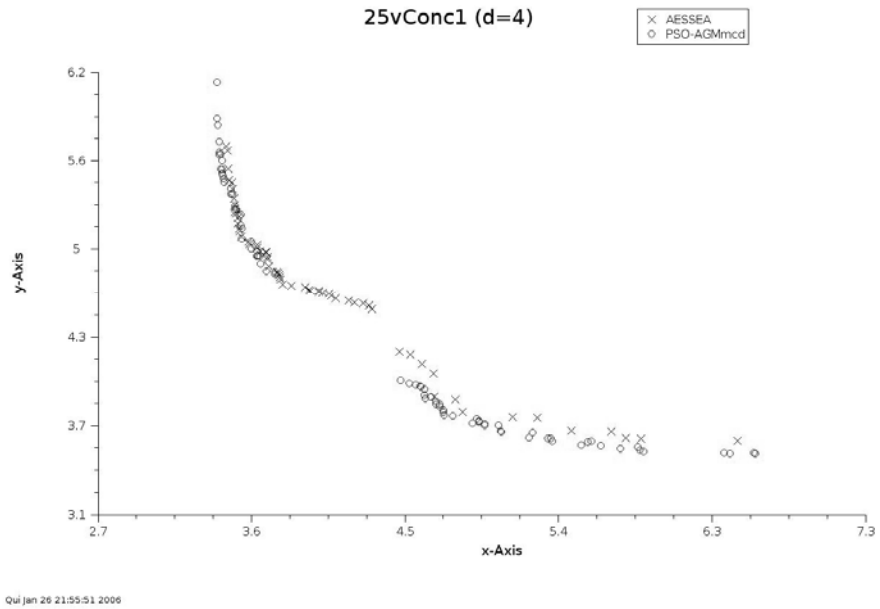


Figura 29: Instância do tipo côncava com grau de restrição 4 (25 vértices)

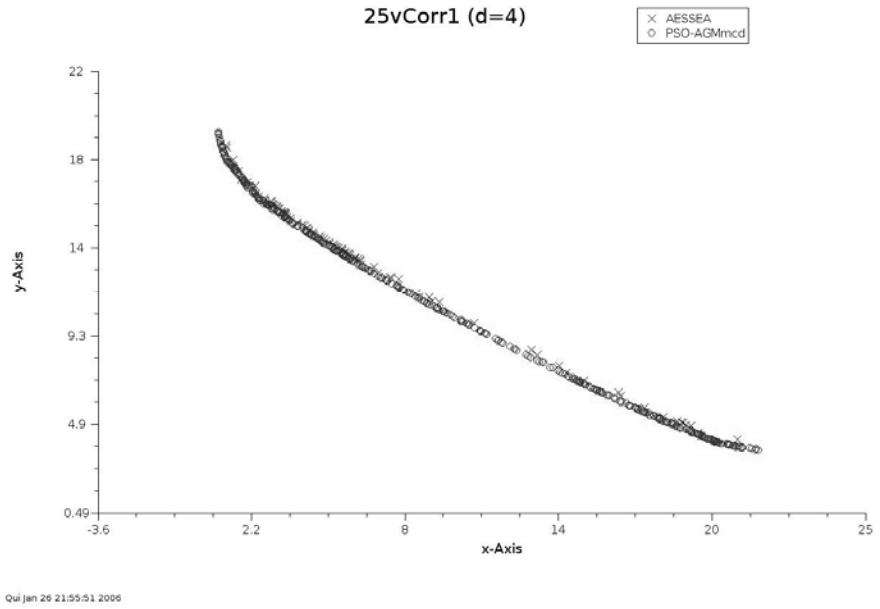


Figura 30: Instância do tipo correlacionada com grau de restrição 4 (25 vértices)

As figuras 31 e 32 mostram os gráficos para um grau de restrição 4 com 50 vértices nos dois tipos de instâncias (côncava e correlacionada). Também é possível notar que o

algoritmo proposto apresenta uma leve melhoria com relação ao AESSEA, mas também não é tão fácil de se afirmar. Com relação ao espalhamento, o AESSEA apresenta uma pequena vantagem na instância 50vConc1 (figura 31), pois apresenta algumas soluções na região côncava.

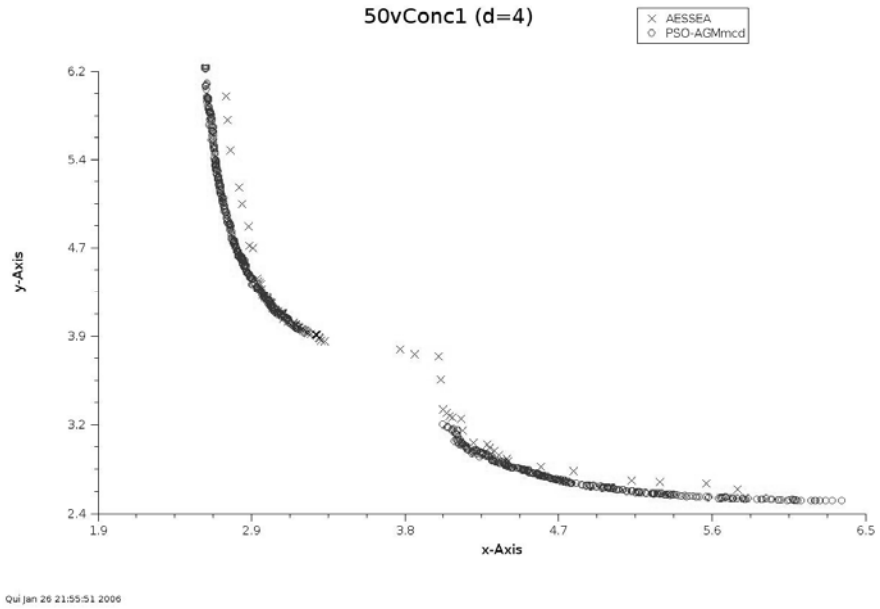


Figura 31: Instância do tipo côncava com grau de restrição 4 (50 vértices)

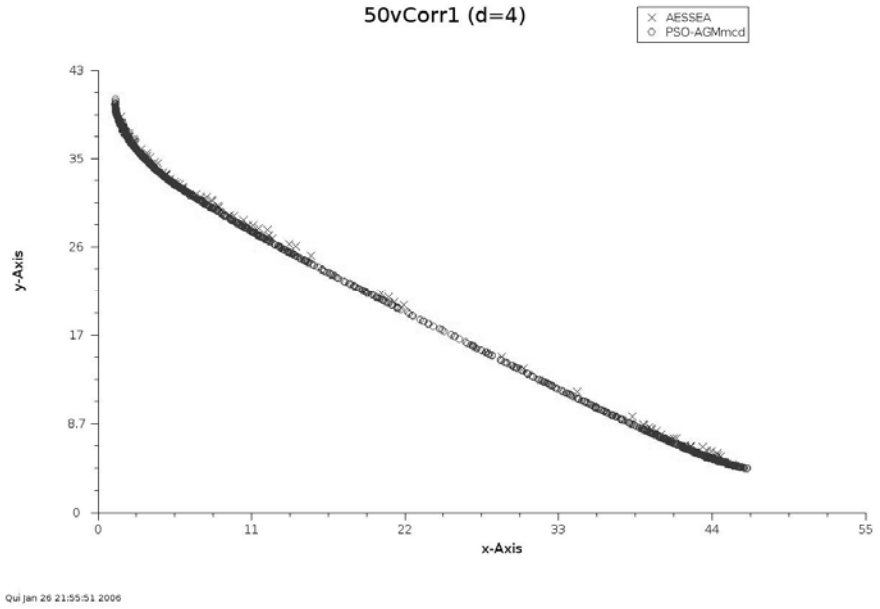


Figura 32: Instância do tipo correlacionada com grau de restrição 4 (50 vértices)

Para instâncias pequenas, realmente é difícil definir graficamente qual o melhor algoritmo dentre o PSO-AGMmcd e o AESSEA. Contudo, ao fazer a comparação com instâncias maiores (100, 200 e 300 vértices), verificou-se pelo gráfico que o PSO-AGMmcd é melhor que o AESSEA. As figuras 33, 34, 35, 36 e 37 ilustram esses casos.

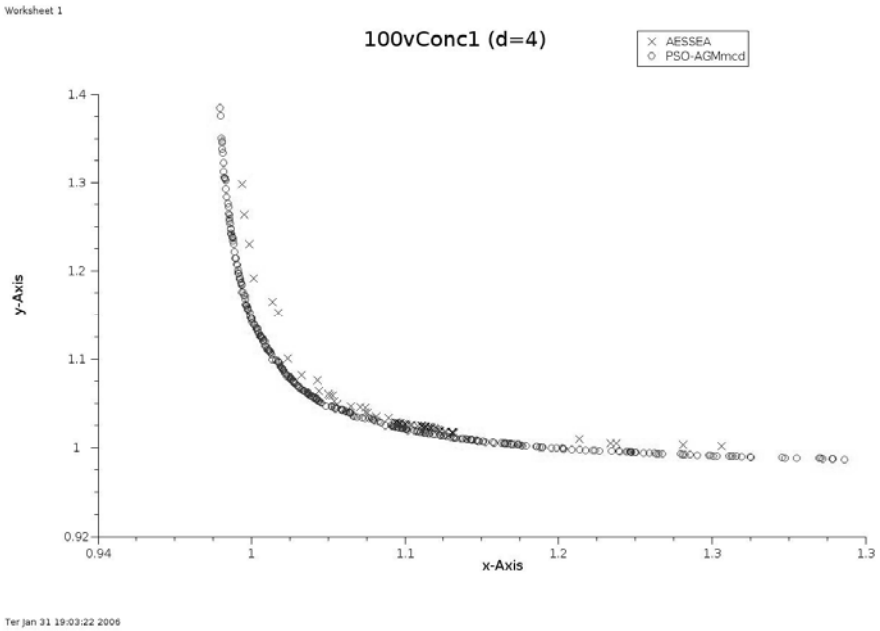


Figura 33: Instância do tipo côncava com grau de restrição 4 (100 vértices)

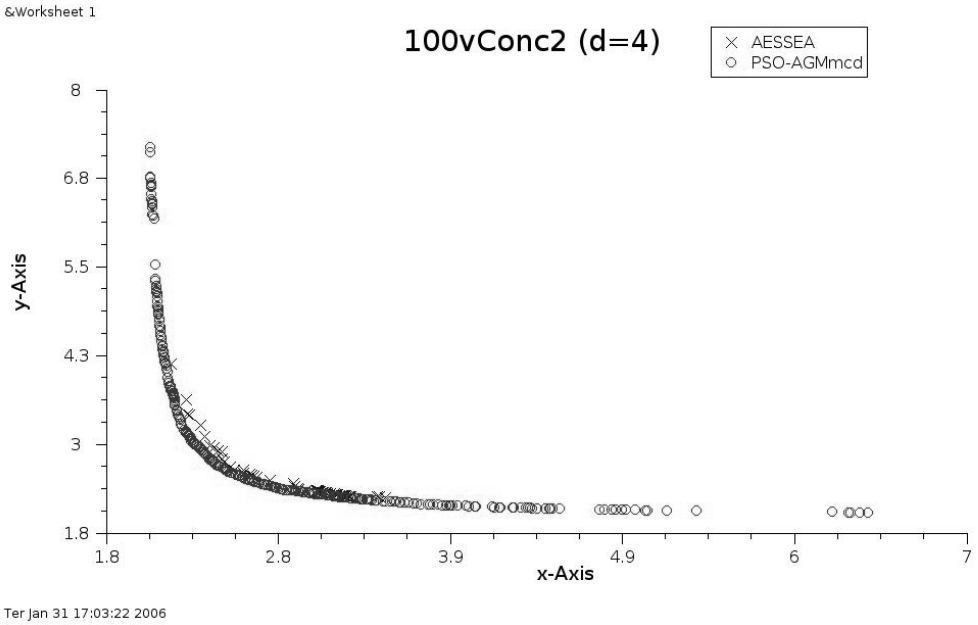


Figura 34: Instância do tipo côncava com grau de restrição 4 (100 vértices)

As fronteiras de Pareto das figuras 33 e 34 para os dois algoritmos ainda se mostram bem próximas, contudo, já se nota uma melhoria do PSO-AGMmcd com relação ao AESSEA. As figuras a seguir (35, 36 e 37) mostram com mais evidência a melhoria do algoritmo PSO-AGMmcd.

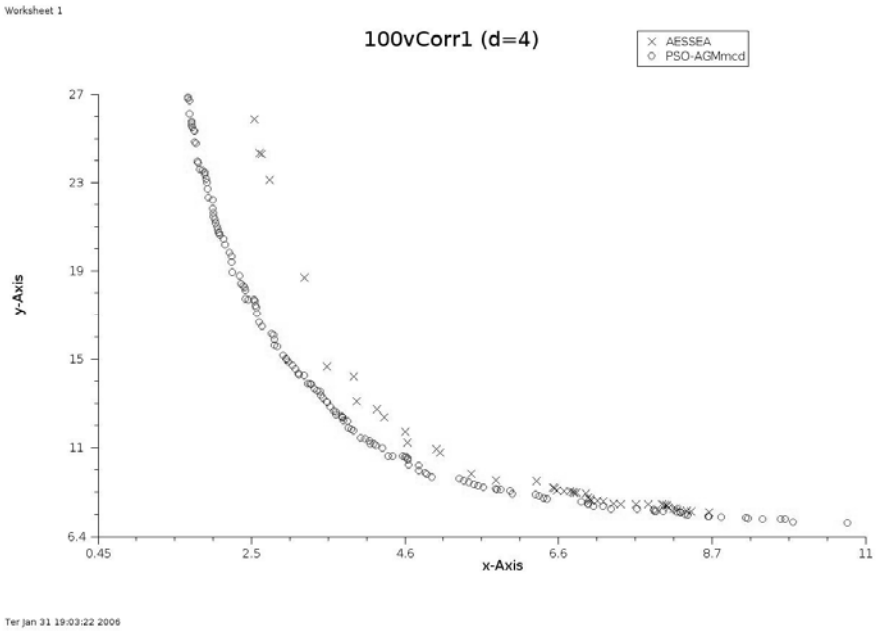


Figura 35: Instância do tipo correlacionada com grau de restrição 4 (100 vértices)

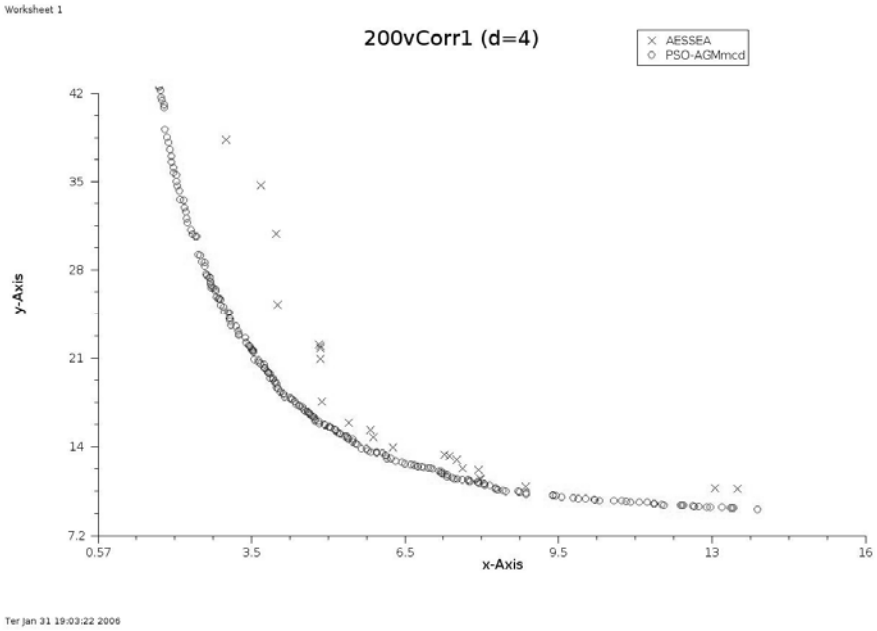


Figura 36: Instância do tipo correlacionada com grau de restrição 4 (200 vértices)

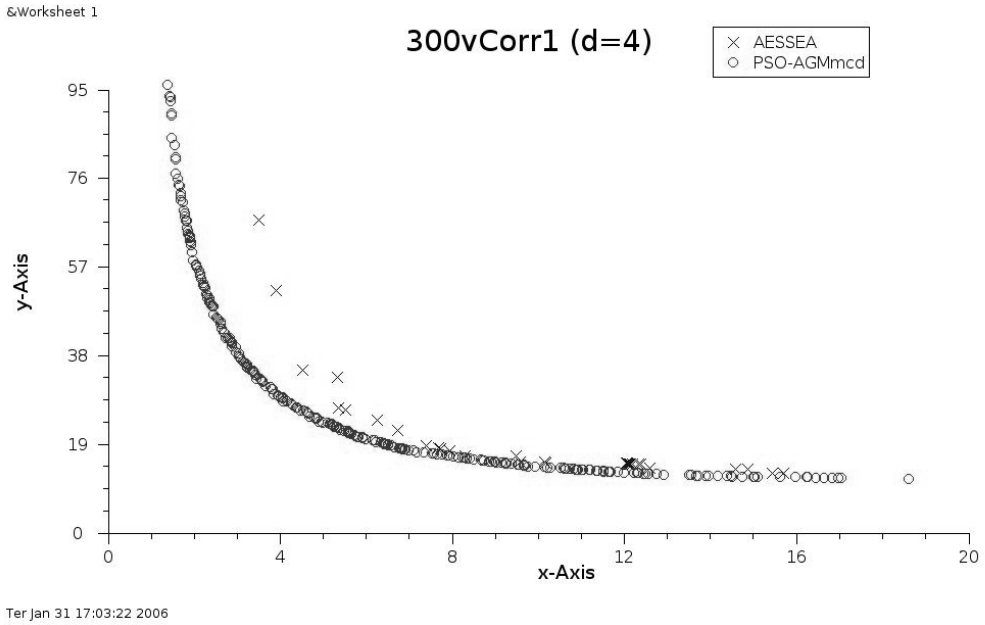


Figura 37: Instância do tipo correlacionada com grau de restrição 4 (300 vértices)

Somente a verificação gráfica não é suficiente para definir qual algoritmo é melhor. Foi feita também uma análise usando o indicador de qualidade I-epsilon aditivo ($I_{\varepsilon+}$), definido por Zitzler [97], a fim de ratificar as informações fornecidas pelos gráficos.

Esse indicador é aplicado às duas fronteiras de Pareto dos algoritmos que se quer comparar: $I_{\varepsilon+}(A, B)$, onde A e B são conjuntos de soluções não-dominadas. A tabela 11 mostra os resultados da aplicação do indicador $I_{\varepsilon+}$. Nesse caso, o conjunto A representará a fronteira de Pareto do algoritmo PSO-AGMmcd; e o conjunto B representará a fronteira de Pareto do algoritmo AESSEA.

Instância	$I_{\varepsilon+}(A, B)$	$I_{\varepsilon+}(B, A)$
10vconc1 (d=3)	0,224156	0,020807
10vcorr1 (d=3)	0,174963	0,152756
10vcorr2 (d=3)	0.058041	0.081451
10vconc1 (d=4)	0,222750	0,000570
10vcorr1 (d=4)	0,112970	0,159899
10vcorr2 (d=4)	0.009665	0.000000
25vconc1 (d=4)	0,213291	0,142495
25vcorr1 (d=4)	0,071930	0,995310
25vcorr2 (d=4)	0.059905	0.057574
50vconc1 (d=4)	0,169281	0,142616
50vcorr1 (d=4)	0,036041	3,156154
50vcorr2 (d=4)	0.198313	0.202730
100vconc1 (d=4)	-0.002650	0.027688
100vconc2 (d=4)	-0.018192	0.221779
100vcorr1 (d=4)	-0.069800	1.218300
200vcorr1 (d=4)	-0.175500	1.773000
300vcorr1 (d=4)	-0.231500	2.162700

A → PSO-AGMmcd ***B*** → AESSEA

Tabela 11: Avaliação dos algoritmos multicritério usando o indicador $I_{\varepsilon+}$

De acordo com a tabela 1, apresentada na seção 3.7.4.2, as instâncias 10vconc1, 10vcorr1, 10vcorr2, 25vconc1, 25vcorr1, 25vcorr2, 50vconc1, 50vcorr1 e 50vcorr2 apresentam resultados não conclusivos, pois $I_{\varepsilon+}(A, B) > 0$ e $I_{\varepsilon+}(B, A) > 0$, sendo considerados dois conjuntos incomparáveis ($A \parallel B$).

Contudo, ao aumentar o tamanho da instância, para 100, 200 e 300 vértices, pôde-se concluir que o algoritmo PSO-AGMmcd é melhor que o algoritmo AESSEA, pois $I_{\varepsilon+}(A, B) \leq 0$ e $I_{\varepsilon+}(B, A) > 0$. Pode-se notar também que, ao aumentar o tamanho de vértices, o afastamento de uma solução para outra também aumenta, sendo o desempenho do algoritmo PSO-AGMmcd ainda melhor.

Por fim, as tabela 12 e 13 mostram o tempo de execução e o número de soluções não-dominadas encontradas para cada instância em ambos os algoritmos para graus de restrição igual a 3 e 4, respectivamente. Para algumas instâncias, já citadas no início desta seção, o AESSEA não conseguiu executar, não tendo então os resultados relatados.

Instância	AESSEA		PSO-AGMmcd	
	Tempo (s)	Nº de Soluções	Tempo (s)	Nº de Soluções
10vconc1	4,40	1920	3,20	540
10vcorr1	4,44	2020	4,28	1102
10vcorr2	3,37	260	2,42	120
25vconc1	–	–	5,60	1256
25vcorr1	–	–	6,10	3752
25vcorr2	–	–	6,60	301
50vconc1	–	–	8,70	2822
50vcorr1	–	–	9,20	4804
50vcorr2	–	–	9,50	564
100vconc1	–	–	16,00	1895
100vconc2	–	–	16,10	1811
100vcorr1	–	–	17,40	872
100vcorr2	–	–	16,90	506
200vconc1	–	–	51,82	2309
200vconc2	–	–	43,92	2427
200vcorr1	–	–	52,02	1281
200vcorr2	–	–	44,12	920
300vconc1	–	–	120,25	2833
300vconc2	–	–	89,13	2760
300vcorr1	–	–	121,05	1526
300vcorr2	–	–	89,33	1101
400vconc1	–	–	191,08	3106
400vconc2	–	–	149,96	3005
400vcorr1	–	–	191,88	1815
400vcorr2	–	–	149,76	1191
500vconc1	–	–	288,53	3191
500vconc2	–	–	227,40	3253
500vcorr1	–	–	289,13	1934
500vcorr2	–	–	227,81	1448

Tabela 12: Tempo de execução e número de soluções encontradas (d = 3)

Instância	AESSEA		PSO-AGMmcd	
	Tempo (s)	Nº de Soluções	Tempo (s)	Nº de Soluções
10vconc1	3,64	1620	2,43	656
10vcorr1	4,72	2020	3,71	1243
10vcorr2	3,27	400	2,21	260
25vconc1	5,20	1240	4,49	1089
25vcorr1	8,64	2020	7,89	3948
25vcorr2	3,85	360	2,76	361
50vconc1	7,77	1700	7,11	3211
50vcorr1	11,37	2020	10,02	5029
50vcorr2	4,72	460	3,89	542
100vconc1	12,69	680	3,70	2114
100vconc2	8,35	640	3,10	1941
100vcorr1	8,72	460	3,43	988
100vcorr2	–	–	4,78	681
200vconc1	–	–	16,14	2584
200vconc2	–	–	13,51	2819
200vcorr1	44,42	230	19,85	1500
200vcorr2	–	–	13,91	971
300vconc1	–	–	87,05	3115
300vconc2	–	–	58,73	3029
300vcorr1	585,19	330	84,85	1709
300vcorr2	–	–	58,93	1299
400vconc1	–	–	160,68	3470
400vconc2	–	–	118,76	3379
400vcorr1	–	–	161,29	2004
400vcorr2	–	–	119,36	1402
500vconc1	–	–	256,33	3548
500vconc2	–	–	195,02	3676
500vcorr1	–	–	257,33	2151
500vcorr2	–	–	196,20	1694

Tabela 13: Tempo de execução e número de soluções encontradas (d = 4)

Capítulo 7

Trabalhos Futuros

Como trabalhos futuros, podem ser citados os seguintes tópicos:

1. Implementar a composição de velocidades para a estratégia adotada para o PSO.
2. Adequação do operador de *path-relinking* utilizado no algoritmo PSO-AGMmcd para ser usado no algoritmo PSO-LK.
3. Simetrização das instâncias assimétricas da TSPLIB, permitindo que se possa aplicar o algoritmo PSO-LK nelas.
4. Investigar outros operadores e outras estratégias para que se possa aumentar ainda mais a eficiência do algoritmo PSO-AGMmcd.
5. Desenvolver uma distância entre as soluções da AGM-mcd para trabalhar com vizinhança física no PSO-AGMmcd.

Capítulo 8

Considerações Finais

Este trabalho apresentou três algoritmos baseados na Otimização por Nuvem de Partículas (PSO). Dois aplicados ao Problema do Caixeiro Viajante (PCV) e um aplicado ao Problema da Árvore Geradora Mínima Restrita em Grau Multicritério (AGM-mc*d*).

Os dois algoritmos PSO aplicados ao PCV apresentaram o conceito de velocidade diferenciado. Quando a partícula segue seu próprio caminho, são aplicados procedimentos de busca local; e quando a partícula segue o caminho de alguma outra posição, o operador de *path-relinking* é aplicado.

Embora alguns trabalhos que descrevem algoritmos PSO aplicados ao PCV foram apresentados anteriormente, estes algoritmos não relataram resultados os quais pudessem ser comparados com resultados obtidos por heurísticas efetivas. Neste trabalho, uma abordagem efetiva baseada no PSO foi apresentada, aplicada ao PCV, onde os resultados foram comparados com três heurísticas de alta qualidade para este problema. A comparação dos resultados mostrou que o algoritmo proposto supera as três heurísticas com relação aos melhores resultados. A comparação entre os resultados médios do PSO-LK e do Tourmerge também mostra que o algoritmo proposto encontra melhores resultados.

O algoritmo PSO aplicado à AGM-mc*d* foi desenvolvido nos mesmos moldes dos algoritmos aplicados ao PCV: um procedimento de busca local para representar a velocidade da partícula quando ela segue seu próprio caminho; e operadores de *path-relinking* para representar a velocidade da partícula quando ela segue o caminho de alguma outra posição.

Este algoritmo mostrou-se superior a outro algoritmo que aborda o mesmo problema, o AESSEA, apresentado por Knowles e Corne [54, 57]. Para instâncias menores, os dois apresentam praticamente o mesmo desempenho. Contudo, ao aumentar o tamanho das instâncias, o algoritmo proposto mostrou-se mais eficiente que o AESSEA.

Referências Bibliográficas

[1] Aarts, E.; Lenstra, J. K. (1997). **Local search in combinatorial optimization**. John Wiley & Sons, Chichester, England.

[2] Arroyo, J. E. C. (2002). **Heurísticas e metaheurísticas para otimização combinatória multiobjetivo**. Tese de Doutorado, Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas, Campinas.

[3] Balas, E.; Toth, P. (1985). **Branch and bound methods**. In The Traveling Salesman Problem, Edited by Lawler, E. L.; Lenstra, J. K.; Rinnooy Kan, A. H. G. and Shmoys, D. B. John Wiley & Sons Ltd, 361-401.

[4] Barrico, Carlos M. C. S. (1998). **Uma abordagem ao problema de caminho mais curto multiobjectivo – Aplicação ao problema de encaminhamento em redes integradas de comunicações**. Dissertação de Mestrado, Departamento de Engenharia Electrotécnica, Faculdade de Ciências e Tecnologia, Universidade de Coimbra, Portugal.

[5] Bazlamaçci, Cüneyt F.; Hindi, Khalil S. (2001). **Minimum-weight spanning tree algorithms: A survey and empirical study**. Computers & Operations Research 28, 767-785.

[6] Bellmore, M.; Nemhauser, G. L. (1968). **The traveling salesman problem: a survey**. Operations Research 16, 538–558.

[7] Bodin, L. D.; Assad, B. L.; Ball, A. (1983). **Routing and scheduling of vehicles and crew, the state of the art**. Computers & Ops. Res. 10, 69-211.

[8] Boese, K.D.; Kahng, A.B.; Muddu, S. (1994). **A new adaptive multistart technique for combinatorial global optimization**. Operations Research Letters 16, 103-113.

[9] Bonomi, E.; Lutton, J. L. (1984). **The n-city traveling salesman problem: statistical mechanics and the metropolis algorithm**. SIAM Review 26, 551-568.

[10] Brady, R. M. (1985). **Optimization strategies gleaned from biological evolution**. Nature 317, 804-806

[11] Campelo, R. E.; Maculan, N. (1994). **Algoritmos e heurísticas: desenvolvimento e avaliação de performance**. Editora da Universidade Federal Fluminense: Niterói, RJ.

[12] Cerny, V. (1985). **Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm**. Journal of Optimization Theory and Applications 15, 41-51.

[13] Chauny, F.; Haurie, A.; Wagneur, E; Loulou, R. (1987). **Punch operations in a flexible manufacturing cell a three-dimensional space-filling curve approach**. INFOR 25 (1), 28-45.

[14] Christofides, N. (1979). **The traveling salesman problem**. In Combinatorial Optimization, Edited by Christofides, N.; Mingozzi, A.; Toth, P. and Sandi, C. John Wiley & Sons Inc., New York, 131-149.

[15] Clerc, Maurice (2000). **Discrete particle swarm optimization: illustrated by the traveling salesman problem**: http://clerc.maurice.free.fr/pso/pso_tsp/Discrete_PSO_TSP.htm. Último acesso em 16/11/2004.

[16] Coello, Carlos A. Coello (2001). **A short tutorial on evolutionary multiobjective optimization**. First International Conference on Evolutionary Multi-Criterion Optimization, 21-40.

[17] Coello, Carlos A. Coello (1999). **An updated survey of evolutionary multiobjective optimization techniques: state of the art and future trends**. Congress on Evolutionary Computation, 3-13.

[18] **Concorde TSP solver**: http://www.tsp.gatech.edu/concorde.html. Último acesso em 18/01/2005.

[19] Cook, W. J., Seymour, P. (2003). **Tour merging via branch-decomposition**. INFORMS Journal on Computing 15, 233-248.

[20] Czyzak, P.; Jaszkiewicz, A. (1998). **Pareto simulated annealing – a metaheuristic technique for multiple objective combinatorial optimization**. Journal of Multi-Criteria Decision Analysis 7, 34-47.

[21] Dantzig, G. B.; Fulkerson, D. R.; Jonson, S. M. (1954). **Solutions of a large scale traveling salesman problem**. Operations Research 12, 393-410.

[22] Daniels, R. L. (1992). **Analytical evaluation of multicriteria heuristics**. Management Science 38, 501-513.

[23] Dijkstra, E. W. (1959). **A note on two problems in connection with graphs**. Numerische Mathematik 1, 269-271.

[24] Dorigo, M.; Gambardella, L. M. (1997). **Ant colony system: a cooperative learning approach to the traveling salesman problem**. IEEE Transactions on Evolutionary Computation 1, n. 1, 53-66.

[25] Eberhart, R.; Shi, Y. (1998). **Comparison between genetic algorithms and particle swarm optimization**. Lecture Notes in Computer Science 1447, Proceedings of the 7th International Conference on Evolutionary Programming VII, 611-616.

[26] Ehrgott, Matthias; Gandibleux, Xavier (2000). **A survey and annotated bibliography of multiobjective combinatorial optimization**. OR Spektrum 22, 425-460.

[27] Esmin, A. A. A.; Aoki, A. R.; Lambert-Torres, G. (2002). **Particle swarm optimization for fuzzy membership functions optimization**. In: IEEE International Conference on Systems, Man and Cybernetics, Tunisia.

[28] Feo, T. A.; Resende, M. G. C. (1989). **A probabilistic heuristic for a computationally difficult set covering problem**. Operations Research Letters 8, 67–71.

[29] Finke, G.; Kusiak, A. (1985). **Network approach to modeling of flexible manufacturing modules and cells**. APII-0399-0516, Department of Applied Mathematics Technical University of Nova Scotia, Nova Escócia, Canadá.

[30] Finke, G.; Kusiak, A. (1987). **Models for the processing planning problem in flexible manufacturing systems**. IJAMT 2 (2), 3-12.

[31] Fiechter, C. N. (1994). **A parallel tabu search algorithm for large traveling salesman problems**. Discrete Applied Mathematics 51, 243-267.

[32] Garey, M; Johnson, D. S. (1979). **Computer and intractibility: a guide to the theory of NP Completeness**. Freeman, San Francisco.

[33] Gendreau, M.; Hertz, A.; Laporte, G. (1992). **New insertion and post optimization procedures for the traveling salesman problem**. Operations Research 40, 1086-1094.

[34] Glover, F. (1963). **Parametric combinations of local job shop rules**. Chapter IV, ONR Research Memorandum N. 117, GSIA, Carnegie Mellon University, Pittsburgh, PA.

[35] Glover, F.; Laguna, M.; Martí, R. (2000). **Fundamentals of scatter search and path relinking**. Control and Cybernetics 29 (3), 653-684.

[36] Glover, F. (1990). **Tabu search: a tutorial**. Interfaces, July-August 1990, 74-94.

[37] Glover, F. (1986). **Future paths for integer programming and links to artificial intelligence**. Computer & Operations Research 13, 533-549.

[38] Glover, F. (1996). **Ejection chains, reference structures and alternating path methods for traveling salesman problems**. Discrete Applied Mathematics 65, 223-253,

[39] Goldbarg, M. C.; Luna, H. P. L. (2000). **Otimização combinatória e programação linear: modelos e algoritmos**. Campus: Rio de Janeiro, RJ.

[40] Goldberg, D. E. (1989). **Genetic algorithms in search**. Optimization & Machine Learning. Addison Wesley, Reading.

[41] Hansen, M. P.; Jaszkiewicz, A. (1998). **Evaluating the quality of approximations to the non-dominated set**. Technical Report, Institute of Mathematical Modeling (1998-7), Technical University of Denmark.

[42] Heppner, F.; Grenander, U. (1990). **A Stochastic nonlinear model for coordinated bird flocks**. In S. Krasner, Ed., The Ubiquity of Caos. AAAS Publications, Washington, DC.

[43] Hagen, L.W.; Kahng, A.B. (1997). **Combining problem reduction and adaptive multistart: A new technique for superior iterative partitioning**. IEEE Transactions on CAD 16, 709-717.

[44] Holland, J. H. (1975). **Adaptation in natural and artificial systems**. University of Michigan Press, Ann Arbor, MI.

[45] Hu, X. (2003). **PSO Tutorial**: http://www.swarmintelligence.org/tutorials.php. Último acesso em 28/12/2005.

[46] Jerald, J.; Asokan, P.; Prabaharan, G.; Saravanan, R. (2004). **Scheduling optimization of flexible manufacturing systems using particle swarm optimization algorithm**. Advanced Manufacturing Technology. Springer-Verlag London Limited.

[47] Johnson, D. S.; McGeoh, L.A. (2002). **Experimental analysis of heuristics for the STSP**. In: Guttin, G.; Punnen, A. P. (eds.): Traveling Salesman Problem and Its Variations, Kluwer Academic.

[48] Johnson, D. S.; Aragon, C. R.; McGeoch, L. A.; Schevon, C. (1997). **Optimization by simulated annealing: an experimental evaluation; part III, the traveling salesman problem,** forthcoming.

[49] Johnson, D. S.; Bentley, J. L.; McGeoch, L. A.; Rothberg, E. E. (1997). **Near-optimal solutions to very large traveling salesman problems,** forthcoming.

[50] Karp, R. M. (1975). **On the computational complexity of combinatorial problems**. Networks 5, 45-68.

[51] Kennedy, J.; Eberhart, R. (1995). **Particle swarm optimization**. Neural Networks. Proceedings, IEEE International Conference on. Perth, WA, Australia, vol. 4, 1942-1948.

[52] Kennedy, J. (1999). **Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance**. Congress on Evolutionary Computation, Washington, DC, IEEE.

[53] Kirkpatrick, S.; Gelatt, C. D.; Vecchi, M. P. (1983). **Optimization by simulated annealing**. Science 220, 671-680.

[54] Knowles, J. D.; Corne, D. W. (2001). **A comparison of encodings and algorithms for multiobjective minimum spanning tree problems**. Congress on Evolutionary Computation, IEEE Press.

[55] Knowles, J. D.; Corne, D. W. (2000). **A new evolutionary approach to the degree-constrained minimum spanning tree problem**. IEEE Transactions on Evolutionary Computation 4 (2), 125–134.

[56] Knowles, J. D.; Corne, D. W. (2000). **Approximating the nondominated front using the Pareto archived evolution strategy**. Evolutionary Computation 8 (2), 149-172.

[57] Knowles, J. D.; Corne, D. W. (2001) **Benchmark problem generators and results for the multiobjective degree-constrained minimum spanning tree problem**. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, 424-431, Morgan Kaufmann Publishers.

[58] Knowles, J. D. (2002). **Local-search and hybrid evolutionary algorithms for Pareto optimization**. PhD Thesis, Department of Computer Science, University of Reading, Reading, United Kingdom.

[59] Kruskal Jr, J. B. (1956). **On the shortest spanning subtree of a graph and the traveling salesman problem**. Proc. ACM 7 (1), 48-50.

[60] Laporte, G; Asef-Vazir, A.; Sriskandarajah, C. (1996). **Some applications of the generalized traveling salesman problem**. JORS 47, 1461-1467.

[61] Lin, S.; Kernighan, B. (1973). **An effective heuristic algorithm for the traveling-salesman problem**. Operations Research 21, 498-516.

[62] Machado, T. R., Lopes, H. S. (2005). **A hybrid particle swarm optimization model for the traveling salesman problem**. In: Ribeiro, H., Albrecht, R. F., Dobnikar, A. (eds.): Natural Computing Algorithms, Wien: SpringerWienNewYork, 255-258.

[63] Misevičius, A. (2004). **Using iterated tabu search for the traveling salesman problem**. Information Technology And Control, Kaunas, Technologija 3(32), 29-40.

[64] Miyazawa, Flávio Keidi. **Otimização Combinatória**: http://www.ic.unicamp.br/~fkm/problemas.html. Último acesso em 26/10/2005.

[65] Moscato, P. (1989). **On evolution, search, optimization, genetic algorithms and martial arts: towards memetic algorithms**. Caltech Concurrent Computation Program, C3P Report 826.

[66] Mühlenbein, H.; Gorges-Schleuter, M.; Krämer, O. (1988). **Evolution algorithms in combinatorial optimization**. Parallel Computing 7, 65-85.

[67] Norback, J. P.; Love, R. F. (1979). **Heuristic for the hamiltonian path problem in euclidean two space**. Journal of the Operational Research Society 30, 363-368.

[68] Pang, W., Wang, K.-P., Zhou, C.-G., Dong, L.-J., Liu, M., Zhang, H.-Y., Wang, J.-Y. (2004). **Modified particle swarm optimization based on space transformation for solving traveling salesman problem**. Proceedings of the Third International Conference on Machine Learning and Cybernetics, 2342-2346.

[69] Pareto, Vilfredo (1896). **Cours D'Economie Politique**. volume I and II. F. Rouge, Lausanne.

[70] Parsopoulos, K. E.; VRAHATIS, M.N. (2002). **Recent approaches to global optimization problems through Particle Swarm Optimization**. Natural Computing 1, 235-306.

[71] Prim, R. C. (1957). **Shortest connection networks and some generalizations**. Bell System Technical Journal 36, 1389-1401.

[72] Prüfer, H. (1918). **Neuer beweis eines satzes über permutationen**. Arch. Math. Phys 27, 742-744.

[73] Raidl, G. R. (2000). **An efficient evolutionary algorithm for the degree-constrained minimum spanning tree problem**. In Proceedings of the 2000 Congress on Evolutionary Computation, Piscataway, NJ, 104-111. IEEE Press.

[74] Reinelt, G. (1995). **TSPLIB**: http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/. Último acesso em 18/01/2005.

[75] Reinelt, G. (1994). **The traveling salesman: computational solutions for the applications**. Lectures Notes in Computer Science 840, Spring-Verlag, Berlim.

[76] Reynolds, C. W. (1987). **Flocks, herds and schools: a distributed behavioral model**. Computer Graphics, 21(4), 24-34.

[77] Reynolds, R. G. (1994) **An introduction to cultural algorithms**. In: Proceedings of Evolutionary Programming, EP94, World Scientific, River Edge, NJ 131-139.

[78] Rocha, D. A. M; Goldbarg, E. F. G.; Goldbarg, M. C. (2006). **A memetic algorithm for the biobjective minimum spanning tree problem**. In: 6th European Conference on Evolutionary Computation in Combinatorial Optimization EvoCOP 2006, Budapest. Lecture Notes in Computer Science.

[79] Rosenberg, R. S. (1967). **Simulation of genetic populations with biochemical properties**. PhD Thesis, University of Michigan, Ann Harbor, Michigan.

[80] Rosenkrantz, D. J.; Stearns, R. E.; Lewis, P. M. (1974). **Approximate algorithms for the traveling salesman salesperson problem**. Fifteenth Annual IEEE Symposium on Switching and Automata Theory, 33-42

[81] Salomon, M; Solomon, M. M.; Van Wassenhove, L. N.; Dumas, Y.; Dauzère-Pérès, S. (1997). **Solving the discrete lotsizing and scheduling problem with sequence dependent set-up costs and set-up times using the traveling salesman problem with time windows**. EJOR 100, 494-513.

[82] Schaffer, J. David (1984). **Multiple Objective optimization with vector evaluated genetic algorithms**. PhD Thesis, Vanderbilt University.

[83] Secrest, B. R. (2001). **Traveling salesman problem for surveillance mission using particle swarm optimization**. AFIT/GCE/ENG/01M-03, Air Force Institute of Technology.

[84] Smith, T. H. C.; Thompson, G. L. A. (1975). **A LIFO implicit enumeration search algorithm for the symmetric traveling salesman problem using Held and Karp's 1-tree relaxation**. Research Report 356, GSIA – Carnegie-Mellon University. Pittsburgh, PA 15213.

[85] Srinivas, N.; Deb, K. (1995). **Multiobjective optimization using nondominated sorting in genetic algorithms**. Evolutionary Computation 2/3, 221-248.

[86] Syswerda, G. (1989). **Uniform crossover in genetic algorithms.** In: J. Schaffer (Ed.), Proceedings of Third International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, 2-9.

[87] Valenzuela, C. L.; Jones, A. J. (1994). **Evolutionary divide and conquer (I): a novel genetic approach to the TSP**. Evolutionary Computation 1, 313-333.

[88] Veldhuizen, D. A. V.; Lamont, G. B. (2000). **Multiobjective evolutionary algorithms: analyzing the state-of-art**. Evolutionary Computation 8(2), 125-147.

[89] Voudouris, C., Tsang, E. (1999). **Guide local search and its application to the traveling salesman problem**. European Journal of Operational Research 113: 469-499.

[90] Wang, Kang-Ping; Huang, Lan; Zhou, Chun-Guang; Pang, Wei (2003). **Particle swarm optimization for traveling salesman problem**. Proceedings of the Second International Conference on Machine Learning and Cybernetics, Xi'an. 1583-1585.

[91] Whitley, D.; Starkwhether, T.; Shaner, D. (1991). **The traveling salesman and sequence scheduling: quality solutions using genetic recombinations**. Handbook of Genetics Algorithms, Edt L. Davis Van Nostrand, 350-372

[92] Wilson, E. O. (1975). **Sociobiology: the new synthesis**. Belknap Press, Cambridge, MA.

[93] Yoshida H.; Kawata K.; Fukuyama Y. (2001). **A particle swarm optimization for reactive power and voltage control considering voltage security assessment**. IEEE Transactions on Power Systems 15, 1232-1239.

[94] Zhou, Genghi; Gen, Mitsuo (1999). **Genetic algorithm approach on multi-criteria minimum spanning tree problem**. European Journal of Operational Research 114, 141-152.

[95] Zitzler, E.; Deb, K.; Thiele, L. (2000). **Comparison of multiobjective evolutionary algorithms: empirical results**. Evolutionary Computation 8(2), 173-195.

[96] Zitzler, E. (1999). **Evolutionary algorithms for multiobjective optimization: methods and applications**. PhD Thesis, Swiss Federal Institute of Technology, Zurich, Switzerland.

[97] Zitzler, E. (2003). **Performance assessment on multiobjetive optimizers: an analysis and review**. IEEE Transactions on Evolutionary Computation 7(2), 117-132.