Estruturas de Dados Eficientes para Algoritmos Evolutivos Aplicados a Projeto de Redes

Telma Woerle de Lima

Estruturas de Dados Eficientes para Algoritmos Evolutivos Aplicados a Projeto de Redes

Telma Woerle de Lima
telma@icmc.usp.br

Orientador:
Prof. Dr. Alexandre Cláudio Botazzo Delbem
acbd@icmc.usp.br

Tese apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC, USP, como parte dos requisitos para a obtenção do título de Doutor em Ciências de Computação e Matemática Computacional.

USP - São Carlos
Abril de 2009

Agradecimentos

Agradeço a toda a minha família pelo carinho e apoio recebidos, em especial a minha mãe, meu pai, meu irmão e minha tia Nilce sem o qual o desenvolvimento deste trabalho não teria sido possível.

Ao meu esposo Anderson da Silva Soares, pelo apoio emocional e valiosa ajuda no desenvolvimento da tese.

Ao meu orientador Prof. Dr. Alexandre Cláudio Botazzo Delbem, pelo apoio e orientação. Após 5 anos de trabalhos conjuntos no mestrado e no doutorado, recebi muito de sua brilhante acuidade científica e do seu entusiasmo pelo trabalho de seus alunos.

Agradeço também a Sra. Aline Bellintani Calligaris Delbem, que sempre recebeu com carinho em sua casa, os alunos do Prof. Alexandre.

Ao Prof. Dr. Franz Rothlauf pela oportunidade de desenvolver parte deste trabalho na Universidade de Mainz na Alemanha.

Ao Prof. Dr. Fernando Federson e esposa pela amizade e apoio.

Aos Professores Cláudio Nogueira de Meneses e Humberto José Longo pelo carinho com que me receberam na Universidade Federal de Goiás.

Aos amigos de laboratório de computação reconfigurável, em especial aos amigos Paulo e Daniel que muito auxiliaram na etapa de conclusão dessa tese, e ao amigo Rodrigo Faccioli da EESC.

Este trabalho contou com o apoio financeiro da Fapesp, agradeço sua essencial contribuição.

LIMA, T. W. *Estruturas de Dados Eficientes para Algoritmos Evolutivos Aplicados ao Projeto de Redes*, São Carlos, 2009. (Tese de Doutoramento) - Instituto de Ciências Matemáticas e de Computação - ICMC, USP.

Resumo

Problemas de projeto de redes (PPRs) são muito importantes uma vez que envolvem uma série de aplicações em áreas da engenharia e ciências. Para solucionar as limitações de algoritmos convencionais para PPRs que envolvem redes complexas do mundo real (em geral modeladas por grafos completos ou mesmo esparsos de larga-escala), heurísticas, como os algoritmos evolutivos (EAs), têm sido investigadas. Trabalhos recentes têm mostrado que estruturas de dados adequadas podem melhorar significativamente o desempenho de EAs para PPRs. Uma dessas estruturas de dados é a representação nó-profundidade (NDE, do inglês *Node-depth Encoding*). Em geral, a aplicação de EAs com a NDE tem apresentado resultados relevantes para PPRs de larga-escala. Este trabalho investiga o desenvolvimento de uma nova representação, baseada na NDE, chamada representação nó-profundidade-grau (NDDE, do inglês *Node-depth-degree Encoding*). A NDDE é composta por melhorias nos operadores existentes da NDE e pelo desenvolvimento de novos operadores de reprodução possibilitando a recombinação de soluções. Nesse sentido, desenvolveu-se um operador de recombinação capaz de lidar com grafos não-completos e completos, chamado EHR (do inglês, *Evolutionary History Recombination Operator*). Foram também desenvolvidos operadores de recombinação que lidam somente com grafos completos, chamados de NOX e NPBX. Tais melhorias tem como objetivo manter relativamente baixa a complexidade computacional dos operadores para aumentar o desempenho de EAs para PPRs de larga-escala. A análise de propriedades de representações mostrou que a NDDE possui redundância, assim, foram propostos mecanismos para evitá-la. Essa análise mostrou também que o EHR possui baixa complexidade de tempo e não possui tendência, além de revelar que o NOX e o NPBX possuem uma tendência para árvores com topologia de estrela. A aplicação de EAs usando a NDDE para PPRs clássicos envolvendo grafos completos, tais como árvore geradora de comunicação ótima, árvore geradora mínima com restrição de grau e uma árvore máxima, mostrou que, quanto maior o tamanho das instâncias do PPR, melhor é o desempenho relativo da técnica em comparação com os resultados obtidos com outros EAs para PPRs da literatura. Além desses problemas, um EA utilizando a NDE com o operador EHR foi aplicado ao PPR do mundo real de reconfiguração de sistemas de distribuição de energia elétrica (envolvendo grafos esparsos). Os resultados mostram que o EHR possibilita reduzir significativamente o tempo de convergência do EA.

LIMA, T. W. *Efficient Data Structures to Evolutionary Algorithms Applied to Network Design Problems.*, Sao Carlos, 2009. (PhD Thesis) - Institute of Mathematics and Computer Sciences, University of Sao Paulo.

Abstract

Network design problems (NDPs) are very important since they involve several applications from areas of Engineering and Sciences. In order to solve the limitations of traditional algorithms for NDPs that involve real world complex networks (in general, modeled by large-scale complete or sparse graphs), heuristics, such as evolutionary algorithms (EAs), have been investigated. Recent researches have shown that appropriate data structures can improve EA performance when applied to NDPs. One of these data structures is the Node-depth Encoding (NDE). In general, the performance of EAs with NDE has presented relevant results for large-scale NDPs. This thesis investigates the development of a new representation, based on NDE, called Node-depth-degree Encoding (NDDE). The NDDE is composed for improvements of the NDE operators and the development of new reproduction operators that enable the recombination of solutions. In this way, we developed a recombination operator to work with both non-complete and complete graphs, called EHR (Evolutionary History Recombination Operator). We also developed two other operators to work only with complete graphs, named NOX and NPBX. These improvements have the advantage of retaining the computational complexity of the operators relatively low in order to improve the EA performance. The analysis of representation properties have shown that NDDE is a redundant representation and, for this reason, we proposed some strategies to avoid it. This analysis also showed that EHR has low running time and it does not have bias, moreover, it revealed that NOX and NPBX have bias to trees like stars. The application of an EA using the NDDE to classic NDPs, such as, optimal communication spanning tree, degree-constraint minimum spanning tree and one-max tree, showed that the larger the instance is, the better the performance will be in comparison whit other EAs applied to NDPs in the literatura. An EA using the NDE with EHR was applied to a real-world NDP of reconfiguration of energy distribution systems. The results showed that EHR significantly decrease the convergence time of the EA.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

ACO – *Ant Colony Optimization* (Otimização por colônia de formigas)
CX – *Cycle Crossover* (Crossover de ciclo)
CV – *Characteristic Vector* (Vetor de características)
dc-MSTP – *Degree-Constrained Minimum Spanning Tree Problem* (Problema de árvore geradora mínima com restrição de grau)
EA – *Evolutionary Algorithm* (Algoritmo evolutivo)
EDAs – *Estimation Distribution Algorithm* (Algoritmo de estimação de distribuição)
EHR – *Evolutionary History Recombination* (Operador de recombinação histórico de evolução)
EP – *Evolutionary Programming*(Programação evolutiva)
ES – *Evolutionary Strategy* (Estratégia evolutiva)
GA – *Genetic Algorithm* (Algoritmo genético)
GP – *Genetic Programming* (Programação Genética)
NPBX – *NDDE PBX* (Operador de recombinação baseado em posição da NDDE)
NOX – *NDDE OX* (Operador de recombinação de ordem da NDDE)
NDE – *Node-Depth Encoding* (Representação nó-profundidade)
NDDE – *Node-Depth-Degree Encoding* (Representação nó-profundidade-grau)
NetKeys – *Network Random Keys* (Chaves aleatórias para redes)
OCSTP – *Optimum communication spanning tree problem* (Problema de árvore geradora de comunicação ótima)
OMTP – *One-max tree problem* (Problema de uma árvore máxima)
OX – *Order Crossover* (Operador de recombinação de ordem)
OBX – *Order Based Crossover* (Operador de recombinação baseado em ordem)
PBX – *Position Based Crossover* (Operador de recombinação baseado em posição)
PMX – *Partial Mapped Crossover* (Operador de recombinação parcialmente mapeado)
PPRs – Problemas de Projeto de Redes
SDE – *Electrical Radial Distribution Systems* (Sistema de distribuição de energia elétrica)

Lista de Símbolos

a – Vértice adjacente a p ou r onde a subárvore podada será religada pelos operadores 1 e 2 da representação nó-profundidade.
A – Matriz de adjacências do grafo.
$\hat{A}$ – Conjunto de todas as arestas válidas para o algoritmo.
B – Orçamento da rede.
C – Critério de limiar da rede.
$C_{n,n}$ – Matriz de custo de um grafo.
C_{max} – Custo máximo das arestas.
$c_{i,j}$ ou $C_{i,j}$ – Custo associado entre dois vértices do grafo.
c – Função de custo $E \rightarrow \mathbb{R}^+$
$\hat{C}$ – Conjunto de vértices considerados pelo algoritmo.
d – Restrição de grau de uma árvore.
$d_{x,y}$ – Distância de Hamming entre as árvores T_x e T_y.
$d_{i,MST}$ – Distância de Hamming da árvore T_i em relação a árvore T_{MST}, que é a árvore geradora mínima do grafo.
$d_{i,STAR}$ – Distância de Hamming da árvore T_i em relação as árvores estrela do grafo.
$\overline{d_{mst-pop}}$ – Média de $d_{i,MST}$ das árvores de uma população de tamanho μ.
$\overline{d_{star-pop}}$ – Média de $d_{i,STAR}$ da árvore de uma população de tamanho μ.
D – Função de custo da árvore de comunicação.
de_i – Profundidade do vértice v_i.
$deg(v_i)$ – Grau do vértice v_i.
$depth_i$ – Profundidade da árvore T_i.
$\overline{depth_{pop}}$ – Média de $depth_i$ na população de tamanho μ.
$diam_i$ – Diâmetro da árvore T_i.
$\overline{diam_{pop}}$ – Média de $diam_i$ na população de tamanho μ.
E – Conjunto finito das arestas ou arcos do grafo.
E_F – Conjunto de arestas da árvore F.
E_H – Conjunto finito das arestas ou arcos do subgrafo H.
E_T – Conjunto de arestas da árvore T.
$E[L_T]$ – Custo ativo esperado.

F ou $F = (V_F, E_F)$ – Floresta composta pelos conjuntos V_F e E_F.
$\hat{F}$ – Conjunto das arestas disjuntas de duas árvores.
$f(G)$ – Soma dos caminhos do grafo G.
g – Tamanho de L_h.
G ou $G = (V, E)$ – Grafo formado pelos conjuntos V e E.
H ou $H = (V_H, E_H)$ – é um subgrafo de $G = (V, E)$ composto pelos conjuntos V_H e E_H.
i_a – Índice do vértice a.
i_l – Índice do último vértice pertencente a subárvore podada.
i_p – Índice do vértice p.
i_r – Índice do vértice r.
i_x – Índice do vértice que possui profundidade menor ou igual a profundidade de p.
k – Constante positiva.
l – Número mínimo de vértices folha.
$l^{opt}_{i,j}$, $l^{b}_{i,j}$ – Indica se a aresta conectando os vértices i e j está presente ($l^{opt}_{i,j} = 1$, $l^{b}_{i,j} = 1$) na rede denominada *opt* (b).
L – Função de custo de um projeto de redes.
$\hat{L}$ – Caminho entre dois vértices.
$L_T(S)$ – Custo das arestas em $T(S)$.
L_D – *Array* circular utilizado para determinar o vértice p para aplicação de CAO na representação nó-profundidade-grau.
L_h – Lista encadeada dos ancestrais de um indivíduo em π.
$L_G(p)$ – *Array* contendo os adjacentes $N_G(p)$.
L_O – *Array* circular contendo uma permutação de $1 \ldots deg(p)$.
L_T – *Array* binário de comprimento n.
m – Tamanho do grafo, que é o número de arestas do grafo.
map_{v_i} – Mapeamento do vértice v_i.
n – Ordem do grafo, que é o número de vértices do grafo.
N – Subconjunto de vértices de fornecimento ou partida.
$N_G(p)$ – Conjunto de vértices adjacentes ao vértice p em G.
$N_T(p)$ – Conjunto de vértices adjacentes ao vértice p que estão em T_{origem}.
p_i – Vértice da *string* de Prüfer.
POP – População de um algoritmo.
p – Raiz da subárvore a ser podada pelos operadores 1 e 2 da representação nó-profundidade.
P – Número de Prüfer composto por $(n-2)$ dígitos inteiros.
$\overline{P}$ – Conjunto de vértices de V não pertencentes a P ordenados de forma crescente.
$P_{s,d}(T)$ – Caminho entre os nós s e d em T.
$q_{i,j}$ – Largura de banda necessária entre i e j.
$Q_{n,n}$ – Matriz de demandas da rede de comunicação.
r – Nova raiz da subárvore podada pelo operador 2 da NDE.
r_i – Vértices no caminho entre os vértice p e r em 2.
root – é o vértice distinto da árvore ou floresta denominado raiz da árvore ou floresta.
R – Seqüência de chaves aleatórias.
R_i – Cada elemento da seqüência de chaves aleatórias R.

R^s – Permutação σR de R em que as chaves estão ordenadas de forma decrescente.
s – $O(|T_{origem}| + |T_{destino}|)$.
s_{op} – Complexidade de tempo para aplicar os operadores PAO e CAO.
s_{tripla} – Complexidade de tempo para determinar os vértices p, r e a.
$\overline{s}$ – Complexidade de qualquer s.
S – Subconjunto de vértices.
$succ(v_i)$ – Função que define o sucesso de um vértice v_i.
$Su(v_x)$ – Sucessores diretos do vértice v_x em $\hat{L}$.
t – Número de árvores que compõem uma floresta.
T ou $T = (V_T, E_T)$ – Árvore composta pelos conjuntos V_T e E_T.
T_x – Representação NDDE (NDE) de uma árvore.
U_k – Componente desconexa do grafo.
v_i – Vértice do grafo.
(v_i, v_j) ou e_i – Aresta ou arco do grafo.
$v_i \rightarrow v_j$ – Arco nos algoritmos da representação *Blob Code*.
V – Conjunto finito de vértices do grafo.
V_F – Conjunto finito de vértices da árvore F.
V_H – Conjunto finito de vértices do subgrafo H.
V_T – Conjunto finito de vértices da árvore T.
$(VA, \leq)$ – Conjunto de vértices parcialmente ordenado da representação precedentes diretos.
$w_{i,j}$ – Peso associado à aresta (v_i, v_j) da matriz de pesos.
W – Matriz de pesos das arestas de um grafo.
W' – Conjunto de pesos associados aos vértices de N.
λ – Quantidade de indivíduos novos gerados em cada iteração de um EA.
μ – Tamanho da população fixa de um EA.
$\phi_D : [2, n-1] \rightarrow [1, n]$ – Função de mapeamento dos vértices da *string* de Prüfer nos vértices da árvore.
Ω – Conjunto de todos os pares (i, j) de árvores na floresta F.
π – *Array* auxiliar da NDE (NDDE) que armazena o ancestral de um indivíduo.
Π_{v_x} – Matriz auxiliar da NDE (NDDE) que armazena para cada vértice v_x de um grafo as informações relativas a sua posição em cada floresta gerada, isto é, o índice de floresta, da árvore, da sua posição na NDE (NDDE).
σR – Permutação de R definida por $\sigma R = R_{\sigma(i)}$.
$\sigma R_i, R_{\sigma(i)}$ – Elementos de σR, que é uma permutação de R.

Sumário

Capítulo

1

Introdução

Problemas envolvendo projeto de redes estão presentes em diversas aplicações do mundo real. As pesquisas nessa área, em geral, envolvem objetivos como, maximização de fluxo ótimo, construção de circuitos, determinação de caminhos mínimos ou de árvore geradora de custo mínimo. Tais objetivos podem estar relacionados a diversas aplicações do mundo real como por exemplo, reconfiguração de sistemas de distribuição de energia elétrica (Delbem et al., 2005), roteamento de veículos (Toth e Vigo, 2001), projeto de redes de computadores (Pióro e Medhi, 2004; Cahn, 1998) e a re-construção de árvores filogenéticas (Current e Marsh, 1993).

Para algumas dessas aplicações existem algoritmos eficientes que produzem soluções exatas. Por exemplo, a árvore geradora de custo mínimo pode ser resolvida por meio dos algoritmos propostos por Prim e Kruskal (Gross e Yellen, 2004) em tempo $O(m \log n)$, onde m é o número de ligações e n o número de pontos da rede. Por outro lado, existem problemas como o caixeiro viajante (Gross e Yellen, 2004), não possuem algoritmos eficientes que os resolvam adequadamente para instâncias envolvendo grafos de alta dimensão (mais de 1 milhão de vértices). Vários desses problemas pertencem à classe NP-Difícil, assim, os algoritmos conhecidos para esses problemas necessitam de um tempo exponencial, o que torna tais algoritmos inadequados para vários problemas de projeto de redes (PPRs) (Raidl e Julstrom, 2001).

Assim, é fundamental o estudo e desenvolvimento de algoritmos mais eficientes para solução desses problemas. Atualmente são utilizadas algumas abordagens para PPRs: algoritmos convencionais de otimização, tais como *branch and cut* e *branch and bound*,

técnicas heurísticas e técnicas híbridas. Os algoritmos convencionais de otimização quando diretamente aplicados a PPRs de larga-escala não são em geral computacionalmente eficientes (Cahn, 1998). Para aplicá-los, nesse caso, é usual utilizar uma série de simplificações no problema, tais como redução do espaço de busca e relaxação de restrições.

Técnicas heurísticas têm sido aplicadas a fim de obter algoritmos computacionalmente mais eficientes para os PPRs. No entanto, mesmo com o uso dessas técnicas, não é possível garantir que a solução factível para esses problemas seja encontrada em um intervalo de tempo aceitável. Em alguns casos, devido à complexidade desses, são também realizadas relaxações do problema mesmo com o uso de heurísticas. Apesar de não serem a solução ideal para tais problemas, as técnicas baseadas em heurísticas têm apresentado resultados relevantes em relação a algoritmos convencionais de otimização, para os casos envolvendo grandes instâncias (Cahn, 1998).

Dentre as técnicas não-convencionais encontram-se os Algoritmos Evolutivos (EAs, do inglês *Evolutionary Algorithms*) (De Jong, 2006; Michalewicz e Fogel, 2004; Deb, 2001; Goldberg, 1989). Tais algoritmos simulam a evolução de uma população de indivíduos (potenciais soluções de um problema) mimetizando o processo de evolução natural (Darwin, 1859, 2004). Essa simulação corresponde a um processo iterativo de criação de conjuntos (populações) de soluções (indivíduos) de forma que, a cada iteração (geração), soluções melhores tendem a ser encontradas. Além disso, os EAs têm apresentado desempenho relevante para problemas complexos de otimização envolvendo características que dificultam o emprego de métodos clássicos de otimização (Deb, 2001; Rothlauf, 2006).

No entanto, a utilização de um EA padrão não é suficiente para garantir a eficiência em PPRs. O uso de representações convencionais para esses problemas diminui a eficiência do EA, sobretudo se o tamanho da rede for grande (nos problemas envolvendo grafos completos com milhares de vértices), em geral, produzindo soluções infactíveis (que representam grafos com ciclos ou grafos desconexos que não são soluções válidas). A produção de tais soluções consome parte do tempo de computação até que seja obtida uma solução factível, reduzindo drasticamente a eficiência do EA. Outro efeito crítico dependente da representação é a produção de redes muito diferentes de seus pais (redes usadas de base para construção de novas redes), aumentando significativamente o tempo de convergência dos EAs. Para esses casos, tem sido necessário encontrar uma representação (estrutura de dados dinâmica (Cormen et al., 2000)) mais adequada e operadores específicos para a representação (Rothlauf, 2006).

Nos últimos anos, várias pesquisas têm sido desenvolvidas com o objetivo de superar essas limitações por meio do desenvolvimento de representações e operadores específicos

para EAs aplicados aos PPRs (Paulden e Smith, 2006a; Soak et al., 2005; Julstrom, 2005; Schindler et al., 2002; Raidl e Julstrom, 2001; Gottlieb et al., 2001; Gaube e Rothlauf, 2001; Knowles e Corne, 2000; Gen e Cheng, 1997; Palmer e Kershenbaum, 1995; Kershenbaum, 1993). Essas representações têm apresentado avanços significativos para o aumento do desempenho dos EAs.

Uma das representações recentemente propostas com o objetivo de melhorar o desempenho dos EAs para os PPRs, é a representação nó-profundidade (NDE - do inglês *Node-Depth Encoding*) (Delbem et al., 2004; Delbem e de Carvalho, 2003). Essa estrutura de dados viabiliza a aplicação de EAs em PPRs modelados por grafos de diversos tipos como: grafos Euclidianos, de pesos aleatórios, e com pesos dependentes da própria topologia da rede. Uma das características dessa representação é a sua independência dos pesos do grafo para a codificação das soluções, tornando a mesma adequada para uma grande diversidade de problemas.

Outro diferencial importante é a facilidade para modelar tanto grafos completos quanto não-completos, sendo esses densos ou esparsos. Esse aspecto é na prática muito importante, uma vez que muitas das redes de larga-escala são esparsas, pois a interconexão completa dos elementos dessas redes é custosa ou fisicamente impossível de ser realizada. As aplicações de EAs com a NDE têm apresentado resultados relevantes para diferentes PPRs de larga-escala (dos Santos et al., 2008; de Lima et al., 2008; Libralao et al., 2005; Delbem et al., 2004).

Esses resultados motivaram a presente tese que busca aperfeiçoar a NDE e estendê-la para lidar com um universo maior de PPRs. Essa pesquisa foi organizada em três etapas descritas a seguir. A primeira etapa consiste da análise da NDE envolvendo propriedades importantes de uma representação, tais como: tendência, localidade e complexidade (Rothlauf, 2006; Raidl e Julstrom, 2001).

A segunda etapa investiga o aperfeiçoamento da representação por meio de melhorias nos operadores existentes, que podem ser classificados como operadores de mutação. Além disso, são desenvolvidos novos operadores de reprodução para essa representação, capazes de recombinar as soluções para obter novas soluções (operadores de recombinação). Com isso, busca-se o aumento significativo do desempenho de EAs. Deve-se observar que os operadores de reprodução propostos, exceto um, baseiam-se nos operadores de recombinação de permutações conhecidos na literatura. A adaptação dos operadores de permutação para PPRs ocorre naturalmente no caso de redes modeladas por grafos completos. Adicionalmente, propõem-se um novo operador de recombinação, o *Evolutionary History Recombination Operator* - EHR, que pode ser utilizado com grafos esparsos, densos ou completos

e que se baseia nos operadores de mutação existentes na NDE. O aperfeiçoamento dessa representação também envolve melhorias nos operadores de mutação que já existiam para a NDE, com o objetivo principal de reduzir a complexidade de tempo média deles.

O conjunto das melhorias resultaram no desenvolvimento de uma nova representação chamada de nó-profundidade-grau (NDDE, do inglês *Node-Depth-Degree Encoding*). Com essa nova representação consegue-se proporcionar um aumento no desempenho dos EAs para PPRs. Foi possível mostrar que os operadores de mutação dessa representação demandam tempo de execução médio $O(\sqrt{n})$, enquanto que as demais representações possuem operadores que são pelo menos $O(n)$, onde n é o número de vértices do grafo (rede). Um dos grandes desafios do projeto foi manter essa complexidade nos operadores de recombinação, objetivo atingido por meio do EHR.

A terceira etapa do trabalho refere-se à avaliação da NDDE. Primeiro são analisadas as propriedades de representações (tendência, localidade, redundância, e outras) da NDDE e dos novos operadores. Em seguida, são desenvolvidos EAs usando a NDDE para os problemas clássicos de projetos de redes de árvore geradora mínima com restrição de grau, árvore geradora de comunicação ótima e uma árvore máxima. Finalmente, um EA com a NDE e o operador EHR é aplicado ao PPR de reconfiguração de sistema de distribuição de energia elétrica de larga-escala.

Este texto está estruturado conforme segue. O Capítulo 2 caracteriza os PPRs considerados neste trabalho. O Capítulo 3 introduz os EAs. O Capítulo 4 revisa os principais tipos de representações para EAs aplicados a PPRs. O Capítulo 5 descreve o desenvolvimento da nova representação, a NDDE. O Capítulo 6 apresenta a análise de propriedades da NDDE e seus operadores. O Capítulo 7 apresenta os resultados e análises da aplicação da NDDE em PPRs clássicos e no problema de reconfiguração de sistemas de distribuição de energia elétrica. Finalmente, o Capítulo 8 apresenta as conclusões deste trabalho e propostas de trabalhos futuros.

Capítulo

2

Projeto de Redes

Problemas de projeto de redes (PPRs) são em geral representados por grafos não-orientados $G = (V, E)$ (ver Apêndice A), onde V é o conjunto de vértices do grafo (ou pontos da rede) e E o conjunto de arestas ou ligações da rede. Um conjunto de pesos W associados às arestas de E também é utilizado em vários desses problemas. Há PPRs em que se associa mais de um peso por aresta ou mesmo em que cada peso varia com a própria topologia da rede. Reconstrução de árvores filogenéticas (Current e Marsh, 1993) e reconfiguração de sistemas de distribuição de energia elétrica (dos Santos et al., 2008; Delbem, 2002) são exemplos desse último tipo de PPR.

A solução de um PPR consiste em encontrar o subgrafo G' de G, que respeite as restrições do problema e otimize um ou mais critérios usados para avaliar a rede. Por exemplo, são comuns as restrições em relação aos pesos das arestas, comprimento do diâmetro ou grau dos vértices e como exemplo de critério para avaliar a rede considere o problema de minimizar o comprimento dos caminhos entre todos os vértices.

O PPR básico é um problema $NP-Completo$ (Johnson et al., 1978), definido conforme segue. Dado um grafo $G = (V, E)$, uma função de pesos $L : E \rightarrow \mathbb{N}$, um orçamento B e um limitante C para o valor critério ($B, C \in N$), o PPR básico consiste em encontrar o subgrafo $H = (V, E_H)$ de G com pesos tais que $\sum_{(i,j)\in E_H} L(i, j) \leq B$ e o valor de critério $f(H) \leq C$, onde $f(H)$ é a soma dos comprimentos de caminhos mínimos entre todos os vértices em H.

No entanto, quando se trata de PPRs no mundo real, minimizar os pesos dos caminhos não é o único fator a ser considerado. Principalmente em telecomunicações, redes de com-

putadores e elétricas, pelo menos outras três características devem ser consideradas (Pióro e Medhi, 2004; Cahn, 1998):

1. O custo da rede, ou seja, o custo para construção ou de manutenção da rede em funcionamento;

2. O desempenho da rede em relação a quantidade de tráfego/fluxo e nível de carregamento de seus componentes;

3. A confiança da rede, isto é, o nível de integridade ou qualidade de um percurso na rede.

Dentre os principais problemas conhecidos e estudados da área de projeto de redes podem ser destacados: árvore geradora de comunicação ótima (Soak, 2006; Rothlauf et al., 2003; Li e Bouchebaba, 2000; Palmer, 1994), reconfiguração de sistema de distribuição de energia elétrica (dos Santos et al., 2008; Delbem et al., 2005; Carvalho et al., 2001), árvore geradora mínima probabilística (Abuali et al., 1994), árvore geradora mínima com restrição de grau (Raidl e Julstrom, 2003), árvore geradora mínima com restrição de diâmetro (de Lima et al., 2008; Singh e Gupta, 2007; Gruber et al., 2006; Julstrom, 2004b; Abdalla e Deo, 2002), problemas envolvendo redes de telecomunicações (Soak et al., 2005; Pióro e Medhi, 2004; Abuali et al., 1995) e roteamento de veículos (Libralao et al., 2005; Voss et al., 1999).

Os algoritmos clássicos de árvore geradora mínima de Prim e Kruskal têm sido utilizados como base para construção de soluções para PPRs básicos. No entanto, esses algoritmos quando aplicados para redes de larga-escala podem obter alguns caminhos muito longos que não são adequados para vários PPRs, pois tais caminhos podem ocasionar problemas de confiança na rede ou aumentar consideravelmente o custo da mesma. Além disso, esses algoritmos dificultam o tratamento de restrições de PPRs (Cahn, 1998). Buscando soluções para PPRs mais complexos, técnicas baseadas em heurísticas têm sido investigadas, bem como a reformulação dos PPRs por meio de relaxações de restrições de forma a viabilizar a obtenção de soluções factíveis em tempo adequado.

Este Capítulo está organizado como segue. A Seção 2.1 apresenta os PPRs envolvendo a construção de uma única árvore geradora. A Seção 2.2 descreve O PPR de florestas geradoras compostas por mais de uma árvore. A Seção 2.3 sintetiza os aspectos principais apresentados neste Capítulo.

2.1 PPRs de Floresta com Uma Única Árvore

Os PPRs que possuem a sua solução representada por uma floresta composta por uma única árvore, em geral, têm um único ponto de partida ou de fornecimento para a rede. Portanto, correspondem a formulação clássica de um PPR. As Seções seguintes descrevem os PPRs desse tipo, que são utilizados em aplicações dos algoritmos desenvolvidos neste trabalho e são utilizados com o objetivo de avaliar o desempenho de representações para PPRs em EAs:

1. Árvore geradora de comunicação ótima (Seção 2.1.1);
2. Árvore geradora mínima com restrição de grau (Seção 2.1.2);
3. Problema de árvore máxima na Seção 2.1.3.

2.1.1 Árvore Geradora de Comunicação Ótima

O problema de árvore geradora de comunicação ótima (OCSTP, do inglês *Optimal Communication Spanning Tree Problem*) consiste em encontrar uma árvore geradora que satisfaz as necessidades de comunicação entre cidades com um custo total mínimo. São fornecidos como dados do problema: o número de vértices (cidades) do grafo, a posição dos mesmos no espaço ou o custo de ligação entre os vértices e as demandas de comunicação. O custo total de comunicação é a soma do produto da demanda de comunicação entre todos os pares pelo custo da ligação entre eles, estejam os vértices diretamente ligados ou não. O OCSTP é um problema mais complexo que o PPR básico e do que outros problemas de árvore geradora com restrição, sendo classificado como NP-Difícil (Soak, 2006; Rothlauf et al., 2003).

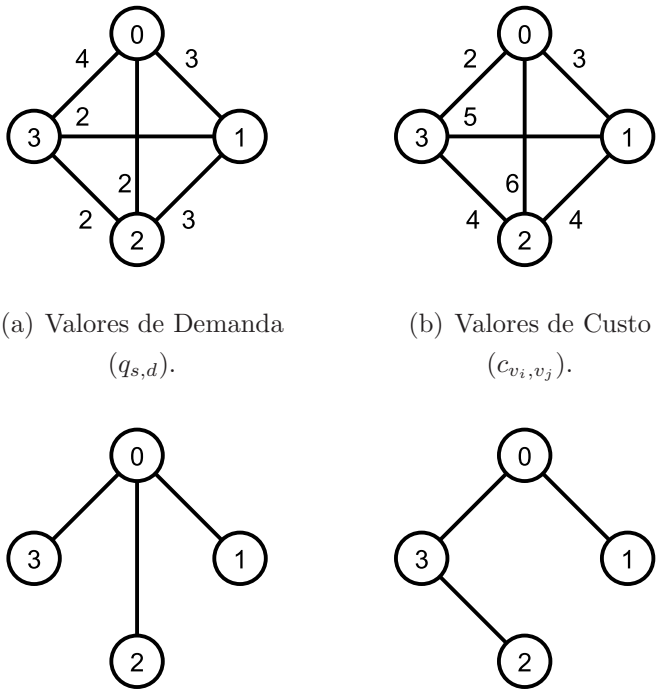
A definição formal do OCSTP é dada a seguir. Dado um conjunto de n cidades, representado por um grafo não-orientado $G = (V, E)$, onde $|V| = n$ e $|E| = m = \frac{n*(n-1)}{2}$. O custo por unidade de ligação de comunicação entre cada par de cidades é representado pela matriz quadrada $C_{n,n}$, onde cada elemento $c_{i,j} > 0$ da matriz é o custo da ligação entre as cidades de índice i e j. O conjunto de demandas é também uma matriz $Q_{n,n}$ onde seus elementos $q_{i,j}$ representam a largura de banda necessária entre todos os pares de cidades. Deve-se encontrar uma árvore geradora $T = (V, E_T)$, com $|E_T| = n - 1$ que conecte essas cidades e atenda a demanda de tráfego de forma mais econômica, ou seja, $D(T) \leq D(T')$, para toda e qualquer árvore geradora T' de G, onde D é a função de custo total da árvore (Palmer, 1994) que é calculado pela Equação 2.1:

$$D(T) = \sum_{s,d \in T} q_{s,d} p_{s,d} \tag{2.1}$$

onde T é uma árvore geradora, $q_{s,d}$ é a demanda entre os vértices s e d de T e

$$p_{s,d} = \sum_{(v_i,v_j) \in P_{s,d}(T)} c_{v_i,v_j},$$

onde c_{v_i,v_j} é um elemento da matriz $C_{n,n}$, e $P_{s,d}(T)$ é o caminho único entre os vértices s, d em T formado pelas arestas (v_i, v_j). Assim, o custo do caminho $P_{s,d}(T)$ é dado por $p_{s,d}$ (Soak, 2006; Soak et al., 2005; Palmer, 1994).



(a) Valores de Demanda ($q_{s,d}$).

(b) Valores de Custo (c_{v_i,v_j}).

(c) Solução 1.

(d) Solução 2.

Figura 2.1: Dados para exemplo de um OCSTP de 4 vértices (Palmer, 1994).

A Figura 2.1 mostra o conjunto de dados de um exemplo do OCSTP. A Figura 2.1(b) apresenta um grafo com 4 vértices e os custos $c_{i,j}$. Os valores de demanda Q são apresentados na Figura 2.1(a). Uma possível solução (árvore geradora) é proposta para esse caso na Figura 2.1(c), com valor de custo:

$$D = p_{0,1} * q_{0,1} + p_{0,2} * q_{0,2} + p_{0,3} * q_{0,3} + p_{1,2} * q_{1,2} + p_{1,3} * q_{1,3} + p_{2,3} * q_{2,3}$$

$$D = 3 * 3 + 6 * 2 + 2 * 4 + 9 * 3 + 5 * 2 + 8 * 2 = 72$$

Outra possível solução é dada na Figura 2.1(d) com valor de custo:

$$D = p_{0,1} * q_{0,1} + p_{0,2} * q_{0,2} + p_{0,3} * q_{0,3} + p_{1,2} * q_{1,2} + p_{1,3} * q_{1,3} + p_{2,3} * q_{2,3}$$

$$D = 3 * 3 + 6 * 2 + 2 * 4 + 9 * 3 + 5 * 2 + 4 * 2 = 65$$

Portanto, a segunda solução apresentada possui menor custo sendo melhor que a primeira. A solução ótima para o problema pode ser obtida construindo todas as árvores geradoras do grafo e escolhendo a de menor custo.

Como a enumeração de todas essas árvores é impraticável para grafos não pequenos (com uma ou mais centenas de vértices), várias heurísticas têm sido aplicadas para resolver o OCSTP, além do uso de propostas de relaxamento de restrições do problema (Cahn, 1998). Dentre essas técnicas, EAs para o OCSTP têm sido desenvolvidos e aplicados a grafos com até uma centena de vértices obtendo resultados relevantes (Soak, 2006; Soak et al., 2005; Rothlauf et al., 2003; Li e Bouchebaba, 2000; Palmer, 1994). Essas pesquisas também mostram a partir da solução ótima encontrada para algumas instâncias do OCSTP, que essas soluções possuem uma tendência para a árvore geradora mínima do grafo (ver Apêndice A) (Rothlauf et al., 2003). Assim, métodos que possuam uma tendência em gerar mais soluções próximas a árvore geradora mínima devem conseguir melhores resultados para esse problema.

2.1.2 Árvore Geradora Mínima com Restrição de Grau

O problema de árvore geradora mínima com restrição de grau (dc-MSTP, do inglês *degree-constrained Minimum Spanning Tree*) é uma extensão do MSTP, para a qual o número de arestas incidentes a um vértice (grau) não deve ultrapassar um limitante superior (Knowles e Corne, 2000). A adição da restrição de grau torna o problema de árvore geradora mínima NP-Difícil (Ausiello et al., 2003).

Formalmente o dc-MSTP pode ser definido como segue. Dado $G = (V, E)$ um grafo não-orientado, seja W uma matriz de pesos, com $w_{i,j}$ o peso do par de vértices v_i, v_j, $deg(v_i)$ o grau do vértice v_i e d a restrição de grau do problema, com $d > 2$. Seja T uma árvore geradora de G. Então, o dc-MSTP pode ser escrito da seguinte forma:

$$min_T \sum_{(i,j)\in T} w_{v_i,v_j} \tag{2.2}$$

sujeito a restrição

$$deg(v_i) \leq d, \text{para todo } v_i \in T.$$

Várias pesquisas utilizando AEs para o dc-MSTP têm sido realizadas nos últimos anos encontrando soluções melhores que algoritmos baseados em heurísticas. Além disso, o dc-MSTP tem sido utilizado para avaliar o desempenho de representações para AEs aplicados a PPRs (Zhou et al., 15-19 Dec. 2007; Han et al., 28-30 May 2005; Delbem et al., 2004; Raidl e Julstrom, 2003; Gottlieb et al., 2001; Krishnamoorthy et al., 2001; Knowles e Corne, 2000; Zhou e Gen, 1997).

2.1.3 Uma Árvore Máxima

O OMTP (do inglês, *One-Max Tree Problem*) foi proposto por Rothlauf et al. (2002) como um teste padrão para avaliar o desempenho de algoritmos de otimização envolvendo o projeto topológico de redes.

Nesse problema, uma árvore geradora T_{opt} de um grafo $G = (V, E)$ é escolhida aleatoriamente como sendo uma árvore ótima. Então, para avaliação de uma nova árvore T_b, utiliza-se a distância $d_{opt,b}$ entre as duas árvores T_{opt} e T_b, que é definida como

$$d_{opt,b} = \frac{1}{2} \sum_{i,j \in V, i<j} |l_{i,j}^{opt} - l_{i,j}^{b}|,$$

onde, $l_{ij}^{opt}(l_{ij}^{b})$ é 1 se a aresta entre os vértices de índice i e j está em $T_{opt}(T_b)$ e 0, caso contrário. A qualidade de uma solução (aptidão de um indivíduo) T_b é definida como a distância $d_{opt,b}$ para a solução ótima T_{opt}.

Rothlauf (2006) utiliza o OMTP para analizar as propriedades de representações computacionais de soluções para AEs, avaliando tanto o desempenho de EAs utilizando a representação quanto a redundância, a localidade e a escalabilidade das representações. Outras pesquisas que utilizam AEs têm investigado o OMTP buscando avaliar as vantagens de diferentes representações (Schindler et al., 2002; Julstrom, 2001).

2.2 PPRs de Floresta com Mais de uma Árvore

Nos PPRs apresentados na Seção 2.1 as soluções são modeladas por uma única árvore de grafo. No entanto, a solução de alguns PPRs corresponde a uma floresta com mais de uma árvore, como, por exemplo, reconfiguração de sistema de distribuição de energia

elétrica (Delbem et al., 2005, 2003; Carvalho et al., 2001), o roteamento de múltiplos veículos para múltiplos destinos (Libralao et al., 2005) e as redes de computadores ou de comunicação com múltiplos centros (Cahn, 1998).

Um PPR básico envolvendo florestas com mais de uma árvore pode ser formulado conforme descrito a seguir. Dado o conjunto de vértices centrais $B = (B_0, \ldots, B_l)$ de um grafo $G = (V, E)$, que são os pontos de fornecimento da rede ou raízes de árvores; o conjunto $N = (n_1, \ldots, n_k)$ com os demais vértices de V; o conjunto de pesos $W = (W_i)$, com cada vértice i pertencente a N; a matriz de custos $w_{i,j}$ entre cada aresta (i, j) de E e W_{max} a capacidade total de fornecimento do sistema. Um PPR básico envolvendo florestas consiste em encontrar o conjunto de árvores $T_1, \ldots, T_l$, que compõem a floresta geradora do grafo G, $F = (V_F, E_F)$, de tal forma que cada vértice central pertença a uma árvore, $\sum_{n_i \in T_s} W_i \leq W_{max}$, e $\sum_{T_s \in F} \sum_{(i,j) \epsilon E_{T_s}} w_{i,j}$ seja mínimo (Cahn, 1998).

Em outras palavras, esse problema consiste em encontrar a floresta geradora F composta por l árvores, onde cada árvore contém um vértice central, a demanda dos demais vértices da árvore não deve ultrapassar a capacidade de fornecimento do sistema e a soma dos custos das ligações de todas as árvores deve ser mínimo.

Um dos importantes problemas desse tipo é a reconfiguração de sistemas de distribuição de energia devido ao seu alto impacto sócio-econômico. Uma reconfiguração apropriada desses sistemas pode minimizar as perdas de potência resultando em economias expressivas, por empresas concessionárias, bem como aumentando a capacidade energética do país. A reconfiguração pode também aumentar a qualidade da energia fornecida ao consumidor.

Um sistema de distribuição de energia elétrica (SDE) é composto por um conjunto de alimentadores agrupados em subestações que fornecem energia para um circuito elétrico. Esse circuito possui chaves (dispositivos que possibilitam conectar e desconectar trechos do circuito), trechos contíguos do circuito sem chaves são chamados de setores. Assim, um alimentador de um SDE pode ser representado por meio de um grafo $G = (V, E)$, onde V corresponde ao conjunto de setores do sistema e E corresponde às chaves que interligam o sistema. Cada setor por sua vez pode ser representado também por um grafo em que cada vértice é um ponto de carga ou de ramificação da rede. Os grafos dos setores são utilizados somente para avaliar a reconfiguração encontrada. Os circuitos dos alimentadores e os circuitos dos setores em geral correspondem a árvores. Assim, cada alimentador pode ser modelado por uma árvore de um grafo e o SDE como uma floresta. A Figura 2.2 ilustra um SDE e a floresta de grafo correspondente.

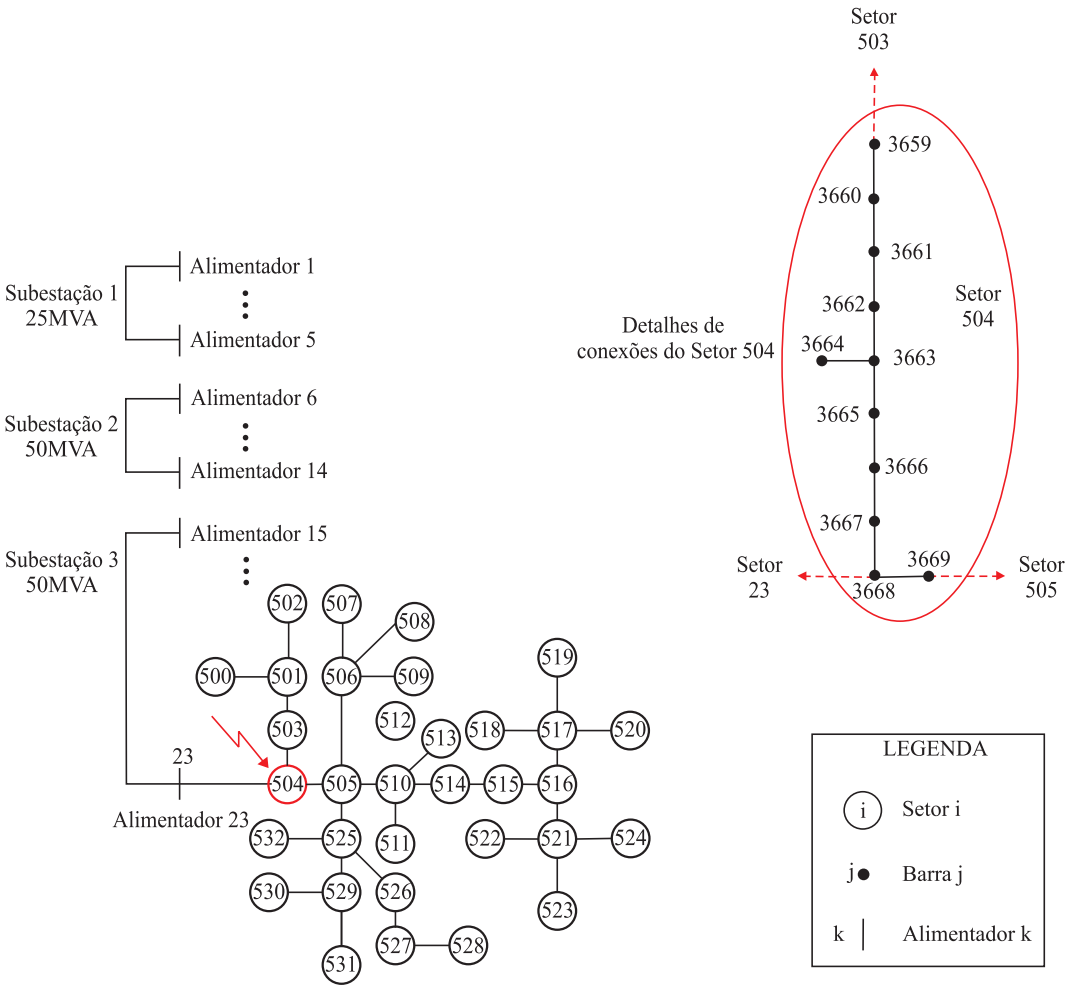


Figura 2.2: Exemplo de um SDE (dos Santos, 2009).

O processo de reconfiguração de um SDE pode ser necessário ou quando ocorre uma falta no SDE (um equipamento é danificado, por exemplo, devido a ventos fortes em uma tempestade, deixando parte de um alimentador desenergizado), ou, simplesmente, buscando melhorar características elétricas do SDE.

Ao reconfigurar o sistema, algumas restrições precisam ser satisfeitas, tais como: o total de carregamento dos alimentadores em um subestação não deve exceder a capacidade limite de cada subestação, a corrente elétrica em cada ramo não deve ultrapassar a capacidade do ramo e a queda de tensão em qualquer barra do SDE também não deve exceder um certo limite.

Dois objetivos podem ser considerados principais para esse problema: minimizar o total de perdas de potência e minimizar o número de chaveamentos, isto é, aberturas e fechamentos de chaves para alterar a configuração do SDE. O último objetivo torna-se

mais importante em caso de faltas gerando apagões, uma vez que nesses casos é preciso reconfigurar o sistema rapidamente para que a energia seja prontamente restabelecida, porém, quanto maior o número de manobras maior é o tempo para implementar a nova configuração.

Diversas técnicas têm sido propostas para solucionar esse problema. As técnicas empregando EAs têm obtido resultados relevantes para a reconfiguração dos sistemas de forma eficiente (Prasad et al., 2008; dos Santos et al., 2008; Delbem et al., 2005, 2003; Carvalho et al., 2001).

2.3 Considerações Finais

Este Capítulo abordou os PPRs que constituem em encontrar um subgrafo de um grafo que obedeça a um conjunto de restrições. Os principais PPRs considerados neste trabalho são árvore geradora de comunicação ótima (Palmer, 1994), reconfiguração de sistema de distribuição de energia elétrica (Delbem et al., 2003; Carvalho et al., 2001), árvore geradora mínima com restrição de grau (Raidl e Julstrom, 2003) e uma árvore máxima (Schindler et al., 2002).

Devido a complexidade computacional desses problemas, diversas técnicas baseadas em heurísticas têm sido propostas para esses problemas, dentre essas destacam-se os EAs apresentados no Capítulo 3.

Capítulo

3

Algoritmos Evolutivos

Algoritmos Evolutivos (EAs, do inglês *Evolutionary Algorithm*) são algoritmos de busca probabilística de soluções baseados no processo de evolução biológica. A Computação Evolutiva é a área de pesquisa que investiga o desenvolvimento desses algoritmos. Várias outras áreas de pesquisa, tais como, ciências naturais, engenharia, biologia e ciência da computação têm empregado amplamente os EAs. As ciências naturais e a biologia utilizam os EAs como ferramentas para simular e estudar sistemas evolutivos naturais. Por outro lado, a ciência da computação e a engenharia aplicam EAs como ferramentas de busca de soluções eficientes para problemas complexos.

Os princípios de EAs tiveram seu início na década de 1930, quando os sistemas evolutivos naturais começaram a ser investigados como algoritmos de exploração de múltiplos picos de uma função objetivo. Na década de 1960, começaram realmente a ser desenvolvidos os primeiros EAs, com o aumento da disponibilidade de computadores digitais. Na década de 1970, foram realizados diversos estudos empíricos e teóricos que resultaram na definição de três linhas de EAs: Programação Evolutiva (EP), Estratégias Evolutivas (ES) e Algoritmos Genéticos (GA) (De Jong, 2006). A década de 1980 caracterizou-se pelo aumento do número de aplicações e aperfeiçoamento dos EAs, destacando-se os desenvolvimentos que resultaram no EA chamado Programação Genética (GP) (Koza, 1992), capaz de elaborar programas de computadores. Desde então, a utilização de EAs tem crescido em diversas áreas da Engenharia e ciências. Nos últimos anos as pesquisas em computação evolutiva tem contribuído para tornar essa área bem consolidada.

Os EAs caracterizam-se pelo uso de conceitos da teoria de evolução e seleção nat-

ural das espécies (Ridley, 2001). Em todos os EAs existe um conjunto de indivíduos (população) representando soluções de um determinado problema. A mudança de características dos indivíduos, ao longo de iterações produz novas soluções. Utilizando um critério de seleção, essas soluções podem evoluir, isto é, caminharem em direção a melhores soluções para o problema. Outro fator comum aos EAs é a pressão de seleção (que simula a atuação do ambiente nos indivíduos) enfatizando os casos de sucesso. Essa ênfase pode ocorrer pela intensificação de comportamentos dos indivíduos de maior sucesso ou pela reprodução (cópia) desses indivíduos (como em GA, ES, EP e GP).

Os indivíduos de um EA precisam ser modelados computacionalmente. Nos EAs a forma mais simples de representação de soluções é um *array* binário, que faz analogia ao cromossomo biológico e, por isso, também é chamado de cromossomo. Assim como na biologia o cromossomo é dito ser o genótipo da solução e a solução no seu domínio real é chamada de fenótipo. Cada posição do cromossomo representa um gene, isto é, uma característica da solução do problema. Cada valor que pode ser assumido pela característica do problema representada pelo gene recebe o nome de alelo, assim como ocorre na Genética. De forma similar ao processo reprodutivo que ocorre na natureza, os novos indivíduos nos EAs podem ser obtidos por meio de operadores de reprodução aplicados aos cromossomos.

O processo de seleção natural pode ser simulado nos EAs por uma função (chamada função de aptidão, avaliação ou *fitness*) que atribui um valor a cada indivíduo. Esse valor, em geral, indica a qualidade da solução que o indivíduo representa. Quão melhor for o valor da aptidão, maior será a chance do indivíduo ser selecionado para reprodução, isto é, transferir suas características para novos indivíduos. O valor de aptidão também é utilizado para determinar quais os indivíduos que irão permanecer (sobreviver) na próxima geração.

A principal aplicação dos EAs têm sido a busca de soluções para problemas de otimização global (De Jong, 2006; Deb, 2001). Esses problemas caracterizam-se por funções objetivo simples ou múltiplas que podem ser multimodais ou mesmo descontínuas, são tratadas como caixa-pretas, que retornam para cada argumento (entrada da caixa) uma resposta (saída da caixa). Assim, o algoritmo de otimização deve encontrar um ótimo global sem informações a priori sobre propriedades da(s) função(ões). Vale destacar que uma das principais diferenças entre EAs e outras técnicas de otimização é que os EAs trabalham com um conjunto de soluções, avaliando simultaneamente vários pontos do espaço de soluções potenciais para um problema. No entanto, os EAs também podem ser utilizados em outras classes de problemas, tais como, sistemas classificadores, ou projeto

de objetos complexos, como por exemplo, redes neurais, definição de conjuntos de regras para executar tarefas e elaboração de códigos Lisp, entre outras (De Jong, 1975).

A Seção 3.1 apresenta as características dos EAs clássicos (também chamados de canônicos), e também os EAs de última geração. A Seção 3.2 apresenta os principais mecanismos de seleção utilizados nos EAs. A Seção 3.3 discute os mecanismos de reprodução. A Seção 3.4 apresenta as considerações finais deste Capítulo.

3.1 Principais Algoritmos Evolutivos

Os primeiros tipos de EAs (EP, ES e GA) são chamados de EAs canônicos (Fogel, 2006). Esses algoritmos possuem um mecanismo de funcionamento similar. A partir de um conjunto de soluções, aplicam-se mecanismos de seleção e reprodução com o objetivo de aprimorar as melhores soluções já encontradas. As Seções 3.1.1, 3.1.2, 3.1.3 apresentam cada um desses algoritmos com as suas particularidades. Na definição dos algoritmos, μ é o tamanho da população e λ é o número de descendentes gerados.

3.1.1 Programação Evolutiva

A programação evolutiva (EP, do inglês *Evolutionary Programming*) foi proposta por Fogel et al. (1966) com o objetivo de utilizar os conceitos de evolução natural para gerar iterativamente soluções apropriadas tanto em ambientes estáticos quanto em ambientes que são alterados dinamicamente (Fogel, 1962). Outro objetivo do desenvolvimento de EP era introduzir os conceitos evolutivos no desenvolvimento da Inteligência Artificial (Fogel, 1962).

Em EP, cada indivíduo da população representa uma máquina de estados finitos, que processa uma seqüência de símbolos. No processo de avaliação, os indivíduos são analisados por uma função de *payoff*. Essa função compara a saída da máquina e a saída esperada para solução do problema. O processo de reprodução dos indivíduos é realizado por meio de operadores de mutação e todos os indivíduos da população atual geram um novo descendente. Na etapa de seleção dos indivíduos para a próxima geração, a população atual e a população de descendentes competem diretamente. O critério de seleção preserva os indivíduos de maior *payoff* para a próxima geração (Fogel, 2006).

A EP pode ser definida como um sistema de reprodução (μ,λ) com $\lambda = \mu$, ou seja, todos os indivíduos da população atual são copiados e modificados pelo operador de mutação. Uma vez que somente os indivíduos de maior *payoff* sobrevivem para a próxima geração, o

processo de seleção é fortemente elitista[1]. Uma maior pressão de seleção na EP diminui o tempo de convergência do algoritmo, mas diminui a diversidade dos indivíduos, o que pode provocar a convergência prematura, prendendo o algoritmo em um ótimo local (De Jong, 2006).

3.1.2 Estratégias Evolutivas

Estratégias evolutivas (ESs, do inglês *Evolutionary Strategy*) foram desenvolvidas com o objetivo de solucionar problemas técnicos de otimização em engenharia e problemas contínuos de otimização paramétrica. A primeira formulação de ES foi proposta por Rechenberg (1965) e Schwefel (1965) e ficou conhecida como $(1 + 1)$-ES. Nessa proposta, um pai gera um novo indivíduo e ambos competem pela sobrevivência, isto é, $\lambda = \mu = 1$.

Nas ESs, cada gene no cromossomo corresponde a uma dimensão do problema no espaço dos números reais com representação de ponto flutuante. Um gene é composto por dois valores, um para a posição na dimensão do problema e outro para o desvio padrão desse valor. A obtenção de um novo indivíduo ocorre pela aplicação de um operador de mutação Gaussiana utilizando média zero e o desvio padrão associado ao gene do pai. Assim como na EP, as ESs selecionam somente os indivíduos de melhor *fitness* para a próxima geração.

No primeiro modelo de ES, denominado $(1 + 1)$-ES, um pai gerava um único novo indivíduo e ambos competiam diretamente pela sobrevivência. Porém, esse modelo possui convergência lenta e, na maioria das vezes, para um ótimo local. Com objetivo de solucionar esse problema, outros dois modelos de ES foram desenvolvidos. O primeiro modelo usa a formulação (μ, λ)-ES, pela qual μ pais morrem e sobrevivem λ filhos. O segundo modelo utiliza a formulação ES-$(\mu + \lambda)$, onde sobrevivem somente os μ melhores indivíduos entre os μ pais e os λ filhos. Em ambos os modelos, λ pode ser maior que μ, ou seja, um pai pode gerar mais de um descendente (Bäck et al., 2000).

Uma contribuição importante de ES é a evolução de um conjunto de desvios padrões associados a cada gene dos indivíduos (Fogel, 2006).

3.1.3 Algoritmos Genéticos

Os Algoritmos Genéticos (GAs, do inglês *Genetic Algorithms*) foram propostos por Holland (1975) e tornaram-se os EAs mais conhecidos. Holland estudou a evolução natural

[1] O processo de seleção é dito elitista quando a cópia dos k melhores ($k \geq 1$) cromossomos de cada geração é copiado para a geração seguinte.

como um processo robusto, simples e poderoso, que poderia ser adaptada para solucionar problemas de otimização encontrando soluções adequadas. A robustez de um GA indica se soluções adequadas são encontradas independente da configuração dos parâmetros iniciais (Goldberg, 1989).

Em relação aos outros EAs, pode-se destacar quatro diferenças dos GAs em sua formulação padrão:

1. GAs utilizam a representação binária para codificar os cromossomos;

2. O valor da função de aptidão é utilizado para selecionar os indivíduos da reprodução, não somente os indivíduos que sobrevivem;

3. Além da mutação, os GAs realizam combinação de informações de dois pais para obter novos indivíduos, esse processo é chamado de recombinação de soluções;

4. Os GAs selecionam para a próxima geração μ indivíduos do total de $(\mu + \lambda)$ indivíduos.

A seleção para reprodução com relação à função de aptidão, em geral, baseia-se na proporção do *fitness* de cada indivíduo em relação ao *fitness* total (soma dos *fitness* de todos os indivíduos da população). Os mecanismos de seleção são discutidos mais detalhadamente na Seção 3.2.

No entanto, o principal diferencial dos GAs é a utilização da recombinação. Essa operação consiste em trocar partes do cromossomo entre os pais. Nos GAs, a operação de recombinação é denominada crossover. A operação de mutação, quando é utilizada a representação binária, consiste em alterar um gene pelo seu complemento.

A utilização combinada das operações de recombinação e mutação equilibra dois objetivos aparentemente conflitantes: o reaproveitamento de informações relevantes contidas nas melhores soluções da população atual e a exploração de novas características ainda não presentes na população.

Dependendo de escolhas específicas de tipo de seleção, número de descendentes gerados, do operador de reprodução e da representação dos indivíduos, pode-se classificar um EA como um dos três algoritmos apresentados. Com base nos EAs apresentados nesta seção, são definidos os principais componentes de um EA. O Algoritmo 3.1.1 apresenta os principais passos de um EA genérico.

Algoritmo 3.1.1: Pseudo-código de um EA típico.

```
//Seja POP(t) a população de indivíduos no tempo t
//Seja POP' a população de descendentes de POP(t)
(1) t := 0
(2) INICIA_POPULACAO(POP(t))
(3) ENQUANTO critério não atingido FAÇA
    (4) POP' := SELECIONA_PAIS(POP(t))
    (5) REPRODUZ(POP')
    (6) AVALIA(POP')
    (7) POP(t+1) := SOBREVIVENTES(POP(t),P')
    (8) t := t+1
FIM
```

3.1.4 EAs de Última Geração

Pesquisas dos últimos dez anos, explorando os aspectos de modelagem estatística e probabilística das populações têm produzido EAs com capacidade de busca maior para problemas de larga-escala e significativamente mais complexos do que a dos EAs da primeira geração. Dentre esses destacam-se CGA (Goldberg, 2002a), ECGA (Goldberg, 2002a), BOA, hBOA (Pelikan et al., 2000), UMDA (Mhlenbein e Paab, 1996) e RSA (Larraenaga e Lozano, 2001).

Esses recentes tipos de EAs têm sido chamados de algoritmos de estimação de distribuição (EDAs, do inglês *Estimation of Distribution Algorithm*). Alguns desses, como o ECGA e o hBOA, podem ser classificados como GA competentes, isto é, que possuem garantias de convergência rápida para a solução ótima em problemas complexos de larga escala (ver (Goldberg, 2002b)). Não foram encontrados registros da aplicação de EAs competentes em PPRs. Apesar desse não ser o foco desta tese, que investiga vários aspectos de representações que estão definidos para EAs da primeira geração, o estudo sobre EDAs, em conjunto com novas representações para PPRs é um tema interessante para trabalhos futuros.

3.2 Mecanismos de Seleção

A seguir são descritos com mais detalhes os principais métodos de seleção de indivíduos utilizados para reprodução ou para escolher os que sobrevivem para a próxima geração. Os principais métodos utilizados, são (De Jong, 2006):

1. Seleção uniforme ou neutra: todos os indivíduos têm igual probabilidade de serem selecionados independente do seu valor de *fitness*. Em geral, esse método é utilizado para selecionar os indivíduos que irão gerar descendentes;

2. Proporcional ao *fitness*: para cada indivíduo é atribuída uma probabilidade $\bar{p}_i$ proporcional ao seu *fitness* definida por f_i/f_{sum}, onde f_i é o *fitness* do indivíduo i e f_{sum} é a soma do *fitness* de todos os indivíduos. Os indivíduos são selecionados de acordo com a probabilidade $\bar{p}_i$ que é alterada dinamicamente. No início do processo evolutivo esse método de seleção é mais elitista e, no final do processo, comporta-se como uma distribuição uniforme, pois os indivíduos tendem a ter valores de *fitness* próximos. Pode ser utilizado para selecionar os indivíduos para reproduzir ou sobreviver;

3. Classificação linear e torneio de dois: dois indivíduos selecionados aleatoriamente competem entre si em um torneio; o vencedor do torneio é o indivíduo de melhor *fitness*. São realizados λ torneios para selecionar λ indivíduos. Pode ser utilizado na seleção dos indivíduos para reprodução ou sobrevivência. O torneio de dois tem o mesmo comportamento do processo de seleção dos indivíduos por meio de uma ordenação decrescente dos indivíduos (classificação linear). De Jong (2006) demonstra matematicamente essa propriedade;

4. Classificação não-linear e torneio de $k > 2$: similar a seleção por torneio de dois, mas com a seleção de $k > 2$ indivíduos que competem entre si; vence o indivíduo de melhor *fitness*. Quanto maior for k, maior será a pressão seletiva, conseqüentemente mais elitista será o processo. Ao contrário do torneio de dois, esse método de seleção tem o comportamento de uma classificação não-linear. De Jong (2006) também prova matematicamente essa característica do torneio de $k > 2$;

5. Truncamento: seleciona somente os μ indivíduos de melhor *fitness*, os demais são descartados. Esse método é o de maior pressão seletiva nos indivíduos, em geral é utilizado para definir os indivíduos que sobreviverão para a próxima geração. Uma combinação bastante comum é utilizar seleção uniforme para selecionar os indivíduos para reprodução e seleção por truncamento para determinar os indivíduos que sobreviverão.

3.3 Mecanismos de Reprodução

Os métodos de reprodução podem simular processos assexuados ou sexuados. No primeiro, um único pai é necessário, assim utilizam-se somente operadores de mutação. No segundo, mais de um pai é necessário, então empregam-se operadores de recombinação. A seguir serão descritos esses dois mecanismos de reprodução e suas formas de aplicação.

3.3.1 Mutação

A mutação é o mecanismo de reprodução no qual um indivíduo reproduz, pela cópia de seu cromossomo, gerando filhos que sofrem posteriormente uma variação de um ou mais genes. A probabilidade de um gene ter o seu valor alterado é chamada de taxa de mutação. Uma vez que essa operação pode gerar indivíduos potencialmente piores do que o seu ancestral, em geral utiliza-se por uma taxa de mutação pequena.

A escolha do operador de mutação depende, principalmente, do tipo de representação utilizada no cromossomo. Para os tipos mais comuns de representação, já existe uma série de operadores de mutação definidos na literatura (De Jong, 2006; Fogel, 2006; Michalewicz e Fogel, 2004).

Nos cromossomos com representação binária, a operação de mutação consiste em determinar as posições que serão alteradas por meio da seleção aleatória com distribuição uniforme dada a taxa de mutação. O novo indivíduo é construído copiando os genes das posições não selecionadas e alterando os genes nas posições selecionadas com o complemento do valor do gene atual (Bäck et al., 2000).

Para EAs que utilizam a representação de ponto flutuante, existe uma grande variedade de métodos para criar novos indivíduos pela modificação de um pai. Nesses casos, em geral, os operadores de mutação alteram todo o cromossomo adicionando uma valor aleatório v obtido a partir de uma distribuição probabilística (Bäck et al., 2000). As diferenças entre esses operadores de mutação ocorrem na escolha da distribuição probabilística utilizada, dentre as quais se destacam a distribuição uniforme com intervalo fixo definido e a distribuição Gaussiana com variância fixa para todo o cromossomo ou variável para cada gene. Além dessas, são utilizadas a distribuição de Cauchy, a distribuição de Laplace e a distribuição de Lèvi.

Uma outra forma de representação utilizada em problemas específicos são as permutações. Em permutações, o operador de mutação deve garantir que o novo indivíduo obtido também represente uma permutação válida. Uma primeira classe de operadores de mutação para permutações escolhe dois ou três pontos de corte no cromossomo e tenta reverter os trechos entre esses pontos. Esses operadores são chamados de 2-opt e 3-opt. Existe também a possibilidade de escolher k pontos, nesse caso, o operador é chamado de k-opt. Porém, esse operador tem alto custo computacional. Mesmo os operadores 2-opt e 3-opt possuem um custo computacional alto devido à necessidade de verificar em todos os trechos quais genes podem ter suas posições modificadas.

Esses operadores têm sido aplicados para o problema do caixeiro viajante em que cada permutação representa um circuito Hamiltoniano. Para problemas em que a permutação

representa uma lista de prioridades, os operadores do tipo k-opt provocam uma grande modificação no cromossomo, não sendo, portanto, indicados para esse tipo de problema. Para esse caso, foram desenvolvidos operadores que trocam duas ou mais posições escolhidas de forma aleatória.

Outros tipos de problemas com representações específicas, tais como máquinas de estados finitos, árvores de análise sintática (*parse*) e muitas outras aplicações do mundo real, necessitam de operadores de mutação especiais que podem ser encontrados na literatura (Bäck et al., 2000).

Nos operadores de mutação, a quantidade de variação, ou seja, o quão diferente o descendente será do pai, pode ser controlada por dois parâmetros: (1) número de genes que poderão ser modificados e (2) tamanho do passo da alteração que será aplicada ao valor do gene. A escolha de qual dos parâmetros alterar para obter uma maior ou menor variação depende em grande parte do problema. Em geral, nos problemas com representação de ponto flutuante, alteraram-se todos os genes do cromossomo e controla-se o tamanho do passo da mutação. Por outro lado, na representação binária, uma vez que a mutação consiste no complemento do gene, deve-se controlar o número de genes alterados para se obter uma maior ou menor exploração do espaço de busca (De Jong, 2006).

3.3.2 Recombinação

A recombinação é o mecanismo de reprodução em que os novos indivíduos são obtidos pela troca ou combinação de genes de dois ou mais indivíduos da população. Em EAs com cromossomos de comprimento fixo, a recombinação é tradicionalmente chamada de crossover[2], pois seleciona-se aleatoriamente um ou mais pontos, que divide os cromossomos pais em trechos que são trocados entre eles para compor os novos indivíduos.

O primeiro tipo de recombinação para cromossomos de comprimento fixo é o crossover de 1-ponto. Este consiste em definir um ponto de corte e trocar os trechos separados por esse ponto. Por exemplo, considere um cromossomo contendo valores de quatro variáveis inteiras (ou genes) de um problema, isto é, $\boxed{x_1}\boxed{x_2}\boxed{x_3}\boxed{x_4}$, onde um conjunto de valores para as variáveis x_1, x_2, x_3 e x_4 define uma solução para o problema. Considere os seguintes indivíduos (cromossomos): $\boxed{12}\boxed{34}\boxed{08}\boxed{71}$ e $\boxed{05}\boxed{17}\boxed{29}\boxed{66}$.

Aplicando-se o operador de 1-ponto a esses indivíduos, com o ponto de corte entre o segundo e o terceiro genes, obtém-se os seguintes descendentes: $\boxed{12}\boxed{34}\boxed{29}\boxed{66}$ e $\boxed{05}\boxed{17}\boxed{08}\boxed{71}$.

Existem outros tipos de crossover de pontos para cromossomos de comprimento fixo:

[2]Crossover é um tipo de recombinação De Jong (2006).

1. O crossover de dois pontos, no qual são escolhidos dois pontos de corte e troca-se o trecho entre eles para compor os novos indivíduos;

2. O crossover de multi-pontos ou uniforme, no qual tradicionalmente define-se uma máscara aleatória que determina de qual dos pais será copiado o gene para compor o novo indivíduo.

Além desses operadores de crossover de ponto foram definidos outros operadores de recombinação para problemas que utilizam uma representação específica.

Em EAs que utilizam a representação de ponto flutuante, os principais operadores de recombinação sugeridos são a média aritmética dos genes dos pais, média ponderada dos genes dos pais, recombinação geométrica e recombinação uniforme. Na média ponderada, cada pai contribui proporcionalmente para a formação do filho e, nesse caso, mais de dois pais podem ser utilizados na operação de recombinação. Na recombinação geométrica, o valor do gene do filho é obtido pela multiplicação dos valores dos genes dos pais e, depois, elevados a uma potência. Por fim, na recombinação uniforme os novos genes são obtidos por meio de uma máscara aleatória que define de qual pai será copiado o gene (Bäck et al., 2000).

Nos problemas representados por permutações, a operação de recombinação deve resultar em uma permutação válida. Com esse objetivo, foram desenvolvidos alguns operadores de recombinação:

- Crossover de Ordem (OX, do inglês *Order Crossover*): seleciona dois pontos de corte que definem uma região denominada região de corte. Essa região é copiada para o filho. No preenchimento das demais posições considera preferencialmente a ordem dos genes no cromossomo;

- Crossover Baseado em Posição (PBX, do inglês *Position Based Crossover*): seleciona um conjunto aleatório de posições a serem copiadas diretamente dos pais no filhos. Considera, preferencialmente a posição dos genes para construir os novos descendentes;

- Crossover Baseado em Ordem (OBX, do inglês *Order Based Crossover*): similar ao PBX, entretanto, considera preferencialmente a ordem dos genes. Também seleciona um conjunto de posições porém para compor O_1 recupera os genes em P_2 e os insere nessas posições do filho só que na ordem de P_1;

- Crossover de Ciclo (CX, do inglês *Cycle Crossover*): utiliza a estratégia de busca por um ciclo nos genes dos pais. A busca pelo ciclo inicia na primeira posição de um

pai (P_1) e prossegue com a busca pela posição do gene que está em P_2 na posição atual de P_1. Esse processo busca evitar a repetição de genes;

- Crossover Parcialmente Mapeado (PMX, do inglês *Partially Mapped Crossover*): similar ao operador OX, porém cria-se um mapeamento dos genes da região de corte de um pai, com os genes da região de corte do outro pai. Ao completar as posições fora da região de corte, se elas contém um gene da região de corte utiliza o mapeamento para selecionar o gene correto;

A seguir, os crossovers baseados em posição e de ordem serão descritos em um nível maior de detalhamento, pois serão utilizados para definir operadores de recombinação propostos nessa tese.

Nos exemplos de aplicação de cada um dos operadores de recombinação, são utilizadas as seguintes permutações como cromossomos pais:

$P_1 =$ | C | E | D | B | F | A |,

$P_2 =$ | E | B | A | C | D | F |.

Crossover de Ordem

No crossover de ordem, os indivíduos filhos (O_1 e O_2) são obtidos da seguinte forma (Michalewicz, 1996; Starkweather et al., 1991):

1. Selecione dois pontos de corte, os genes entre esses pontos definem uma região de corte;

2. Copie a região de corte de P_1 (P_2) para O_1 (O_2);

3. Marque em P_2 (P_1) os genes da região de corte de P_1 (P_2);

4. Considere o cromossomo como *arrays* circulares (ver Figura 3.1). Preencha as posições em O_1 (O_2) a partir do segundo ponto de corte com os genes de P_2 (P_1) não marcados a partir do segundo ponto de corte.

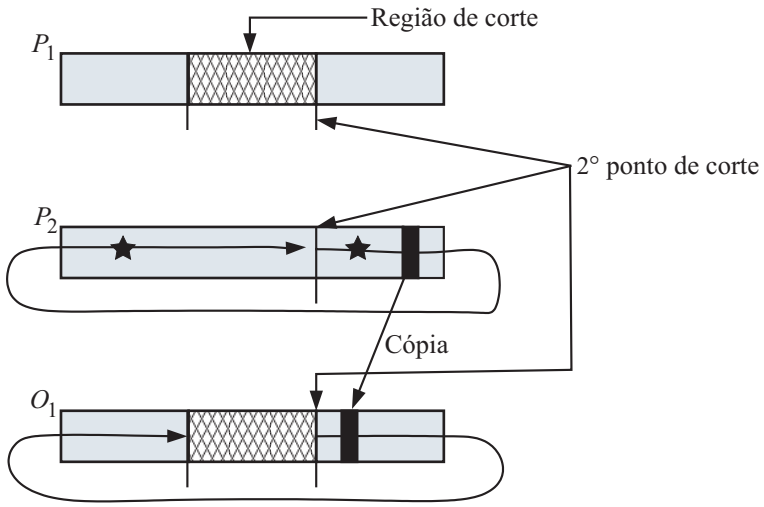


Figura 3.1: Processo de cópia usando *arrays* circulares para gerar O_1.

Por exemplo, considere P_1 e P_2 como sendo os cromossomos definidos no início desta seção. No Passo 1, definem-se os pontos de corte entre as posições (2 e 3) e (4 e 5). No Passo 2, os filhos são formados somente pelas regiões de corte entre os pontos de corte: $O_1 =$ X X D B X X e $O_2 =$ X X A C X X, onde X indica posições não definidas. No Passo 3, os genes D e B são marcados (com ★) no P_2 (posições 5 e 2 do P_2, respectivamente) e os genes A e C no P_1 (posições 6 e 1 do P_1, respectivamente), resultando nos seguintes cromossomos marcados:

$P_1 =$ ★ E D B F ★

$P_2 =$ E ★ A C ★ F.

No Passo 4, a primeira posição de P_2, a partir do segundo ponto de corte, foi marcada; assim, a segunda posição (com o gene F) é copiada para O_1 resultando em X X D B F X. Em seguida, copia-se o gene E de P_2 ($O_1 =$ X X D B F E) e pula-se a segunda posição de P_2 (marcada), copiando então o gene A de P_2 ($O_1 =$ A X D B F E). Por fim, copia-se o gene C de P_2 produzindo $O_1 =$ A C D B F E. De forma similar, é obtém-se $O_2 =$ D B A C F E.

Esse processo de marcação dos genes para depois ignorá-los no processo de cópia tem como objetivo evitar que os valores dos genes sejam repetidos, pois no caso de permutações, o gene em cada posição deve ser único no cromossomo.

Crossover Baseado em Posição (PBX)

Este operador considera a posição dos genes para construir os novos descendentes. As etapas para aplicação desse operador são (Michalewicz e Fogel, 2004; Starkweather et al.,

1991):

1. Defina um conjunto de posições aleatórias;
2. Copie os genes dessas posições de $P_1(P_2)$ nas mesmas posições no filho;
3. Remova de $P_2(P_1)$ os genes que possuem os mesmos valores dos genes nas posições selecionadas em $P_1(P_2)$;
4. Preencha as posições vazias do filho da esquerda para a direita com os valores que restaram de $P_2(P_1)$ ainda não utilizado, também da esquerda para a direita.

Por exemplo, considere P_1 e P_2 definidos anteriormente. No Passo 1, definem-se as posições 2, 4 e 5 escolhidas de forma aleatória. No Passo 2, para o primeiro filho são copiadas as posições do P_1 e obtém-se $O_1 =$ | X | E | X | B | F | X | e, para o segundo filho, as posições são copiadas do P_2: $O_2 =$ | X | B | X | C | D | X |.

A seguir, as posições vazias são preenchidas pelos genes do outro pai ignorando os genes já inseridos no filho e, assim, obtém-se: $O_1 =$ | A | E | C | B | F | D | e $O_2 =$ | E | B | F | C | D | A |.

3.4 Considerações Finais

Neste Capítulo foram introduzidos os EAs, que são ferramentas de busca que têm sido amplamente aplicadas principalmente a problemas de otimização complexos. Os EAs têm como base o processo de evolução dos sistemas naturais.

Foram apresentados os tipos de EAs mais comuns: EP, ES e AG. Foram definidos os principais mecanismos de seleção e reprodução nos EAs. Por fim, foram descritos os operadores de recombinação baseados em permutações, base dos operadores propostos neste trabalho para a NDDE (ver Capítulo 5).

O Capítulo 4 apresenta as principais representações para projeto de redes utilizadas em EAs.

Capítulo

4

Representações de Algoritmos Evolutivos para Projeto de Redes

Representações são importantes para o desempenho dos Algoritmos Evolutivos (EAs). A escolha de uma representação inapropriada para um problema pode afetar o tempo de convergência dos EAs. Dependendo da representação utilizada, os EAs podem ter comportamento similar ao de métodos de busca aleatória (Rothlauf, 2006).

Problemas de projetos de redes (PPRs) são representados, em geral, por árvores geradoras de um grafo. Uma representação para essa classe de problemas deve incorporar as restrições necessárias para um grafo corresponder a uma árvore. Ao utilizar uma *string* como genótipo (codificação da solução de um problema no EA) para a representação dos PPRs, tem-se uma grande distância entre a estrutura da árvore no fenótipo (solução em seu domínio real) e o genótipo utilizado. Dessa forma, representações convencionais em EAs para PPRs não tem produzido resultados relevantes (Rothlauf, 2006). Nesse contexto, diversas pesquisas têm investigado representações mais adequadas para os EAs aplicados a PPRs e operadores de reprodução para modificar árvores geradoras produzindo novas árvores geradoras.

As representações de PPRs para EAs podem ser classificadas em diretas, indiretas ou mistas. As do tipo direta codificam uma árvore por meio de seu conjunto de arestas e aplicam os operadores de reprodução diretamente ao conjunto de arestas. Conseqüentemente, os processos de codificação e decodificação não são necessários. Por outro lado, essas representações precisam de operadores de busca específicos que possam ser aplicados

diretamente ao conjunto de arestas.

As representações indiretas geralmente codificam uma árvore (fenótipo) como uma lista de *strings* (genótipo) e aplicam os operadores de reprodução clássicos no genótipo, ou seja, utilizam crossover de pontos e mutação de perturbação dos genes (Tzschoppe et al., 2004; Rothlauf e Tzschoppe, 2005).

As representações mistas possuem características tanto de representações diretas quanto de indiretas. Em geral, essas representações utilizam uma função de codificação do fenótipo em um genótipo, mas utilizam operadores de busca específicos em alternativa aos operadores de busca clássicos.

Os primeiros critérios para a avaliação de uma representação de árvores para EAs foram propostos por Palmer (1994). Posteriormente, Raidl e Julstrom (2003) complementaram e melhor definiram um conjunto de critérios que avaliam se a representação é adequada ou não para problemas representados por árvores. Esses critérios são descritos a seguir:

1. Espaço: as representações não devem utilizar quantidades excessivas de memória. O espaço necessário para codificar uma árvore pela representação deve ser o mínimo necessário;

2. Tempo: a complexidade de tempo para avaliar, recombinar e mutar as soluções deve ser baixa. Em PPRs o processo de avaliação pode incluir a decodificação da representação, assim, a decodificação também deve ser eficiente;

3. Factibilidade: todas as representações, principalmente as obtidas por recombinação e mutação, devem representar soluções factíveis do problema. Em outras palavras, a representação deve ser capaz de codificar somente árvores. Nos casos em que a representação não (de)codifica exclusivamente árvores, métodos alternativos para trabalhar com essas soluções infactíveis devem ser especificados;

4. Cobertura: a representação deve ser capaz de codificar todo o espaço de busca, ou seja, capaz de representar todas as árvores possíveis;

5. Tendência: todas as soluções devem ser igualmente prováveis de serem representadas. Uma representação que possui uma tendência maior em codificar certos tipos de solução só é favorável se essas soluções estão próximas de uma solução ótima;

6. Localidade: pequenas alterações na representação devem produzir pequenas alterações na árvore representada. O desejável é uma representação com alta localidade. A baixa localidade pode direcionar o EA a um processo de busca aleatória;

7. Hereditariedade: as operações de recombinação devem preservar a maioria das arestas dos pais nos filhos. Considere E_{T_1} e E_{T_2} os conjuntos de arestas dos pais e $E_{T_1'}$ e $E_{T_2'}$ os conjuntos de arestas dos filhos, o ideal é que $E_{T_1} \cup E_{T_2}$ fosse igual $E_{T_1'} \cup E_{T_2'}$, ou seja, todas as arestas dos filhos estivessem presentes nos pais;

8. Restrições: devem possibilitar a inclusão de restrições dos problemas com facilidade, sem a necessidade de grandes modificações;

9. Hibridismo: os operadores da representação devem possibilitar a inserção de heurísticas do problema;

10. Grafos esparsos e densos: a representação deve codificar soluções em grafos não-completos (com menos de $n(n-1)/2$ arestas) preservando as características anteriores.

Um EA que utiliza uma representação com as características enumeradas deveria ser eficiente para PPRs. A última propriedade em especial não tem sido avaliada na literatura, uma vez que as representações existentes foram aplicadas somente para problemas de grafos completos (Raidl e Julstrom, 2003). Deve-se observar que essa propriedade é muito importante, pois uma diversidade de PPRs do mundo real são modelados por grafos esparsos, principalmente, se envolver redes de larga-escala.

Rothlauf (2006), analisa três propriedades como fundamentais para o desempenho dos EAs:

1. Localidade: possui o mesmo significado da localidade proposta por Raidl e Julstrom (2003). Representações com baixa localidade, ou seja, pequenas alterações na codificação produzem grandes alterações na árvore correspondente e fazem com que os EAs assemelhem-se a processos de busca aleatória;

2. Escalabilidade: as representações que tornam genes mais significativos que outros para a solução do problema diminuem a velocidade de convergência dos EAs. Assim, é desejável que as representações não possuam escalabilidade dos genes. Por exemplo, um fenótipo composto por um lista de número inteiros e todos os alelos (inteiros) são igualmente importantes para o cálculo do *fitness*. Quando todos os alelos possuem igual importância, os *building blocks*[1] são solucionados em paralelo, ou seja, igualmente escalados. No entanto, ao utilizar uma representação binária

[1] *Building blocks* são subconjuntos de genes no cromossomo fortemente correlacionados. A determinação correta de alelos de um *building block* contribuem significativamente para o aumento do valor de *fitness* (Goldberg e Holland, 1988)

para esse problema, alguns bits passam a ter maior importância (*bits* mais significativos) para a solução do problema do que outros (*bits* menos significativos). Nesse caso, os *building blocks* são solucionados de forma seqüencial. Por isso, o tempo de convergência do EA aumenta;

3. Redundância: ocorre se mais de uma codificação é traduzida para a mesma árvore. Se a redundância for sinônima (codificações similares representam árvores iguais) a complexidade computacional do EA para o problema não é alterada; caso contrário a complexidade do algoritmo é alterada, podendo diminuir ou aumentar.

A Seção 4.1 apresenta as principais representações que são classificadas como indiretas e suas características. A Seção 4.2 aborda as representações diretas e seus operadores. A Seção 4.3 descreve as representações mistas. A Seção 4.4 apresenta uma breve comparação das representações de PPRs. A Seção 4.5 apresenta as considerações finais deste Capítulo.

4.1 Representações Indiretas

Representações indiretas são aquelas onde existe um mapeamento genótipo-fenótipo e podem ser aplicados os operadores de recombinação e mutação convencionais. As principais representações para PPRs que podem ser classificadas como indiretas são: Número de Prüfer (Zhou e Gen, 1997; Abuali et al., 1994), Vetor de Características (Palmer, 1994; Davis et al., 1993), Tendência de Ligação e Nó (Palmer e Kershenbaum, 1995), Chaves Aleatórias para Redes (Rothlauf et al., 2002; Schindler et al., 2002), *Blob Code* (Julstrom, 2005, 2004a, 2001), *Dandelion Code* (Thompson et al., 2007; Paulden e Smith, 2006a,b).

As Seções 4.1.1, 4.1.2 e 4.1.3 descrevem algumas representações indiretas para PPRs que são importantes e ilustram como tais representações podem ser distintas.

4.1.1 Vetor de Características

A representação Vetor de Características (CV, do inglês *Characteristic Vector*) é um *array* binário. A cada aresta de um grafo é associado um índice de $1 \ldots m$ de uma posição do *array* (Palmer, 1994; Davis et al., 1993), onde m é o número de todas as possíveis arestas do grafo. Uma árvore é codificada pela inserção de 1s na posição correspondente a cada uma de suas arestas e 0s para as demais posições. A Tabela 4.1 ilustra a CV da árvore com nove vértices apresentada na Figura 4.1.

A primeira propriedade da CV a ser destacada é a sua capacidade de representar todas as árvores possíveis. No entanto, a representação também pode codificar grafos que

Tabela 4.1: Representação por CV utilizando *array* para a árvore da Figura 4.1.

1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0
A-B	A-C	A-D	A-E	A-F	A-G	A-H	A-I	B-C	B-D	B-E	B-F	B-G	B-H	B-I	C-D	C-E	C-F

0	0	0	0	1	0	0	1	0	0	0	0	1	0	0	0	0	0
C-G	C-H	C-I	D-E	D-F	D-G	D-H	D-I	E-F	E-G	E-H	E-I	F-G	F-H	F-I	G-H	G-I	H-I

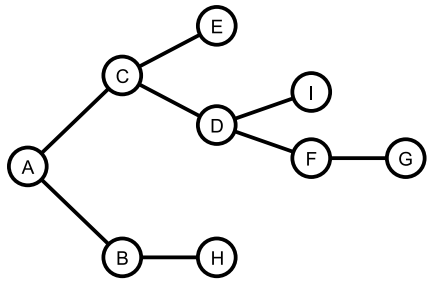


Figura 4.1: Exemplo de uma árvore com 9 vértices.

não representam árvores. Toda CV que representa uma árvore deve possuir $(n-1)$ 1s, que representem arestas de um grafo conexo e sem ciclos, onde n é o número de vértices do grafo. Em um grafo completo existem $2^{n(n-1)/2}$ CVs, porém existem somente $n^{(n-2)}$ árvores (Prüfer, 1918). A probabilidade de aleatoriamente criar uma árvore em CV é $\frac{n^(n-2)}{2^{n(n-1)/2}}$, que possui limite superior $\frac{2\ln(n)}{n\ln(2)n}$, que por sua vez é também limitado por $\frac{3\ln(n)}{n}$. Por isso, a chance de construir aleatoriamente uma CV que represente uma árvore decresce exponencialmente com o tamanho n do problema (Palmer, 1994).

As CVs aleatoriamente construídas podem ser inválidas em dois sentidos diferentes:

- Existem ciclos no grafo representado;
- O grafo não é conexo.

Caso não sejam removidas da população as soluções infactíveis, existem duas maneiras de lidar com esse problema. A primeira consiste em ignorar essas soluções e deixá-las na população. A segunda é a aplicação de um método de correção das soluções infactíveis. Na primeira estratégia, deve-se encontrar um meio de calcular o *fitness* dessas soluções e garantir que ao final do EA exista uma solução válida. A segunda forma é mais utilizada e as soluções são corrigidas em dois passos:

- Remova as arestas da CV que formam ciclos;

- Adicione arestas para obter uma árvore conexa.

Esse método de correção das soluções inválidas mostrou não possuir tendência, podendo, então, ser utilizado para qualquer tipo de problemas independente da topologia da solução ótima (Rothlauf, 2006).

A segunda propriedade importante de CV é que esta possui uma localidade naturalmente alta. A alteração de uma posição do *array* corresponde diretamente a inclusão ou remoção de uma aresta. A localidade de CV é independente da topologia da árvore representada (Rothlauf et al., 2002).

Além da alta representatividade de soluções infactíveis, outro fator crítico para CV é a sua complexidade de tempo e espaço relativamente alta. A complexidade de espaço necessária para CV é $O(m)$ para grafos esparsos e em $O(n^2)$ para grafos densos e completos. Uma vez que, para decodificar a árvore representada é necessário percorrer todo o *array* de CV de comprimento m, a complexidade de tempo é $O(m)$ e, para grafos completos, é $O(n^2)$ (Raidl e Julstrom, 2001). A complexidade de CV está entre as mais altas das representações encontradas na literatura da área.

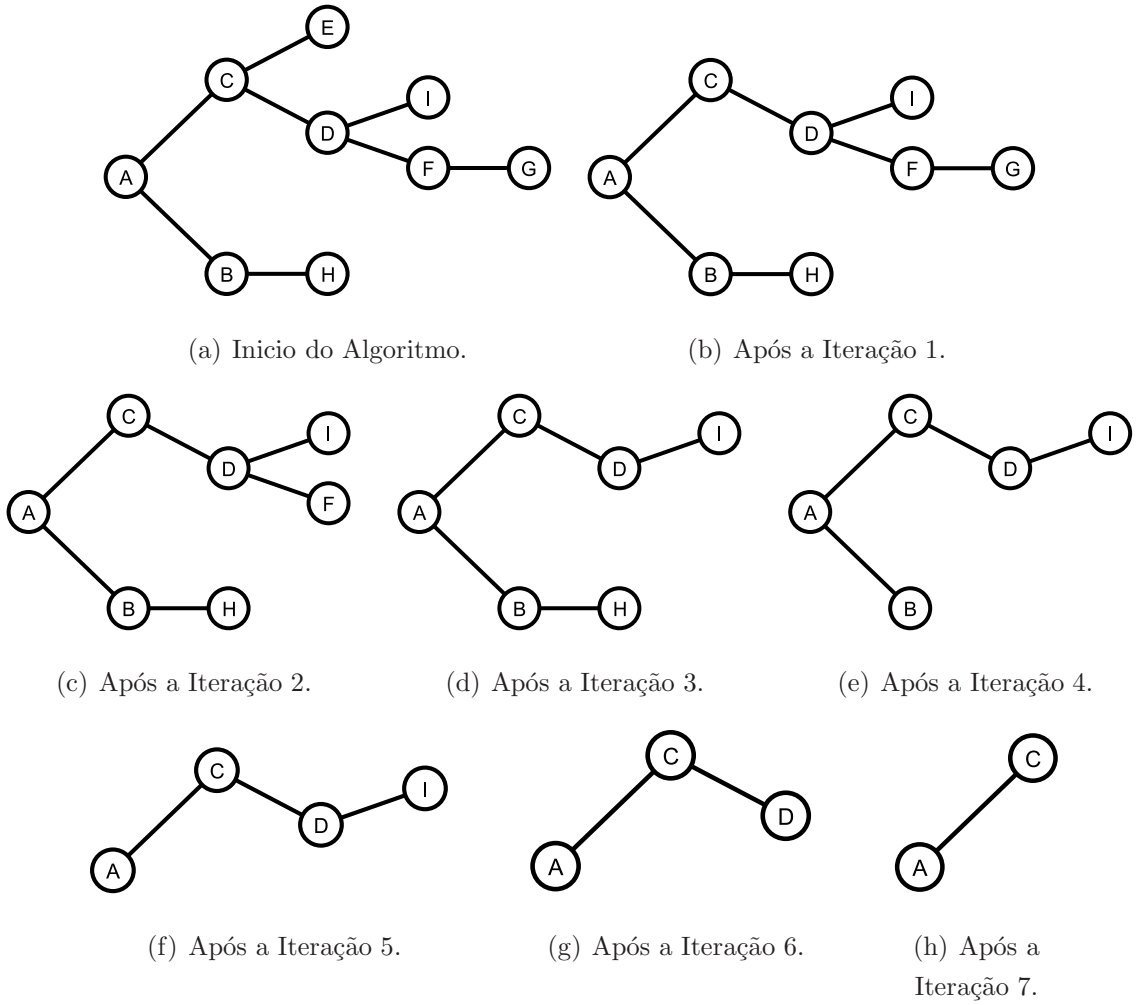
4.1.2 Número de Prüfer

Número de Prüfer é uma das representações para árvore mais conhecidas (Zhou e Gen, 1997; Palmer e Kershenbaum, 1995; Gen e Zhou, 1995; Abuali et al., 1994). Essa representação tem como base a propriedade da correspondência única entre uma *string* de comprimento $n-2$ e uma única árvore geradora (Prüfer, 1918). No entanto, a representação Número de Prüfer não é o único mapeamento de correspondência $1-1$ entre a *string* de comprimento $n-2$ e uma árvore. Picciotto (1999) propôs outros mapeamentos como *Blob Code* e *Dandelion Code*.

Define-se o número de Prüfer como segue: dada uma árvore T com n vértices, o número de Prüfer (P) é um número de $n-2$ dígitos, cujos dígitos possuem valores entre 1 e n. A codificação de uma árvore em número de Prüfer é definida pelos passos do Algoritmo 4.1.1 (Zhou e Gen, 1997; Abuali et al., 1994).

Algoritmo 4.1.1: - Codificação por Número de Prüfer.

//Seja $T = (V, E)$ a árvore a ser codificada (Entrada)
//Seja P o Número de Prüfer correspondente a T (Saída)
//Seja $\overline{P}$ o complemento do Número de Prüfer
//Seja $v_i, v_j \in V$
(1) $P \leftarrow \emptyset$
(2) $\overline{P} \leftarrow \emptyset$
(3) Enquanto (houver mais de dois vértices a serem considerados)
  (4) $v_i \leftarrow$ vértice folha de menor rótulo $\in V$; (ver Apêndice A)
  (5) $v_j \leftarrow$ vértice predecessor de v_i se $(v_i, v_j) \in E$;
  (6) $P \leftarrow$ concatena (P, v_j);
  (7) $\overline{P} \leftarrow$ concatena $(\overline{P}, v_i)$, se $v_i \notin P$;
  (8) Remova o vértice v_i e a aresta (v_i, v_j) de T.



(a) Inicio do Algoritmo. (b) Após a Iteração 1.

(c) Após a Iteração 2. (d) Após a Iteração 3. (e) Após a Iteração 4.

(f) Após a Iteração 5. (g) Após a Iteração 6. (h) Após a Iteração 7.

Figura 4.2: Exemplo de aplicação do Algoritmo 4.1.1 para a árvore da Figura 4.1.

Considere a árvore da Figura 4.1 para um exemplo de aplicação do algoritmo de

codificação. Para a construção do Número de Prüfer, é utilizada a ordem alfabética para decidir se um rótulo de vértice é menor ou maior do que outro. A Figura 4.2 apresenta o estado da árvore após cada iteração do Algoritmo 4.1.1. Inicia-se o processo pela busca do vértice folha de menor rótulo que é o vértice E. No próximo passo, o vértice predecessor a E é adicionado a P, nesse caso $P = C$. Em seguida o vértice E é removido da árvore (ver Figura 4.2(b)). Na próxima iteração, G é o vértice folha de menor rótulo. Então, seu predecessor, F é adicionado a $P = F$ e G é removido da árvore (ver Figura 4.2(c)). O processo continua considerando sempre o vértice folha de menor rótulo e inserindo o seu predecessor em P. Ao final do processo, quando restar somente 2 vértices na árvore (ver Figura 4.2(h)), é obtido número de Prüfer $P = CFDBADC$ com $\bar{P} = EGH$.

Para decodificar um Número de Prüfer gerando uma árvore, utilizam-se os passos apresentados no Algoritmo 4.1.2. Os vértices de $\overline{P}$ são denominados elegíveis para posteriores considerações. O processo de decodificação continua até não haver mais nenhum dígito em P, e dois vértices restarem em $\overline{P}$.

Algoritmo 4.1.2: - Decodificação do Número de Prüfer.

```
//Seja P o Número de Prüfer (Entrada)
//Seja P̄ o complemento do Número de Prüfer
//Seja T = (V, E) a árvore codificada (Saída)
//Seja v_i, v_j ∈ V
(1) Enquanto (restar pelo menos um dígito em P)
    (2) v_i ← vértice elegível de P̄ com o menor rótulo;
    (3) v_j ← dígito mais a esquerda de P;
    (4) Adicione a aresta (v_i, v_j) em T;
    (5) Remova v_i de P̄ e v_j de P;
    (6) Se (v_j ∉ P)
        (7) Coloque v_j em P̄;
//Sejam v_r e v_s os dois vértices restantes, então:
(8) Adicione a aresta (v_r, v_s) na árvore T.
```

Considere o Número de Prüfer $P = CFDBADC$ do exemplo anterior e $\overline{P} = EGHI$ para decodificarmos a árvore da Figura 4.1. Primeiro, escolhe-se o vértice de menor rótulo de $\overline{P}$ que é E. A seguir, o primeiro dígito do número de Prüfer que é C. Obtém-se a aresta (C, E) que é inserida na árvore. Remove-se os dígitos E de $\overline{P}$ e C de P. Como o dígito C ainda está presente em P, esse não é adicionado a $\overline{P}$. Na próxima iteração os vértices de $\overline{P}$ (elegível) e P são respectivamente G e F. Então, a aresta (F, G) é adicionada a árvore, e os dígitos F e G removidos de P e $\overline{P}$. Como o dígito F não está mais presente em P, esse digito é adicionado a $\overline{P}$. O processo contínua dessa forma, até que todos dos dígitos de P tenham sido processados.

Uma implementação eficiente dos algoritmos de codificação e decodificação utiliza uma

lista de prioridades implementada por uma *heap*. Assim, a complexidade de tempo dos algoritmos é $O(n \log n)$ (Gottlieb et al., 2001).

As principais vantagens da representação por Número de Prüfer são:

- Toda árvore pode ser representada por Número de Prüfer. A análise do algoritmo de codificação comprova essa propriedade, pois toda árvore possui pelo menos um vértice de grau 1 (folha), possibilitando a aplicação;

- Somente árvores são representadas por Número de Prüfer. Essa propriedade foi provada em Prüfer (1918) na demonstração do teorema de Cayley sobre a existência de n^{n-2} árvores geradoras para um grafo completo com n vértices. Assim, um Número de Prüfer pode ser aleatoriamente construído e sempre representará uma árvore. Por conseguinte, essa representação permite a utilização dos operadores convencionais de mutação e recombinação;

- Todo Número de Prüfer representa somente uma árvore. Isso é uma conseqüência direta do mapeamento $1 - 1$ da proposta de Prüfer (Prüfer, 1918);

- Todas as árvores são representadas uniformemente, ou seja, não há tendência. Novamente, uma conseqüência direta do mapeamento $1-1$. Existem n^{n-2} árvores para um grafo completo de tamanho n e também existem exatamente n^{n-2} Números de Prüfer. Por isso, EAs não possuem problemas de redundância e não são afetados pela representação excessiva ou inexpressiva de alguns indivíduos.

Essas propriedades fazem do Número de Prüfer uma representação interessante para árvores. No entanto, essa representação também possui algumas desvantagens. A principal dessas desvantagens é sua localidade relativamente baixa, uma vez que a mudança de somente um dígito de um Número de Prüfer pode mudar drasticamente a árvore resultante (Gottlieb et al., 2001). Por isso, um operador de mutação convencional que explora variações em um indivíduo não obtém em geral descendentes que são similares aos pais. Assim, mutações no Número de Prüfer aproximam-se a um algoritmo de busca aleatória; ao contrário do seu propósito de realizar uma busca local.

Uma outra desvantagem do Número de Prüfer ocorre para grafos esparsos. Nesse caso, a geração de árvores válidas utilizando Número de Prüfer é improvável. É importante salientar que há diversos PPRs que não correspondem a grafos completos.

4.1.3 Chaves Aleatórias para Redes

A representação Chaves Aleatórias para Rede (NetKeys, do inglês, *Network Random Keys*) utiliza chaves aleatórias como uma alternativa ao CV, uma vez que este gera um grande conjunto de soluções infactíveis (ver Seção 4.1.1) (Rothlauf et al., 2002; Schindler et al., 2002).

NetKeys, assim como CV, representa uma árvore por um *array*, em que cada índice é associado com uma aresta. No entanto, ao contrário de CV, que utiliza 0 ou 1 para indicar a presença ou ausência de uma aresta, NetKeys utiliza números reais entre 0 e 1 para cada uma das arestas. Os valores reais associados às arestas podem ser interpretadas como a importância da aresta. Assim, é possível distinguir quais arestas são mais ou menos importantes. A importância atribuída as arestas pelos valores das chaves aleatórias evita problemas de redundância, ou a representação excessiva ou inexpressiva de um tipo de solução. Em outras palavras, um tipo de árvore ser menos ou mais representado do que outro, além de não gerar soluções infactíveis. Assim, os operadores convencionais de mutação e recombinação podem ser aplicados a NetKeys sem a necessidade de nenhum mecanismo de correção (Rothlauf et al., 2002).

A representação NetKeys consiste de um *array* de números reais de comprimento $m = n(n-1)/2$, onde cada posição representa uma das arestas do grafo, como em CV. Os números reais associados a cada aresta seguem o princípio de chaves aleatórias (Bean, 1994).

Uma seqüência de chaves R de comprimento m possui m diferentes valores. Para qualquer seqüência de chaves $R = R_0, \ldots, R_{m-1}$, a permutação σR de R é definida como a seqüência com elementos $\sigma R_i = R_{\sigma(i)}$, de forma que σR corresponde a uma seqüência de chaves em ordem decrescente (Rothlauf et al., 2002).

Dada uma seqüência R de chaves de tamanho m e a permutação dessa seqüência como descrito no parágrafo acima, uma árvore representada por NetKeys pode ser decodificada pelo Algoritmo 4.1.3.

Como exemplo da utilização de NetKeys, a árvore da Figura 4.1 pode ser obtida a partir da seqüencia de chaves da Tabela 4.2, da permutação σR = (2, 23, 26, 14, 17, 31, 16, 18, 1, 9, 11, 36, 22, 24, 35, 29, 21, 12, 28, 15, 6, 3, 8, 7, 25, 13, 30, 19, 32, 10, 4, 20, 33, 27, 34, 5) e da aplicação do Algoritmo 4.1.3.

Pela aplicação do Algoritmo 4.1.3 à seqüência R anterior, as arestas são inseridas na seguinte ordem A-C, D-F, D-I, B-H, C-E, F-G, C-D, sem formar nenhum ciclo. Porém, ao tentar inserir a aresta C-F, que seria a próxima na permutação, essa forma ciclo e é

Algoritmo 4.1.3: Pseudo-código de Decodificação da Representação NetKeys.

//Seja R uma seqüência de chaves de comprimento m (entrada)
//Seja G um grafo com n vértices (saída)
(1) Faça $i = 0$;
(2) Ordene R obtendo a permutação σR. Todas as arestas de G são numeradas de 1 a m;
(3) Faça $j = \sigma R_i$;
(4) Se a inserção da aresta com rótulo j em G não forma ciclo, então insira a aresta j em G;
(5) Pare, se existem $(n-1)$ arestas em G;
(6) Incremente i e retorne ao Passo 3.

Tabela 4.2: Representação por NetKeys para a árvore da Figura 4.1.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0.82	0.98	0.35	0.12	0.03	0.37	0.28	0.31	0.79	0.17	0.76	0.52	0.23	0.93	0.47	0.86	0.91	0.85
A-B	A-C	A-D	A-E	A-F	A-G	A-H	A-I	B-C	B-D	B-E	B-F	B-G	B-H	B-I	C-D	C-E	C-F

19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
0.21	0.11	0.55	0.66	0.97	0.65	0.26	0.94	0.09	0.48	0.61	0.22	0.88	0.19	0.10	0.05	0.62	0.73
C-G	C-H	C-I	D-E	D-F	D-G	D-H	D-I	E-F	E-G	E-H	E-I	F-G	F-H	F-I	G-H	G-I	H-I

descartada. O processo continua pela aresta A-B seguida pela aresta B-C e termina, pois foram inseridas $n-1$ arestas na árvore.

O processo de construção da permutação σR, ou seja, a ordenação a partir de R, é a etapa do algoritmo que requer maior complexidade de tempo no processo de decodificação. Essa etapa requer tempo $O(m \log(m))$, que resulta na complexidade do algoritmo de decodificação (Rothlauf et al., 2002).

A representação NetKeys possui as seguintes propriedades:

- A representação possui alta localidade, a mutação em NetKeys pode resultar na alteração de uma aresta ou de nenhuma;
- A representação permite uma distinção entre as arestas importantes e não importantes;
- Não existem soluções infactíveis. O método de construção garante que somente árvores serão decodificadas a partir da representação.

Apesar dos seus claros benefícios em relação a CV, essa representação ainda necessita de complexidade de espaço e tempo superiores a de outras representações presentes na

literatura, conforme apresenta este Capítulo (ver Seção 4.4).

4.2 Representações Diretas

As representações diretas utilizam o próprio conjunto de arestas da árvore para representá-la e, por isso, necessitam de operadores especiais de recombinação e mutação. As principais representantes desse tipo são Conjunto de Arestas (Raidl, 2000) e NetDir (Rothlauf, 2006). A seguir será apresentada a representação conjunto de arestas e seus operadores.

A representação Conjunto de Arestas proposta em Raidl (2000) consiste em representar uma árvore diretamente por meio de seus conjuntos de arestas. A representação por conjunto de arestas da árvore da Figura 4.1 é: { (A,B), (A,C), (B,H), (C,D), (C,E), (D,I), (D,F) e (F,G)}.

Essa representação pode ser implementada de duas formas: por meio de um *array* ou de uma tabela de dispersão (*hash*) (Cormen et al., 2000) em que as entradas são os pares de vértices que representam a aresta. A implementação por tabela de dispersão possibilita que as operações de inserção, remoção e busca das arestas de um indivíduo ocorram em tempo constante. Ambas as formas de implementação possuem complexidade de espaço $O(n)$, onde n é o número de vértices do grafo (Raidl e Julstrom, 2003).

Para garantir a produção exclusiva de soluções factíveis foram propostos algoritmos específicos para geração aleatória de soluções e para as operações de recombinação e mutação. Para cada um desses algoritmos existem duas variantes: uma baseada em heurísticas e outra sem heurísticas. A Seção 4.2.1 apresentada a versão sem heurística desses algoritmos. A Seção 4.2.2 descreve como alterar os operadores para que esses utilizem heurística.

4.2.1 Conjunto de Arestas: sem Heurísticas

Com o objetivo de gerar soluções factíveis tanto para grafos completos quanto não completos são propostos e analisados três algoritmos para geração das soluções: o PrimRST e o KruskalRST e o RandWalkST. Os dois primeiros são modificações nos algoritmos de árvore geradora mínima de Prim e Kruskal respectivamente, em relação ao critério de escolha das arestas para compor a árvore. O terceiro é o algoritmo de geração por caminhos aleatória (Raidl e Julstrom, 2003). As análises de PrimRST mostram que esse operador possui uma tendência em representar árvores estrela, que possuem um vértice central e os demais conectados a ele, e representam menos árvores parecidas com árvores linha (que

têm o grau máximo da árvore igual a dois). Assim como o PrimRST, o KruskalRST apresenta uma tendência para árvores do tipo estrela. No entanto, a tendência de KruskalRST é menor do que a do PrimRST. O RandWalkRST não possui tendência e sua complexidade média de tempo é $O(n\log(n))$, porém não possui um limitante superior de pior caso. Raidl e Julstrom (2003) recomendam o uso do algoritmo KruskalRST, que garante complexidade $O(n\log(n))$ no pior caso. A seguir serão apresentados maiores detalhes somente sobre esse operador de inicialização.

O KruskalRST (ver Algoritmo 4.2.1) para geração das soluções iniciais seleciona uma aresta aleatoriamente e, então, verifica se esta forma um ciclo. Se a aresta não formar ciclo ela é incluída na árvore. O processo repete-se até que $n-1$ arestas tenham sido adicionadas a árvore. A complexidade de tempo para verificar se uma aresta forma ou não ciclo é $O(m)$. O passo do algoritmo proposto que possui maior complexidade é o de verificação de ciclo formado pela aresta incluída, Assim, a complexidade de tempo do KruskalRST é $O(m)$ (Raidl, 2000; Raidl e Julstrom, 2003).

Algoritmo 4.2.1: - Geração das Soluções para Conjunto de Arestas, KruskalRST.

```
//Seja E o conjunto de arestas do grafo (entrada)
//Seja V o conjunto de vértices do grafo (entrada)
//Seja E_T o conjunto de arestas da árvore (saída)
(1) E_T ← ∅;
(2) Â ← E;
(3) Enquanto |E_T| < n − 1
    (4) Escolha aleatoriamente um aresta (v_u, v_x) ∈ Â
    (5) Â ← Â − {(v_u, v_x)};
    (6) Se v_u e v_x não são conectados em E_T
        (7) E_T ← E_T ∪ {(v_u, v_x)};
(8) Retorne E_T;
```

O operador de recombinação deve garantir uma alta hereditariedade, ou seja, deve construir descendentes que, juntos, possuam a maioria ou todas as suas arestas presentes nos pais. Uma forma de obter o resultado esperado, é aplicar um dos algoritmos PrimRST, KruskalRST ou RandWalkRST no grafo $G' = (V, E_{T1} \cup E_{T2})$, onde E_{T1} e E_{T2} são os conjuntos de arestas dos pais. Porém, quando o problema possui restrições, nem sempre é possível obter soluções factíveis utilizando somente as arestas dos pais, assim é necessário um método diferente do descrito para o algoritmo de recombinação (Raidl e Julstrom, 2003).

O algoritmo desenvolvido por Raidl (2000) inclui as arestas presentes em ambos os pais (E_{T1} e E_{T2}) na nova árvore (E_T). Em seguida, insere as arestas do conjunto $\hat{F}$ que corresponde à disjunção das arestas de seus pais. Se após a inclusão desses conjuntos, uma árvore geradora válida não foi formada, então, determina cada componente desconexa U_k

e seleciona arestas aleatoriamente para conectar cada componente à árvore que está sendo formada. O Algoritmo 4.2.2 descreve os passos da aplicação do operador de recombinação. Para ilustrar o funcionamento do operador, considere a Figura 4.3.

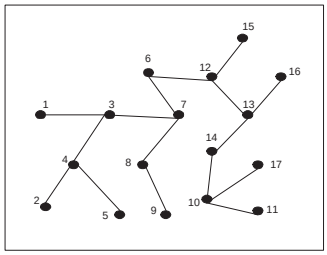
Algoritmo 4.2.2: - Recombinação por Conjunto de Arestas.

//Seja E_{T1} e E_{T2} os conjuntos de arestas dos pais (entrada)
//Seja E_T o conjunto de arestas da nova árvore (saída)
(1) $E_T \leftarrow E_{T1} \cap E_{T2}$
(2) $\hat{F} \leftarrow E_{T1} \cup E_{T2} \backslash E_T$;
(3) Para (todas arestas $(v_i, v_j) \in \hat{F}$ em ordem aleatória)
    (4) Se $(v_i, v_j) \notin E_T$)
        (5) $E_T \leftarrow E_T \cup \{(v_i, v_j)\}$;
        (6) Se ($|E_T| = n - 1$)
            (7) Retorne E_T;
/* determine todas componentes desconexas: U_k */
(8) $U \leftarrow \{U_k\}$, $\forall v_i, v_j \in V, v_i \neq v_j$:
(9) $v_i \in U_k \wedge$ conectado$(v_i, v_j, E_T) \longrightarrow v_j \in U_k$,
(10) $v_i \in U_k \wedge$ **not** (conectado $(v_i, v_j, E_T)) \longrightarrow v_j \notin U_k$,
(11) $\bigcup_k U_k = V$;
/* conecte as componentes aleatoriamente */
(12) Para (todo $U_k \in U \backslash U_1$ em ordem aleatória)
    (13) Escolha $v_i \in U_1|$ aleatoriamente;
    (14) Escolha $v_j \in U_k|$ aleatoriamente;
    (15) $E_T \leftarrow E_T \cup (v_i, v_j)$;
    (16) $U_1 \leftarrow U_1 \cup U_k$;
(17) Retorne E_T;

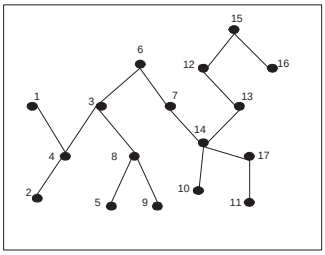
As Figuras 4.3(a) e 4.3(b) representam as árvores que serão recombinadas. A Figura 4.3(c) mostra o resultado da inclusão das arestas presentes em ambos os pais do Passo 1 do algoritmo. O Passo 2 é representado na Figura 4.3(d). A seguir são determinadas as componentes da árvore (ver Figura 4.3(e)) que são conectadas por arestas aleatórias. Por fim, a Figura 4.3(f) apresenta a árvore resultante da recombinação.

O operador de mutação deve ter alta localidade, isto é, uma pequena alteração no cromossomo provoca uma pequena mudança na árvore. O operador de mutação para conjunto de arestas consiste em escolher aleatoriamente uma aresta a ser inserida e remover uma aresta pertencente ao ciclo formado pela inclusão da nova aresta. Os passos do processo de mutação são descritos pelo Algoritmo 4.2.3. Um exemplo de funcionamento desse operador é ilustrado na Figura 4.4. Os operadores de mutação e recombinação podem ser implementados em tempo $O(n)$ (Raidl, 2000).

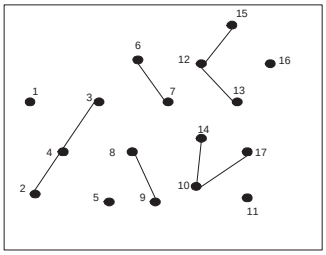
A inclusão de restrições do problema pode ser feita em ambos os métodos tanto no KruskalRST quanto no PrimRST para geração de soluções (Raidl e Julstrom, 2003).



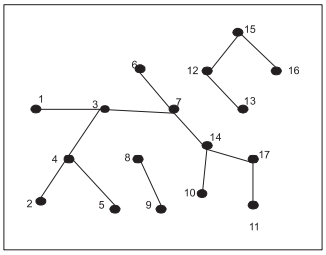
(a) Árvore relativa a E_{T1}.



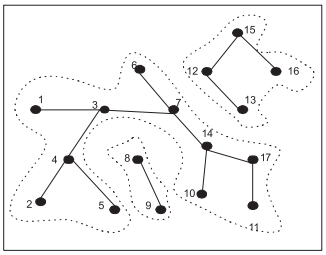
(b) Árvore relativa a E_{T2}.



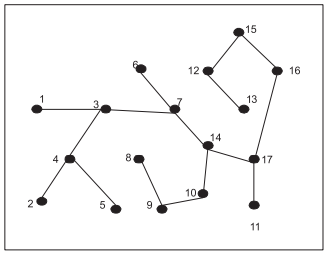
(c) Árvore (E_T) formada por arestas de E_{T1} e E_{T2}.



(d) Inclusão de arestas que pertencem a somente um dos pais $\hat{F}$.



(e) Determinação das componentes.



(f) Árvore resultante.

Figura 4.3: Ilustração do funcionamento do operador de recombinação para Conjunto de Arestas.

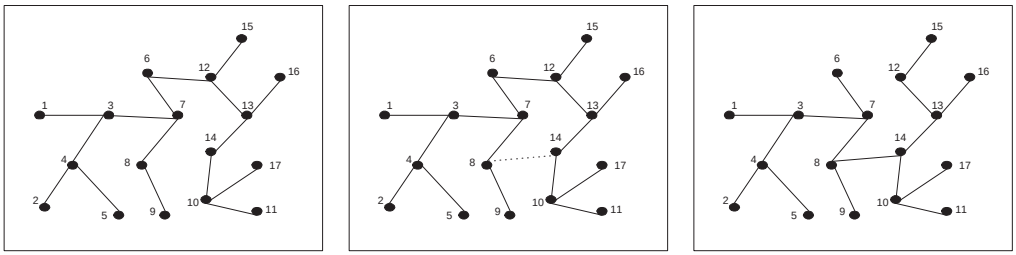
Algoritmo 4.2.3: - Mutação para Conjunto de Arestas.

//Seja V o conjunto de vértices do grafo (entrada)
//Seja E_T o conjunto de arestas da árvore (entrada e saída)
// Escolha da aresta (v_i, v_j) para inserção:
(1) Escolha $v_i \in V$ aleatoriamente;
(2) Escolha $v_j \in V \backslash \{v_i\} |\ (v_i, v_j) \notin E_T$ aleatoriamente;
// Escolha da aresta (v_a, v_b) para ser apagada:
(3) $L \leftarrow \{(k,l) \in E_T | (k,l)$ compõe o caminho de v_i a $v_j\}$;
(4) $(v_a, v_b) \leftarrow (v_a, v_b) \in L | v_a = v_i \vee v_b = v_i$;
(5) $E_T \leftarrow E_T \cup \{(v_i, v_j)\} \backslash \{(v_a, v_b)\}$;
(6) Retorne E_T;

4.2.2 Conjunto de Arestas: com Heurísticas nos Operadores

Buscando melhorar o desempenho do algoritmo, foi proposta em (Raidl, 2000) a incorporação de heurísticas para construção das soluções iniciais e para os operadores de mutação e recombinação. As heurísticas adicionadas aos operadores permitem que arestas de menor custo tenham maior probabilidade de serem incluídas do que arestas de maior custo.

A inclusão de heurísticas na inicialização consiste em, no lugar de processar todas as arestas de forma aleatória, ordenar as arestas de forma crescente de peso. Para o



(a) Árvore referente à E_T. (b) Uma aresta é escolhida para ser inserida em E_T formando um ciclo. (c) Árvore resultante, após a eliminação de uma aresta do ciclo.

Figura 4.4: Ilustração do operador de mutação por Conjunto de Arestas.

primeiro indivíduo são analisadas as arestas de menor peso. Com o objetivo de garantir a diversidade da população inicial, para os demais indivíduos, a ordem das arestas é alterada para os indivíduos possuirem menor tendência. As k primeiras arestas de menor custo da ordenação inicial são permutadas. O valor de k é aumentado com o tempo, isto é, a i-ésima solução inicial da população de tamanho P ($i = 1, \ldots, P$) é criada usando $k = \alpha(i-1)n/P$, onde α é o parâmetro estratégico que controla a tendência média da heurística e n é o número de vértices do grafo.

Para a inclusão das heurísticas na recombinação, altera-se a ordem das arestas do conjunto $\hat{F}$ para às arestas de menor custo. Enquanto $\hat{F}$ for não vazio, aplica-se um torneio binário nas arestas de $\hat{F}$, isto é, duas arestas de $\hat{F}$ são selecionadas aleatoriamente, então, a aresta de menor custo é escolhida. Em seguida, é verificado se essa aresta pode ser incluída na árvore.

O operador de mutação é alterado para trabalhar com heurísticas aumentando a probabilidade de inserir arestas de menor custo. Para esse propósito, pode-se aplicar o torneio nas arestas disponíveis. Porém, dependendo do número de vértices, o número de arestas participando do torneio deve ser muito grande para favorecer as arestas de menor custo. Assim, outra estratégia para trabalhar de forma apropriada é necessária. Selecione a i-ésima aresta de menor custo com $i = \lfloor |\mathcal{N}(0, \beta n)| \rfloor mod(n(n-1)/2) + 1$, onde $\mathcal{N}(0, \beta n)$ é uma distribuição normal com média zero e desvio padrão βn. O valor R_i é um número inteiro positivo aleatório em $[1, n(n-1)/2] = [1, m]$, com uma distribuição que favorece arestas de baixo custo dependendo do parâmetro estratégico β. Pequenos valores de β geram uma forte tendência em torno das arestas de baixo custo (Raidl, 2000). Infelizmente, a aresta sorteada ou pode já estar na árvore ou pode ser uma aresta que não respeita uma restrição do problema. Nesses casos, uma nova aresta é sorteada.

Pode-se fazer uma implementação que possibilite que cada aresta do grafo seja sorteada

uma única vez durante a aplicação do algoritmo. Com isso, a complexidade de tempo dos operadores permanece inalterada (Raidl, 2000).

Rothlauf e Tzschoppe (2005); Tzschoppe et al. (2004) mostraram que a representação de Conjunto de Arestas com inicialização, e os operadores de recombinação e mutação utilizando heurísticas possuem tendência. A utilização das heurísticas, direciona a representação para a codificação de árvores próximas a árvore geradora mínima. Assim, as heurísticas devem ser implementadas somente quando a solução do problema for parecida com a árvore geradora mínima do grafo.

4.3 Representações Mistas

As representações mistas são aquelas que possuem características tanto de representação indiretas quando de diretas. Em outras palavras, utilizam um mapeamento fenótipo-genótipo e também possuem operadores de reprodução especiais. Alguns exemplos dessas representações são: Precedentes Diretos (Carvalho et al., 2001, 1999), Ajuste Adaptativo das Ligações (Soak et al., 2005), Permutação Baseada em Árvore (Zhou e Gen, 2003), D-Based (Soak et al., 2004), Sub-Conjunto de Comprimento Fixo (Julstrom, 2004b), Nó-Profundidade (Delbem e de Carvalho, 2003; Delbem et al., 2004) e Predecessores ou Codificação Determinante (Lin e Gen, 2006; Raidl e Drexel, 2000; Palmer e Kershenbaum, 1995; Abuali et al., 1995). A Seção 4.3.1 descreve a representação Predecessores com o objetivo de ilustrar o grupo de representações mistas. A Seção 4.3.2 apresenta a representação Nó-Profundidade, que é base das propostas de codificação abordadas nesta pesquisa.

4.3.1 Predecessores

A representação por Predecessores ou Codificação Determinante foi inicialmente proposta em Abuali et al. (1995) e tem como base os teoremas de Even (ver (Even, 1979)) sobre como determinar o número de árvores geradoras de um grafo orientado com uma determinada raiz a partir de sua matriz de adjacências. Na representação por predecessores, para codificar uma árvore, deve-se escolher um vértice arbitrário e chamar de raiz (*root*). Para cada vértice v_i, exceto o *root*, a representação armazena o seu predecessor que é o vértice v_j imediatamente anterior a v_i no caminho da raiz até v_i (Raidl e Drexel, 2000). Em outras palavras, se $Pred[v_i] = v_j$, então v_j é o primeiro vértice no caminho de v_i até *root* em T. O *array* da representação possui comprimento $n-1$, pois o vértice raiz não é armazenado, visto que esse não possui predecessor. Como exemplo, na Tabela 4.3

mostra um *array* representando a árvore da Figura 4.1 com o vértice A sendo a raiz da árvore.

Tabela 4.3: Representação por Predecessores para a árvore da Figura 4.1.

A	B	A	C	D	F	D	C

Uma árvore enraizada é representada por uma única seqüência de $n-1$ dígitos, onde os dígitos são números entre 1 e n. Além disso, qualquer árvore pode ser representada por $n-1$ desses dígitos. Portanto, esta é uma representação que cobre todo o espaço de solução (Lin e Gen, 2006; Raidl e Drexel, 2000; Palmer e Kershenbaum, 1995; Abuali et al., 1995). O processo de codificação/decodificação, a partir da lista de arestas pertencentes à árvore, pode ser realizado em tempo $O(n)$ (Palmer e Kershenbaum, 1995; Abuali et al., 1995).

Uma das dificuldades da representação por predecessores é a grande quantidade de soluções infactíveis que podem ser obtidas pela representação (com ciclos ou representando árvores desconexas). Os primeiros EAs que utilizaram essa representação usaram mecanismos de correção após a aplicação do operadores de crossover e mutação convencionais (Palmer e Kershenbaum, 1995; Abuali et al., 1995). Os EAs mais recentes utilizam operadores de inicialização da população, de recombinação e de mutação específicos para evitar a geração de representações que não codifiquem árvores (Lin e Gen, 2006; Raidl e Drexel, 2000).

Em Raidl e Drexel (2000) é proposto um algoritmo de inicialização que mantém dois conjuntos de vértices: um para os vértices na árvore e outro para os demais vértices ainda não conectados a árvore. Aleatoriamente, seleciona-se um vértice de cada conjunto e uma aresta entre eles é inserida sempre que um vértice é conectado a árvore, este é removido do conjunto de vértices não conectados a árvore. Antes de conectar o vértice à árvore, é verificado se a aresta é válida. O processo repete-se até que todos os vértices estejam na árvore.

Em Lin e Gen (2006) é utilizado um algoritmo de inicialização baseado no algoritmo de Prim similar ao algoritmo utilizado por conjunto de arestas, onde as arestas são escolhidas de forma aleatória para compor a árvore. Esse algoritmo tem complexidade de tempo de $O(n \log(n))$, onde n é o número de vértices do grafo.

Na operação de recombinação utiliza-se um algoritmo com base no algoritmo de Prim, porém, o conjunto de arestas possui somente as arestas pertencentes aos indivíduos pais (Lin e Gen, 2006). A operação de recombinação é realizada diretamente sobre o conjunto

de arestas. Primeiramente, inserem-se as arestas em comum em ambos os pais. Em seguida, para os vértices sem predecessor (desconectados), escolhem-se aleatoriamente um predecessor a partir das arestas dos indivíduos pais. Esse processo continua até que todos os vértices tenham seus predecessores definidos (Raidl e Drexel, 2000).

A operação de mutação consiste em remover uma aresta, obtendo duas componentes, e inserir uma nova aresta válida ligando as duas componentes por diferentes vértices (Raidl e Drexel, 2000). O método é similar ao proposto por Raidl e Drexel (2000). O algoritmo de mutação também remove uma aresta e, em seguida, reconecta as componentes por meio de uma aresta. No entanto, nesse caso, é escolhida a aresta de menor custo que incide em um dos vértices da aresta removida.

4.3.2 Nó-profundidade

A representação nó-profundidade (NDE, do inglês, *Node-Depth Encoding*) é baseada nos conceitos de caminhos e de profundidade do vértice em um grafo (árvore) (Delbem e de Carvalho, 2003). A representação consiste de pares contendo um vértice e sua profundidade (v_i, de_i). Uma árvore pode ser representada por um *array* com os pares (v_i, de_i), de tamanho n. Para representar uma árvore pela NDE, deve-se escolher um vértice (arbitrário) para ser a raiz da árvore, o qual terá a profundidade 0. A ordem dos pares dos demais vértices no *array* pode ser determinada por uma busca em profundidade (Gross e Yellen, 2004; Cormen et al., 2000) ou um percurso em pré-ordem para árvores (Valiente, 2002; Atallah e Fox, 1998). Em ambos os métodos, o processo deve iniciar pelo vértice definido como raiz. A Tabela 4.4 apresenta a NDE para a árvore apresentada na Figura 4.1, obtida por meio do percurso pré-ordem a partir do nó A (raiz).

Tabela 4.4: Representação NDE para a árvore da Figura 4.1.

i	1	2	3	4	5	6	7	8	9
de	0	1	2	1	2	3	4	3	2
v	A	B	H	C	D	F	G	I	E

Com o objetivo de gerar novas soluções de forma eficiente e de garantir que exclusivamente soluções factíveis sejam obtidas, foram propostos dois operadores de mutação. Apesar desses operadores não serem parecidos com operadores de mutação mais comuns em representações de grafos, esses operadores são chamados de mutação pois geram uma nova solução a partir de uma única outra solução. Ambos os operadores propostos produzem resultados similares e são denominados operador 1 e 2 (Delbem et al., 2004).

Esses operadores produzem novas árvores T' de um grafo G quando aplicadas a outra árvore T do mesmo grafo (Delbem et al., 2004). Ambos os operadores transferem uma subárvore entre árvores, no caso de florestas, ou de uma parte da árvore para outra. Enquanto que no operador 1 a raiz da subárvore podada não é alterada; no operador 2, um novo vértice (diferente da raiz) é escolhido para ser a nova raiz da subárvore podada (Delbem et al., 2004).

O operador 1 utiliza dois vértices em especial: o vértice p, que indica a raiz da subárvore a ser transferida e o vértice adjacente a p, chamado a, onde a subárvore podada será conectada. O operador 2 requer, além desses dois vértices, o vértice r que corresponde à nova raiz da subárvore podada.

Operador 1

Considere que os vértices p e a foram previamente escolhidos, que a implementação da representação utiliza *array* e que são conhecidos os índices de p (i_p) no *array* de T_{origem} e a (i_a) no *array* de $T_{destino}$. Então, o Operador 1 pode ser descrito pelos passos do Algoritmo 4.3.1.

Algoritmo 4.3.1: Mutação para a NDE - operador 1

//Seja F o *array* de ponteiros para as NDEs da floresta geradora (entrada)
//Seja F' o *array* de ponteiros para as NDEs (árvores) da floresta geradora resultante (saída)
//Sejam T_{origem} e $T_{destino}$ as NDEs das árvores dos vértices p e a, respectivamente
//Seja $|T_x|$ o tamanho da NDE de T_x
//Seja i_p e i_a índices dos vértices nos *arrays* (entrada)
(1) Determine o intervalo $(i_p, \ldots, i_l)$ em T_{origem} percorrendo a NDE, a partir de i_p, enquanto $de_i > de_{i_p}$ e $i < |T_{origem}|$, faça $i_l = i - 1$
//o intervalo $(i_p, \ldots, i_l)$ correspondente aos vértices da subárvore enraizada em p;
(2) Copie em $T'_{destino}$ as posições de $T_{destino}$ de $i = 1$ até i_a;
(3) Copie as posições de $(i_p, \ldots, i_l)$ de T_{origem} em $T'_{destino}$ a partir da posição $i_a + 1$ e atualize os valores de profundidade: $de_i = de_i - de_{i_p} + de_{i_a} + 1$;
(4) Copie as posições de $T_{destino}$ de $i = i_a + 1$ até $|T_{destino}|$, em $|T'_{destino}|$ a partir da posição $i_a + (i_e - i_p) + 2$;
(5) Copie em T'_{origem} as posições de T_{origem} que não pertencem ao intervalo $(i_p, \ldots, i_l)$;
(6) Copie os ponteiros das NDEs de F para F' e atualize os ponteiros de T_{origem} e $T_{destino}$ para T'_{origem} e $T'_{destino}$;

Por exemplo, considere F uma estrutura de ponteiros para uma floresta representada pelas NDEs T_{origem} e $T_{destino}$ apresentadas na Tabela 4.5, cujas árvores podem ser vistas

na Figura 4.5(a). O vértice $p = B$ em T_{origem} possui índice $i_p = 2$ (em vermelho) e o vértice $a = L$ em $T_{destino}$ possui índice $i_a = 5$ (em azul).

Tabela 4.5: NDEs de T_{origem} e $T_{destino}$.

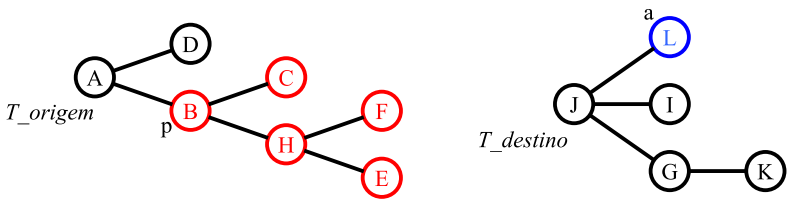
(a) T_{origem}

i	1	2	3	4	5	6	7
de	0	**1**	2	2	3	3	1
v	A	**B**	C	H	F	E	D

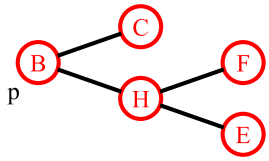
(b) $T_{destino}$

i	1	2	3	4	5
de	0	1	2	1	**1**
v	J	G	K	I	**L**

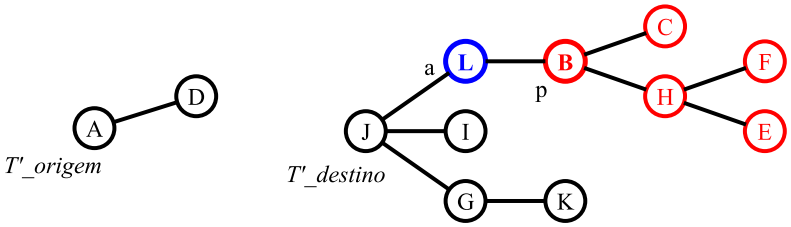
No Passo 1 do Algoritmo 4.3.1, a NDE de T_{origem} é percorrida até encontrar o índice $i = 7$ que possui profundidade igual a profundidade de p ($de_7 = de_2 = 1$) e atribui $i_l = i-1 = 6$. O intervalo $(2, \ldots 6)$, corresponde a subárvore apresentada na Figura 4.5(b).



(a) Árvores representadas por T_{origem} e $T_{destino}$.



(b) Subárvore correspondente ao intervalo $(i_p, \ldots, i_l)$.



(c) Árvores representadas por T'_{origem} e $T'_{destino}$.

Figura 4.5: Árvores utilizadas no exemplo de aplicação do operador 1.

Tabela 4.6: NDE de $T'_{destino}$ após o Passo 2.

i	1	2	3	4	5	6	7	8	9	10
de	0	1	2	1	**1**					
v	J	G	K	I	**L**					

No Passo 2, copia-se em $T'_{destino}$ as posições de $T_{destino}$ até $i = i_a = 5$ (ver Tabela 4.6).

O Passo 3 copia as posições de T_{origem} do intervalo $(i_p, \ldots, i_l$ para $T'_{destino})$ a partir da posição 6 $(i_a + 1)$ e atualiza os valores de profundidade. Por exemplo, a profundidade do vértice p em $T'_{destino}$ é $de_6 = 1 - 1 + 1 + 1 = 2$, para os demais vértices a profundidade é calculada de forma similar. O passo 4 copiaria as posições à direita de $i_a = 5$, de $T_{destino}$ para $T'_{destino}$, mas nesse caso não há tais posições. Após esse passo, $T'_{destino}$ está completa. O Passo 5 obtém T'_{origem} pela cópia de T_{origem} sem os dados no intervalo $(2, \ldots, 6)$. As NDEs resultantes podem ser vistas na Tabela 4.7. As árvores representadas por essas NDEs estão presentes na Figura 4.5(c).

Tabela 4.7: NDEs de T'_{origem} e $T'_{destino}$ após a aplicação do operador 1.

(a) T'_{origem}

i	1	2
de	0	1
v	A	D

(b) $T'_{destino}$

i	1	2	3	4	5	6	7	8	9	10
de	0	1	2	1	**1**	**2**	3	3	4	4
v	J	G	K	I	**L**	**B**	C	H	E	F

Operador 2

As diferenças entre os operadores 1 e 2 está no Passo 3, ou seja, somente a cópia do intervalo da NDE de T_{origem} para $T'_{destino}$, que corresponde a subárvore podada, é diferente (Delbem et al., 2004).

Para o operador 2, o procedimento de cópia da subárvore podada pode ser definido com mais etapas. Nesse caso, usa-se uma NDE temporária (tmp). Primeiramente, copia-se em tmp o intervalo de T_{origem} que contém as posições em que $de_i > de_{i_r}$ e $i_r < i < |T_{origem}|$, lembrando que i_r é o índice da nova raiz da subárvore com raiz atual em p. Esse intervalo corresponde aos índices $(i_r, \ldots, i_{l_r})$, que corresponde a subárvore com raiz em r.

A próxima etapa faz $i_k = i_r$ e busca-se na NDE a primeira posição i, com $i_p < i < i_k$ que possui $de_{i_p} < de_i < de_{i_k}$. Chame i de i_{r1}. Copie para a próxima posição livre em tmp o intervalo de T_{origem} definido por $(i_{r1}, \ldots, i_{l_{r1}})$ sem as posições do intervalo $(i_r \ldots i_{l_r})$. O valor de $i_{l_{r1}}$ é obtido como no Passo 1 do algoritmo do operador 1. Agora faz-se $i_k = i_{r1}$ e

começa o processo novamente incrementando i_{r1} para i_{r2} (assim por diante) e utilizando rx no lugar de r em todos os índices até que $i_{rx} = i_p$. Com esse processo, obtém-se o intervalo da NDE de T_{origem} correspondente à subárvore podada de forma que a nova raiz seja r. Esse intervalo reorganizado, então é copiado para $T'_{destino}$.

Finalizada essa etapa de reconstrução da NDE, o algoritmo do operador 2 prossegue como o algoritmo do operador 1 no Passo 3, apenas trocando os dados do intervalo $(i_p, \ldots, i_l)$ pelos dados de *tmp*.

Tabela 4.8: NDEs de T_{origem} e $T_{destino}$.

(a) T_{origem}

i	1	2	3	4	5	6	7
de	0	**1**	2	2	3	**3**	1
v	A	**B**	C	H	F	**E**	D

(b) $T_{destino}$

i	1	2	3	4	5
de	0	1	2	1	**1**
v	J	G	K	I	**L**

Como exemplo de aplicação do operador 2, considere as NDEs de T_{origem} e $T_{destino}$ da Tabela 4.8, onde a posição em vermelho corresponde ao vértice $p = B$ com $i_p = 2$, a posição em verde ao vértice $r = E$, com $i_r = 6$ em T_{origem} e a posição em azul ao vértice $a = L$ com $i_a = 5$ em $T_{destino}$. As NDEs são as mesmas do exemplo para o operador 1. Ao aplicar o Passo 1, que é o mesmo para ambos os operadores, tem-se o intervalo $(2, \ldots, 6)$. As árvores correspondentes a essas NDEs estão na Figura 4.6(a). O Passo 2 também é o mesmo do operador 1 e obtém-se, novamente, a $T'_{destino}$ parcial que está na Tabela 4.6. A partir dessa etapa, tem-se a diferenciação entre os operadores 1 e 2.

Tabela 4.9: NDE *tmp* após a primeira etapa do Passo 3.

i	1	2	3	4	5
de	**3**				
v	**E**				

Primeiramente, encontra-se na NDE de T_{origem} o intervalo que corresponde a subárvore com raiz r, nesse exemplo, esse intervalo é somente a posição $i_r = 6$, pois não existe na NDE nenhuma posição à direita de i_r com profundidade maior do que do vértice r. Na seqüência, esse intervalo é copiado na NDE temporária *tmp* (ver Tabela 4.9). A próxima etapa do Passo 3 é encontrar a primeira posição em T_{origem}, entre $i_p(2)$ e $i_r(6)$, que possui profundidade menor do que $i_r(3)$ e maior do que $i_p(1)$, que é $i_k = 4$, e atribui para $i_{r1} = 4$. A seguir encontra-se o intervalo da NDE como definido no Passo 1 e copia-se em *tmp* esse

intervalo (ver Tabela 4.10), que corresponde a subárvore da Figura 4.6(c). Nessa etapa a profundidade das posições é corrigida de forma similar à definida no Passo 3 do operador 1. Agora faz-se $i_r = i_{r1}$ e o processo recomeça. O novo i_k válido é o próprio $i_p = (2)$ e, então, copia-se o intervalo (2,3) sem as posições que já foram copiadas nas etapas anteriores, ou seja, copia-se as posições 2 e 3, que correspondem à subárvore da Figura 4.6(d). Após essas etapas, obtém-se a NDE temporária da Tabela 4.11 que corresponde a subárvore da Figura 4.6(e).

Tabela 4.10: NDE *tmp* após a segunda etapa do Passo 3.

i	1	2	3	4	5
de	**3**	4	5		
v	**E**	H	F		

Tabela 4.11: NDE *tmp* após a segunda parte do Passo 3.

i	1	2	3	4	5
de	3	4	5	5	6
v	E	H	F	B	C

A partir desse ponto, os algoritmos dos operadores 1 e 2 passam a ser similares. A única diferença é que no lugar do intervalo $(i_p, \ldots, i_l)$ copia-se a NDE *tmp* para $T'_{destino}$. As NDEs resultantes podem ser vistas na Tabela 4.12 e as árvores correspondentes são apresentadas na Figura 4.6(f). Deve-se observar que esses operadores sempre produzem novas soluções factíveis sem a necessidade de algoritmos de reparo ou de verificação de ciclos.

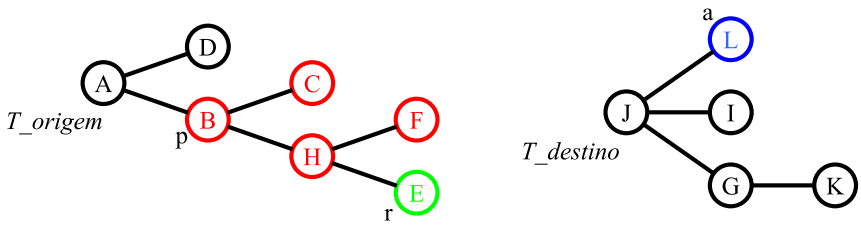
Tabela 4.12: NDEs de T'_{origem} e $T'_{destino}$ após a aplicação do operador 2.

(a) T'_{origem}

i	1	2
de	0	1
v	A	D

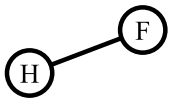
(b) $T'_{destino}$

i	1	2	3	4	5	6	7	8	9	10
de	0	1	2	1	1	2	3	4	4	5
v	J	G	K	I	L	E	H	F	B	C

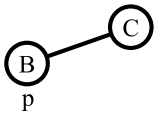


(a) Árvores representadas por T_{origem} e $T_{destino}$.

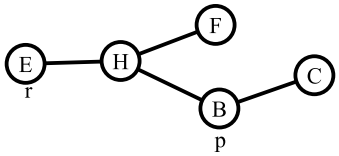
(b) Subárvore de $r0 = r = E$.



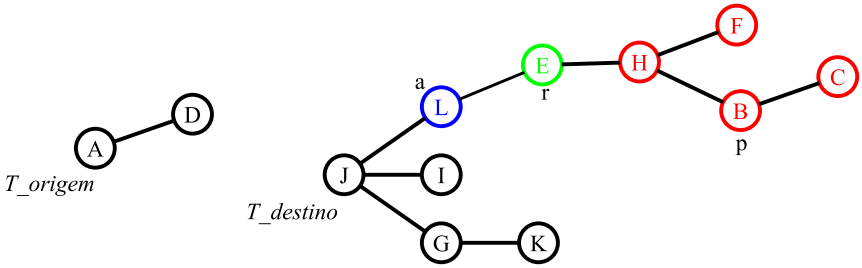
(c) Subárvore de $r1$=H.



(d) Subárvore de $r2 = p = B$.



(e) Subárvore correspondente ao *array* tmp.



(f) Árvores representadas por T'_{origem} e $T'_{destino}$.

Figura 4.6: Exemplo de aplicação do operador 2.

Aplicação dos Operadores 1 e 2 em uma Árvore

Quando a transferência da subárvore podada ocorre dentro da mesma árvore, os operadores 1 e 2 consideram T_{origem} e $T_{destino}$ como a mesma NDE. Deve-se verificar, cuidadosamente, a ordem dos índices i_p (i_r) e i_a na NDE para não copiar trechos repetidos. Por exemplo, se $i_p < i_a$, ao copiar as posições até i_a, é preciso pular o intervalo $(i_p, \ldots, i_l)$. Além disso, a não pode corresponder a um vértice no intervalo que representa a subárvore de p.

Determinação dos Vértices p, r e a

O algoritmo para determinação eficiente dos vértices p, r e a apropriados utiliza duas estruturas auxiliares: a matriz Π_{v_x} e o *array* π . A matriz Π_{v_x} contém a identificação do vértice, sua posição, sua floresta e sua árvore. Sempre que um vértice tem sua posição alterada pelos operadores 1 e 2, uma nova coluna com as informações atualizadas é inserida na matriz. O *array* π armazena, em cada posição, o ancestral da floresta representada naquela posição (Delbem et al., 2004). Os passos para a determinação dos vértices p, r e a são descritos pelo Algoritmo 4.3.2.

Algoritmo 4.3.2: - Determinação dos vértices p, r e a (Delbem et al., 2004)

//Seja T a NDE de T_{origem} (entrada)
//Sejam p, r e a os vértices escolhidos (saída)
//Seja i_x o índice do vértice x
(1) Escolha aleatoriamente um índice de T diferente da raiz e chame esse vértice de p;
(2) Se for o operador 2 escolha aleatoriamente um índice no intervalo de $(i_p, \ldots, i_l)$ e chame o vértice da posição de r;
(3) Escolha aleatoriamente um vértice da lista de adjacências de p no caso do operador 1 ou da lista de adjacências de r no caso do operador 2 e chame de a.
(4) Recupere posição de a utilizando a matrix Π_a e o *array* π. No caso de florestas com mais de uma árvore verifique se $a \notin T$, em caso afirmativo, retorne a e seu índice i_a; Para floresta com uma única árvore verifique se $i_a \notin (i_p, \ldots i_l)$, em caso afirmativo, retorne a e seu índice i_a. Caso contrário, retorne ao Passo 3.

Localização de um Vértice na NDE

Seja F_0 uma floresta de G codificada pela NDE e T_i a i-ésima árvore em F_0 e F_j a j-ésima floresta gerada depois de F_0. A localização de um vértice em F_0 utiliza uma matriz auxiliar Π_{v_x} para cada vértice v_x de G.

Para F_0, Π_{v_x} possui só uma coluna $\Pi_{v_x} = \begin{bmatrix} 0 \\ t_0 \\ i_0 \end{bmatrix}$, onde t_0 é um índice da árvore que contém v_x e i_0 é o índice correspondente a v_x no *array* T_{t_0} que é a NDE da árvore.

Suponha que uma floresta $F_{posterior}$ é obtida a partir de $F_{anterior}$, $anterior < posterior$, e que o vértice v_x pertence à subárvore transferida para uma nova árvore em $F_{posterior}$. Então, v_x deve ter uma posição em $F_{posterior}$ que é diferente de sua posição em $F_{anterior}$. Com o objetivo de armazenar essa mudança, insere-se uma nova coluna na matriz Π_{v_x} com os valores de sua nova posição. A matriz alterada resulta em:

$$\Pi_x = \begin{bmatrix} 0 & \dots & anterior & \dots & posterior \\ t_0 & \dots & t_{anterior} & \dots & t_{posterior} \\ i_0 & \dots & i_{anterior} & \dots & i_{posterior} \end{bmatrix}.$$

A atualização de colunas deve ser efetuada pelo menos para os vértices da subárvore podada. No pior caso, todos os vértices de T_{origem} e $T_{destino}$ precisam ter suas matrizes atualizadas.

Na primeira linha de Π_{v_x} é armazenado um identificador da floresta, que teve o vértice v_x alterado para uma nova posição. O último predecessor $F_{anterior}$ de $F_{posterior}$ pode ser determinado a partir de π (vetor que armazena a relação de ancestralidade (Delbem et al., 2004)). Assim, para localizar a posição de um vértice v_x em F_i, primeiro busca-se em Π_{v_x}, se F_i existe; caso contrário localiza-se em π, o predecessor de F_i chamado de F_k, com $k < i$ e novamente busca-se em Π_{v_x}. Se a busca novamente retorna um valor nulo, o processo recomeça buscando do predecessor de F_k até que seja encontrada uma coluna para v_x em alguma floresta. A busca em Π_{v_x} pode ser efetuada usando um método de busca para conjunto de dados ordenados como, por exemplo, a busca binária (Cormen et al., 2000), pois as colunas são inseridas de forma ordenada pelo identificador da floresta.

Decodificação

A decodificação de uma floresta geradora F representada na NDE é realizada aplicando o Algoritmo 4.3.3 a NDE de cada uma de suas árvores.

Algoritmo 4.3.3: Decodificação da NDE.

//Seja T_x a NDE a ser decodificada (entrada)
//Seja $|T_x|$ o tamanho da NDE T_x
//Sejam $V(T)$ e $E(T)$ os conjuntos de vértices e arestas que definem a árvore codificada (saída)
//Seja *raizes* um *array* de tamanho $|T_x|$ que armazena as raízes de subárvores
(1) $V(T) = \emptyset$
(2) $E(T) = \emptyset$
(3) Se $|T_x| > 0$
    (4) Inclua vértice v_i em $V(T)$
(5) $raizes_1 = v_1$
(6) $i = 2$ //(segunda posição da NDE T_x)
(7) Enquanto $i < |N|$
    (8) Incluí o vértice v_i em $V(T)$
    9. $raizes_{de_i} = v_i$
    10. Incluí a aresta $(raizes_{de_i - 1}, v_i)$ em $E(T)$
    11. $i = i + 1$

4.4 Comparação das Representações

As representações para serem eficientes quando aplicadas a EAs devem possuir um conjunto de propriedades, dentre as quais podem ser destacadas: tendência, hereditariedade, localidade e redundância. Além disso, a complexidade de espaço para representar a árvore e a complexidade de tempo para codificação, decodificação e aplicação dos operadores são também um forte indicativo da eficiência da representação. A Tabela 4.13 apresenta um resumo das principais propriedades das representações de PPRs para EAs. O cálculo da complexidade da NDE é apresentado no Apêndice B. A eficiência das demais representações na Tabela 4.13 é apresentada por Raidl e Julstrom (2001).

Tabela 4.13: Complexidade de espaço e tempo, factibilidade, localidade e hereditariedade.

Representação	Espaço	Tempo [1]	Factibilidade [2]		Localidade/ Hereditariedade
			Completo	Esparso	
CV (Seção 4.1.1)	$O(m)$	$O(m)$	Sim	Não	Alta
Predecessores (Seção 4.3.1)	$O(n)$	$O(n^2)$	Sim	Não	Alta
Número de Prüfer (Seção 4.1.2)	$O(n)$	$O(n \log n)$	Sim	Não	Baixa
NetKeys (Seção 4.1.3)	$O(m)$	$O(m \log m)$	Sim	Sim	Alta
Conjunto de Arestas (Seção 4.2.1)	$O(n)$	$O(n)$	Sim	Sim	Alta
NDE (Seção 4.3.2)	$O(\lvert T_{origem}\rvert + \lvert T_{destino}\rvert + t^3)$	$O(\lvert T_{origem}\rvert + \lvert T_{destino}\rvert + t)$	Sim	Sim	Alta

[1] Complexidade de tempo necessária para obter somente redes factíveis em grafos completos.
[2] Capacidade de gerar somente redes factíveis para grafos completos e não-completos.
[3] t é o número de florestas, isto é, tamanho de F.

Com base na Tabela 4.13, observa-se que as representações Conjunto de Arestas e NDE podem ser consideradas as mais adequadas, uma vez que são eficientes (no pior caso, $O(|T_{origem}| + |T_{destino}| + t) = O(n)$) e podem lidar com grafos completos ou não, além de possuírem alta localidade e hereditariedade. A propriedade de factibilidade para grafos completos e não-completos é muito importante para representações em EAs aplicados para PPRs do mundo real, pois vários desses problemas são representados por grafos não-completos, esparsos ou densos (Cormen et al., 2000), uma vez que diversas conexões em uma rede real grande podem ser fisicamente inviáveis ou muito custosas. Com base na formulação original, as representações NetKeys e Conjunto de Arestas podem ser consideradas capazes de trabalhar com grafos não-completos. No entanto, demandam um custo computacional extra para garantir a produção de soluções factíveis.

4.5 Considerações Finais

Este Capítulo apresentou as principais representações utilizadas para EAs aplicados a PPRs. As representações podem ser classificadas em diretas, indiretas ou mistas, de acordo com a codificação utilizada no cromossomo e nos operadores de recombinação e mutação.

Deve-se destacar que a NDE tem apresentado resultados relevantes para diversos PPRs de larga-escala, podendo trabalhar adequadamente com PPRs correspondentes a florestas geradoras, bem como lidar com redes modeladas por grafos não-completos. No Capítulo 5 é apresenta a representação Nó-Profundidade-Grau desenvolvida a partir da NDE.

Capítulo

5

Representação Nó-Profundidade-Grau

Este Capítulo propõe a estrutura de dados chamada representação nó-profundidade-grau (NDDE, do inglês *Node-Depth-Degree Encoding*) desenvolvida nesta pesquisa. A NDDE consiste no desenvolvimento de operadores de recombinação e modificações dos operadores 1 e 2 da representação nó-profundidade (NDE, ver Seção 4.3.2). Esses operadores modificados para a NDDE são denominados PAO (do inglês, *Preserve Ancestor Operator*) e CAO (do inglês, *Change Ancestor Operator*). O uso de operadores de recombinação busca a redução do tempo de convergência de EAs para PPRs. Os operadores PAO e CAO buscam diminuir a complexidade de tempo em relação aos operadores 1 e 2 desenvolvidos para a NDE.

A Seção 5.1 apresenta as melhorias nos operadores 1 e 2 da NDE que resultaram nos operadores PAO e CAO. A Seção 5.2 apresenta a formulação dos operadores de recombinação para a NDDE que estão entre as principais contribuições desta pesquisa. Seção 5.2.1 apresenta os operadores de recombinação para grafos completos. A Seção 5.2.2 propõe um operador para a NDDE (denominado EHR) capaz de recombinar florestas de grafos tanto completos quanto não-completos de forma eficiente sem necessitar de algoritmos de reparos. A Seção 5.3 apresenta as conclusões deste Capítulo.

5.1 Construindo PAO e CAO para a NDDE

As melhorias básicas nos operadores 1 e 2 da NDE foram inicialmente propostas pelos pesquisadores Alexandre C. B. Delbem e Guilherme P. Telles, que elaboraram a base da NDDE apresentada no Apêndice B. Neste trabalho vários aspectos dos operadores foram aperfeiçoados e corrigidos, além disso foram definidas formas mais adequadas para a implementação das melhorias.

A seguir, é apresentado o resultado final que produziu os operadores PAO e o CAO. Como parte das modificações, a codificação de uma árvore pela NDDE contém, além dos valores de vértice e profundidade, o grau do vértice. Assim, cada vértice é representado pela trinca ordenada *(nó,profundidade,grau)*.

Algoritmo 5.1.1: - Pseudocódigo de PAO.

//Seja F a estrutura para a floresta geradora com os ponteiros para as NDDEs (entrada)
//Seja F' a estrutura para a floresta geradora resultante com os ponteiros para as NDDEs (saída)
//Sejam T_{origem} e $T_{destino}$ as NDDEs das árvores dos vértices p e a, respectivamente
//Sejam T'_{origem}, $T'_{destino}$ as NDDEs das novas árvores após a aplicação da operação.
//Seja $|T_x|$ o tamanho da NDDE de T_x
//Seja i_p e i_a índices dos vértices nas NDDEs (entrada)
(1) Determine o intervalo $(i_p, \ldots, i_l)$ em T_{origem} percorrendo a NDDE, a partir de i_p, enquanto $de_i > de_{i_p}$ e $i < |T_{origem}|$, faça $i_l = i - 1$;
//o intervalo $(i_p, \ldots, i_l)$ correspondente aos vértices da subárvore enraizada em p;
(2) Copie em $T'_{destino}$ as posições de $T_{destino}$ de $i = 1$ até i_a. Incremente o grau de i_a por 1, $deg_{i_a} = deg_{i_a} + 1$;
(3) Copie os dados de $(i_p, \ldots, i_l)$ para $T'_{destino}$ a partir da posição $i_a + 1$ e atualize os valores de profundidade desses vértices $de_x = de_x - de_{i_p} + de_{i_a} + 1$;
(4) Copie em $T'_{destino}$, a partir de $i_a + i_l - i_p + 1$, as posições de $T_{destino}$ de $i = i_a + 1$ até $|T_{destino}|$;
(5) Copie em T'_{origem} as posições de T_{origem} que não pertencem ao intervalo $(i_p, \ldots, i_l)$. O grau do ancestral de p deve ser diminuído de um;
(6) Copie a floresta da estrutura de dados F e gere F' alterando os ponteiros das NDDEs T_{origem} e $T_{destino}$ para os ponteiros das NDDEs T'_{origem} e $T'_{destino}$, respectivamente. Calcule o número de vértices adjacentes a T'_{origem} e $T'_{destino}$ e armazene na estrutura F';
(7) Atualize as matrizes Π_{v_x} para cada v_x em T'_{origem} e $T'_{destino}$.

Assim como os operadores 1 e 2 na NDE, PAO e CAO na NDDE transferem subárvores de uma árvore T_{origem} para uma árvore $T_{destino}$. Em PAO, a raiz da subárvore transferida não é alterada. Em CAO, a subárvore transferida passa a ter como nova raiz que é outro vértice da subárvore podada diferente da raiz anterior. Assim, PAO gera, em geral, pequenas mudanças nas árvores; enquanto CAO pode produzir mudanças maiores e mais complexas.

Observe que o conjunto das NDDEs (árvores) representando uma floresta é referenciado por uma estrutura F que armazena um ponteiro para cada NDDE. Além disso, F

armazena a soma do número de vértices adjacentes no grafo para cada vértice de cada NDDE. Essa informação é importante para garantir a eficiência computacional de PAO e CAO para quaisquer tipo de grafo, conforme apresentado no Apêndice B. Essa constitui uma das principais diferenças dos operadores 1 e 2 para PAO e CAO. A informação dos vértices adjacentes de uma árvore em conjunto com o grau do vértice possibilitam determinar, de forma eficiente, um vértice a adjacente a p (r) tal que a está em uma árvore diferente de p (ver Apêndice B).

A seguir são apresentados os algoritmos dos operadores de PAO e CAO. Considere que o vértice p e seu índice i_p em T_{origem} e o vértice a e seu índice i_a em $T_{destino}$ já foram adequadamente escolhidos. O Algoritmo 5.1.1 apresenta o pseudocódigo do operador PAO.

Para CAO, são necessários três vértices como entrada: o vértice de corte p, o novo vértice raiz r e o vértice adjacente a com seus respectivos índices i_p, i_r e i_a, sendo que a em $T_{destino}$ deve ser adjacente a r, com p e r em T_{origem}. O Algoritmo 5.1.2, descreve os passos para aplicação de CAO considerando que esses vértices já foram previamente encontrados.

Outra diferença entre dos operadores 1 e 2 para PAO e CAO é referente a escolha dos vértices p, r e a. O procedimento para esse Passo é apresentado na Seção 5.1.1.

5.1.1 Obtenção Eficiente dos Vértices p, a e r

Um aspecto importante para a eficiência de algoritmos usando PAO e CAO é a obtenção de forma eficiente os vértices necessários para aplicação desses operadores. No caso de PAO, necessita-se selecionar dois vértices p e a que podem ser determinados pelo Algoritmo 5.1.3.

Para CAO são necessários 3 vértices p, r e a. O Algoritmo 5.1.4 apresenta os passos para obter os vértices CAO de forma eficiente.

A eficiência dos algoritmos de PAO (CAO) depende de como é executado o Passo 3 do Algoritmo 5.1.3 (Passo 3 (ii) do Algoritmo 5.1.4). Para a implementação desse passo são necessários três *arrays*: L_G, L_O e L_T, descritos a seguir:

- L_G tem tamanho n (número de vértices) e armazena $N_G(p)$[1] (p deve ser trocado por r quando se considera o Passo 3(ii) do Algoritmo 5.1.4). A codificação é tal que $L_G[i] = i$ se i está em $N_G(p)$, caso contrário $L_G[i] = -1$;

[1] $N_G(p)$ é o conjunto dos vértices adjacentes de p em G

Algoritmo 5.1.2: - Pseudocódigo de CAO

//Seja F a estrutura para a floresta geradora com os ponteiros para as NDDEs (entrada)
//Seja F' a estrutura para a floresta geradora resultante com os ponteiros para as NDDEs (saída)
//Sejam T_{origem} e $T_{destino}$ as NDDEs das árvores dos vértices p e a, respectivamente
//Sejam T'_{origem}, $T'_{destino}$ as NDDEs das novas árvores após a aplicação da operação.
//Seja $|T_x|$ o tamanho da NDDE de T_x
//Seja i_p, i_r e i_a índices dos vértices nas NDDEs (entrada)
(1) Determine o intervalo $(i_r, \ldots, i_{lr})$ em T_{origem} percorrendo a NDDE, a partir de i_p, enquanto $de_i > de_{i_p}$ e $i < |T_{origem}|$, faça $i_l = i - 1$. Esse Passo é similar ao Passo 1 de PAO;
//o intervalo $(i_r, \ldots, i_{lr})$ correspondente aos vértices da subárvore enraizada em r;
(2) Copie em $T'_{destino}$ as posições de $T_{destino}$ de $i = 1$ até i_a. Incremente o grau de i_a por 1, $deg_{i_a} = deg_{i_a} + 1$;
(3) Copie os dados de $(i_r, \ldots i_{lr})$ para uma NDDE temporária T_{tmp1} e atualize os valores de profundidade $de_{v_x} = de_{v_x} - de_{i_r} + de_{i_a} + 1$;
(4) Considere os índices entre r e p em T_{origem}, que possuem $de_{i_p} \leq de_i < de_{i_r}$, como $i_{r_0}, i_{r_1}, \ldots, i_{r_{\hat{n}}}$, $i_r = i_{r_0}$ e $i_p = i_{r_{\hat{n}}}$. Para encontrar os índices i_{r_y}, começando de $y = 0$ faça $i_k = i_{r_y}$ e decremente i_k até $de_{i_k} = de_{i_{r_y}} - 1$ ou $i_k = i_p$, e faça $i_{r_{y+1}} = i_k$. Construa T_{tmp2} copiando os intervalos dos i_{r_y}, calculados como no Passo 1, sem o intervalo de $i_{r_{y-1}}$, com y começando de 1, e atualize a profundidade dos vértices por $de_x = de_x - de_{i_{r_y}} + i + de_{i_a} + 1$. O grau de r deve ter um incremento de um e o grau de p deve ter um decremento de um;
(5) Copie os dados de $[T_{tmp1}, T_{tmp2}]$ para $T'_{destino}$ a partir da posição $i_a + 1$. O grau de a deve ser incrementado de um;
(6) Copie em $T'_{destino}$, a partir de $i_a + i_l - i_p + 1$, as posições de $T_{destino}$ de $i = i_a + 1$ até $|T_{destino}|$;
(7) Construa a nova representação T'_{origem} removendo os dados de T_{tmp1} e T_{tmp2} de T_{origem}. O grau do ancestral de p deve ser diminuído de um;
(8) Copie a floresta da estrutura de dados F e gere F' alterando os ponteiros das árvores T_{origem} e $T_{destino}$ para os ponteiros das árvores T'_{origem} e $T'_{destino}$ respectivamente. Calcule o número de vértices adjacentes a T'_{origem} e $T'_{destino}$ e armazene na estrutura F';
(9) Atualize as matrizes Π_{v_x} para cada v_x em T'_{origem} e $T'_{destino}$.

- L_O um *array* circular de comprimento $deg_G(p)$, cujos elementos são uma permutação dos índices i de L_G tal que $L_G[i] \neq -1$;

- L_T um *array* auxiliar binário com comprimento n e usado para marcar os vértices de G que estão em uma NDDE.

O pseudocódigo do Algoritmo 5.1.5 mostra como obter um vértice a adequado de forma eficiente. A Figura 5.1 ilustra como usam-se L_O, L_G e L_T para obter um vértice a adequado.

Na Figura 5.1(a) têm-se um grafo G não-completo de 4 vértices e na Figura 5.1(b) uma floresta geradora de G com duas árvore T_1 e T_2. Supondo C o vértice p, a Figura 5.1(c) apresenta os *arrays* $L_G[C]$, L_O e L_{T_1}. Em L_O armazena-se a permutação das 3 posições de $L_G[C]$ que são diferentes de -1. E L_{T_1} preenche com 1 os adjacentes de B que estão em T_1. Construídos os *arrays*, seleciona-se uma posição aleatória em L_O, supondo essa posição 2, busca-se em $L_G[C]$ o vértice D e em seguida verifica-se em L_{T_1}

Algoritmo 5.1.3: - Obtenção dos vértices p e a para PAO.

//Seja F a estrutura com os ponteiros para as NDDEs das árvores (entrada)
//Seja $deg_T(p)$ o grau de p na árvore T e $deg_G(p)$ o grau de p na no grafo G
//Seja $N_T(p)$ o conjunto de vértices adjacentes de p que estão em T_{origem} e $N_G(p)$ o conjunto de vértices adjacentes a p em G
//Seja T_{origem} e $T_{destino}$ as NDDEs para aplicação de PAO (saída)
(1) Escolha de forma aleatória uma árvore T_{origem} na floresta F que tenha pelo menos uma aresta incidente em um vértice de outra árvore. Se tal árvore não existe, o próprio G é uma floresta geradora;
(2) Escolha de forma aleatória um vértice em T_{origem}, para chamar de p, diferente da raiz, tal que $deg_T(p) < deg_G(p)$;
(3) Selecione aleatoriamente um vértice a de $N_G(p) \setminus N_T(p)$;
(4) Determine a posição de a na estrutura da floresta utilizando a matriz Π_a e o *array* π e retorne $T_{destino}$ e o índice de a. //(ver Seção 4.3.2).

Algoritmo 5.1.4: - Obtenção dos vértices p, r e a para CAO

//Seja F a estrutura com os ponteiros para as NDDEs das árvores (entrada)
//Seja $deg_T(r)$ o grau de r na árvore T e $deg_g(r)$ o grau de r na no grafo G.
//Seja $N_T(r)$ o conjunto de vértices adjacentes de r que estão em T_{origem} e $N_G(r)$ o conjunto de vértices adjacentes a r em G.
//Sejam de_p e de_r as profundidades dos vértices p e r.
//Seja T_{origem} e $T_{destino}$ as NDDEs para aplicação de CAO (saída)
(1) Escolha aleatoriamente uma árvore T_{origem} na floresta F que tenha pelo menos uma aresta incidente em um vértice em outra árvore. Se tal árvore não existe G é a própria floresta;
(2) Escolha aleatoriamente um vértice em T_{origem}, para chamar de r, diferente da raiz, tal que $deg_T(r) < deg_G(r)$;
(3)

(i) Selecione de forma aleatória um vértice p de T_{origem}, diferente da raiz, tal que $de_p < de_r$. Se este vértice não existe, faça $p = r$;

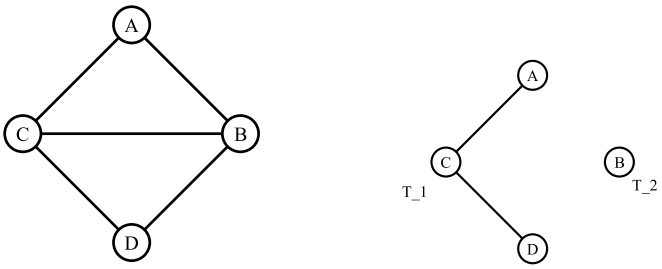
(ii) Selecione de forma aleatória um vértice a de $N_G(r) \setminus N_T(r)$;

(4) Determine o índice de a na estrutura da floresta utilizando a matriz Π_a e o *array* π definidos para a NDE e retorne $T_{destino}$ e o índice i_a //(ver Seção 4.3.2).

que D é um vértice marcado (linha verde). A busca pelo vértice a adequado continua incrementando o índice de L_O e agora têm-se o vértice B, que em L_{T_1} corresponde a um vértice não-marcado (linha tracejada em vermelho) e assim chama-se $a = B$.

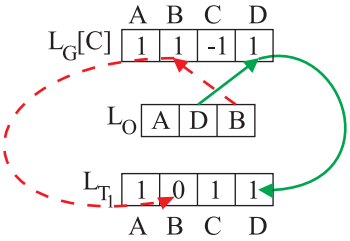
Algoritmo 5.1.5: - Detalhamento do passo 3 do Algoritmo 5.1.3 e 3(ii) do Algoritmo 5.1.4

//Seja T_{origem} uma NDDE (entrada)
//Seja n o número de vértices do grafo
//Seja $L_G(p)$ ($L_G(r)$) um *array* com os vértices adjacentes de $N_G(p)$ ($N_G(r)$) (entrada)
//Seja L_O é um *array* com a permutação dos índices i de L_G (entrada)
//Seja L_T é um *array* auxiliar binário de comprimento n
//Seja a o vértice adjacente (saída)
(1) Percorra a NDDE de T_{origem} marcando em L_T cada vértice v_x da NDDE que está em $L_G(p)$;
(2) Selecione aleatoriamente um inteiro i no intervalo $1 \ldots deg_G(p)$;
(3) Se $L_T[L_G[L_O[i]]]$ é não marcado, chame $L_G[L_O[i]]$ de a e vá para o Passo 4; senão continue no Passo 3 até percorrer todo o *array* L_T;
(4) Desmarque os vértices da vizinhança de p em L_T utilizando a NDDE.



(a) Grafo G.

(b) Floresta F com as árvore T_1 e T_2.



(c) *Arrays* L_G, L_O e L_T.

Figura 5.1: Exemplo de utilização dos *arrays* L_G, L_O e L_T.

Para implementação do Passo 3(i) do Algoritmo 5.1.4 é utilizado um *array* circular L_D de zeros com tamanho $i_r - 1$, onde i_r é o índice do vértice r. O passos para encontrar o vértice p são definidos pelo Algoritmo 5.1.6.

O Passo 4 dos Algoritmos 5.1.3 e 5.1.4 pode ser uma etapa de alta complexidade do algoritmo dependendo da forma como for implementada. Na NDE e também na NDDE é utilizada uma estrutura auxiliar para cada vértice mantendo atualizada a sua posição em cada indivíduo (floresta geradora). Essa estrutura é a mesma descrita para a NDE na Seção 4.3.2. A cada novo indivíduo gerado têm-se um crescimento da matriz Π_{v_x} e do

Algoritmo 5.1.6: - Detalhamento do Passo (3)(i) do Algoritmo 5.1.4

//Seja L_D o *array* de zeros com tamanho $i_r - 1$ (entrada)
//Seja de_i a profundidade do vértice i
//Seja p um vértice (saída)
(1) Faça $p = 0$;
(2) Percorra a NDDE da posição 1 até i_r marcando para cada posição i, $L_D[de_i] = i$ se $de_i < de_r$;
(3) Escolha aleatoriamente j entre 1 e $i_r - 1$;
(4) Percorra L_D até encontrar um $L_D[k] \neq 0$ fazendo, nesse caso, $p = L_D[k]$, ou percorra todo o *array*;
(5) Se $p == 0$, faça $p = r$.

arrayπ. Porém, esse crescimento pode ser limitado por um procedimento de reinicialização aplicado após $k\sqrt{n}$ novas florestas terem sido construídas, $k > 1$. O Apêndice B mostra como o custo desse processo de reinicialização pode ser amortizado, tornando o uso da matriz Π_{v_x} computacionalmente eficiente.

Aplicação de PAO e CAO com uma Árvore

PAO e CAO também podem ser aplicados utilizando somente uma árvore, para isso são necessárias algumas alterações nos algoritmos apresentados anteriormente. Para aplicar os Algoritmos 5.1.1 e 5.1.2 com uma única árvore considere $T_{destino}$ como T_{origem}, $T'_{destino}$ como T'_{origem} e ignore o Passo 6 do primeiro e no segundo o Passo 7.

Nos Algoritmos 5.1.3 e 5.1.4 alguns passos são modificados. Primeiramente, no Passo 1 se a estrutura for uma floresta com mais de uma árvore é suficiente escolher uma árvore de forma aleatória sem a necessidade de verificar a existência de um adjacente fora da árvore. O outro caso é quando a estrutura da floresta é composta por uma única árvore e, nesse caso, não existe escolha de árvore.

Para o Passo 3 do Algoritmo 5.1.3 existem dois casos a serem considerados: quando o grafo é completo e quando o grafo é não-completo. O procedimento para cada caso é definido a seguir:

1. **Grafo completo**: utiliza-se L_O com tamanho n para acessar os índices dos vértices na NDDE. Seleciona-se de forma aleatória um inteiro i no intervalo $1 \ldots n$. Se $L_O[i]$ não está no intervalo $[i_p, i_l]$, onde i_p é índice de p e i_l é índice de l (último vértice da subárvore de p) e $L_O[i]$ é diferente do índice do ancestral de p então chame de i_a o índice presente em $L_O[i]$. Caso contrário, incremente i até percorrer todo L_O. O procedimento é o mesmo para os Passos 3(i) ou 3(ii), utilizando em ambos os casos p e l como fim da subárvore de p, pois mesmo no caso de CAO, a subárvore total a ser transferida é a subárvore de p e o vértice a não pode estar nesse intervalo.

2. **Grafo não-completo**: utiliza-se o Algoritmo 5.1.5, porém, em vez de chamar de a um vértice *não-marcado* no Passo 3 deve-se chamar $L_G[L_O[i]]$ de a se $L_T[L_G[L_O[i]]]$ estiver **marcado.**

O Algoritmo 5.1.6 é utilizado para selecionar o vértice p mesmo quando o operador CAO é aplicado para efetuar a transferência da subárvore para outro ponto da mesma árvore.

5.2 Operadores de Recombinação

Operadores de recombinação são muito importantes no desenvolvimento de EAs, pois preservam informações existentes na população ao mesmo tempo que exploram novas soluções obtidas da combinação de características de ambos os pais. Além disso, o uso de recombinação em um EA tende a diminuir o tempo de convergência do mesmo (De Jong, 2006; Michalewicz e Fogel, 2004).

São propostas três estratégias para o desenvolvimento de operadores de recombinação para a NDDE. Duas estratégias são específicas para grafos completos e são descritas na Seção 5.2.1. A terceira estratégia descrita na Seção 5.2.2, busca desenvolver um operador de recombinação que possa ser utilizado tanto para grafos completos quanto não-completos.

5.2.1 Recombinação para Grafos Completos - NOX e NPBX

No caso de representações utilizadas em EAs para problemas de permutação, assim como no caso da NDDE para PPRs, a ordem em que os genes estão dispostos define como é o fenótipo codificado pelo genótipo. Além disso, em ambos os casos não pode haver a repetição de valores, no caso da NDDE os vértices não podem ser repetidos. Assim, os operadores de recombinação para a NDDE são baseados em operadores de recombinação presentes na literatura para os problemas de permutação (ver Seção 3.3.2).

Dentre os diversos operadores de recombinação apresentados para as permutações, selecionou-se para essa pesquisa os operadores: crossover baseado em ordem (OX) e crossover baseado em posição (PBX). Em uma análise inicial, ambos são os que apresentam uma conversão mais direta para serem adaptados a NDDE. Com isso, obtém-se dois operadores de recombinação para a NDDE que são chamados de NOX (NDDE OX), que é baseado no OX, e NPBX (NDDE PBX) que é baseado em PBX. Esses operadores são descritos a seguir.

NOX

O operador de recombinação NOX é inspirado no operador de crossover de ordem (OX). O operador de recombinação OX seleciona dois pontos de corte e, então a região de corte definida por esses pontos é copiada do pai para compor o filho. As demais posições do filho são preenchidas, a partir do segundo ponto de corte, pelas posições do outro pai considerando os cromossomos como *arrays* circulares. Essas posições do outro pai podem conter genes da região de corte assim, esses genes são pulados no pai e os genes seguintes são utilizados para preencher as posições relativas a eles no filho (ver Figura 3.1 e Seção 3.3.2).

Para a formulação do NOX considere: P_1 e P_2 as NDDEs dos pais, O_1 e O_2 as NDDEs dos filhos, A_1 e A_2 *arrays* auxiliares e que a floresta possui uma única árvore. O pseudo-código do Algoritmo 5.2.1 apresenta os passos para aplicação do operador NOX na NDDE.

Algoritmo 5.2.1: Operador de Recombinação NOX.

//Sejam P_1 e P_2 as NDDEs dos indivíduos pais (entrada)
//Sejam O_1 e O_2 as NDDEs dos indivíduos filhos resultado da aplicação de NOX (saída)
//Sejam A_1 e A_2 os *arrays* auxiliares para marcação
//Gene significa a trinca da NDDE
(1) Determine de forma aleatória os dois pontos de corte;
(2) Marque em A_2 (A_1) os vértices que estão na região entre os pontos de corte em P_1 (P_2);
(3) Preencha os genes de O_1 (O_2) com os genes de P_2 (P_1), ignorando os vértices marcados, até o primeiro ponto de corte em O_1. Atribua 1 ao grau do gene i e incremente o grau do gene que contém o vértice raiz de i. Para o gene da primeira posição atribua profundidade 0 e para o gene da segunda posição profundidade 1. Se necessário aplique o processo de correção de profundidade;
(4) Preencha a região de corte em O_1 (O_2) com os genes da região de corte de P_1 (P_2). Atribua 1 ao grau do gene i e incremente o grau do gene que contém o vértice raiz de i. Se necessário aplique o processo de correção de profundidade;
(5) Preencha os genes de O_1 (O_2) depois do segundo ponto de corte, com os genes de P_2 (P_1), a partir da última posição utilizada no Passo 3, ignorando os vértices marcados. Atribua 1 ao grau do gene i e incremente o grau do gene que contém o vértice raiz de i. Se necessário aplique o processo de correção de profundidade;
(6) Verifique se foram gerados indivíduos com redundância (ver Seção 6.1) e aplique o método de correção da mesma;

Destacam-se os seguintes pontos como diferenças entre os operadores de recombinação NOX e OX :

1. Em NOX, às duas primeiras posições do *array* da NDDE possuem valor fixo para a profundidade. A primeira posição deve ser sempre 0 e a segunda posição sempre 1;

2. Em NOX, utilizam-se as posições à partir da primeira posição do cromossomo para completar o filho nas posições externas à região de corte e não a partir do segundo ponto de corte como ocorre em OX. Com isso, evitam-se NDDEs que correspondam

a árvores com raízes diferentes, garantindo também as profundidades 0 e 1 para as duas primeiras posições do cromossomo.

Com o objetivo de garantir sempre a criação de NDDEs válidas, após a aplicação de NOX, pode ser necessário aplicar um método de correção da profundidade, ver Passos 3, 4 e 5 do Algoritmo 5.2.1. O método de correção deve ser aplicado quando, ao copiar os genes dos pais, acontecer de de_i e de_{i-1} possuírem uma diferença maior que 1 ($de_i - de_{i-1} > 1$). A correção da profundidade consiste em transformar o valor da profundidade de_i inválida em um valor válido fazendo $de_i = de_{i-1} + 1$.

O valor do grau na NDDE não deve ser copiado diretamente dos pais e sim recalculado durante a aplicação do operador. O cálculo do grau é efetuado atribuindo 1 ao grau do gene na posição i, ($deg_1 = 1$) e grau do seu vértice raiz na posição x da NDDE é alterado da seguinte forma ($deg_x = deg_x + 1$). O vértice raiz v_x de um vértice v_i é aquele que na NDDE possui profundidade $de_{v_x} = de_{v_i} - 1$. O índice x do vértice raiz pode ser obtido mantendo um *array* auxiliar que armazena na posição k o índice do último gene com a profundidade k e atribui-se a x o índice armazenado na posição $k = de_i - 1$.

Esse processo de cálculo do valor de grau necessita de $2n - 2$ acessos, visto que em uma árvore existem $n - 1$ arestas que conectam dois vértices e o valor de grau precisa ser atualizado sempre que a inserção de um vértice gera uma nova aresta na árvore.

Na seqüência apresenta-se um exemplo da aplicação de NOX para uma NDDE de uma floresta com uma única árvore. Com o objetivo de facilitar o entendimento e para evitar erros de interpretação com relação ao que é índice, profundidade e vértice, os vértices foram representados por letras.

Considere para o exemplo os cromossomos pais (P_1 e P_2) das Tabelas 5.1(a) e 5.1(b). As árvores codificadas por esses cromossomos estão presentes nas Figuras 5.2(a) e 5.2(b). O primeiro ponto de corte foi escolhido entre as posições 2 e 3. O segundo ponto de corte foi selecionado entre as posições 6 e 7. Assim, a região de troca é composta pelas posições 3, 4, 5 e 6 dos cromossomos.

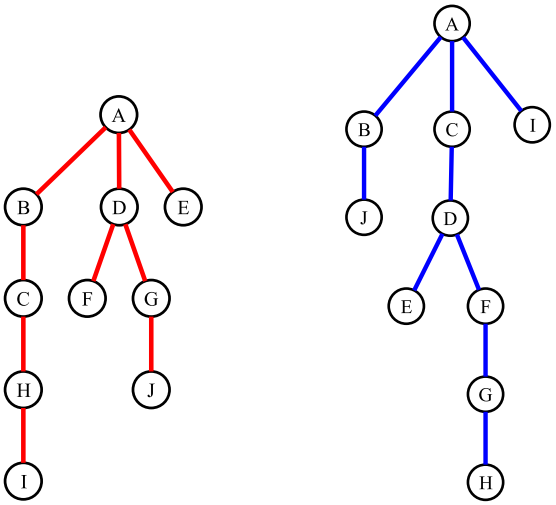
Tabela 5.1: NDDEs dos indivíduos pais.

(a) NDDE de P_1.

i	1	2	3	4	5	6	7	8	9	10
de	0	1	2	3	4	1	2	2	3	1
v	A	B	C	H	I	D	F	G	J	E
deg	3	2	2	2	1	3	1	2	1	1

(b) NDDE de P_2.

i	1	2	3	4	5	6	7	8	9	10
de	0	1	2	1	2	3	3	4	5	1
v	A	B	J	C	D	E	F	G	H	I
deg	3	2	1	2	3	1	2	2	1	1



(a) Árvore de P_1. (b) Árvore de P_2.

Figura 5.2: Árvores representadas pelos pais P_1 e P_2 utilizados no exemplo da recombinação por NOX.

No *array* auxiliar A_1, marcam-se os vértices J, C, D e E [2] que pertencem a região de corte em P_2. No *array* auxiliar A_2, marcam-se os vértices C, H, I, D da região de P_1. As Tabelas 5.2(a) e 5.2(b) representam os *arrays* auxiliares A_1 e A_2 respectivamente.

Tabela 5.2: *Arrays* auxiliares do NOX.

(a) *Array* auxiliar A_1 do NOX.

A	B	C	D	E	F	G	H	I	J
		X	X	X					X

(b) *Array* auxiliar A_2 do NOX.

A	B	C	D	E	F	G	H	I	J
		X	X				X	X	

Após marcar os vértices nos *arrays* auxiliares, o próxima Passo do algoritmo é construir os novos indivíduos. Para obter O_1 até o primeiro ponto de corte, copiam-se as posições 1 e 2 de P_2. No Passo 4 copia-se de P_1 os genes da região de corte que são as posições de 3 a 6. Por fim, o Passo 5 preenche as posições de O_1 depois da região de corte com as posições 3, 6, 7 e 8 de P_2. Ressalta-se que ao copiar posições de P_2 são ignoradas as posições referentes a vértices marcados em A_2, por isso os índices não são lineares. No caso de O_1 não foi necessário aplicar o mecanismo de correção da profundidade e a verificação de redundância não encontra nenhum vértice em posição que torne a NDDE obtida redundante.

[2] Para simplificar o exemplo, os *arrays* A_1 e A_2 são indexados pelas letras em ordem alfabética.

Para obter O_2 no Passo 3 são copiados os genes das posições 1 e 2 de P_1. No Passo 4 copiam-se os genes da região de corte, que correspondem as posições de 3 a 6 do P_2. O Passo 5 termina de preencher O_2 com os genes das posições 4, 5, 7 e 8 do P_1. Novamente, não foi necessário aplicar o mecanismo de correção da profundidade e também o indivíduo obtido não possui nenhum vértice em posição que cause a redundância da representação. As NDDEs obtidas após a aplicação de NOX podem ser vistas nas Tabelas 5.3(a) e 5.3(b) e as árvores codificadas pelas NDDEs na Figura 5.3.

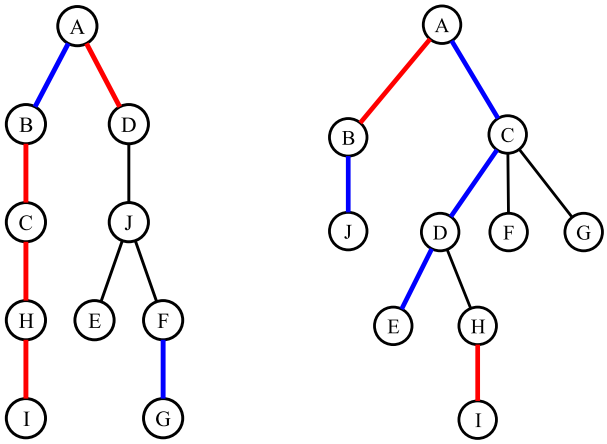
Tabela 5.3: NDDEs dos indivíduos obtidos da aplicação de NOX.

(a) NDDE de O_1.

i	1	2	3	4	5	6	7	8	9	10
de	0	1	2	3	4	1	2	3	3	4
v	A	B	C	H	I	D	J	E	F	G
deg	2	2	2	2	1	2	3	1	2	1

(b) NDDE de O_2.

i	1	2	3	4	5	6	7	8	9	10
de	0	1	2	1	2	3	3	4	2	2
v	A	B	J	C	D	E	H	I	F	G
deg	2	2	1	4	3	1	1	2	1	1



(a) Árvore de O_1. (b) Árvore de O_2.

Figura 5.3: Árvores representadas pelos filhos obtidos usando o NOX.

Observa-se que a maioria das arestas dos filhos está presente nos pais (arestas em azul e vermelho na Figura 5.3). No entanto, cada indivíduo possui três arestas que não estavam presentes em nenhum dos pais (ver arestas em preto na Figura 5.3). Esse fato será avaliado mais profundamente na análise de propriedades da NDDE (ver Capítulo 6). Ressalta-se que nesse exemplo, não foi necessário nenhum mecanismo de correção para evitar a redundância.

O Algoritmo 5.2.1 pode ser aplicado a florestas com uma árvore. Isso não significa que o NOX não possa ser aplicado para florestas com mais de uma árvore. Para isso, considera-se um *array* virtual que concatena as NDDEs de todas as árvores. Então, aplica-se o Algoritmo 5.2.1 ao *array* virtual. Além disso, devem ser considerados fixos os valores das profundidades das duas primeiras posições de cada NDDE. Após a aplicação de NOX, as NDDEs que compõem o *array* virtual devem ser separadas adequadamente, respeitando possíveis novos tamanhos para as mesmas, visto que as árvores podem possuir tamanhos diferentes em cada indivíduo pai. O separador de cada NDDE (árvore) pode ser identificado pelo vértice raiz que é mantido fixo para cada árvore em todos os indivíduos.

NPBX

O operador de recombinação NPBX corresponde a uma adaptação do operador de crossover baseado em posição (PBX) para a NDDE. O PBX seleciona um conjunto de posições de forma aleatória e copia os genes dessas posições de um pai para o filho. Em seguida o PBX remove os genes das posições selecionadas de um pai P_1 (P_2) no outro pai P_2 (P_1). As demais posições dos filhos são preenchidas com os genes das posições restantes do outro pai P_2 (P_1).

O Algoritmo 5.2.2 apresenta o pseudo-código para aplicação de NPBX. Assim como em NOX, a principal diferença entre PBX e NPBX é que as duas primeiras posições do cromossomo devem possuir profundidades com valor fixo. A primeira posição deve ser 0 e a segunda posição deve ser 1. No NPBX, também é necessário aplicar o mecanismo de correção da profundidade ao construir os novos indivíduos.

Algoritmo 5.2.2: - Operador de Recombinação NPBX.

//Sejam P_1 e P_2 as NDDEs dos indivíduos pais (entrada)
//Sejam O_1 e O_2 as NDDEs dos indivíduos filhos resultantes da aplicação de NPBX (saída)
//Sejam A_1 e A_2 os *arrays* auxiliares para marcação.
(1) Determine de forma aleatória as posições a serem copiadas do pai P_1 (P_2);
(2) Marque em A_2 (A_1) os vértices de P_2 (P_1) que correspondem aos genes das posições selecionadas no Passo 1 em P_1 (P_2);
(3) Obtenha O_1 (O_2) pela cópia das posições selecionados de P_1 (P_2) e os demais genes de P_2 (P_1) ignorando os genes marcados no *array* auxiliar. Mantenha as duas primeiras posições do *array* de profundidades fixas. Aplique o método de correção da profundidade sempre que necessário.
(4) Verifique se existem vértices em posições que causem redundância (ver Seção 6.1) e troque as posições quando necessário.

A seguir é apresentado um exemplo de aplicação do NPBX. Para isso, utiliza-se os mesmos cromossomos pais definidos para o exemplo de NOX. Para facilitar o entendimento, as NDDEs são apresentadas novamente nas Tabelas 5.4(a) e 5.4(b). As posições

escolhidas são 2, 4, 6, 9 e 10.

Tabela 5.4: NDDEs dos indivíduos pais.

(a) NDDE de P_1.

i	1	2	3	4	5	6	7	8	9	10
de	0	1	2	3	4	1	2	2	3	1
v	A	B	C	H	I	D	F	G	J	E
deg	3	2	2	2	1	3	1	2	1	1

(b) NDDE de P_2.

i	1	2	3	4	5	6	7	8	9	10
de	0	1	2	1	2	3	3	4	5	1
v	A	B	J	C	D	E	F	G	H	I
deg	3	2	1	2	3	1	2	2	1	1

No Passo 2 do algoritmo são marcados os vértices das posições selecionados. Em A_1 são marcados os vértices das posições selecionadas em P_2 que são: B, C, E, H e I. Já em A_2, são marcados os vértices das posições selecionadas em P_1 que são: B, H, D, J e E. Os *arrays* auxiliares podem ser vistos nas Tabelas 5.5(a) e 5.5(b).

Tabela 5.5: *Arrays* auxiliares de NPBX.

(a) *Array* auxiliar A_1 em NPBX.

A	B	C	D	E	F	G	H	I	J
	X	X		X			X	X	

(b) *Array* auxiliar A_2 em NPBX.

A	B	C	D	E	F	G	H	I	J
	X		X	X			X		X

Para compor o primeiro filho (O_1), as posições do filho são preenchidas com a cópia das posições dos pais na seguinte seqüência: 1 de P_2; 2 de P_1; 4 de P_2; 4 de P_1; 7, de P_2; 6 de P_1; 8 e 10 de P_2 e, para finalizar, 9 e 10 de P_1. No processo de construção de O_1, os vértices marcados em A_2 são ignorados em P_2. Ao preencher a posição 4 de O_1 com a posição 4 de P_1, 7 de O_1 com 8 de P_2 e 9 de O_1 com 9 de P_1, as profundidades precisam ser corrigidas, respectivamente de $de_4 = 3$ para $de_4 = 2$; $de_8 = 4$ para $de_7 = 2$ e $de_9 = 3$ para $de_9 = 2$. Na Tabela 5.6(a), as posições em negrito indicam onde foram efetuadas mudanças na profundidade. Nesse filho, a verificação de redundância encontrou que os vértices da posições 8 e 10 (I, E) possuem o mesmo ancestral e não estão em ordem crescente. Para evitar que a NDDE seja redundante, trocam-se os intervalos da NDDE que representam as subárvores desses vértices, apesar de a NDDE sem essa inversão ser válida. A necessidade desse mecanismo será explicada na Seção 6.1.

Para o O_2, os genes são preenchidos com os genes das seguintes posições de cada um dos pais: 1 de P_1; 2 de P_2; 6 de P_1; 4 de P_2; 7 de P_1; 6 de P_2; 8, e 9 de P_1 e por último 9 e 10 de P_2. Para este filho são ignoradas em P_1 as posições dos vértices marcados em A_1. O método de correção da profundidade precisou ser aplicado ao copiar o gene da posição

9 de P_2 para a posição 9 de O_2. A profundidade foi alterada de $de_9 = 5$ para $de_9 = 4$. A verificação de redundância também detectou que nesse filho os vértices das posições 3 e 4 (D e C) não estão em ordem crescente, com isso necessita-se corrigir essas posições para não gerar NDDEs redundantes.

Os filhos obtidos pela aplicação de NPBX (sem a correção da redundância) são apresentados nas Tabelas 5.6(a) e 5.6(b). Após a correção das posições para evitar redundância, esse filhos ficam como nas Tabelas 5.7(a) e 5.7(b), onde as posições alteradas estão em itálico. As árvores representadas por esses indivíduos podem ser vistas na Figura 5.4.

Tabela 5.6: NDDEs dos indivíduos obtidos da aplicação do NPBX.

(a) NDDE de O_1.

i	1	2	3	4	5	6	7	8	9	10
de	0	1	1	**2**	3	1	**2**	1	**2**	1
v	A	B	C	**H**	F	D	**G**	I	**J**	E
deg	2	2	2	**2**	1	2	**3**	1	**2**	1

(b) NDDE de O_2.

i	1	2	3	4	5	6	7	8	9	10
de	0	1	1	1	2	3	2	3	**4**	1
v	A	B	D	C	F	E	G	J	**H**	I
deg	2	2	1	4	3	1	1	2	**1**	1

Tabela 5.7: NDDEs dos indivíduos do NPBX depois da correção de redundância.

(a) NDDE de O_1 sem redundância.

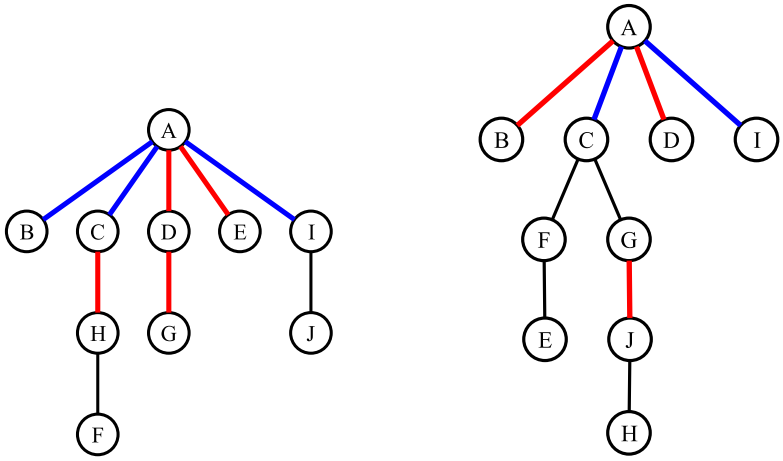
i	1	2	3	4	5	6	7	8	9	10
de	0	1	1	**2**	3	1	**2**	*1*	*1*	***2***
v	A	B	C	**H**	F	D	**G**	*E*	*I*	***J***
deg	2	2	2	**2**	1	2	**3**	*1*	*2*	***1***

(b) NDDE de O_2 sem redundância.

i	1	2	3	4	5	6	7	8	9	10
de	0	1	*1*	*2*	*3*	*2*	*3*	***4***	*1*	1
v	A	B	*C*	*F*	*E*	*G*	*J*	***H***	*D*	I
deg	2	2	*4*	*3*	*1*	*1*	*2*	***1***	*1*	1

Novamente, observa-se que, apesar de serem topologicamente diferentes dos pais, a maioria das arestas dos filhos estão presentes nos pais (aresta em azul e vermelho na Figura 5.4). No entanto, em média, os filhos possuem 3 arestas que não pertencem a nenhum dos pais (arestas em preto da Figura 5.4). O comportamento relativo ao surgimento de arestas diferentes dos pais será avaliado nos testes de propriedades da representação (ver Capítulo 6).

Assim como em NOX, o algoritmo de NPBX prevê que a floresta possua apenas uma única árvore. A estratégia para aplicar NPBX para florestas com mais de uma árvore é a mesma definida para NOX. Considera-se um *array* virtual que concatena todas as árvores em seqüência e aplica-se a operação a essa seqüência como se fosse uma única árvore de modo a garantir que as raízes das NDDEs sem mantenham fixas, assim como a profundidade das duas primeiras posições de cada árvore.



(a) Árvore de O_1 de NPBX. (b) Árvore de O_2 de NPBX.

Figura 5.4: Árvores representadas pelos cromossomos filhos resultantes da aplicação do NPBX.

5.2.2 Recombinação para Grafos Completos e Não-completos - EHR

Uma vez que NOX e NPBX não podem ser aplicados em casos onde os grafos são não-completos (densos ou esparsos), foi proposto outro operador de recombinação que solucionasse esse problema. Recordando que os operadores PAO e CAO da NDDE podem ser aplicados tanto a grafos completos como não completos, buscou-se uma solução de recombinação baseada no PAO. Esse operador foi denominado EHR (do inglês, *Evolutionary History Recombination Operator*).

O EHR baseia-se no histórico evolutivo de aplicação de PAO, isto é, na seqüência de pares de vértices (a, p) (ver Seção 4.3.2) aplicada para gerar cada indivíduo (ver Figura 5.5). O histórico de cada indivíduo pode ser recuperado por meio da matriz Π_{v_x} (que armazena a localização do vértice v_x de cada indivíduo) e do *array* π (que armazena o índice do ancestral de cada indivíduo). A Figura 5.5 apresenta um exemplo de histórico de evolução por meio de aplicações de PAO. Na Figura, a partir da árvore $T0$ com a aplicação de PAO utilizando os vértices $p = D$ e $a = A$, obtém-se a árvore $T2$ (ramo a direita do histórico). O ramo a esquerda é formado pela aplicação de PAO com os vértice $p = D$ e $a = B$ em $T0$, obtendo $T1$ e $T3$ é obtida pela aplicação de PAO e $T1$ com os vértices $p = C$ e $a = D$.

No entanto, para facilitar a utilização do EHR, é proposta uma modificação no *array* π, que passa a armazenar, além do índice do ancestral, o par de vértices (a e p) que foram

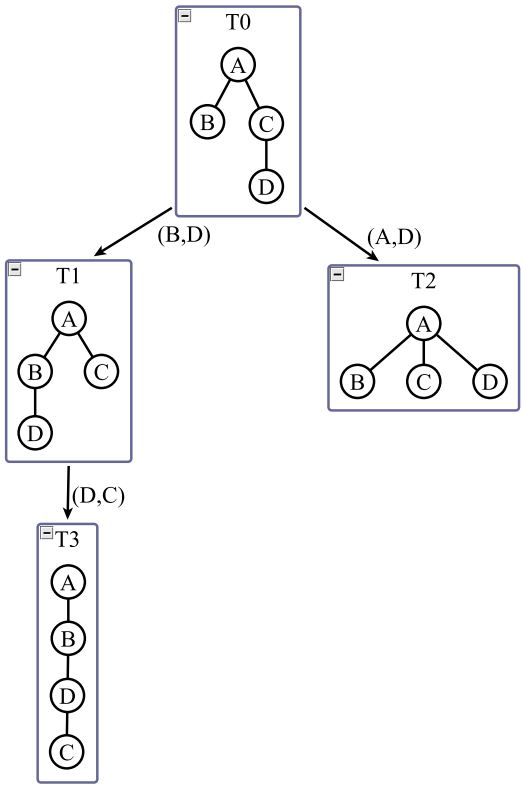


Figura 5.5: Histórico de aplicações de PAO.

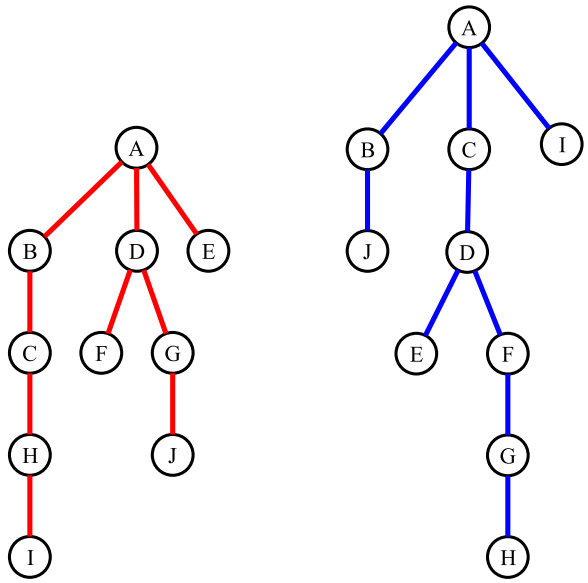
utilizados na aplicação de PAO. O EHR é apresentado no Algoritmo 5.2.3.

Algoritmo 5.2.3: - Pseudocódigo do EHR.

//Sejam P_1 e P_2 as NDDEs dos pais (entrada)
//Seja c a NDDE do ancestral comum de P_1 e P_2
//Seja π o *array* da NDDE que armazena o índice do ancestral de cada indivíduo e o par (a, p) utilizado
//Sejam p e a os vértices necessários para aplicação de PAO
//Seja O_1 a NDDE resultante (saída)
(1) Selecione dois indivíduos P_1 e P_2 como pais;
(2) Determine o ancestral comum c entre P_1 e P_2 utilizando o *array* π, caso não exista um ancestral retorne ao Passo 1;
(3) Determine as seqüências s_1 e s_2 de pares (a, p) que geram P_1 e P_2 a partir de c o ancestral comum (isto é, os conjuntos de pares de vértices p e a armazenados na π) até obter o indivíduo atual;
(4) Selecione aleatoriamente de s_1 e s_2 um subconjunto s_s de pares (a, p) para ser aplicado a c, tal que $|s_s| \leq k$, onde $|s_s|$ é o tamanho de s_s e k é uma constante positiva;
(5) Aplique s_s utilizando PAO, obtendo O_1.

Para exemplificar o funcionamento do EHR, utilizaram-se os mesmos pais dos exemplos

de NOX e NPBX. Nesse exemplo, são apresentadas somente as árvores e não as suas codificações como NDDE, pois as modificações na NDDE consistem de aplicações do operador PAO. As Figuras 5.6(a) e 5.6(b) reapresentam as árvores dos indivíduos pais.



(a) Primeiro pai. (b) Segundo pai.

Figura 5.6: Exemplo dos pais para aplicação de EHR.

Por meio do *array* π, recupera-se o ancestral comum dos indivíduos pais representado pela árvore da Figura 5.7. Ao percorrer a lista encadeada de ancestralidade armazenada no *array* π, obtém-se que para P_1 (ver Figura 5.6(a)), são necessárias 5 modificações a partir do ancestral comum para gerá-lo, e para P_2 (ver Figura 5.6(b)), são necessárias 7 modificações.

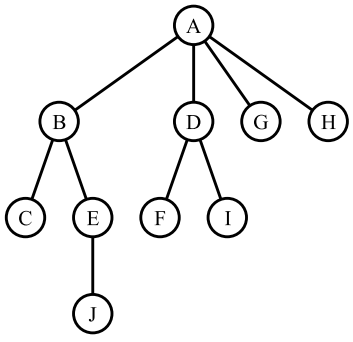
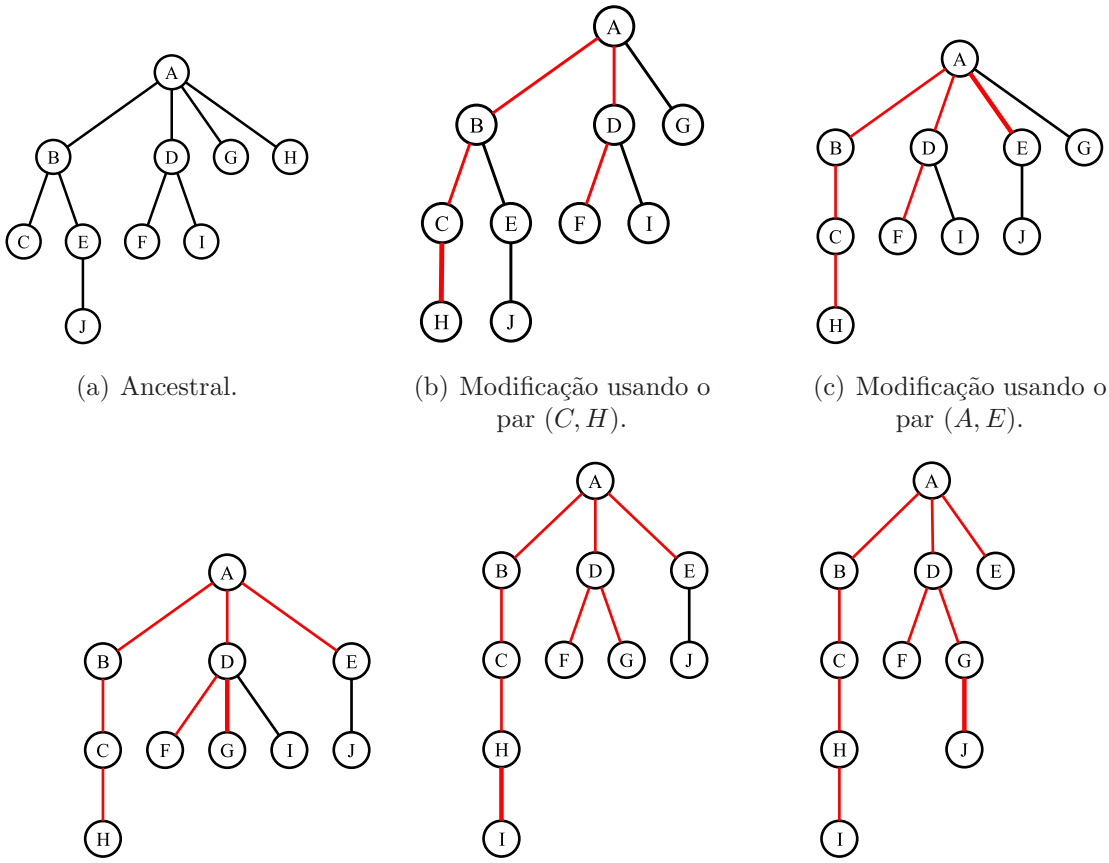


Figura 5.7: Ancestral comum entre os pais.

As modificações para obter P_1 são apresentadas na Figura 5.8, em que as arestas em vermelho indicam as arestas do indivíduo pai que já estão presentes no processo de modificação e as arestas em negrito, onde foi modificada a árvore. A seqüência de pares aplicados é: (C, H) (ver Figura 5.8(b)), (A, E) (ver Figura 5.8(c)), (D, G) (ver Figura 5.8(d)), (H, I) (ver Figura 5.8(e)) e (G, J) (ver Figura 5.8(f)), finalmente obtendo o indivíduo P_1.

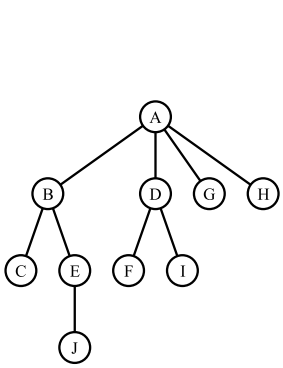


(a) Ancestral. (b) Modificação usando o par (C, H). (c) Modificação usando o par (A, E).

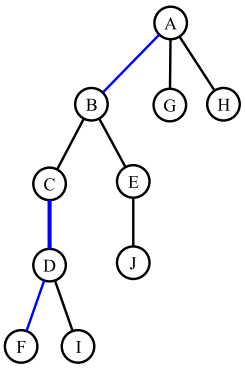
(d) Modificação usando o par (D, G). (e) Modificação usando o par (H, I). (f) Modificação usando o par (G, J).

Figura 5.8: Modificações (arestas em negrito) aplicados ao ancestral comum (ver Figura 5.7) que produziram P_1.

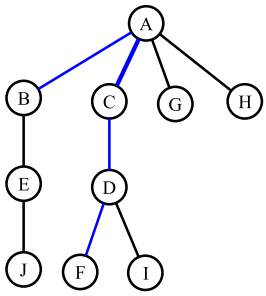
A seqüência de modificações para obter P_2 são apresentadas na Figura 5.9. As arestas em azul indicam as arestas de P_2 que já estão presentes no processo de modificação e as arestas em negrito mostram modificações. A seqüência de modificações a partir do ancestral comum foi: (C, D) (ver Figura 5.9(b)), (A, C) (ver Figura 5.9(c)), (F, G) (ver Figura 5.9(d)), (A, I) (ver Figura 5.9(e)), (G, H) (ver Figura 5.9(f)), (B, J) (ver Figura 5.9(g)), (D, E) (ver Figura 5.9(h)), obtendo o indivíduo P_2.



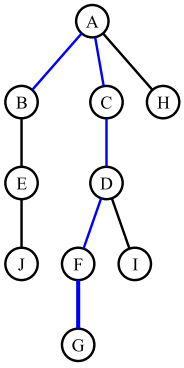
(a) Ancestral.



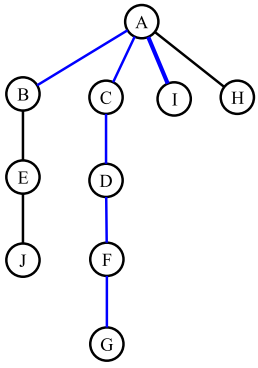
(b) Modificação (C, D).



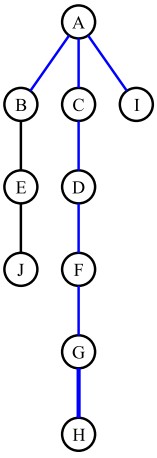
(c) Modificação (A, C).



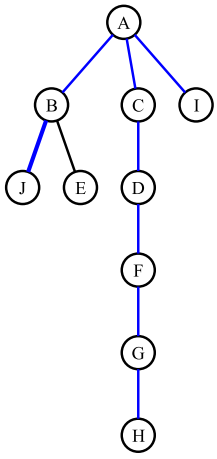
(d) Modificação (F, G).



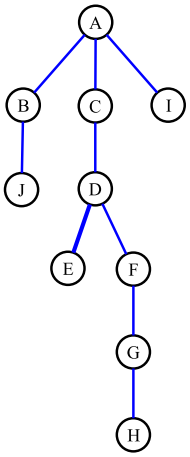
(e) Modificação (A, I).



(f) Modificação (G, H).



(g) Modificação (B, J).



(h) Modificação (D, E).

Figura 5.9: Modificações (arestas em negrito) aplicados ao ancestral comum (ver Figura 5.7) que produzem P_2.

Dada as seqüências s_1 e s_2, construiu-se a subseqüência s_s com 5 pares: (A, I), (D, E), (D, G), (B, J) e (C, H). A Figura 5.10 apresenta o indivíduo resultante da aplicação das modificações ao ancestral comum.

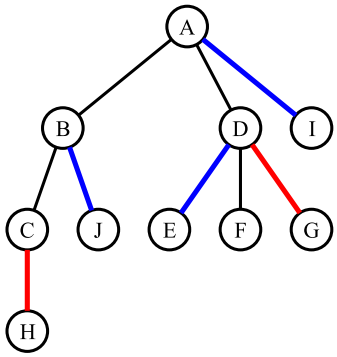


Figura 5.10: Novo indivíduo depois da aplicação de um subconjunto de modificações (A, I), (D, E), (D, G), (B, J) e (C, H) ao ancestral comum.

Um ponto importante que vale ressaltar é que a escolha ideal de modificações são as que não envolvem o mesmo vértice, como o vértice p, pois a contribuição de informação de alguma das modificações pode ser perdida. Uma das principais propriedades desse operador é que grande parte das informações comuns entre os pais é preservada. Além disso, todas as arestas do indivíduo gerado estão presentes em um dos pais. Novamente, ressaltando, a característica mais importante é de que o EHR pode ser aplicado para problemas de grafos não-completos (esparsos ou densos) sem a necessidade de mecanismos de correção de soluções infactíveis. Outro aspecto importante é que se pode aplicar o EHR para florestas com uma ou mais árvores sem a necessidade de nenhum mecanismo auxiliar.

5.3 Considerações Finais

Este Capítulo apresentou o desenvolvimento da estrutura de dados NDDE para EAs aplicados a problemas de projetos de redes. Essa nova estrutura de dados tem como base a NDE. Nessa proposta são desenvolvidos os operadores de recombinação NOX, NPBX e EHR, bem como as melhorias implementadas para os operadores de mutação gerando PAO e CAO.

A NDDE é avaliada por meio da aplicação de EAs usando a NDDE para problemas clássicos de projeto de redes, como árvore geradora mínima com restrição de grau e árvore de comunicação ótima (ver Capítulo 7), e pela análise de suas propriedades

(ver Capítulo 6).

Capítulo

6

Análise de Propriedades da NDDE

A análise de propriedades tais como, redundância, tendência, localidade, hereditariedade e complexidade são muito importantes para auxiliar na escolha da representação para um determinado problema (Rothlauf, 2006). A partir desse tipo de análise pode-se também avaliar o desempenho de um EA que utilize certa representação e, com isso, obter indicativos sobre a capacidade do EA em solucionar o problema para o qual foi proposto.

A Seção 6.1 descreve a propriedade de redundância e sua análise aplicada à NDDE. A Seção 6.2 aborda a propriedade de tendência e analisa empiricamente os operadores de mutação PAO e CAO, além dos operadores de recombinação NOX e NPBX da NDDE. A Seção 6.3 apresenta a propriedade de localidade e sua análise para os operadores PAO e CAO da NDDE. A Seção 6.4 descreve a propriedade de hereditariedade com uma análise empírica da mesma para os operadores de recombinação NOX e NPBX. A Seção 6.5 apresenta a análise complexidade de tempo dos operadores da NDDE, bem como resultados empíricos que mostram coerência deles com a análise teórica. Por fim, a Seção 6.6 destaca os resultados mais relevantes descritos neste Capítulo.

6.1 Redundância

A propriedade de redundância avalia o mapeamento fenótipo-genótipo. Uma representação é dita redundante se um fenótipo é representado por mais de um genótipo, ou seja, dois ou mais genótipos codificam o mesmo fenótipo. A redundância pode ser clas-

sificada em redundância sinônima e não-sinônima. Na redundância sinônima, genótipos similares codificam o mesmo fenótipo e, na não-sinônima, genótipos significativamente diferentes codificam o mesmo fenótipo. A redundância sinônima não altera o desempenho dos EAs, ao contrário da redundância não-sinônima que pode influenciar a eficiência dos EAs (Rothlauf, 2006).

A redundância em uma representação pode afetar o desempenho dos EAs, principalmente em relação às operações de recombinação. Se diferentes genótipos correspondem ao mesmo fenótipo, a operação de recombinação pode estar recombinando as informações do mesmo fenótipo e, assim, produzindo novamente o mesmo indivíduo ou, até mesmo, dependendo da representação, gerando soluções inválidas.

A representação NDE é uma representação redundante, pois dependendo do vértice selecionado como raiz da árvore para iniciar a codificação e da ordem em que os vértices irmãos (que possuem o mesmo vértice pai) são inseridos na NDE, diferentes códigos podem corresponder a mesma árvore. A redundância da NDE é classificada como uma redundância sinônima, pois genótipos similares codificam o mesmo fenótipo e genótipos significativamente distintos correspondem a fenótipos bem diferentes. Assim, como a NDE, a NDDE sem as modificações descritas na seqüência também é redundante, pois a diferença básica no processo de codificação das árvores na NDDE em relação a NDE é a adição do valor de grau do vértice.

Tabela 6.1: NDDEs redundantes para a árvore da Figura 6.1.

(a) NDDE 1.

i	1	2	3	4	5	6	7	8	9
de	0	1	2	1	2	3	4	3	2
v	A	B	H	C	D	F	G	I	E
deg	2	2	1	3	3	2	1	1	1

(b) NDDE 2

i	1	2	3	4	5	6	7	8	9
de	0	1	2	3	4	3	2	1	2
v	A	C	D	F	G	I	E	B	H
deg	2	3	3	2	1	1	1	2	1

Por exemplo, considere as NDDEs das Tabelas 6.1(a) e 6.1(b) que representam a árvore da Figura 6.1. Apesar de representarem a mesma árvore as NDDEs são diferentes, na NDDE da Tabela 6.1(a) o vértice B, por exemplo, aparece na posição 2 e na NDDE da Tabela 6.1(b) o vértice B está na posição 8. Visto que existem duas NDDEs para uma

mesma árvore a NDDE é uma representação redundante.

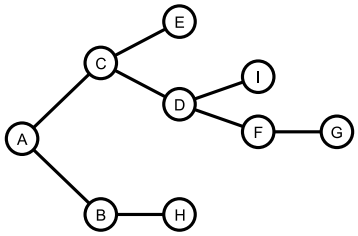


Figura 6.1: Árvore com 9 vértices.

A redundância na NDDE pode ser evitada fazendo as modificações descritas a seguir. Primeiro deve-se garantir que todos os indivíduos da população inicial possuam o mesmo vértice raiz da árvore. Deve-se garantir também que os vértices irmãos estejam em ordem crescente de rótulo (ou algum outro tipo de ordem fixa) na representação. Para isso, se for utilizada uma busca em profundidade para codificar os indivíduos, deve-se criar a lista de adjacências da árvore em ordem crescente. Segundo esse critério estabelecido a NDDE a ser utilizada para codificar a árvore da Figura 6.1 é a presente na Tabela 6.1(a).

Com o primeiro objetivo alcançado, ou seja, os indivíduos da população inicial são não redundantes, é preciso, garantir, então que os operadores de mutação PAO e CAO e os operadores de recombinação NOX e NPBX não gerem indivíduos redundantes. Para o EHR não é necessário aplicar nenhum processo especial, pois esse utiliza aplicações de PAO para construir o novo indivíduo.

Em PAO e CAO, a redundância pode ser gerada pela inserção da subárvore podada em um ponto inadequado da $T_{destino}$. Com o objetivo de evitar a redundância, para cada vértice na NDDE, armazena-se a informação do índice de seu pai na própria NDDE. Quando for inserir uma subárvore com raiz p (r) ligando essa ao vértice a, busca-se no intervalo de $i_a + 1$ até o índice do último descendente de a pelo primeiro vértice v_x à direita de a cujo pai é a, tal que $v_x > p$ ($v_x > r$) e determina que a subárvore de p/r seja inserida a partir desse ponto. O custo para efetuar essa busca é $O(|T_{destino}|)$, pois no pior caso o vértice a é a raiz da árvore e a posição correta para inserir o vértice p/r é a última posição de $T_{destino}$.

Por fim, resta garantir que os operadores NOX e NPBX não criem NDDEs que representem indivíduos redundantes. No pseudo-código de ambos os operadores, foi definido um passo que verifica e corrige possíveis vértices em posições redundantes (ver Seções 5.2.1 e 5.2.1). O Algoritmo 6.1.1 apresenta os passos para verificar e corrigir se existem vértices em posição que gere a redundância da NDDE.

Algoritmo 6.1.1: - Verificação e correção da redundância

//Seja T a NDDE da árvore a qual será verifica a redundância (entrada e saída)
//Seja T_{tmp} a NDDE auxiliar para armazenar a subárvore a ser trocada
//Seja de_{v_x} a profundidade do vértice v_x
//Seja deg_{v_x} o grau do vértice v_x em T
//Seja $raizes$ o $array$ que armazena na posição i o vértice mais a direita com profundidade i
//Considera-se que T é redundante
(1) Enquanto existir redundância execute os Passos de 2 a 7
(2) Para todo v_x de T recupere o ancestral de v_x: $ancestral = raizes[de_{v_x} - 1]$;
(3) Se ($deg_{ancestral} > 2$) ou ($deg_{ancestral} > 1$ se $ancestral$ é o vértice raiz da árvore), verifique se $raizes[de_{v_x}] \neq -1$, caso afirmativo $irmao = raizes[de_{v_x}]$; caso contrário se $v_x < irmao$, os vértices estão em posição redundante, assim, continue no Passo 4, senão continue no Passo 7
(4) Copie a subárvore de $irmao$ até v_x em T_{tmp};
(5) Copie de v_x até v_k tal que ($v_k > irmao$, se $de_{v_k} == de_{irmao}$), ou ($de_{v_k} < de_{v_x}$), ou chegou ao fim de T, para as posições de T a partir da posição de $irmao$;
(6) Copie T_{tmp} a partir da posição atual em T;
(7) Se não é fim de T Retorne ao Passo 2; caso contrário verifique se executou o Passo 4 ao menos uma vez, se sim retorne ao Passo 1, caso contrário termine.

O Algoritmo 6.1.1 verifica possíveis vértices redundantes comparando os vértices 2 a 2 e troca as posições quando necessário. No pior caso, tem-se uma árvore estrela, $n-1$ vértices estão ligados ao vértice raiz e estão ordenados de forma decrescente. Nesse caso, a cada passagem são corrigidos metade dos casos de redundância assim, o número de vértices em posição redundante diminui pela metade a cada passagem. Dessa forma, a complexidade de tempo do Algoritmo 6.1.1 é no pior caso $O(n \log(n))$. No entanto, verifica-se que na prática é raro acontecer o pior caso ou casos próximos dele.

6.2 Tendência

Em uma representação sem tendência todos os possíveis fenótipos são codificados uniformemente. Assim, operadores de reprodução sem tendência não privilegiam a representação de soluções específicas. A aplicação dos operadores sem nenhuma pressão de seleção não modifica as propriedades estatísticas da população. Representações e operadores de busca tendenciosos podem ser utilizados se existe conhecimento a priori de que a solução ótima é similiar ao conjunto de soluções que são preferidas pela tendência (Rothlauf e Tzschoppe, 2005). Por outro lado, representação e operadores de reprodução não-tendenciosos devem ser utilizadas se nenhum conhecimento específico sobre problema está disponível.

Com o objetivo de analisar empiricamente a tendência da NDE e da NDDE para problemas envolvendo árvores, são geradas soluções aleatórias com a representação Número de Prüfer em grafos com pesos distribuídos uniformemente (ver Seção 4.1.2). A representação

de Número de Prüfer não possui tendência na criação de soluções dada a correspondência 1 : 1, ou seja, cada Número de Prüfer corresponde a uma única árvore e cada árvore corresponde a um único Número de Prüfer. A essas soluções aleatórias são aplicados os operadores da representação. As soluções geradas são avaliadas de acordo com algumas métricas: distância para a árvore geradora mínima ($\widehat{d}_{i,MST}$) e a distância para estrelas ($\widehat{d}_{i,STAR}$). A distância entre duas árvores geradoras T_x e T_y é medida pela distância definida por Hamming (1980), que é o número de arestas diferentes entre as duas árvores, calculada por:

$$d_{x,y} = \frac{1}{2} \sum_{v_u,v_v \in V, v_u < v_v} |l^x_{v_u,v_v} - l^y_{v_u,v_v}|,$$

onde $l^x_{v_u,v_v}$ é 1 se existe a aresta de v_u para v_v em T_x e 0 se não existe a aresta. A distância mínima para estrelas é calculada por:

$$\widehat{d}_{i,STAR} = min(d_{i,star_j}),$$

onde $j = 1 \ldots n$ e $star_j$ é a árvore estrela que possui como vértice central o vértice v_j.

Para analisar a potencial tendência dos operadores de mutação e recombinação da NDDE, as medidas estatísticas utilizadas na população são: média da distância para árvore geradora mínima $\overline{d}_{mst-pop} = \frac{1}{\mu} \sum_{i=1}^{\mu} \widehat{d}_{i,MST}$, onde $\widehat{d}_{i,MST}$ é a distância entre a árvore T_i e a árvore geradora mínima do grafo (ver Apêndice A) e μ é o número de indivíduos na população, bem como a média da distância mínima para estrelas $\overline{d}_{star-pop} = \frac{1}{\mu} \sum_{i=1}^{\mu} \widehat{d}_{i,STAR}$, onde $\widehat{d}_{i,STAR}$ é a distância mínima entre a árvore T_i e todas as árvores estrela do grafo. Nesse processo, nenhum método de seleção é utilizado, os novos indivíduos gerados pelos operadores substituem os indivíduos pais.

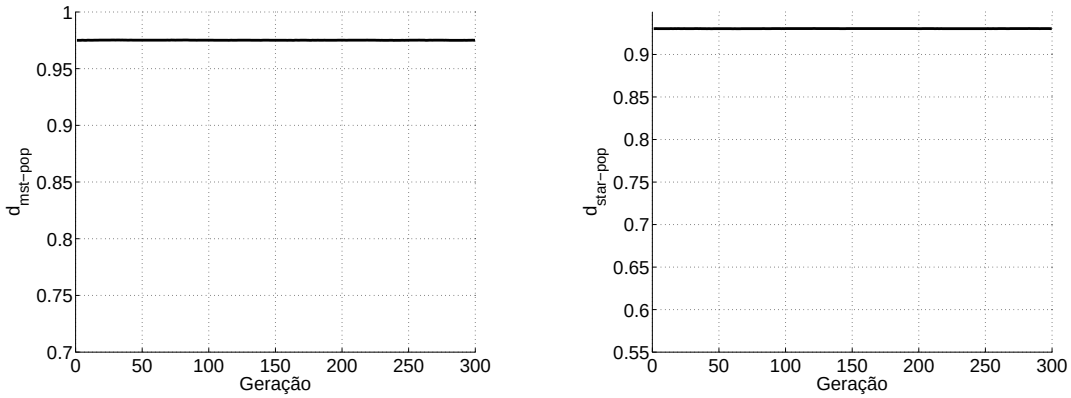
Se $\overline{d}_{mst-pop}$ diminui, os operadores são tendenciosos em direção a árvore geradora mínima. Se $\overline{d}_{star-pop}$ diminui, os operadores são tendenciosos em direção a estrelas. Se todas as medidas ficam constantes, os operadores são não-tendenciosos.

Nos testes realizados foram utilizados grafos Euclidianos definidos para o problema de árvores de Steiner e disponíveis na biblioteca *OR-Library* [1] com número de vértices $n = 10$, 20, 40 e 80. Além disso, foi utilizada a geração de soluções iniciais por Número de Prüfer. Os resultados apresentam as medidas $\overline{d}_{mst-pop}$ e $\overline{d}_{star-pop}$ de forma normalizada, ou seja, $d_{mst-pop} = \overline{d}_{mst-pop}/(n-1)$ e $d_{star-pop} = \overline{d}_{star-pop}/(n-1)$, assim obtém-se valores comparáveis independente do número de vértices do grafo.

[1] http://people.brunel.ac.uk/ mastjjb/jeb/info.html

PAO e CAO

Para os operadores 1 e 2 da NDE (ver Seção 4.3.2), foi verificado, por meio de testes empíricos, que esses não possuem nenhuma tendência para os parâmetros analisados (ver (de Lima et al., 2008)). No entanto, como os operadores PAO e CAO da NDDE possuem algumas modificações em relação aos operadores 1 e 2 da NDE, foram realizados novos testes para PAO e CAO. Os resultados com esses operadores são bastante similares, assim são apresentados na seqüência somente os resultados para PAO. Para os demais grafos, com diferentes número de vértices (10, 20, 40), os resultados são similares ao grafo com 80 vértices e por isso não são apresentados.



(a) Distância para a árvore geradora mínima. (b) Mínima Distância para árvore estrela.

Figura 6.2: Análise de tendência para os operadores PAO e CAO da NDDE em um grafo Euclidiano com 80 vértices.

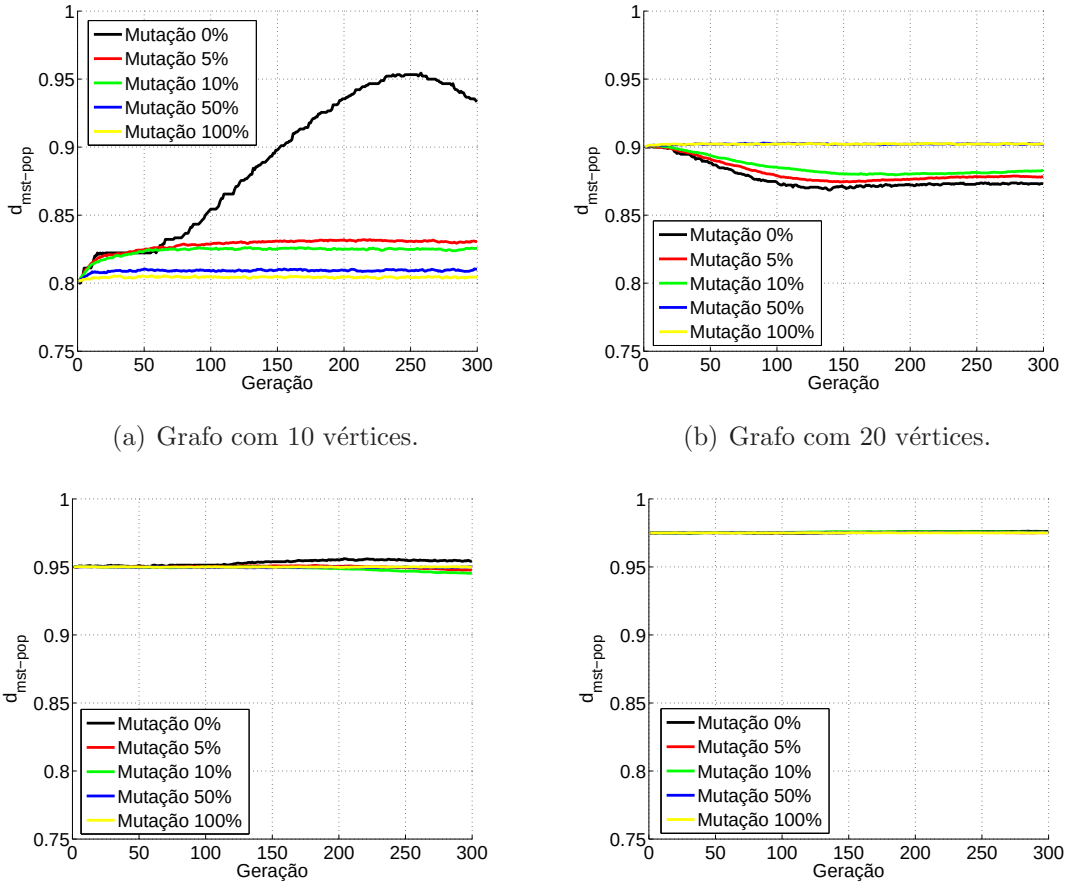
A Figura 6.2(a) apresenta a média de $d_{mst-pop}$ em 30 execuções para o grafo com 80 vértices com 300 gerações, 10 mil indivíduos em cada geração, e o operador de mutação PAO. Observa-se que a medida $d_{mst-pop}$ manteve-se com valor constante ao longo das gerações. Como essa medida corresponde a $\overline{d}_{mst-pop}$, conclui-se que o operador PAO e, por similaridade CAO, não possuem tendência em relação à árvore geradora mínima.

A Figura 6.2(b) mostra média de $d_{star-pop}$ com os mesmos parâmetros da Figura 6.2(a) e, novamente, observa-se que a medida analisada mantém-se com valor constante no decorrer das gerações. Assim, conclui-se que os operadores PAO e CAO também não possuem nenhuma tendência com relação a árvores estrela.

NOX e NPBX

Os mesmos testes aplicados aos operadores PAO e CAO foram aplicados para os operadores de recombinação NOX e NPBX. Para estes operadores, além de analisar o

seu comportamento separadamente, também é avaliado seu comportamento em relação a tendência em conjunto com os operadores de mutação PAO e CAO utilizando diferentes taxas de aplicação (5%, 10%, 50%, 100%). Deve-se observar que PAO e CAO possuem probabilidade igual de serem aplicados. Na apresentação desses resultados, são utilizados os valores normalizados de $d_{mst-pop}$ e $d_{star-pop}$.



(a) Grafo com 10 vértices.

(b) Grafo com 20 vértices.

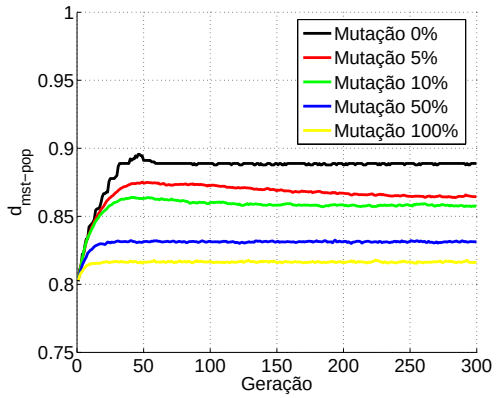
(c) Grafo com 40 vértices.

(d) Grafo com 80 vértices.

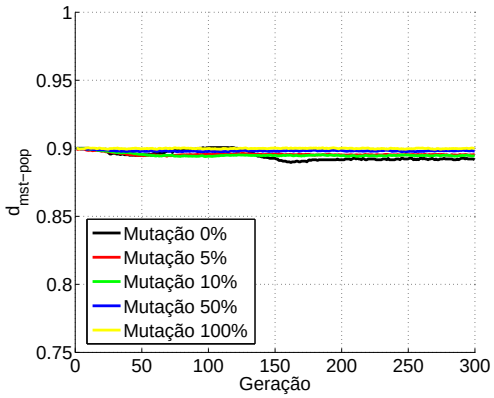
Figura 6.3: Distância para a MST utilizando o operador NOX.

A Figura 6.3 apresenta os resultados para NOX com relação a $d_{mst-pop}$ com grafos com 4 números de vértices (n = 10, 20, 40 e 80). Observa-se que para os grafos com menor número de vértices, houve uma tendência em relação a $d_{mst-pop}$. No caso com 10 vértices (ver Figura 6.3(a)) o NOX sem auxílio de mutação tende a um afastamento da árvore geradora mínima. No caso com 20 vértices (ver Figura 6.3(b)), ocorre uma aproximação em relação a árvore geradora mínima ao longo das gerações quando se utiliza menos PAO e CAO. Conforme aumenta o número de vértices do grafo (ver Figuras 6.3(c) e 6.3(d)), nota-se que essa tendência em relação a árvore geradora mínima diminui, de

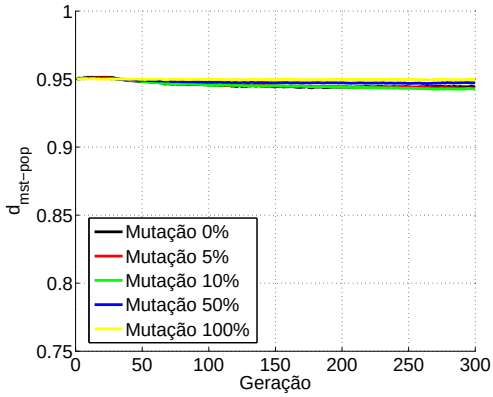
forma praticamente independente do nível de utilização de PAO e CAO em conjunto com NOX.



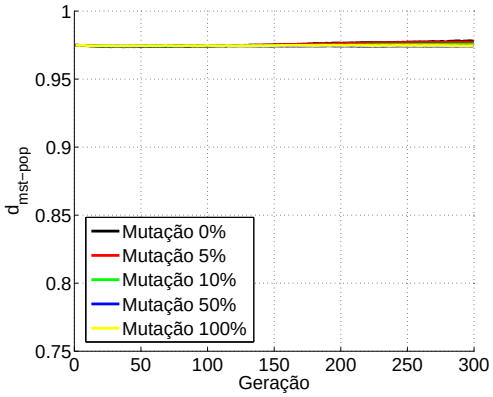
(a) Grafo com 10 vértices.



(b) Grafo com 20 vértices.



(c) Grafo com 40 vértices.

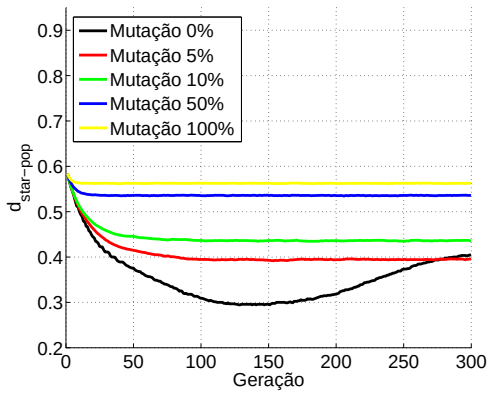


(d) Grafo com 80 vértices.

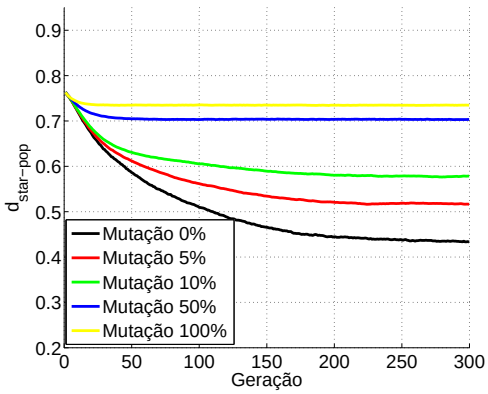
Figura 6.4: Distância para a MST utilizando o operador NPBX.

Os resultados de $d_{mst-pop}$ para o NPBX são apresentados na Figura 6.4. O NPBX, assim como o NOX, apresenta para os grafos menores uma tendência em distanciar (ver Figura 6.4(a)) ou em aproximar (ver Figura 6.4(b)) da árvore geradora mínima. Novamente, conforme o número de vértices do grafo aumenta, a tendência em relação a árvore geradora mínima diminui. Vale ressaltar, que em relação a propriedade de tendência para árvore geradora mínima, ambos os operadores NOX e NPBX possuem comportamentos similares.

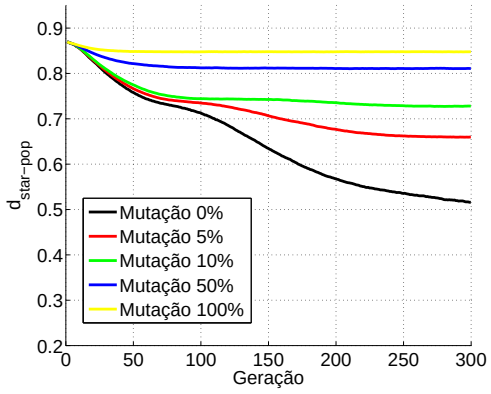
A seguir são apresentados os resultados de análise da tendência em relação a métrica $d_{star-pop}$.



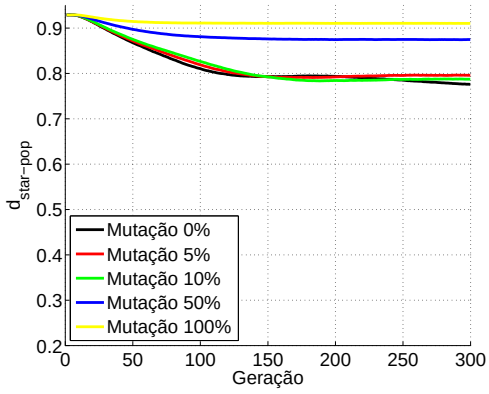
(a) Grafo com 10 vértices.



(b) Grafo com 20 vértices.



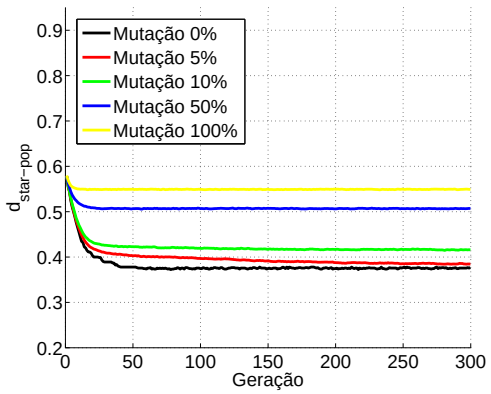
(c) Grafo com 40 vértices.



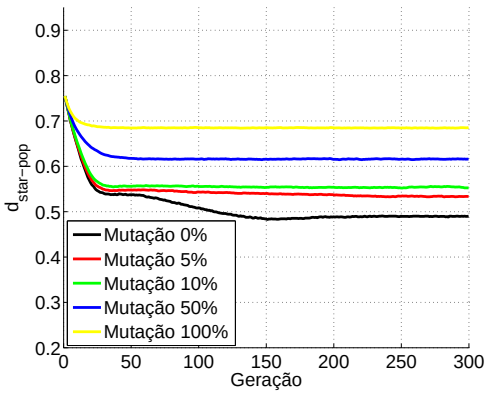
(d) Grafo com 80 vértices.

Figura 6.5: Distância para árvores estrela utilizando o operador NOX.

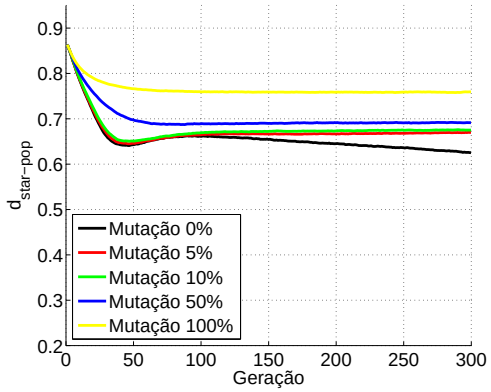
Os resultados para $d_{star-pop}$ usando NOX são apresentados na Figura 6.5. Analisando as Figuras 6.5(a), 6.5(b), 6.5(c) e 6.5(d), nota-se que o NOX possui uma tendência em aproximar as soluções encontradas à árvore estrela, pois no decorrer das gerações o valor de $d_{star-pop}$ diminui. O uso dos operadores de mutação PAO e CAO combinados com NOX auxilia na diminuição da tendência do NOX para árvores estrela conforme a taxa de aplicação da mutação aumenta.



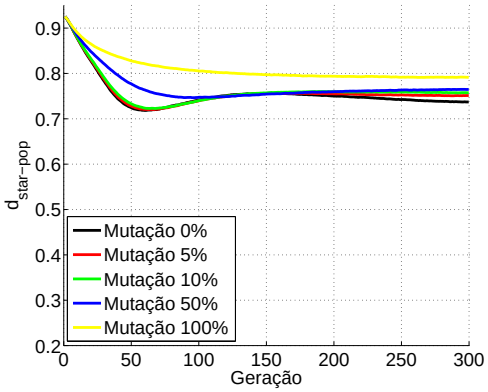
(a) Grafo com 10 vértices.



(b) Grafo com 20 vértices.



(c) Grafo com 40 vértices.



(d) Grafo com 80 vértices.

Figura 6.6: Distância para árvores estrela utilizando o operador NPBX.

A Figura 6.6 apresenta os resultados de $d_{star-pop}$ para o NPBX. Assim, como em NOX, as Figuras 6.6(a), 6.6(b), 6.6(c) e 6.6(d) mostram que o NPBX possui tendência em aproximar as soluções à árvores estrela. No entanto, para os casos analisados, a tendência de NPBX é menor do que a tendência do NOX, pois a aproximação do NPBX é menor do que a do NOX, sem a aplicação de PAO e CAO. Novamente, o uso da mutação combinada com NPBX diminui o efeito da tendência do operador conforme a taxa de aplicação aumenta.

Com base, nesses resultados conclui-se que NOX e NPBX possuem uma tendência em aproximar as soluções a árvores estrela. No entanto, essa tendência diminui conforme aumenta o número de vértices do grafo. Além disso, o uso combinado NOX e NPBX com PAO e CAO auxilia na diminuição da tendência.

Com relação a medida de distância para a árvore geradora mínima, nos casos apresentados, verificou-se para ambos os operadores uma tendência em aproximar ou distanciar

da árvore geradora mínima para os grafos menores. Essa variação em aproximar ou distanciar, deve-se provavelmente à similaridade de topologia da árvore geradora mínima, nos grafos analisados, com as árvores estrela. Felizmente, a tendência diminui com o aumento no número de vértices, uma vez que a proposta desse trabalho é obter representações mais adequadas para PPRs de larga-escala.

EHR

Com relação ao EHR, a propriedade de tendência não é analisada empiricamente, pois cada aplicação do EHR consiste em conjuntos de aplicações combinadas de PAO. Assim, como PAO não possui tendência, o EHR também não possui.

6.3 Localidade

A localidade de uma representação descreve o quão bem genótipos vizinhos correspondem a fenótipos vizinhos. A localidade de uma representação é alta se genótipos vizinhos correspondem diretamente a fenótipos vizinhos.

Trabalhos prévios tem indicado que a alta localidade de uma representação é necessária para uma busca evolutiva eficiente (Gottlieb et al., 2001; Rothlauf, 2006). Os resultados dessas pesquisas mostram que a alta localidade não altera a complexidade dos problemas para os quais a representação é aplicada. Por outro lado, representações de baixa localidade podem alterar a complexidade do problema pois os *building blocks* (ver Seção 4) podem ser mais facilmente quebrados na falta de localidade. Por isso, somente representações de alta localidade podem garantir a solução de problemas de alta complexidade (Rothlauf, 2006).

Chamemos de d_m a localidade de uma representação, $d^p_{x,y}$ a distância entre os fenótipos x e y, d^p_{min} a distância mínima entre dois fenótipos vizinhos, $d^g_{x,y}$ a distância entre dois genótipos e d^g_{min} a distância mínima entre dois genótipos vizinhos. A localidade d_m é dada por:

$$d_m = \sum_{d^g_{x,y}=d^g_{min}} |d^p_{x,y} - d^p_{min}|$$

.

Para $d_m = 0$, todos os genótipos vizinhos correspondem a fenótipos vizinhos e a representação possui uma localidade perfeita. Quando menor o valor d_m, maior é a localidade da representação (Rothlauf, 2006).

A propriedade de localidade é medida principalmente com relação à operação de

mutação da representação, pois ao aplicar mutação deve-se garantir que pequenas alterações no genótipo reflitam em pequenas alterações no fenótipo. Isso somente é possível se a representação possuir uma alta localidade.

Para a NDE e a NDDE ainda não foi possível estabelecer uma métrica ideal para medir a distância entre os genótipos. Por exemplo, após uma aplicação de PAO, tem-se que a distância entre as árvores é de uma aresta. No entanto, ao analisar as NDDEs de árvores diferentes os mesmos vértices podem estar em posições e profundidade distintas de uma árvore para outra, o que poderia parecer um indicativo de baixa localidade.

É possível estimar o comportamento da representação com relação aos operadores de mutação PAO e CAO. Considerando, que grafos não-orientados a cada aplicação dos operadores PAO e CAO, tem-se que a menor diferença do fenótipo entre o indivíduo pai e o novo individuo gerado é de uma aresta. Assim, sempre que a operação de mutação é aplicada tem-se que os fenótipos diferem na menor distância possível para árvores, não importando quão diferentes possam ser os seus genótipos. Pode-se dizer, com base nessa estimativa, que a NDE e a NDDE possuem uma localidade alta, praticamente perfeita, pois em todos os casos a distância entre os fenótipos será mínima.

6.4 Hereditariedade

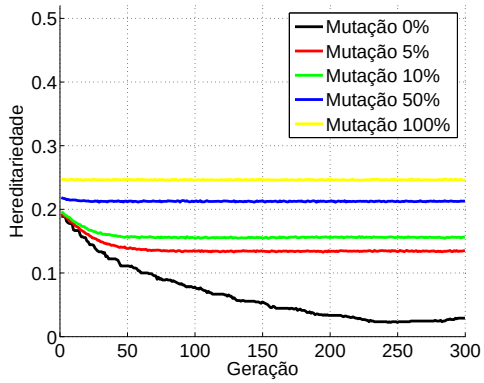
A hereditariedade mede a capacidade dos operadores de recombinação em preservar as características dos indivíduos pais nos novos indivíduos (Raidl e Julstrom, 2001).

No caso de representações para PPRs, a hereditariedade pode ser avaliada pelo tamanho do conjunto de arestas que estão presentes nos filhos e que não estão presentes nos pais (Carrano et al., 2007). Quanto menor o tamanho desse conjunto, maior é a hereditariedade do operador de recombinação e, portanto, da representação.

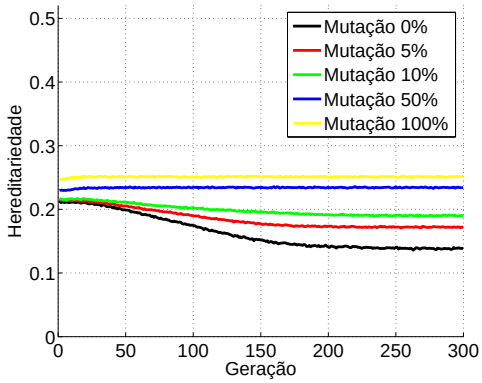
Para calcular o número de arestas dos filhos que não estão presentes nos pais, consideram-se $E_P = E_{P_1} \cup E_{P_2}$ a união dos conjuntos de arestas dos pais, $E_O = E_{O_1} \cup E_{O_2}$ a união dos conjuntos de arestas dos filhos e $|E_{O-P}| = |E_O - E_P|$ o total de arestas da união das arestas dos filhos que não estão na união dos conjuntos de arestas dos pais. A medida apresentada nos resultados desta Seção é o valor normalizado de $|E_{O-P}|$, definido como $Hereditariedade = |E_{O-P}|/(n-1)$.

Os testes empíricos para verificar a hereditariedade dos operadores de recombinação NOX e NPBX são realizados utilizando os grafos Euclidianos com $n = 10,\quad 20,\quad 40,\quad 80$ vértices (Ver Seção 6.2), com os filhos substituindo todos os pais na população, sem o uso de nenhuma pressão seletiva. A cada geração, são geradas 10000 soluções e são execu-

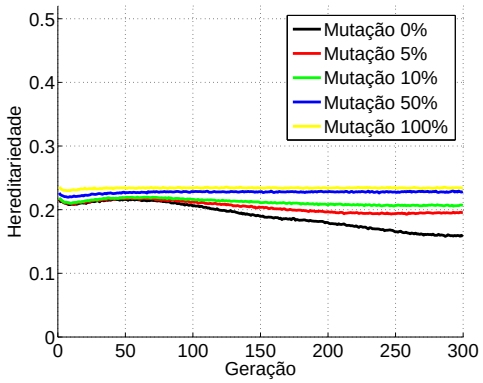
tadas 300 gerações. Para cada aplicação do operador, são calculados a *Hereditariedade*, conforme definido anteriormente, e o seu valor médio na população.



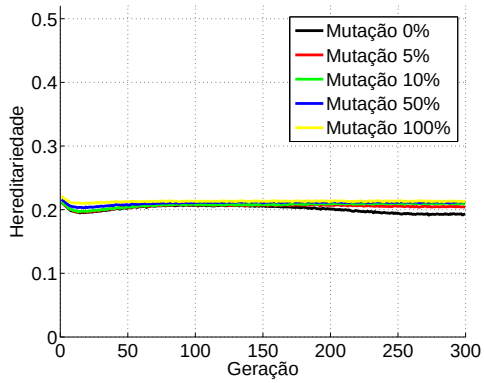
(a) Grafo com 10 vértices.



(b) Grafo com 20 vértices.



(c) Grafo com 40 vértices.

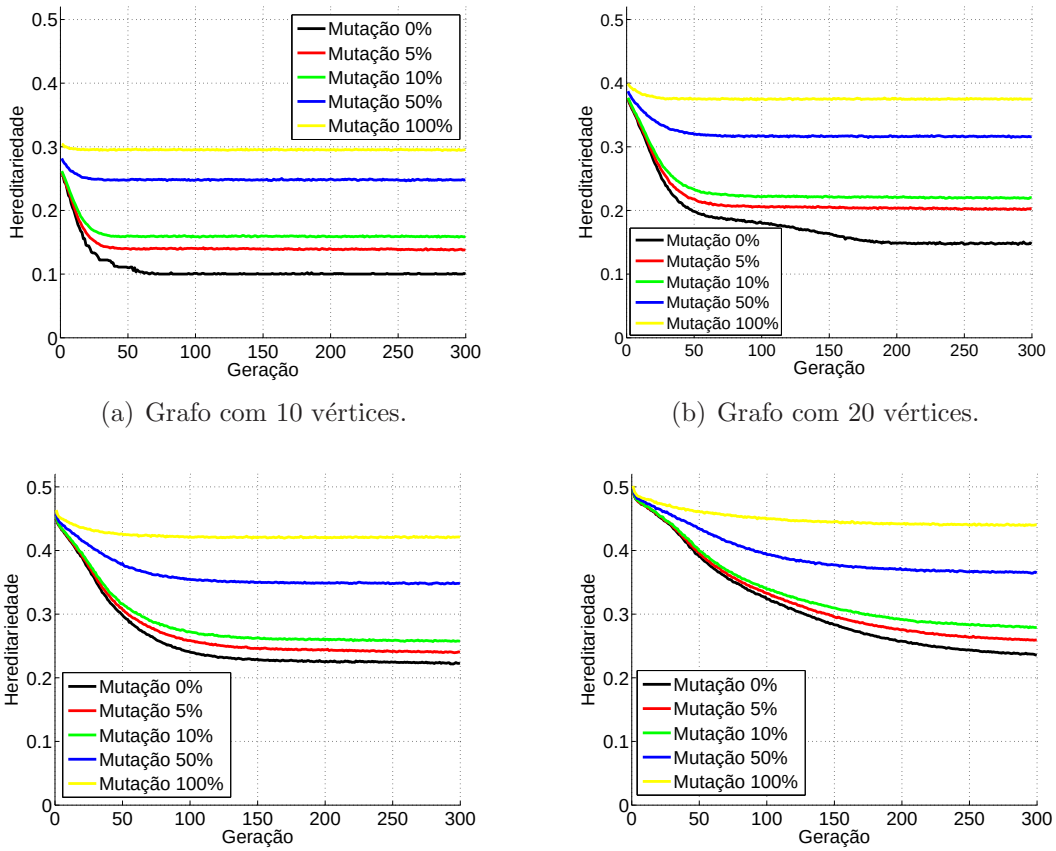


(d) Grafo com 80 vértices.

Figura 6.7: Hereditariedade do operador NOX.

A Figura 6.7 apresenta os resultados de *Hereditariedade* para o NOX. Observa-se que para o grafo com 10 vértices (ver Figura 6.7(a)) que, no decorrer das gerações, a *Hereditariedade* melhora a ponto de alcançar valor quase zero. No entanto, conforme aumenta o número de vértices, a *Hereditariedade* do NOX tende a estabilizar próximo a 0.2. Como era de esperar, o uso da mutação combinada com NOX, diminuir a *Hereditariedade*. Porém, esse efeito não é relevante para os grafos maiores.

A Figura 6.8 mostra os resultados de *Hereditariedade* para o NPBX. Observa-se que a *Hereditariedade* do NPBX, no melhor caso, é em torno de 0.1 para o grafo com 10 vértices (ver Figura 6.8(a)). Com o aumento do número de vértices do grafo, o melhor valor de *Hereditariedade* obtido por NPBX é de 0.2. Assim como no NOX, o uso da mutação em conjunto com o NPBX aumenta o número de arestas dos filhos não presentes nos pais conforme aumenta-se a taxa de aplicação da mutação de PAO e CAO.



(a) Grafo com 10 vértices.

(b) Grafo com 20 vértices.

(c) Grafo com 40 vértices.

(d) Grafo com 80 vértices.

Figura 6.8: Hereditariedade do operador NPBX.

Em comparação com NOX, a hereditariedade de NPBX é ruim, pois com o aumento do número de vértices os valores iniciais de *Hereditariedade* pioram, enquanto que no NOX esses valores mantém-se estáveis. Com isso, o NOX mostra-se um operador de recombinação mais adequado do que NPBX.

6.5 Complexidade de Tempo

A complexidade de tempo de uma representação é determinada por meio do tempo envolvido na codificação e decodificação, bem como na aplicação dos operadores de recombinação e mutação (Raidl e Julstrom, 2003). A complexidade é calculada com base no número de vértices (n) ou arestas (m) do grafo ou no tamanho $|T|$ das árvores envolvidas.

O processo de codificação de uma árvore na representação NDDE pode ocorrer por meio da aplicação de uma busca em profundidade em grafos ou por um percurso em pré-ordem (Gross e Yellen, 2004; Cormen et al., 2000). A busca em profundidade em um

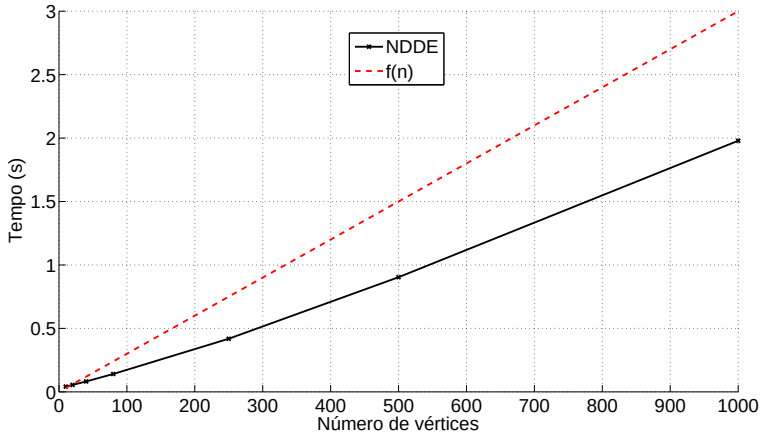
grafo tem complexidade $O(n+m)$ (ver Apêndice A). No caso de uma árvore $m = n-1$. Assim, o algoritmo para codificar a NDDE baseado na busca em profundidade é $O(n)$. Se for utilizado o percurso em pré-ordem, a complexidade é a mesma, pois esse percurso é $O(n)$ (ver Apêndice A).

O processo de decodificação, seguindo o Algoritmo 4.3.3, possui complexidade de tempo $O(|T_x|)$, pois esse processo necessita de percorrer a NDDE uma única vez para construir a árvore representada pela NDDE. O processo de decodificação é utilizado principalmente no cálculo de *fitness* do problema, quando a representação é utilizada em um EA. No caso de problemas em que a função de $fitness$ é linear e são aplicados os operadores PAO e CAO, é possível atualizar o $fitness$ dos novos indivíduos obtidos em tempo constante $O(1)$, pois sabe-se exatamente quais as arestas removidas e inseridas na árvore, bastando aumentar ou diminuir o *fitness* de forma proporcional à inclusão ou à exclusão dessas arestas. Assim, o algoritmo de decodificação só precisa ser utilizado no cálculo de *fitness* das soluções iniciais e na decodificação da solução final obtida para o problema.

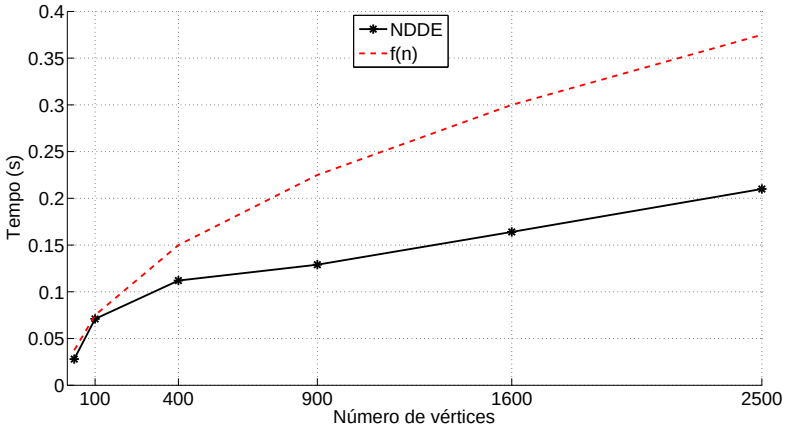
PAO e CAO

Na seqüência é analisada a complexidade de tempo para aplicação dos operadores de recombinação e mutação da NDDE. A análise de complexidade dos operadores de mutação PAO e CAO foi desenvolvida pelos pesquisadores Alexandre C. B. Delbem e Guilherme P. Telles e é apresentada no Apêndice B. Têm-se pela análise apresentada que PAO e CAO possuem complexidade de tempo média $O(|T_{origem}|+|T_{destino}|+t)$, onde $|T_{origem}|$ e $|T_{destino}|$ são os tamanhos das NDDEs T_{origem} e $T_{destino}$ respectivamente e t é o número de árvores na floresta. No caso em que a floresta possui somente uma árvore, a complexidade de tempo de PAO e CAO é $O(n)$, pois o tamanho da árvore, nesse caso, é o próprio número de vértices do grafo. No caso em que o problema possui $O(\sqrt{n})$ árvores, de acordo com a análise apresentada no Apêndice B, têm-se que o tamanho médio das árvores também é $O(\sqrt{n})$. Assim, a complexidade de tempo de PAO e CAO passa a ser em média $O(\sqrt{n})$.

A Figura 6.9 apresenta um teste empírico avaliando o tempo necessário para aplicação dos operadores PAO e CAO em (i) florestas com somente uma árvore e em (ii) florestas com $O(\sqrt{n})$ árvores. No caso (i), utilizam-se grafos completos com número de vértices que variam de 10 a 100 vértices. No caso (ii), empregam-se os grafos completos com número de vértices entre 10 e 2500. As curvas da Figura 6.9 foram construídas com os valores médios de 30 execuções a partir do tempo acumulado para aplicar o operador PAO/CAO 10000 vezes. O operador CAO possui comportamento similar ao de PAO, por isso, são apresentados somente os resultados referentes ao PAO. Na Figura 6.9(a),



(a) Tempo de execução para florestas com uma árvore.



(b) Tempo de execução para florestas com $O(\sqrt{n})$ árvores.

Figura 6.9: Tempo de execução de PAO na NDDE para grafos completos.

observa-se o resultado para o caso (ii) na curva preta, a curva vermelha tracejada indica a função $f(n) = 0.003n$ como limitante superior do tempo de computação medido para o PAO. Com isso observa-se que o operador PAO possui o comportamento linear previsto pela análise de complexidade de tempo.

A Figura 6.9(b) mostra o caso (ii) com $O(\sqrt{n})$ árvores. A curva preta foi construída com base nos valores médios obtidos por aplicações do PAO. A curva vermelha corresponde à função $f(n) = 0.0000017\sqrt{n}$ que é um limitante superior para o tempo de computação esperado para o operador. Novamente, observa-se que o comportamento do operador na prática segue a complexidade de tempo determinada por análise.

A seguir é analisada a complexidade de tempo para os operadores de recombinação propostos para a NDDE. A análise de complexidade de tempo dos operadores NOX e

NPBX é mais simples e similar entre eles, por isso será efetuada primeiro. Por fim, apresenta-se a analise a complexidade de tempo de EHR.

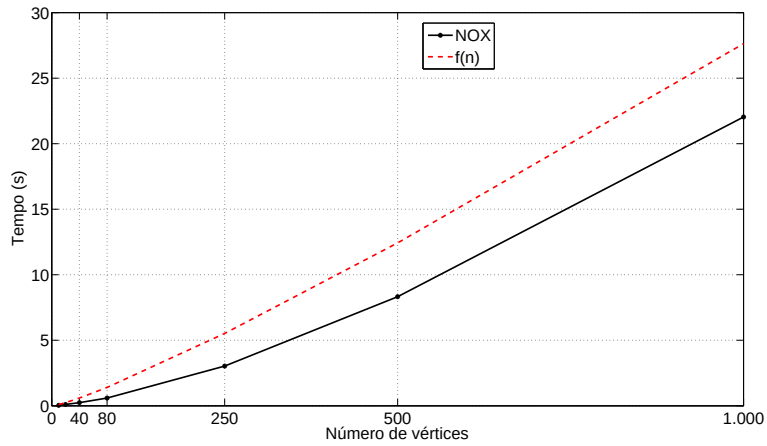
NOX e NPBX

Seguindo os passos dos Algoritmos 5.2.1 e 5.2.2 (ver Seção 5.2.1), tem-se que o Passo 1 de NOX requer tempo constante O(1), porém, no NPBX, o Passo 1 depende do número de posições que serão selecionadas para copiar do pai no filho. No pior caso, esse número de posições é n. Portanto, para o NPBX, o Passo 1 é O(n).

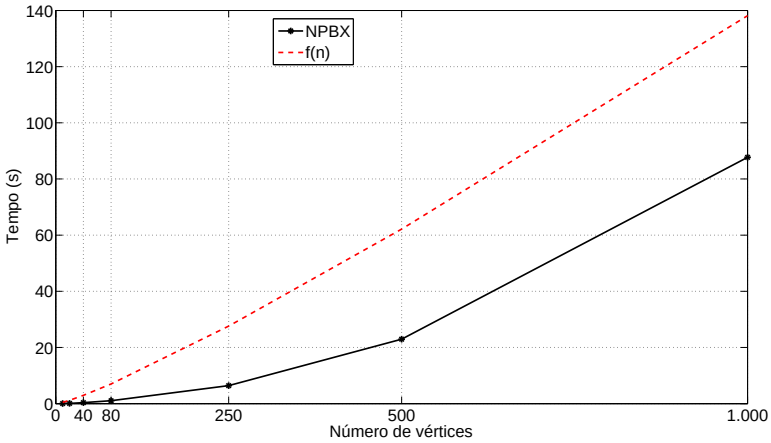
Em ambos os operadores, o Passo 2 consiste em marcar os vértices ou da região de corte ou das posições selecionadas. A região de corte tem, no pior caso, tamanho $n-2$, pois o vértice raiz e o último vértice obrigatóriamente não pertence a região de corte. Supondo no NPBX um número variável de posições, pode-se dizer que no pior caso são selecionadas n posições. Como para o acesso ao *array* auxiliar de marcação A_1 (A_2) utiliza-se o rótulo do vértice como índice, os acessos para cada vértice são efetuados em tempo constante. Assim, o Passo 2 requer no máximo n acessos e, portanto, é $O(n)$.

Os Passos 3, 4 e 5 de NOX são responsáveis por construir a NDDE do filho a partir dos pais. A cada cópia de gene para o filho, os Passos 3 e 5 verificam se o vértice é marcado, caso afirmativo incrementa-se o índice do pai; caso contrário, copia-se o gene. O processo de verificação, dado o acesso direto pelo rótulo do vértice ao *array* auxiliar, é efetuado em tempo constante: $O(1)$. Com os Passos 3 e 5 pode acontecer de ser percorrido todo o pai para preencher as posições do filho fora da região de troca, então para executar esses dois passos necessita-se de um tempo $O(n)$. No Passo 4, são copiados os genes da região de troca, não ocorrendo o processo de verificação e, como essa região possui tamanho médio $n/2$, esse passo é $O(n)$. No Passo 6, para verificar se o indivíduo gerado é redundante, a NDDE é percorrida uma vez verificando se o vértice está em posição redundante em relação aos seus irmãos. Esse processo necessita de tempo de execução $O(n)$. Como apresentado na Seção 6.1, o Algoritmo 6.1.1 que corrige a NDDE redundante necessita de tempo $O(n \log n)$. Não considerando a eliminação de redundância, tem-se que o NOX possui complexidade de tempo médio de execução $O(n)$, porém, com a eliminação da redundância a complexidade do NOX passa a ser $O(n \log n)$.

No NPBX o Passo 3 é similar ao conjunto dos Passos 3, 4 e 5 do NOX. Assim, o Passo 3 do NPBX necessita de tempo de execução $O(n)$. O Passo 4 do NPBX é igual ao Passo 6 do NOX, com tempo $O(n)$. Novamente, sem a correção de redundância, tem-se que o NPBX possui complexidade de tempo de execução $O(n)$ e, com a aplicação da correção de indivíduos redundantes, $O(n \log n)$.



(a) Tempo de execução de NOX.



(b) Tempo de execução de NPBX.

Figura 6.10: Tempo de execução NOX e NPBX na NDDE.

A Figura 6.10 apresenta um teste empírico avaliando o tempo necessário para aplicação dos operadores NOX e NPBX utilizando grafos completos com número de vértices que variam de 10 a 1000 vértices. As curvas da Figura 6.10 foram construídas com os valores médios de 30 execuções a partir do tempo acumulado para aplicar os operadores NOX (NPBX) 10000 vezes. Observa-se o resultado empírico de cada um dos operadores mostrado na curva preta das Figuras 6.10(a) e 6.10(b). Para NOX, a curva vermelha tracejada indica a função $f(n) = 0.004n\log(n)$ como limitante superior do tempo de computação medido. Para NPBX, a curva vermelha tracejada indica a função $f(n) = 0.02n\log(n)$. Com isso, observa-se que tanto NOX quanto NPBX possuem o comportamento coerente com a análise de complexidade de tempo.

EHR

A análise da complexidade de tempo para o EHR, com base no Algoritmo 5.2.3, é um pouco mais complicada para os Passos 2 e 3. A operação do Passo 2 pode ser executada, no pior caso, em tempo $O(g)$, onde g é o número de indivíduos gerados, pois, nesse caso, o ancestral comum é o primeiro indivíduo gerado.

Para o Passo 3, é necessário buscar na Π_{v_x} a posição e árvore de cada um dos vértices armazenados para cada modificação a ser aplicada. No pior caso, são 3 os vértices armazenados, o tempo de busca na Π_{v_x} para cada vértice e cada ancestral é $O(log|\Pi_{v_x}|)$. Considerando que serão aplicadas k (constante positiva) modificações, então o tempo total para executar o Passo 3 é $3k \log|\Pi_{v_x}|$, portanto, $O(\log|\Pi_{v_x}|)$. Assim, a complexidade dos Passos 2 e 3 é $O(log|\Pi_{v_x}| + g)$, a qual resulta em $O(g)$, uma vez que $|\Pi_{v_x}| = O(g)$ (ver Apêndice B). Por meio da aplicação de um processo de reinicialização da população (ver Seção 5.1), é possível limitar o número de indivíduos gerados a partir de um ancestral comum, g, por uma constante. Com isso, a complexidade desses passos resulta em $O(1)$.

O Passo 4 pode ser executado em tempo $O(1)$ pois consiste em selecionar um número aleatório de modificações. O Passo 5 corresponde a k aplicações do operador PAO. Cada aplicação é $O(|T_{origem}| + |T_{destino}|)$ (ver Apêndice B). Como k é uma constante positiva, a complexidade de tempo do EHR resulta em $O(|T_{origem}| + |T_{destino}|)$. Esse resultado é o mesmo obtido para o PAO (CAO).

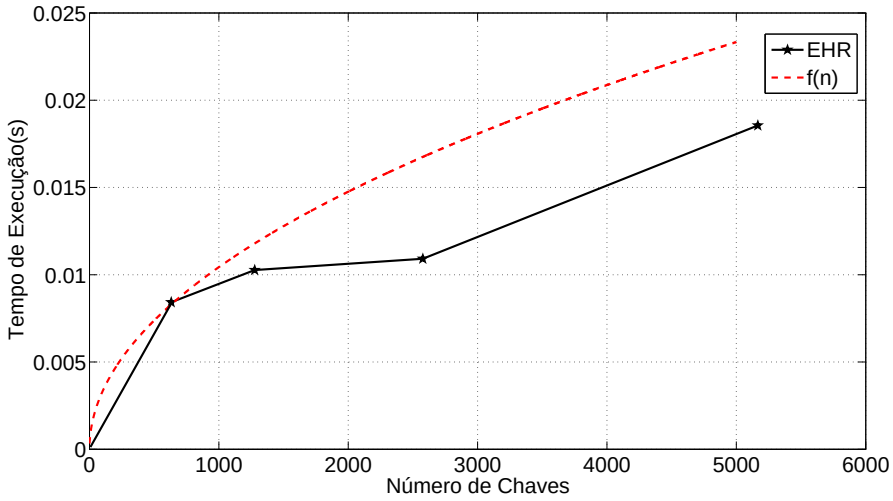


Figura 6.11: Tempo de execução do EHR.

A Figura 6.11 apresenta os resultados de tempo de computação do EHR aplicado ao PPR de reconfiguração de sistema de distribuição de energia elétrica (ver Seção 2.2), cujo

grafo possui naturalmente $O(\sqrt{n})$ árvores. Observa-se que o EHR para esse problema é limitado pela curva ($f(n) = 0.00033\sqrt{n}$), cujo resultado é coerente com a análise teórica da complexidade de tempo apresentada para EHR.

Para problemas que não são naturalmente divididos em $O(\sqrt{n})$ árvores, o Apêndice B apresenta uma forma de modelá-los utilizando $O(\sqrt{n})$ árvores. Assim, os operadores PAO, CAO e EHR mantém a limitante da sua complexidade de tempo em $O(\sqrt{n})$.

6.6 Considerações Finais

Este Capítulo apresentou a análise das propriedades de tendência, redundância, localidade, hereditariedade e complexidade de tempo para a representação NDDE e seus operadores. Vale ressaltar dessa análise que os operadores NOX e NPBX possuem tendência de aproximar as soluções a árvores estrela e que o uso dos operadores de mutação PAO e CAO em conjunto com NOX ou NPBX diminui o efeito dessa tendência, pois os operadores PAO e CAO não possuem tendência. O EHR, por ser baseado no PAO, também não possui tendência.

Outro ponto importante é que o operador PAO possui hereditariedade em torno de 0.2. Por outro lado, o NPBX possue em geral hereditariedade inferior a do NOX. No inicio do processo evolutivo, a hereditariedade do NPBX é relativamente baixa e pode diminuir com o aumento do número de vértices do grafo. Felizmente, para PAO, CAO e NOX a tendência e a hereditariedade melhoram com o aumento do número de vértices do grafo. Isso é adequado uma vez que a NDDE foi desenvolvida para lidar com PPRs de larga-escala.

A complexidade de tempo é outro fator importante a ser analisado. Verificou-se que PAO e CAO tem complexidade $O(|T_{origem}| + |T_{destino}| + t)$; NOX e NPBX são $O(n)$. Esses mesmos operadores passam a ter complexidade $O(n \log n)$ se for aplicado o método de correção da redundância. O EHR possui complexidade de tempo igual a do PAO, $O(n)$ para a floresta com uma única árvore e $O(\sqrt{n})$ para floresta com $O(\sqrt{n})$. Mesmo para grafos com menos de $O(\sqrt{n})$ árvores, é possível modelá-los de forma que os operadores trabalhem com se houvesse $O(\sqrt{n})$ árvores. Assim, é possível manter a complexidade de tempo $O(\sqrt{n})$ para PAO, CAO e EHR.

Deve-se ressaltar também que a NDDE possui alta localidade, quase perfeita.

O Capítulo 7 apresenta os resultados da NDDE aplicadas a PPRs importantes presentes na literatura.

Capítulo

7

Resultados Experimentais em Problemas de Projeto de Redes

Este Capítulo avalia a representação Nó-Profundidade-Grau (NDDE) para Algoritmos Evolutivos (EAs) aplicados a diferentes tipos de problemas de projeto de redes (PPRs). Os problemas utilizados nos testes podem ser classificados segundo o tipo de rede envolvida:

1. Redes modeladas por florestas geradoras com uma única árvore em grafos completos;

2. Redes modeladas por florestas geradoras com mais de uma árvore, para grafos não-completos.

Nos testes para os PPRs do Item 1, utiliza-se o operador de recombinação NOX, visto que este operador apresentou melhores resultados segundo as propriedades avaliadas no Capítulo 6. O EA para esses problemas é denominado EA**ND**, indicando que utiliza operadores da **ND**DE (PAO, CAO e NOX). Para o PPR do Item 2, utiliza-se o operador de recombinação EHR, que possibilita lidar com florestas geradoras com mais de uma árvore e com grafos não-completos. O EA para esse problema é chamado de EA**N**, destacando que utiliza um operador da **N**DE, no caso o operador 1. Deve-se observar que o EHR foi implementado com base no operador 1 (ver Seção 4.3.2).

As Seções deste Capítulo estão organizadas como segue. A Seção 7.1 apresenta os parâmetros de configuração do EAND e os resultados para os seguintes PPRs de floresta com uma única árvore: uma árvore máxima (OMTP), árvore geradora mínima com

restrição de grau (dc-MSTP) e árvore de comunicação ótima (OCSTP). A Seção 7.2 apresenta o EAN e sua aplicação para o PPR real com mais de uma árvore na floresta, o de reconfiguração de sistemas de distribuição de energia elétrica. Por fim, a Seção 7.3 destaca os principais resultados deste Capítulo.

7.1 Resultados para Florestas com Uma Única Árvore

Esta Seção apresenta os resultados para os PPRs cujas redes são modeladas por florestas geradoras com uma única árvore. Os problemas considerados são: uma árvore máxima (OMTP, ver Seção 2.1.3), árvore geradora mínima com restrição de grau (dc-MSTPP, ver Seção 2.1.2) e árvore geradora mínima de comunicação ótima (OCSTP, ver Seção 2.1.1).

A Seção está organizada como segue. A Seção 7.1.1 apresenta os parâmetros de configuração do EAND aplicado aos problemas de teste. A Seção 7.1.2 mostra o desempenho do EAND para o OMTP. A Seção 7.1.3 descreve os resultados do EAND aplicado ao problema dc-MSTP. A Seção 7.1.4 apresenta os resultados do EAND para o OCSTP.

7.1.1 Parâmetros do EAND

O EAND é um EA *steady-state*, em que a cada geração são gerados 1 (no caso de aplicar somente mutação) ou 2 (no caso de aplicar a recombinação) novos indivíduos com elitismo total, ou seja, somente os melhores indivíduos do conjunto envolvendo pais e filhos sobrevivem. Além disso, não existem indivíduos repetidos na população. Se um novo indivíduo possui valor da função objetivo igual a outro indivíduo presente na população, os dois indivíduos são comparados e o novo indivíduo entra na população somente se possuir genótipo diferente do que já está presente na população (Raidl e Julstrom, 2003). Os pais são selecionados por meio do torneio de 3 com reposição. O critério de parada verifica se ocorre um certo número de avaliações sem que haja melhoria do melhor indivíduo encontrado. O valor escolhido para o critério de parada é de 10000 avaliações sem melhoria. O algoritmo 7.1.1 apresenta o pseudõ-código de EAND.

Os testes com o EAND envolvem populações de tamanho 10, 100 e 500 indivíduos. As populações iniciais são geradas utilizando Número de Prüfer para garantir soluções que não possuam nenhum tendência em privilegiar algum tipo de topologia de árvore. Os testes envolvem variação da taxa de mutação com valores 0, 0.01, 0.05, 0.1, 0.5, e 1.0,

Algoritmo 7.1.1: - Pseudocódigo de EAND.

(1) Gera a população inicial;
(2) Avalia a população inicial;
(3) Ordena a população inicial pelo valor da função objetivo;
(4) Enquanto não for atingido o critério de parada;
    (5) Seleciona 2 pais por torneio;
    (6) Se pertence a taxa de recombinação aplica a recombinação;
    (7) Se pertence a taxa de mutação aplica a mutação;
    (8) Avalia os novos indivíduos;
    (9) Se o valor da função objetivo dos novos indivíduos pertence a população verifica se os indivíduos são iguais. Em caso afirmativo não entra na população. Caso contrário insere na população de forma ordenada;
    (10) Caso o valor da função objetivo dos novos não pertence a população, insere os novos indivíduos na população de forma ordenada;

considerando taxa de recombinação fixa em 1.0. Os testes também envolvem a variação da taxa de recombinação com os valores 0, 0.1, 0.3, 0.5, e 0.8, com taxa de mutação fixa em 1.0[1]. Em todos os testes, os operadores de mutação PAO e CAO são aplicados com a mesma probabilidade.

As configurações do EAND são aplicadas para todos os PPRs de floresta com uma árvore em que é utilizado. Nos gráficos de resultados deste Capítulo (Figuras 7.1 a 7.13) a legendas utilizam a seguinte notação:

- c0m1: taxa de recombinação 0 e taxa de mutação 1;
- c0.1m1: taxa de recombinação 0.1 e taxa de mutação 1;
- c0.3m1: taxa de recombinação 0.3 e taxa de mutação 1;
- c0.5m1: taxa de recombinação 0.5 e taxa de mutação 1;
- c0.8m1: taxa de recombinação 0.8 e taxa de mutação 1;
- c1m1: taxa de recombinação 1 e taxa de mutação 1;
- c1m0: taxa de recombinação 1 e taxa de mutação 0;
- c1m0.01: taxa de recombinação 1 e taxa de mutação 0.01;
- c1m0.05: taxa de recombinação 1 e taxa de mutação 0.05;

[1]Quando não é aplicada a recombinação, a mutação é aplicada diretamente para um dos pais selecionados para a recombinação.

- c1m0.1: taxa de recombinação 1 e taxa de mutação 0.1;
- c1m0.5: taxa de recombinação 1 e taxa de mutação 0.5.

7.1.2 Uma Árvore Máxima

O OMTP consiste em, dada uma árvore geradora T_{opt} chamada de árvore ótima, encontrar a árvore T_b que possui as mesmas arestas da árvore ótima (ver Seção 2.1.3). O valor da função objetivo para o OMTP utilizado no EAND, é dado pelo número de arestas de T_b que correspondem a arestas de T_{opt} dividido pelo total de arestas da árvore, isto é, $\frac{|T_{opt}-T_b|}{n-1}$, em que n é o número de vértices do grafo. O objetivo é maximizar o valor da função objetivo, assim o valor ótimo é atingido se o valor da função objetivo é igual a 1. Vale ressaltar nesse problema que o espaço de busca cresce exponencialmente em relação ao número de vértices do grafo, pois o número de árvores geradoras de um grafo completo com n vértices é n^{n-2} (Gross e Yellen, 2004). Em outras palavras, encontrar a árvore T_{opt} para n grande é um problema difícil.

O conjunto de testes do OMTP para avaliar o desempenho do EAND é composto por três tipos de T_{opt}:

1. Árvores geradoras do tipo linha;
2. Árvores geradoras aleatórias;
3. Árvores geradoras do tipo estrela.

Nas árvores geradoras tipo linha o grau máximo dos vértices é dois. As árvores geradoras com topologia aleatória são quaisquer árvores geradoras diferentes de árvores linha e estrelas. Nas árvores geradoras do tipo estrela existe um vértice central e os demais vértices estão conectados nele. O número de vértices presentes em cada tipo de árvore geradora testadas são $n = 10$, 20, 30, 40, 50, 100, 200, 300, 400, 500, 1000, 2500 e 5000. Para cada um dos três tipos de árvore geradora são construídas, de forma aleatória, 10 árvores geradoras diferentes (instâncias) para cada número de vértices.

Todas as configurações de taxa de mutação e taxa de recombinação (ver Seção 7.1.1) são aplicadas para todas as instâncias definidas para o problema. São realizadas 30 execuções de cada combinação de teste (tamanho da população, tipos de árvores T_{opt}, número de vértices, taxa de recombinação e taxa de mutação).

A seguir são apresentados os resultados do EAND para os casos propostos para o OMTP. Cada conjunto de resultados é sintetizado em uma Figura com dois gráficos:

1. O gráfico à esquerda mostra a avaliação em que foi obtido o melhor indivíduo, conforme o número de vértices do problema aumenta;

2. O gráfico à direita mostra o valor da função objetivo do melhor indivíduo encontrado conforme aumenta o número de vértices do problema.

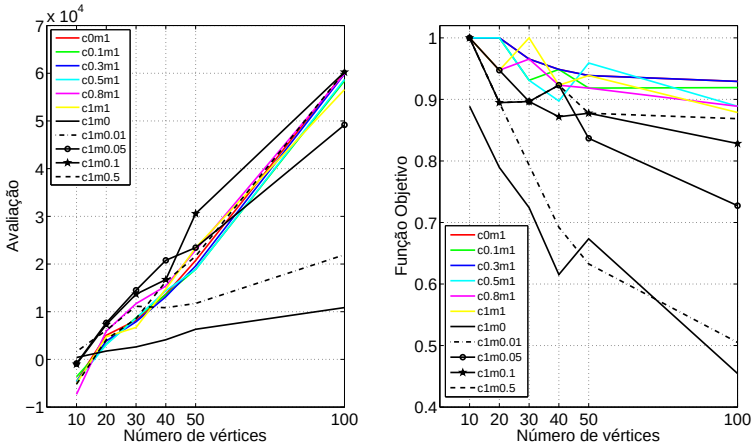
Árvores geradoras do tipo linha

A Figura 7.1 apresenta os resultados do EAND com os diferentes tamanhos de população para o conjunto de árvores linha com número de vértices até $n = 100$. Primeiramente, analisou-se o ajuste de parâmetros para 100 vértices, pois para até esse número de vértices há resultados na literatura utilizando EAs (Rothlauf, 2006; Raidl e Julstrom, 2003).

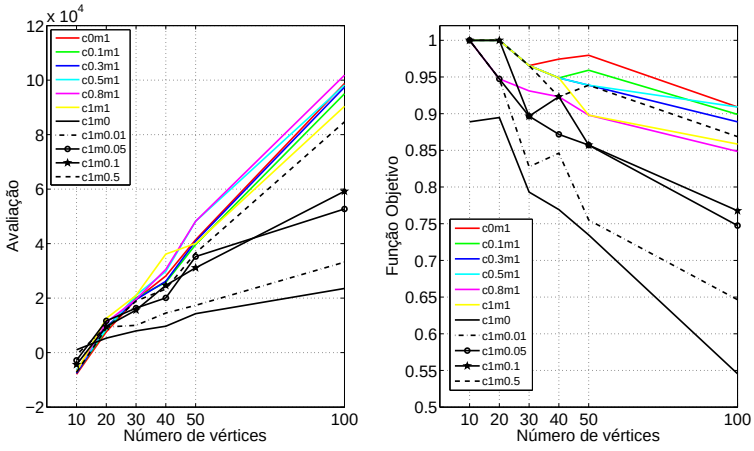
O primeiro parâmetro a ser analisado é o tamanho da população. A Figura 7.1(a) apresenta os resultados quando a população possui 10 indivíduos. Observa-se que nesse caso, poucas configurações conseguiram encontrar a solução ótima do problema (função objetivo = 1). As soluções ótimas foram encontradas, somente, para os grafos com até 30 vértices. No entanto, ao observar as Figuras 7.1(b) e 7.1(c), para as quais o EAND utilizou populações com 100 e 500 indivíduos, nota-se que apesar de melhorar o valor da função objetivo da melhor solução encontrada por algumas configurações, houve a piora da função objetivo de configurações que obtiveram o ótimo na população com 10 indivíduos. Assim, para o EAND aplicado ao OMTP com árvores geradoras do tipo linha, seria mais adequado utilizar população com 10 indivíduos.

Fixado o parâmetro de tamanho da população, analisa-se as taxas de recombinação e mutação. Comparando os dois gráficos da Figura 7.1(a), conclui-se que as configurações com taxa de aplicação da recombinação em 1.0 e de mutação pequena apresentam os piores resultados. O uso de mutação com taxa 1.0 produziu melhores resultados da função objetivo independente do número de vértices do grafo. A configuração c0.3m1 produz valores da função objetivo expressivos e ao mesmo tempo não necessita de um número excessivo de avaliações para encontrar essas soluções.

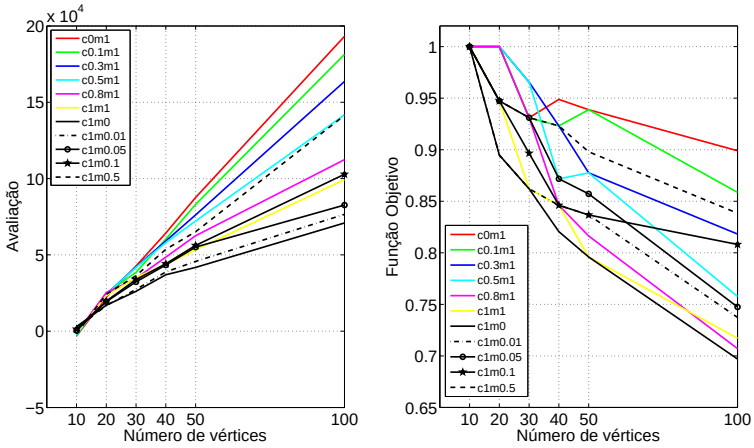
Nos resultados apresentados na literatura para árvores do tipo linhas, os EAs possuem como critério de parada encontrar a solução ótima sendo necessárias em torno de 1.2 milhões de avaliações para $n = 100$ (Raidl e Julstrom, 2003). O EAND não encontrou a solução ótima para esse número de vértices devido a diferença no critério de parada. Contudo, o EAND obteve o valor da função objetivo de 0.9 com apenas 60000 avaliações, indicando que o ótimo poderia ser encontrado com um número relativamente baixo de avaliações.



(a) População com 10 indivíduos.

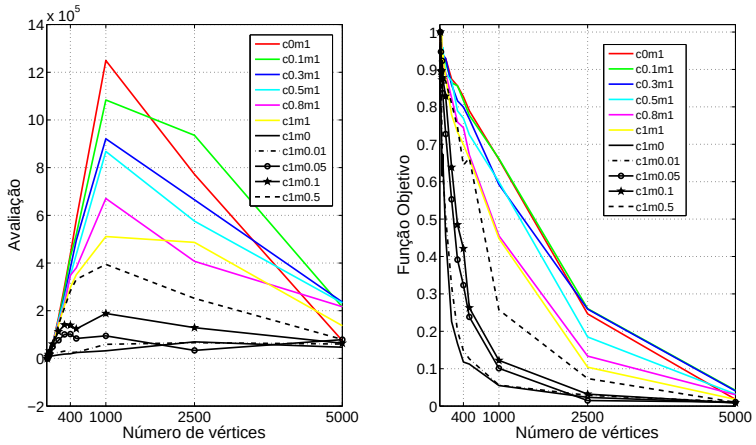


(b) População com 100 indivíduos.

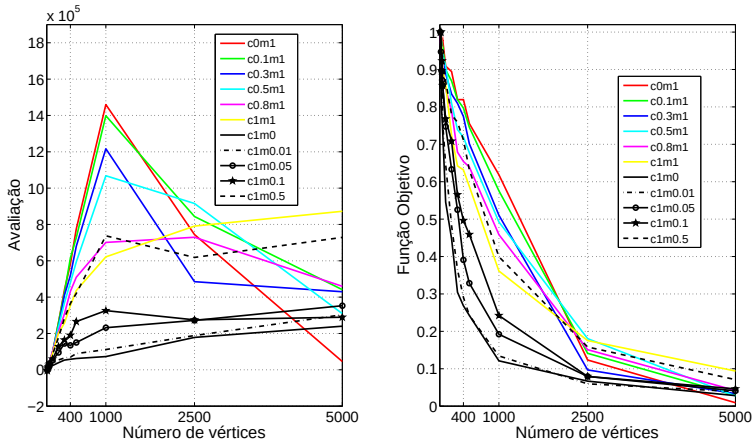


(c) População com 500 indivíduos.

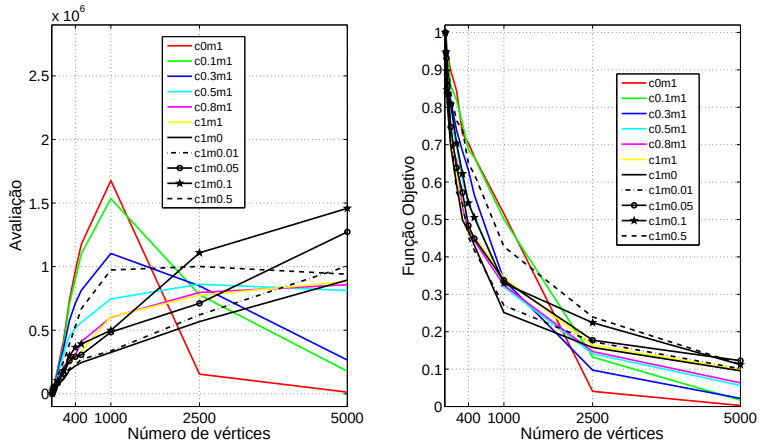
Figura 7.1: EAND aplicado ao OMTP com árvores geradoras do tipo linha com até 100 vértices.



(a) População com 10 indivíduos.



(b) População com 100 indivíduos.



(c) População com 500 indivíduos.

Figura 7.2: EAND aplicado ao OMTP com árvores geradoras do tipo linha com até 5000 vértices.

Deve-se observar que o objetivo principal dos testes apresentados é avaliar a contribuição dos novos operadores de recombinação no OMTP e não efetivamente encontrar a solução ótima.

A Figura 7.2 apresenta os resultados do EAND para todos os grafos propostos (n variando de 10 a 5000). Verifica-se novamente que com a população de 10 indivíduos, é possível obter melhores valores da função objetivo com um menor número de avaliações. Vale ressaltar que, com o critério de parada utilizado, para os casos com mais de 1000 vértices, o número de avaliações para atingir a melhor solução encontrada começa a decair, pois a dificuldade do problema aumenta demasiadamente.

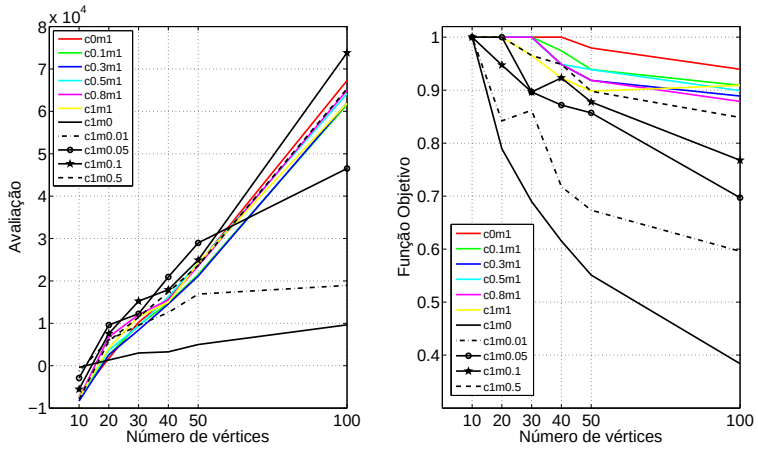
Conclui-se do conjunto de resultados apresentados para o OMTP com árvores geradoras do tipo linha que uma população de 10 indivíduos encontra melhores soluções do que populações maiores. Vale ressaltar também que a mutação mostrou-se muito importante para encontrar a solução ótima. Possivelmente, isso deve-se a tendência que o NOX possui em relação a árvores estrela, pois árvores do tipo linhas são o oposto de árvores do tipo estrela em termos de topologia. Assim, o uso excessivo de recombinação tende a gerar soluções que não contribuem para melhorar a qualidade das soluções.

Com relação ao aumento no tamanho da população, verifica-se que esse aumento pode contribuir para melhorar o valor da função objetivo obtido pelas configurações que possuem taxa de recombinação 1.0 e taxa de mutação pequena, diminuindo o efeito da convergência prematura.

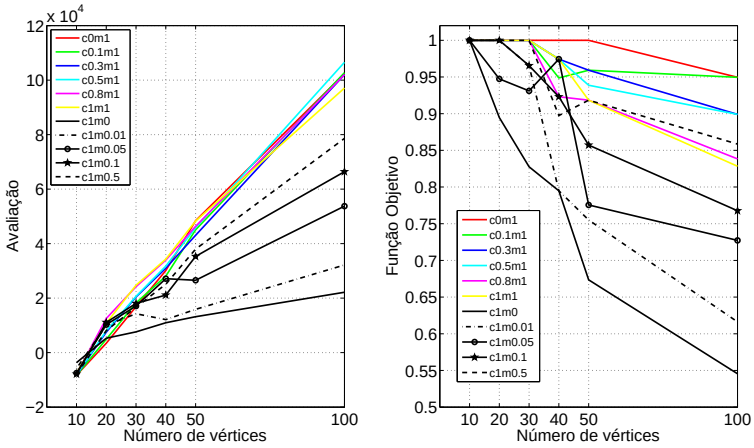
Árvores geradoras aleatórias

A Figura 7.3 apresenta os resultados do EAND aplicado ao OMTP com T_{opt} pertencente ao conjunto de árvore geradoras aleatórias com diferentes tamanhos de população. Começa-se analisando, novamente, o tamanho da população. As Figuras 7.3(a), 7.3(b) e 7.3(c) mostram os gráficos com os tamanhos de população 10, 100 e 500 indivíduos, respectivamente para grafos com até 100 vértices.

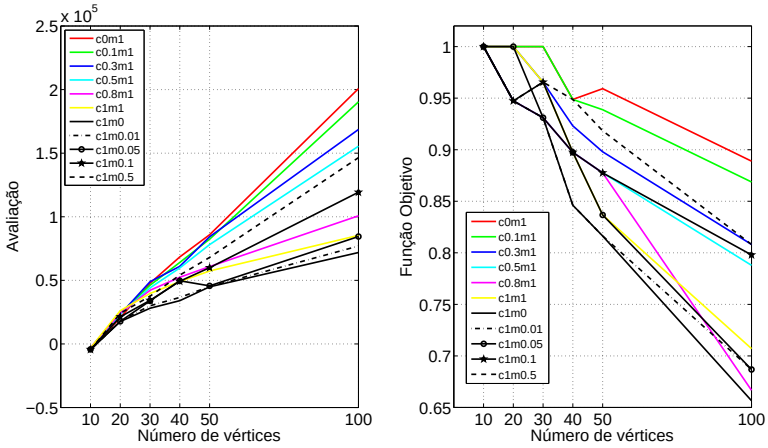
As populações com 10 e 100 indivíduos apresentaram valores da função objetivo melhores em relação aos da função objetivo da população com 500 indivíduos (Figura 7.3(c)). A principal diferença, entre a população com 10 e com 100 indivíduos é que com a segunda, foi possível obter a solução ótima para grafos com até 50 vértices. Porém, o tempo de convergência, medido pelo número de avaliações necessárias para encontrar o melhor indivíduo, é superior com a população maior. Levando em consideração que na população com 10 indivíduos a configuração c0m1 possui um valor da função objetivo próximo do ótimo e que, com população de tamanho 10, o tempo de convergência é menor, sugere-se



(a) População com 10 indivíduos.



(b) População com 100 indivíduos.



(c) População com 500 indivíduos.

Figura 7.3: EAND aplicado ao OMTP com árvores geradoras aleatórias com até 100 vértices.

novamente a utilização da população com 10 indivíduos.

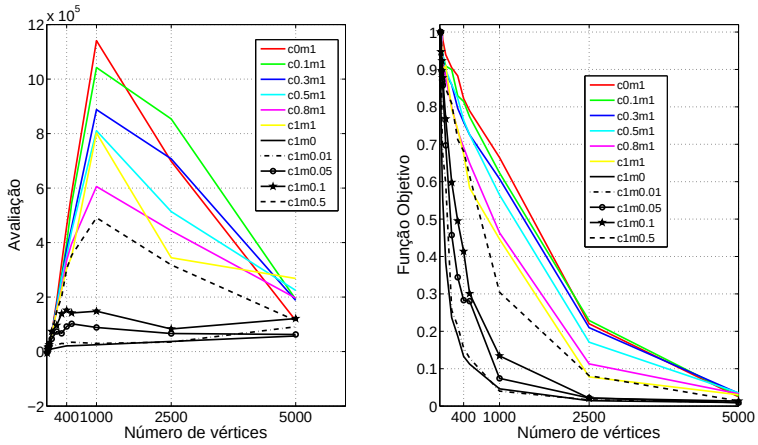
Escolhido um valor para o tamanho da população, analisa-se então qual a melhor opção de configuração para as taxas de mutação e recombinação. Observando os gráficos de número de avaliações e dos valores da função objetivo da Figura 7.3(b), a configuração mais indicada é c0m1. Essa configuração encontra, para todos os números de vértices, o melhor valor de função objetivo em comparação com as outras configurações. Adicionalmente, o tempo de convergência não é alto, apesar de necessitar de um número um pouco maior de avaliações do que as configurações que possuem valores próximos para a função objetivo.

De acordo com a Figura 7.3(c), o EAND possui comportamento coerente com o de EAs em geral, uma vez que com o aumento da população e o uso prioritário de recombinação, os valores da função objetivo são melhores dos que os obtidos com populações menores.

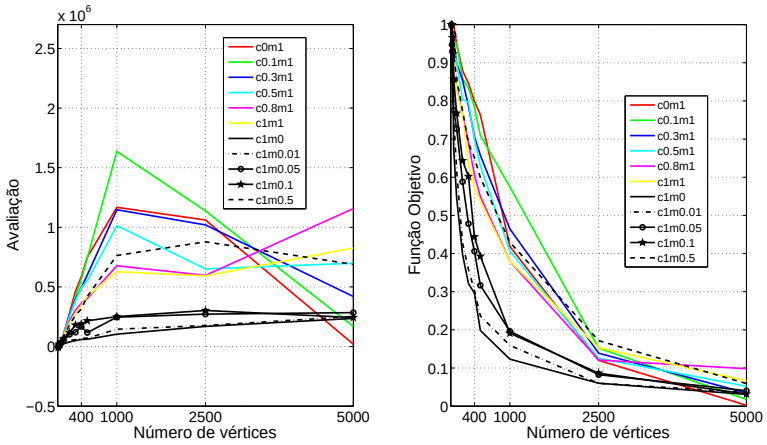
Assim como no caso de árvores geradoras do tipo linha, a Figura 7.4 apresenta o desempenho do EAND para os grafos com até 5000 vértices e diferentes tamanhos de população. Ao contrário do que ocorre para os casos menores, a população com 500 indivíduos encontra um valor de função objetivo ligeiramente superior para os grafos com 2500 e 5000 vértices, sendo o uso da recombinação fundamental para encontrar esses valores. Porém, para os demais grafos, o valor da função objetivo encontrado para população com 10 indivíduos é melhor.

Conclui-se dos testes com árvores geradoras aleatórias que o EAND consegue encontrar melhores soluções para casos maiores do que nos testes com geradoras do tipo linha. Considerando somente o valor da função objetivo obtido, a população de tamanho 100 encontrou o valor ótimo para um maior número de grafos. Por outro lado, para os casos com mais de 2500 vértices, a população com 500 indivíduos obteve a melhor solução encontrada.

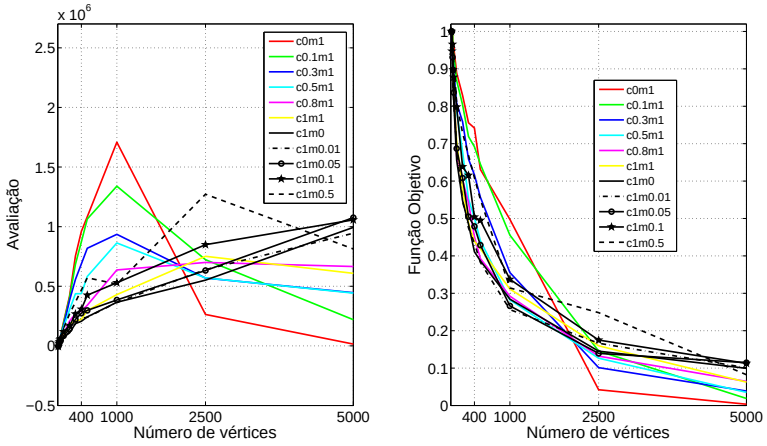
A literatura reporta que para encontrar o valor ótimo com 100 vértices são necessárias em torno de 700 mil avaliações (Raidl e Julstrom, 2003). O EAND aproxima-se do valor ótimo da função objetivo com aproximadamente 80 mil avaliações, devido ao critério de parada utilizado. Esse resultado sugere que o EAND, possivelmente atingiria o ótimo com um valor bem inferior de avaliações.



(a) População com 10 indivíduos.



(b) População com 100 indivíduos.



(c) População com 500 indivíduos.

Figura 7.4: EAND aplicado ao OMTP com árvores geradoras aleatórias com até 5000 vértices.

Árvores geradoras do tipo estrela

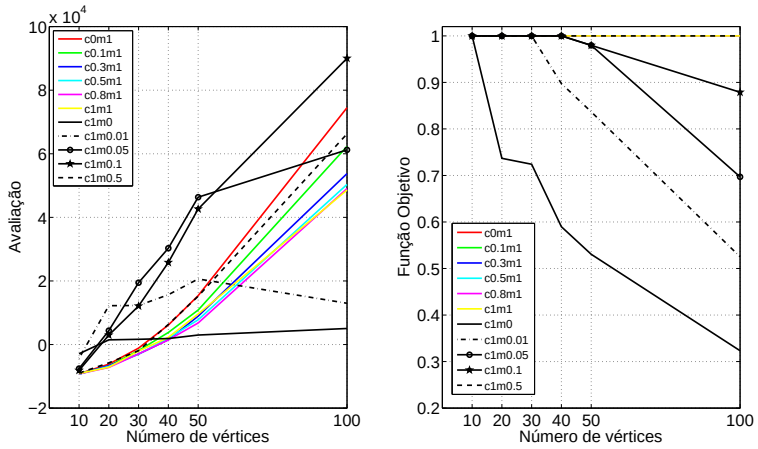
A Figura 7.5 mostra os resultados obtidos para os diferentes tamanhos de população utilizados para o EAND aplicado ao OMTP com T_{opt} pertencente ao conjunto de árvores geradoras do tipo estrela. Em comparação com os resultados obtidos para as outras duas topologias de árvores apresentadas, os valores da função objetivo obtidos pelo EAND para as diferentes configurações é significativamente melhor, encontrando o ótimo para todos os grafos com até 100 vértices.

Primeiramente, analisa-se qual o melhor tamanho de população a ser utilizado no EAND aplicando a árvore estrela. As Figuras 7.5(a), 7.5(b) e 7.5(c) apresentam os resultados para as populações com 10, 100 e 500 indivíduos respectivamente. Em todos os tamanhos de população, pelo menos uma configuração do EAND encontrou a solução ótima para todos os grafos com número de vértices até 100. Por outro lado, com a população de tamanho 10 a maioria das configurações, exceto as configurações c1m0, c1m0.01, c1m0.01 e c1m0.1, apresentaram melhores resultados usando recombinação com baixa mutação.

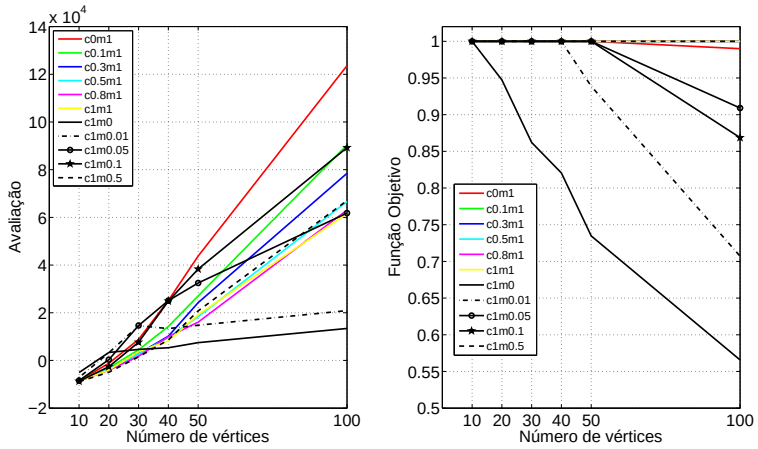
Esse resultado, contraria a expectativa de que com alta taxa de aplicação do NOX, poderia-se encontrar a solução ótima rapidamente. Essa expectativa baseava-se na tendência apresentada por NOX para soluções com topologia similar a estrelas. Com o aumento do tamanho da população, algumas configurações que encontravam a solução ótima passaram a não encontrar mais, além de ocorrer o aumento do número de avaliações necessárias para encontrar o melhor indivíduo. Assim, sugere-se que seja utilizada a população com 10 indivíduos.

Selecionado o tamanho de população, deve-se escolher a melhor configuração de taxa de aplicação de recombinação e mutação. Tendo em vista que a maioria das configurações obteve a solução ótima em todos os casos, utiliza-se a que precisou do menor número de avaliações para encontrar a solução ótima. Analisando o gráfico de avaliações, conclui-se que a melhor configuração é c0.8m1, que é também a configuração sugerida para a representação Conjunto de Arestas em (Raidl e Julstrom, 2003) para o OMTP.

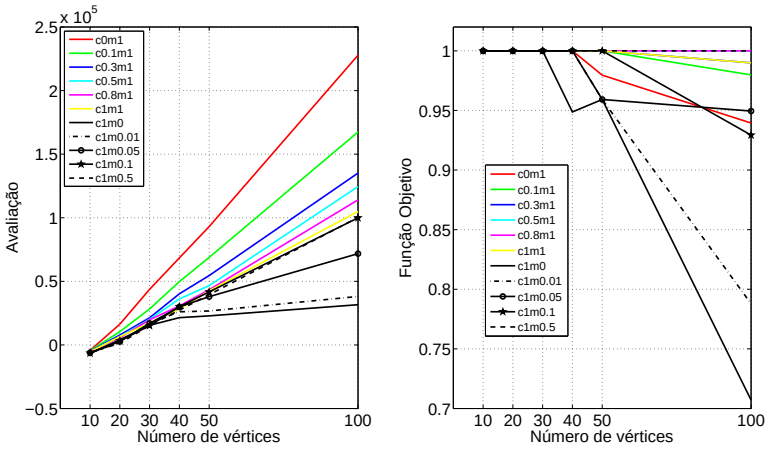
A Figura 7.6 apresenta o resultado do EAND para diferentes tamanhos de população. Ao contrário do ocorrido com resultados envolvendo T_{opt} de árvores geradoras do tipo linha e aleatórias, a população com melhor valor da função objetivo é a de tamanho 500. Além disso, altas taxas de recombinação e de mutação mostraram-se fundamentais para se obter os melhores resultados com os grafos maiores.



(a) População com 10 indivíduos.

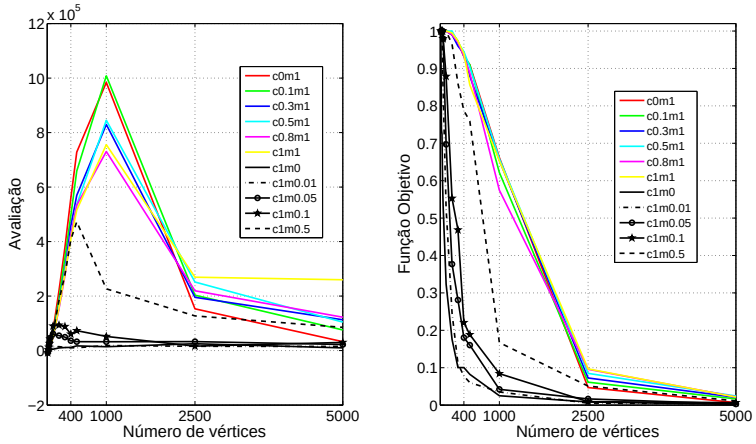


(b) População com 100 indivíduos.

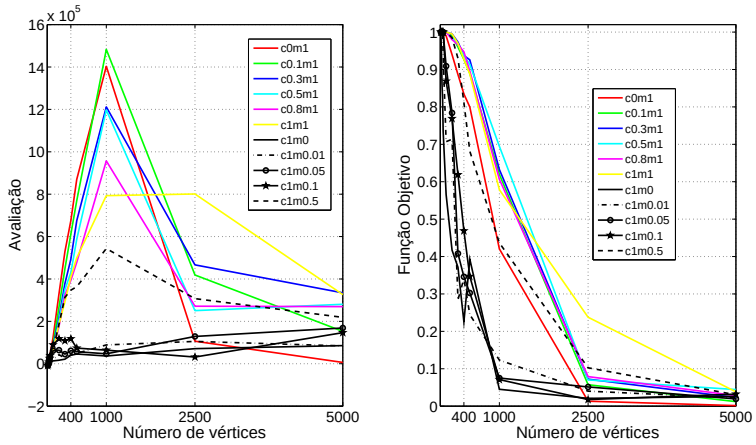


(c) População com 500 indivíduos.

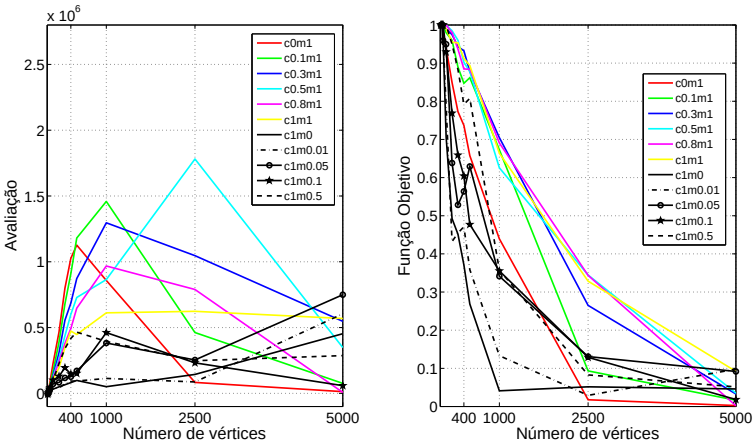
Figura 7.5: EAND aplicado ao OMTP com árvores geradoras do tipo estrela com até 100 vértices.



(a) População com 10 indivíduos.



(b) População com 100 indivíduos.



(c) População com 500 indivíduos.

Figura 7.6: EAND aplicado ao OMTP com árvores geradoras do tipo estrela com até 5000 vértices.

Os resultados apresentados na literatura para EAs aplicados ao OMTP com árvores do tipo estrela para $n = 100$ vértices requerem número de avaliações médio para obter o ótimo em torno de 190 mil (Raidl e Julstrom, 2003). O EAND com uma população de 10 indivíduos, consegue, em um caso de convergência média, obter o ótimo com aproximadamente, 50 mil avaliações, apresentando um número de avaliações relativamente baixo. Esse também é mais um indicativo de que o EAND deve encontrar o ótimo com menos avaliações para o OMTP de árvores geradoras tipo linha e aleatória.

Conclui-se de todos os casos de teste analisados que o uso da recombinação auxilia o EAND a encontrar melhores valores da função objetivo para casos com maior número de vértices e que sua contribuição é mais expressiva com populações maiores. Para o caso de T_{opt} pertencente ao conjunto de árvores do tipo estrela, a tendência do NOX para árvores estrela possibilitou, que nos casos até 100 vértices, um maior número de configurações encontrasse a solução ótima. Porém, foi preciso uma taxa de mutação expressiva acima de 50%. Para os diferentes tipos de topologia, em grafos com até 100 vértices, a população com 10 indivíduos mostrou-se a melhor opção.

Tabela 7.1: Número médio de avaliações em 30 execuções para OMTP utilizando EAND e GAES.

Instância		EAND	GAES
Topologia	n	Avaliações	Avaliações
Linha	10	1083	1026
	20	12599	5134
	50	205435	127065
	100	975638	1188204
Aleatória	10	958	836
	20	3939	3318
	50	61624	66353
	100	483732	625461
Estrela	10	484	446
	20	1613	1062
	50	13919	14742
	100	55171	113654

A Tabela 7.1 apresenta a média do número de avaliações necessárias para que EAND e GAES (um EA com a representação Conjunto de Arestas) encontrem a solução ótima

para algumas instâncias dos três tipos de topologias de árvores. Os resultados para GAES estão disponíveis na literatura em Raidl e Julstrom (2003). Observa-se que para todos as instâncias com 100 vértices EAND necessita de um número menor de avaliações do GAES. Para as topologias de árvore geradora aleatória e árvore geradora do tipo estrela EAND apresenta número menor de avaliações para instâncias com 50 vértices.

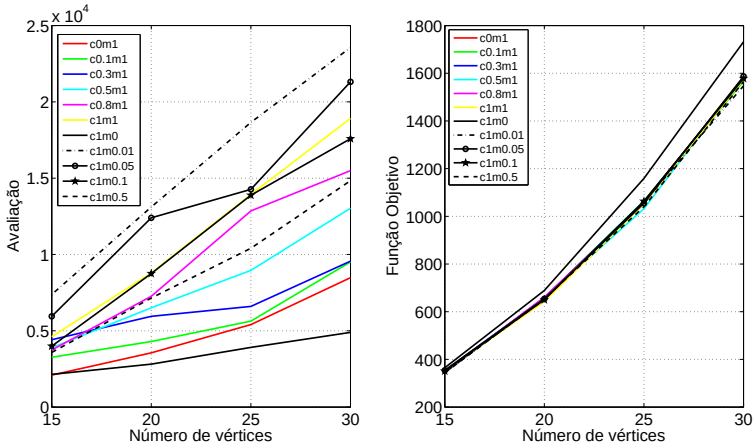
7.1.3 Árvore Geradora Mínima com Restrição de Grau

O dc-MSTP é um PPR em que dado um grafo, um conjunto de pesos positivos para as arestas e uma restrição d para o grau máximo da árvore, busca-se encontrar a árvore geradora que possua a soma de pesos mínima de tal forma que o grau máximo (deg_{max}) dos vértices da árvore seja $deg_{max} \leq d$ (ver Seção 2.1.2). Assim, esse é um problema de minimização no qual o valor da função objetivo de um indivíduo é a soma dos pesos da árvore.

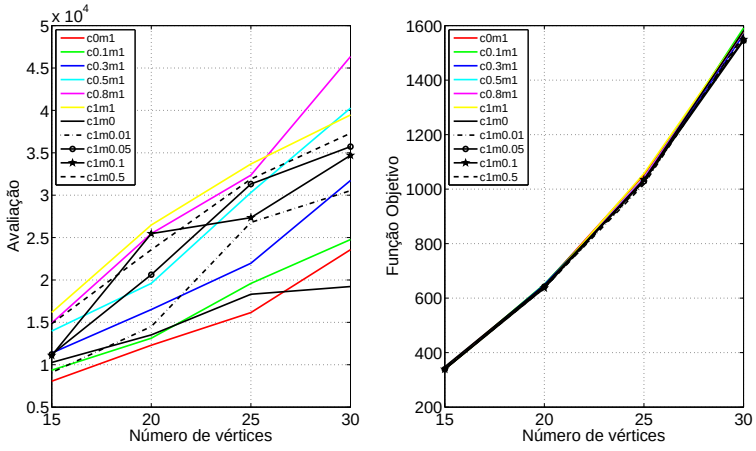
O EAND foi aplicado ao dc-MSTP em três conjuntos de grafos, sendo dois deles grafos de *benchmark* utilizados na literatura para avaliar o desempenho de representações de PPRs em EAs. O terceiro conjunto é composto por grafos de pesos aleatórios desenvolvidos para este trabalho.

O primeiro conjunto de casos de teste chamado de SHRD é composto por grafos com $n = 15, \quad 20, \quad 25, \quad 30$ vértices e foi proposto em (Krishnamoorthy et al., 2001) como instâncias difíceis para o dc-MSTP, pois possuem árvore geradora mínima com valores altos de grau máximo. O peso das arestas desses grafos é construído da seguinte forma: o primeiro vértice é conectado a todos os demais vértices com peso k, o segundo vértice é conectado com todos os outros (menos o vértice 1) por peso $2k$, e o i-ésimo vértice é conectado a todos os outros (menos os $i - 1$ vértices anteriores) por peso ik. Por fim, são obtidos valores diferentes para os pesos subtraindo de cada peso um valor aleatório uniforme entre 1 e 18 para $k = 20$. Os grafos que compõem esse conjunto de testes foram obtidos em (Raidl, 2008). Cada uma das configurações do EAND com os tamanhos da população 10, 100 e 500 (ver Seção 7.1.1) foi aplicada para o conjunto SHRD com restrição $d = 5$.

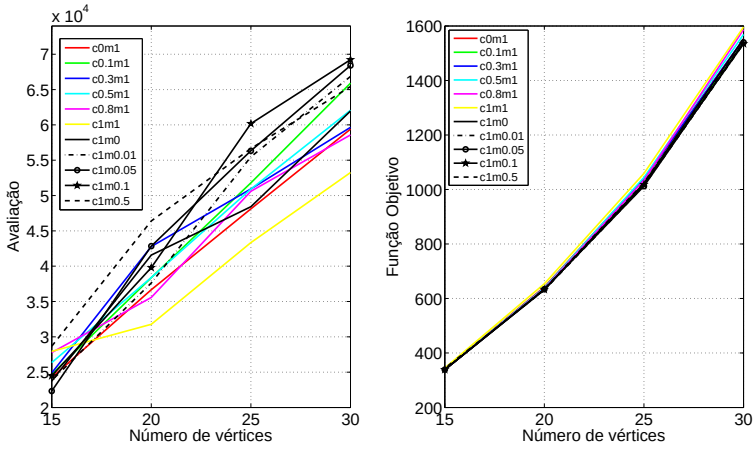
A Figura 7.7 apresenta os resultados do EAND com os diferentes tamanhos de população. Os gráficos dessa figura mostram que o EAND com uma população de 10 indivíduos possui convergência bem distinta para cada uma das configurações de taxa de recombinação e mutação. Os valores da função objetivo das diferentes configurações são similares. Conforme aumenta o tamanho da população, as diferentes configurações passam a apresentar comportamentos mais próximos em número de avaliações, porém, com



(a) População com 10 indivíduos.



(b) População com 100 indivíduos.



(c) População com 500 indivíduos.

Figura 7.7: EAND aplicado ao dc-MSTP com restrição 5 e os grafos SHRD.

valores maiores se comparados aos obtidos com 10 indivíduos na população.

Ao analisar os valores da função objetivo obtidos, conclui-se que o aumento no tamanho da população não resulta em melhora, assim, sugere-se utilizar o tamanho de 10 indivíduos para a população. Observa-se também que a configuração c1m0 possui a melhor convergência, porém, os valores da função objetivo são piores dos que os encontrados pelas demais configurações. Ponderando entre obter um equilíbrio entre o melhor desempenho na função objetivo e em convergência, a configuração c0m1 é mais indicada.

A configuração do EAND com taxa de recombinação 0, taxa de mutação 1 e população de 10 indivíduos foi aplicada para os grafos de SHRD com restrição $d = 3$ e $d = 4$, além da restrição $d = 5$ já apresentada. A aplicação de uma algoritmo de *branch-and-cut* (Nemhauser e Wolsey, 1988) encontrou a solução ótima para os grafos SHRD com as restrições utilizadas (Raidl e Julstrom, 2003; Krishnamoorthy et al., 2001).

Tabela 7.2: Valores de *gap* (média de 30 execuções) para os grafos SHRD utilizando EAND, GAES e GANK.

Instância				EANDD	GANK	GAES
Nome	n	d	C_{opt}	%-*gap*	%-*gap*	%-*gap*
SHRD150	15	3	582	1.71	0.03	0.02
		4	430	1.86	0.00	0.00
		5	339	1.76	0.00	0.00
SHRD200	20	3	1088	3.95	0.18	0.06
		4	802	2.61	0.17	0.04
		5	627	3.34	0.26	0.05
SHRD250	25	3	1745	3.72	0.44	0.01
		4	1276	4.23	0.42	0.05
		5	999	5.30	0.47	0.00
SHRD300	30	3	2592	3.31	0.74	0.08
		4	1905	2.62	1.02	0.05
		5	1504	4.92	1.19	0.11

A Tabela 7.2 apresenta os valores da métrica *gap*, em porcentagem, que é dada por $((C - C_{opt})/C_{opt}) * 100$, onde C_{opt} é o valor da solução ótima e C é o valor da função objetivo da média do melhor individuo encontrado na execuções do EA. Os grafos SHRD com as restrições $d = 3$, 4 e 5, utilizando o EAND, GAES (um EA com a representação Conjunto de Arestas) e GANK (um EA com a representação *NetKeys*), os resultados de GAES e GANK estão disponíveis em (Raidl e Julstrom, 2003). Observa-se da Tabela 7.2 que os valores de *gap* do EAND são maiores do que os de GAES e GANK, apesar de o

EAND encontrar a solução ótima em pelo menos uma das execuções para todos os casos.

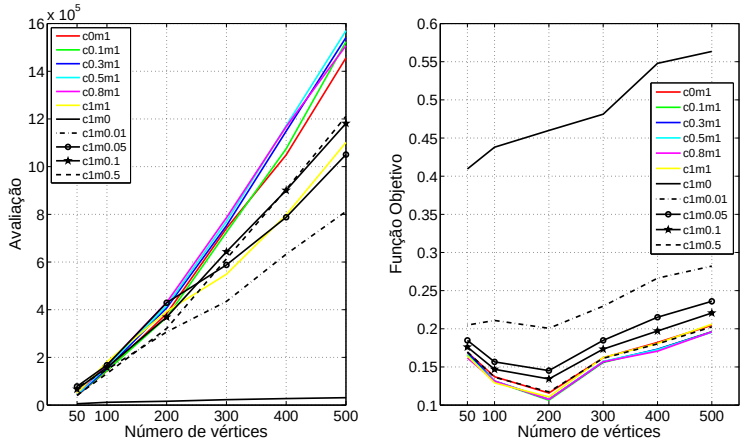
O segundo conjunto de grafos de teste, também é reportado na literatura como grafos de *benchmark* e são chamados de *m-Graphs* por possuirem armadilhas para algoritmos gulosos, como os baseados no algoritmo de Prim ou Kruskal. O conjunto de *m-Graphs* é composto por nove casos definidos por Knowles et al. (1999) com $n = 50,\quad 100,\quad 200$ (com três grafos para cada número de vértices) e três casos definidos por Raidl e Julstrom (2003) com $n = 300,\quad 400,\quad 500$, utilizando um processo de construção similar, totalizando 11 grafos completos. As armadilhas nesses grafos são obtidas por meio de um conjunto de vértices com alto número de arestas de baixo custo e com as demais arestas que incidem nesses vértices possuindo custo alto em relação às demais. Assim, ao buscarem atender a restrição de grau, algoritmos gulosos acabam não escolhendo certas arestas de baixo custo que formariam uma solução melhor. Uma descrição detalhada de como construir esses grafos é dada por Knowles et al. (1999). O conjunto de *m-Graphs* foi disponibilizado em (Raidl, 2008).

O EAND é aplicado para os *m-Graphs* com todas as configurações de parâmetros e tamanhos de população utilizados para os testes com restrição $d = 5$. Os gráficos de resultados contendo número de avaliações para encontrar o melhor indivíduo e o valor da função objetivo desses indivíduos são apresentados na Figura 7.8.

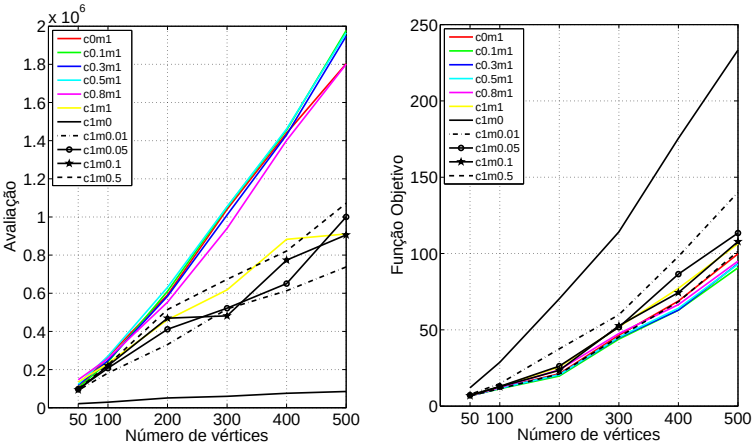
Primeiramente analisam-se os resultados obtidos pelo EAND comparando os diferentes tamanhos de população. Observa-se que o aumento do tamanho da população, além de aumentar o número de avaliações para obter o melhor indivíduo, também aumentam para algumas configurações o valor da função objetivo do melhor indivíduo. A única configuração que apresenta uma clara melhora no valor da função objetivo quando a população aumenta é a que utiliza somente recombinação (c1m0). Assim, os experimentos indicam que o tamanho de população adequada seria 10 indivíduos.

Selecionado o tamanho mais adequado para a população, analisam-se as configurações de taxa de aplicação de recombinação e mutação. Com exceção da configurações que priorizam a recombinação, as demais configurações obtém valores similares para a função objetivo, então o fator decisivo para selecionar a melhor configuração torna-se o número de avaliações para obter o melhor indivíduo. Segundo esse critério, a configuração c0.1m1 é uma escolha adequada, ponderando entre bons valores para a função objetivo e número de avaliações. Se somente o valor da função objetivo fosse considerado, a configuração c0.8m1, poderia ser utilizada.

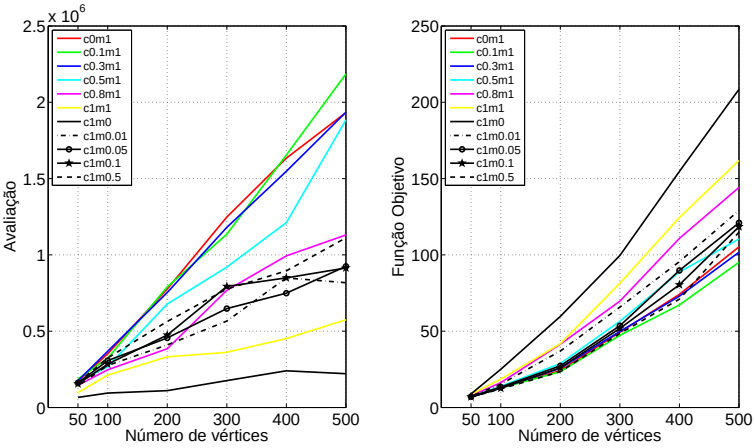
Assim como o conjunto de grafos SHRD, os *m-Graphs* possuem para as nove primeiras instâncias o valor da solução ótima encontrada por meio de um algoritmo *branch-and-cut*.



(a) População com 10 indivíduos.



(b) População com 100 indivíduos.



(c) População com 500 indivíduos.

Figura 7.8: EAND aplicado ao dc-MSTP com restrição 5 para os *m-Graphs*.

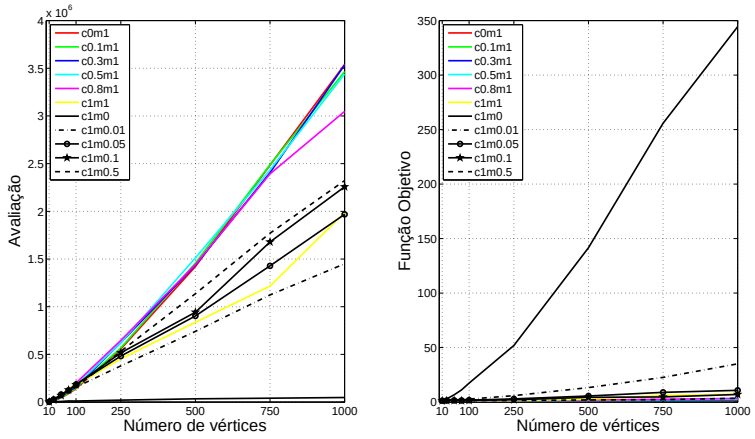
Tabela 7.3: Valores de *gap* (média de 30 execuções) para os *m-Graphs* utilizando EAND, GAES e GANK.

Instância			EAND	GANK	GAES
Nome	n	C_{opt}	%-gap	%-gap	%-gap
M1	50	6.60	2.72	7.85	1.19
M2	50	5.78	3.80	11.75	1.45
M3	50	5.50	2.72	5.4	0.92
M4	100	11.08	3.97	53.88	6.59
M5	100	11.33	2.38	58.62	10.78
M6	100	10.19	10.19	71.33	14.70
M7	200	18.33	16.54	146.60	17.64
M8	200	19.16	22.84	166.17	26.3
M9	200	16.13	7.19	153.89	9.78
M10	300	40.69	16.93	89.67	19.99
M11	400	54.69	15.43	137.93	37.84
M12	500	79.34	-14.18(*)	117.38	34.04

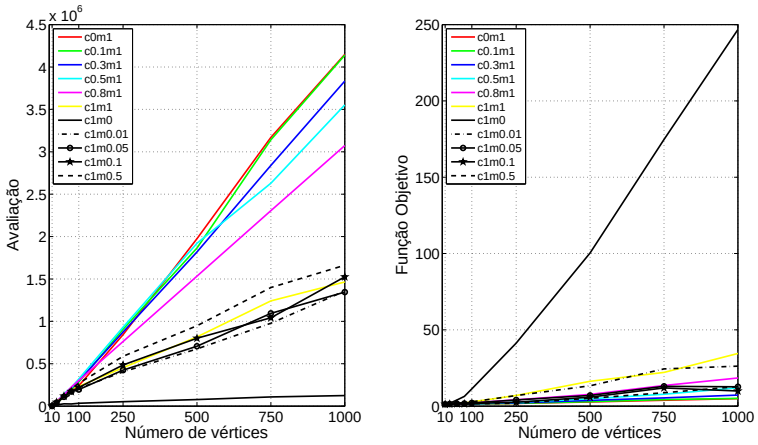
Para as outras três instâncias, é utilizado o melhor valor encontrado na literatura, reportado por Raidl e Julstrom (2003), como C_{opt}. Para comparação de resultados entre técnicas, utilizou-se configuração c0.8m1. A Tabela 7.3 apresenta os resultados do *gap* médio de 30 execuções do EAND, GAES e GANK. Destaca-se o *gap* negativo para o $m - Graph$ M12 obtido pelo EAND, indicando que foi encontrada uma solução melhor do que a da literatura para esse caso.

Observa-se na Tabela 7.3 que para todos os casos, o *gap* do EAND é menor que o do GANK. Em comparação com GAES, o EAND apresentou um resultado inferior apenas para os grafos com $n = 50$. O desempenho do EAND é consideravelmente melhor para os grafos com maior número de vértices, indicando que o EAND é mais adequado para problemas mais difíceis.

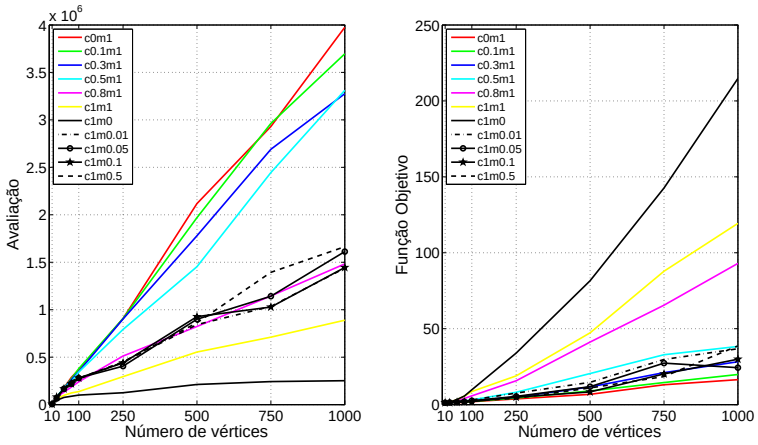
O terceiro conjunto de grafos utilizados para avaliar o desempenho do EAND consiste de grafos completos com pesos aleatórios entre $[0 \ldots 1]$ com número de vértices $n = 10$, 25, 50, 100, 250, 500, 750 e 1000. De acordo com Knowles et al. (1999), grafos com pesos aleatórios possuem em geral as árvores geradoras mínimas com grau máximo baixo, tornando o dc-MSTP para esses grafos de fácil solução. Assim, é importante observar que o intuito de utilizar esses grafos é verificar se o uso de recombinação pode aumentar a convergência em casos como grafos com maior número de vértices, visto que os casos anteriores possuem no máximo 500 vértices. Para esse conjunto de grafos também foi utilizada a restrição $d = 5$. Nos testes foram aplicadas todas as configurações e tamanhos



(a) População com 10 indivíduos.



(b) População com 100 indivíduos.



(c) População com 500 indivíduos.

Figura 7.9: EAND aplicado ao dc-MSTP com restrição $d = 5$ para grafos com pesos aleatórios.

de população proposto, os resultados podem ser vistos na Figura 7.9.

Primeiramente, comparam-se os resultados com relação ao tamanho da população. Da Figura 7.9, é nítido que o valor da função objetivo aumenta para a maioria das configurações quando o tamanho da população aumenta. Dessa forma, é mais adequado utilizar a população de tamanho 10, evitando uma variação grande nos valores da função objetivo obtido. Definido o tamanho da população, verifica-se então qual é a configuração mais adequada em termos de taxa de recombinação e mutação.

Na Figura 7.9(a) para o gráfico de número de avaliações, observa-se que existe uma separação entre as configurações com taxa de recombinação em 1.0 e as com taxa de mutação em 1.0. As primeiras possuem um número de avaliações menor, como era esperado com o uso de recombinação. Por outro lado, observando o gráfico dos valores da função objetivo nota-se que conforme aumenta a taxa de recombinação o valor da função objetivo do melhor indivíduo aumenta. Ressalta-se também que com uma taxa de recombinação mediana com taxa de mutação em 1.0, obtém-se um equilíbrio entre número de avaliações e valores da função objetivo. Assim, a configuração c0.8m1 é uma das mais adequadas para o EAND aplicado em grafos de pesos aleatórios. Essa configuração é utilizada na maioria dos EAs que utilizam a representação Conjunto de Arestas para PPRs (Raidl e Julstrom, 2003).

Na Seção 6.5, os testes empíricos mostraram que quando um problema possui $O(\sqrt{n})$ árvores, os operadores PAO e CAO possuem uma curva de tempo de execução que possui limitante superior dado por uma função $c\sqrt{n}$, onde c é uma constante positiva. No Apêndice B, é mostrado que, mesmo problemas com menos de $O(\sqrt{n})$ árvores, podem ser modelados de forma que o EAND trabalhe como se houvesse $O(\sqrt{n})$ árvores. Uma versão do EAND utilizando esse processo de modelagem foi implementado para resolver o dc-MSTP. Os resultados obtidos por esse EAND são comparados com testes efetuados, anteriormente, utilizando EAN (o EA utilizando a NDE), o GAES (um EA utilizando a representação Conjunto de Arestas) e GAPE (um AE utilizando a representação de Número de Prüfer (Knowles e Corne, 1999), ver Seção 4.1.2).

O conjunto de dados de teste utilizados consiste de 10 grafos completos, com pesos obtidos de forma aleatória entre $0 \ldots 1$, com $n = 15$, 20, 30, 50, 100, 200, 300, 400, 500, 1000. Os testes foram realizados com restrição de grau $d = 5$. Para cada caso de teste, são efetuadas 30 execuções com 10000 avaliações. O EAN, o GAES e o GAPE utilizam seleção por torneio para escolher os indivíduos para reprodução e selecionam para sobreviver os indivíduos de melhor função objetivo. O EAND utiliza torneio para selecionar os indivíduos para reprodução e os indivíduos que sobreviveram para a próxima

geração. O EAN e o EAND possuem população de tamanho 10, e o GAES e o GAPE, população de tamanho 100.

A Figura 7.10 apresenta um gráfico comparativo do tempo de execução de cada um dos algoritmos. Observa-se que o EAND possui o melhor desempenho de todos os AEs testados. Para os casos maiores de 500 e 1000 vértices, o EAN obteve um desempenho 10 vezes melhor do que GAES, que utiliza a representação Conjunto de Arestas que é uma das melhores representações, em termos de eficiência computacional, da literatura (Raidl e Julstrom, 2003).

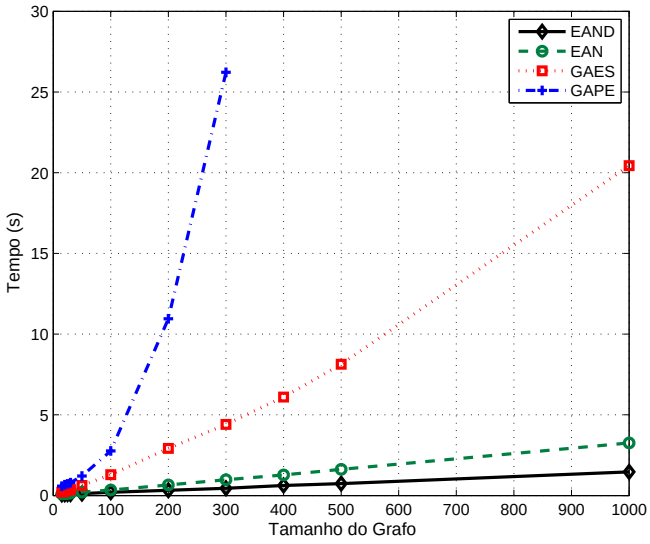
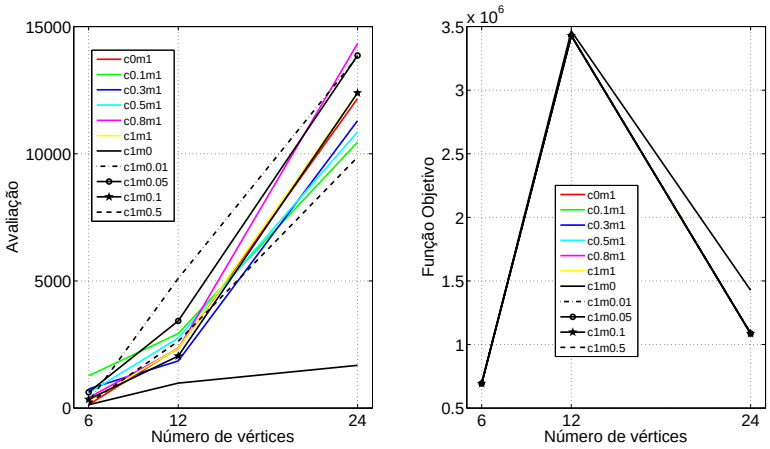


Figura 7.10: Tempo de execução de EAND, EAN, GAES e GAPE.

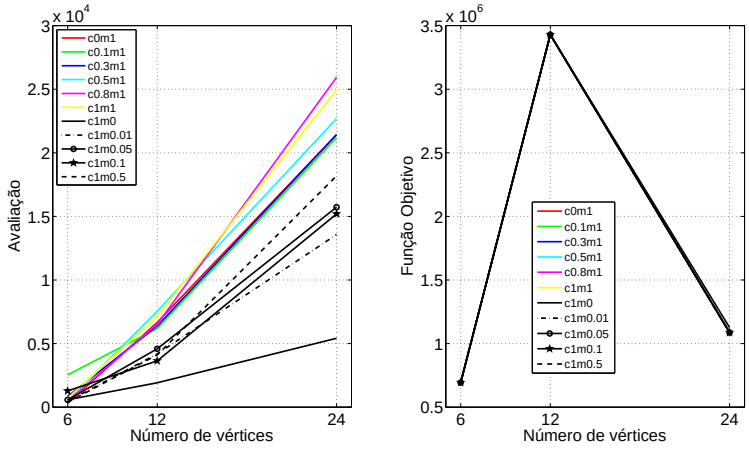
7.1.4 Árvore Geradora de Comunicação Ótima

O OCSTP consiste em encontrar a árvore geradora que atenda a demanda de comunicação com o menor custo (ver Seção 2.1.1). Esse, também, é um problema de minimização da função objetivo, que é dado pela soma dos produtos da demanda pelo custo do caminho entre todos os pares de demanda de comunicação.

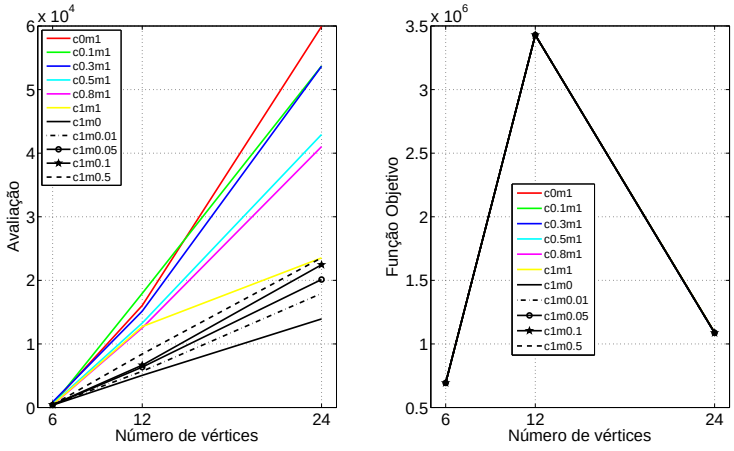
O conjunto de testes é composto por dados de três conjunto de problemas apresentados na literatura (Soak, 2006; Rothlauf, 2006), que em sua maioria possuem solução ótima conhecida. Os dados desses problemas foram disponibilizados em (Raidl, 2009). E como na na literatura são reportados testes com população de 20 indivíduos para o OCSTP (Rothlauf, 2006), optou-se por utilizar 20 indivíduos em vez de 10 indivíduos.



(a) População com 20 indivíduos.



(b) População com 100 indivíduos.



(c) População com 500 indivíduos.

Figura 7.11: EAND aplicado ao OCSTP de Palmer.

O primeiro conjunto de teste foi proposto por Palmer (1994) em sua tese e possui grafos com $n = 6, \quad 12, \quad 24$. Na tese de Palmer, há um caso de $n = 48$, que no entanto não foi disponibilizado. Esses problemas são chamados de Palmer.

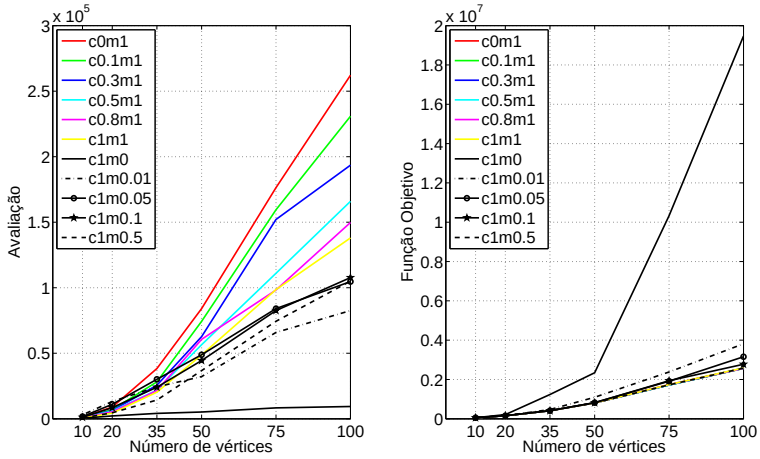
A seguir são apresentados os resultados do EAND para esse conjunto de problemas com os diferentes tamanhos de população (20, 100 e 500). Analisando esses resultados observa-se que a população com 500 indivíduos obtém os mesmos valores da função objetivo encontrados pelo EAND com população menor e, além disso, o número de avaliações é superior. Assim, claramente, o uso desse tamanho de população não é aconselhável.

Vale ressaltar da Figura 7.11 que, para a população com 20 indivíduos, somente a configuração c1m0 não alcançou o mesmo valor da função objetivo que as outras configurações para o caso de $n = 24$. Quando se utiliza a população com 100 indivíduos, todas as configurações obtêm valor da função objetivo similar para o melhor indivíduo encontrado. Porém, o número de avaliações para essa população é superior ao da população com 20 indivíduos. Assim, sugere-se o uso da população menor.

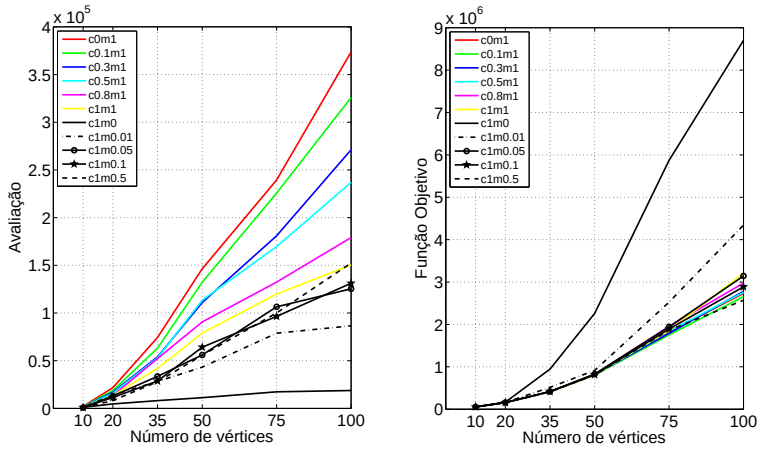
Com o tamanho da população definido, o próximo passo é analisar os valores da função objetivo e número de avaliações. Como a maioria das configurações obteve o valor da função objetivo próximo para o melhor indivíduo encontrado, a escolha do conjunto de parâmetros ideal deve ser definida pelo número de avaliações. A configuração que possui um desempenho mediano em termos do número de avaliações para todos diferentes números de vértices é c1m0.5, nota-se que essa configuração possui taxa de recombinação 1.0 indicando que, para o OCSTP, a recombinação produz uma melhora considerável no tempo de convergência do EAND. Essa configuração é utilizada para o cálculo de *gap* em relação a solução ótima. Os resultados do *gap* são apresentados na Tabela 7.4.

O segundo conjunto de problemas OCSTP foi desenvolvido por Raidl. As matrizes de demanda e de custo foram geradas de forma aleatória usando uma distribuição uniforme (Rothlauf, 2006). As instâncias possuem $n = 10$, 20, 35, 50, 75, e 100. Esses problemas são chamados simplesmente Raidl.

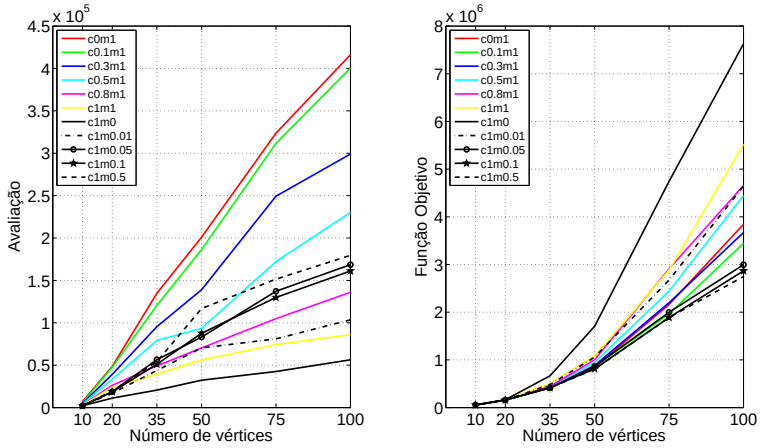
Analisando os gráficos da Figura 7.12 que apresenta os resultados do EAND para esse problema com os diferentes tamanhos de população, observa-se que as populações de 20 e de 500 indivíduos não obtiveram valores da função objetivo com aptidão relativamente alta para as instâncias com maior número de vértices. Para as instâncias menores, os valores da função objetivo são similares em todas as populações. Como o número de avaliações não aumenta excessivamente para a população com 100 indivíduos em relação a população com 20 e a população com 100 apresenta melhor valor para a função objetivo, sugere-se o uso de população com 100 indivíduos.



(a) População com 20 indivíduos.



(b) População com 100 indivíduos.



(c) População com 500 indivíduos.

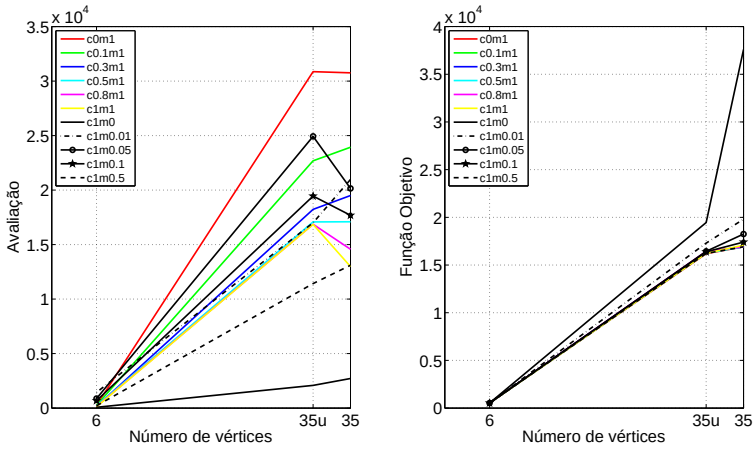
Figura 7.12: EAND aplicado ao OCSTP de Raidl.

Definido o tamanho da população, deve-se determinar a configuração que tenha melhor valor da função objetivo com um menor número de avaliações. É possível observar que conforme aumenta o tamanho do problema, o número de avaliações necessário para que cada configuração encontre a melhor solução é bem diverso, no entanto, os valores da função objetivo da melhor solução encontrada estão próximos, com exceção das configurações c1.0m1 e c1.0m0.01 que estão mais distantes das demais. Essas duas configurações são justamente as que convergem mais rapidamente. Assim, pode-se dizer que a melhor configuração, considerando os valores da função objetivo e convergência é c1m0.5, pois apresenta valor da função objetivo próximo do melhor encontrado em cada caso e o número de avaliações para obter o melhor indivíduo encontrado é menor do que o das outras configurações que também possuem um bom valor da função objetivo. O resultado dessa configuração é utilizado para calcular o *gap* da Tabela 7.4.

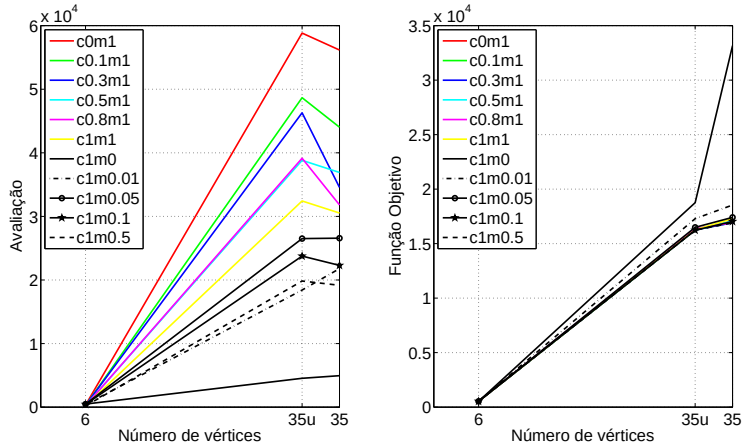
O terceiro conjunto de testes para o OCSTP foi proposto por Berry et. al (Soak, 2006; Rothlauf, 2006) e possuem $n = 6$ e 35. Para $n = 35$, existem duas matrizes de distância diferentes, uma composta por custos aleatórias com distribuição uniforme e a outra é uma matriz de custos unitária, ou seja, o custo entre todos os pontos é 1. Esse conjunto de problemas é chamado de Berry.

O EAND foi aplicado para esse conjunto de instâncias. Os resultados são apresentados na Figura 7.13. Observa-se que o aumento no tamanho da população não propiciou nenhuma melhora no valor da função objetivo encontrado para os melhores indivíduos, com exceção da configuração c1m0, que obtém um melhor valor da função objetivo com o aumento da população. Em relação aos valores encontrados pelas outras configurações esse valor está muito distante. Assim, sugere-se a utilização do EAND com uma população de 20 indivíduos.

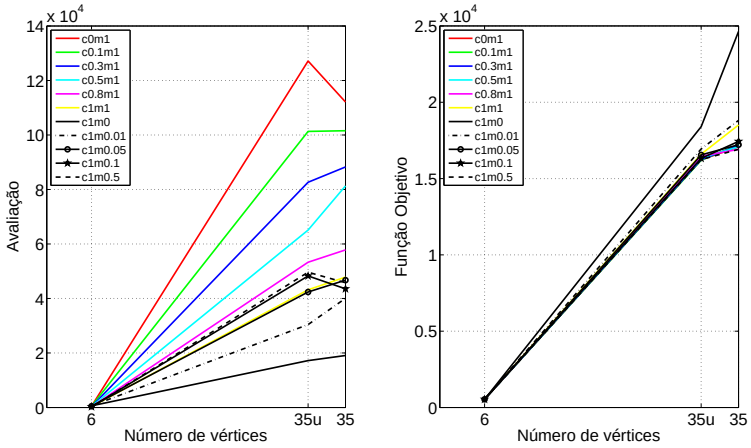
Nos gráficos da Figura 7.13, o valor 35u do eixo de número de vértices indica o caso em que a matriz de custos é unitária. É possível observar que as diferentes configurações possuem o número de avaliações bem distinto para encontrar a solução ótima e que, no entanto, os valores da função objetivo do melhor indivíduo encontrado por cada uma delas é similar. Outro fato interessante, é que o número de avaliações para encontrar a solução no caso da matriz de custo unitária na maioria das configurações foi maior ou igual ao do caso com matriz de custos aleatória. Para esse problema, repete-se o perfil de que a configuração c1m0 converge rapidamente para uma solução subótima. Observando o valor da função objetivo do melhor indivíduo e o menor número de avaliações, sugere-se o uso da configuração c1m0.5, pois essa possui tempo de convergência mediano e valor da função objetivo próximo do mínimo. Essa configuração é utilizada para o cálculo do *gap*.



(a) População com 20 indivíduos.



(b) População com 100 indivíduos.



(c) População com 500 indivíduos.

Figura 7.13: EAND aplicado ao OCSTP de Berry.

Conclui-se dos testes para o OCSTP que o uso do operador de recombinação NOX melhora o tempo de convergência do EAND e mantém a qualidade da solução encontrada pelo melhor indivíduo, visto que a configuração que obtém o melhor desempenho considerando convergência e valor da função objetivo é a c1m0.5. Uma justificativa para o bom desempenho de NOX nesse problema seria a sua tendência para árvores estrela. Deve-se observar que, segundo Rothlauf et al. (2003) as soluções ótimas do OCSTP para os problemas analisados possui uma tendência para a árvore geradora mínima e não para árvores estrela.

A Tabela 7.4 apresenta os valores ótimos encontrados na literatura para cada um dos problemas analisados, bem como o *gap* (como calculado para o dc-MSTP, ver Seção 7.1.3) entre as soluções ótimas e a média dos melhores indivíduos obtidos em cada execução de cada algoritmo (EAND, GANK e GAES). Para o EAND utilizou-se a configuração c1m0.5 e os tamanhos de população indicados para cada tipo de problema (20 para Palmer e Berry; 100 para Raidl). Os valores de *gap* para GAES e GANK estão disponíveis em Soak (2006). Não é possível comparar os algoritmos em número de avaliações, pois GAES e GANK nos testes reportados na literatura não são EAs *steady-state*.

Tabela 7.4: Valores médios de *gap* em 30 execuções para OCSTP utilizando EAND, GAES e GANK.

Instância			EAND	GANK	GAES
Nome	n	C_{opt}	%-gap	%-gap	%-gap
Palmer	6	693180	0.00	0.00	0.00
	12	3428509	0.37	0.58	0.00
	24	1086656	0.15	0.04	0.04
Raidl	10	53674	0.00	0.00	0.00
	20	157570	4.52	1.57	4.45
	35	412167	3.13	-	-
	50	806864	5.41	44.11	29.22
	75	1862988	7.79	-	-
	100	2570451	10.23	-	-
Berry	6	534	0.00	0.00	0.00
	35u	16190	3.48	2.52	1.58
	35	16915	6.76	2.95	0.00

Para os casos de Raidl com $n = 35, \quad 50, \quad 100$, não existe na literatura um valor

ótimo reportado, assim utilizou-se para o cálculo do *gap* o menor valor encontrado pelo EAND com a configuração selecionada. Em todos os problemas analisados, o EAND consegue encontrar a solução ótima em pelo menos uma das execuções. No entanto, na média, o valor da função objetivo do melhor indivíduo possui o *gap* informado na Tabela 7.4. Em geral, o EAND com a configuração c1m0.5 não obtém um *gap* médio menor do que GAES e GANK nos casos testados. Para a maior instância (Raidl com 50 vértices), o EAND apresenta *gap* médio significativamente menor. Esse resultado, reforça as conclusões anteriores (ver Seções 7.1.2 e 7.1.3) de que quanto maior o tamanho do problema, ou seja, para problemas mais difíceis, o EAND tem um desempenho superior ao de outros EAs aplicados aos PPRs. Os *gaps* médios do EAND para Raidl com $n = 75$ e 100 são menos do que o *gap* de GAES para Raidl com $n = 50$, reforçando a indicação de que o EAND possui um diferencial maior para grafos grandes.

7.2 Problema de Floresta com Mais de uma Árvore

Nesta Seção são realizados testes utilizando um EA baseado na NDE com o operador 1 e o operador de recombinação EHR para o PPR de reconfiguração de sistemas de energia elétrica (ver Seção 2.2). Esse algoritmo baseia-se no EA desenvolvido por dos Santos et al. (2008) que usa a NDE com os operadores 1 e 2.

O EA implementado por dos Santos et al. (2008), utiliza 5 subpopulações referentes aos diferentes objetivos do problema. A estratégia de subpopulações (também chamada de tabelas (Benayoun et al., 1971)) possibilita lidar com vários objetivos de forma computacionalmente eficiente e simples. A primeira subpopulação refere-se aos melhores indivíduos obtidos em termos da função objetivo, que é uma ponderação entre os múltiplos objetivos o número de chaves (num_{ma}) que precisam ser manobradas para obter a configuração do sistema. As outras subpopulações armazenam os melhores indivíduos nos seguintes objetivos: perdas (P_e), queda de tensão (Q), carregamento da rede (C_R) e carregamento das subestações (C_S). O EA resultante da inclusão do operador EHR, também é *steady-state* como o EAND (ver Seção 7.1) e é chamado de EAN.

Os sistemas de energia elétrica utilizados foram testados com as variações dos parâmetros do EAN descritas a seguir. Os parâmetros de teste modificam o tamanho de cada subpopulação, a taxa de aplicação do EHR e o número máximo de modificações que podem ser aplicadas a cada chamada do EHR. Os tamanhos de população testados são 5, 10 e 25 indivíduos em cada subpopulação. O número máximo de modificações (aplicações do operador 1 por aplicação do EHR) é 4, 6 e 10. Para a taxa de aplicação

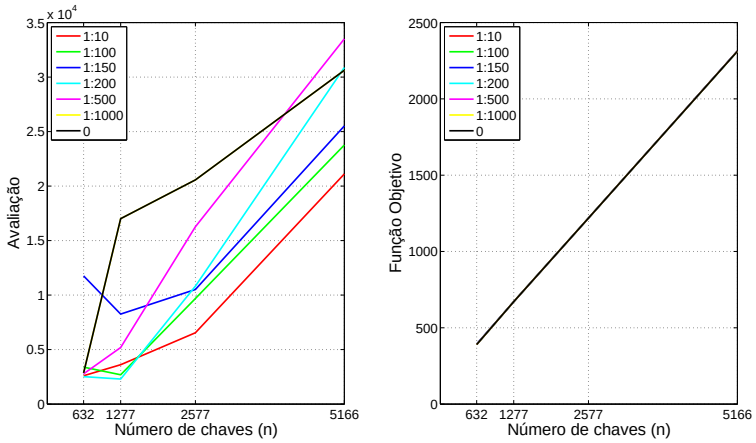
de EHR, têm-se os seguintes valores (que estão presentes nas legendas dos gráficos de resultado desta Seção):

- 0, EHR não será aplicado;
- 1:10, EHR é aplicado para gerar 1 em cada 10 indivíduos gerados no EA;
- 1:100, EHR é aplicado para gerar 1 em cada 100 indivíduos gerados no EA;
- 1:150, EHR é aplicado para gerar 1 em cada 150 indivíduos gerados no EA;
- 1:200, EHR é aplicado para gerar 1 em cada 200 indivíduos gerados no EA;
- 1:500, EHR é aplicado para gerar 1 em cada 500 indivíduos gerados no EA;
- 1:1000, EHR é aplicado para gerar 1 em cada 1000 indivíduos gerados no EA.

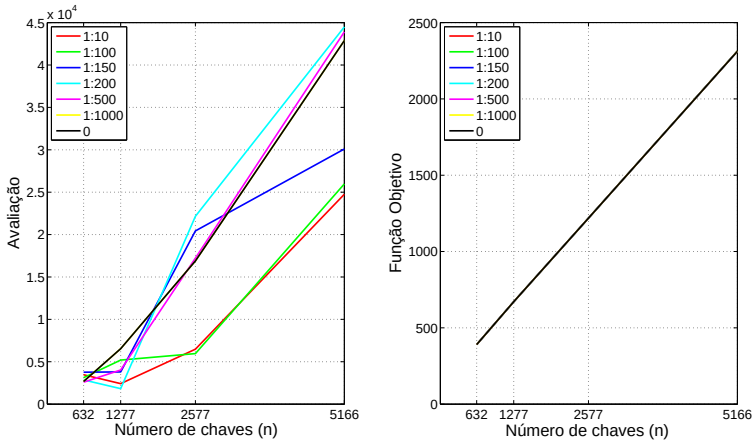
Os casos de teste são compostos por quatro sistemas de distribuição de energia elétrica (ver (dos Santos, 2009)):

- Sistema 1: é uma rede real da cidade de São Carlos em 1994. Esse sistema possui 3860 barras, 533 setores, 632 chaves, sendo 509 chaves NF e 123 chaves NA, 3 subestações e 23 alimentadores (dos Santos et al., 2008). A função objetivo utilizada é $10^{-3} * P_e + a * C_R + b * C_S + 4 * num_{ma}$, com $a = 100$ se $C_R > 0.5$, caso contrário $a = 0$ e $b = 100$ se $C_S > 0.5$; caso contrário $b = 0$;
- Sistema 2: consiste do Sistema 1 duplicado com a adição de 13 chaves NA que conectam o sistema original com a parte duplicada totalizando 1277 chaves. A função objetivo utilizada é $0.5 * 10^{-3} * P_e + 2 * num_{ma}$;
- Sistema 3: consiste do Sistema 2 duplicado com a adição de 23 chaves NA, totalizando 2577 chaves. A função objetivo utilizada é $0.25 * 10^{-3} * P_e + a * C_R + num_{ma}$, com $a = 100$ se $C_R > 0.5$; caso contrário $a = 0$;
- Sistema 4: consiste do Sistema 3 duplicado e 12 chaves NA adicionadas totalizando 5166 chaves. A função objetivo utilizada é $0.25 * 10^{-3} * P_e + a * C_R + num_{man}$, com $a = 100$ se $C_R > 0.5$; caso contrário $a = 0$.

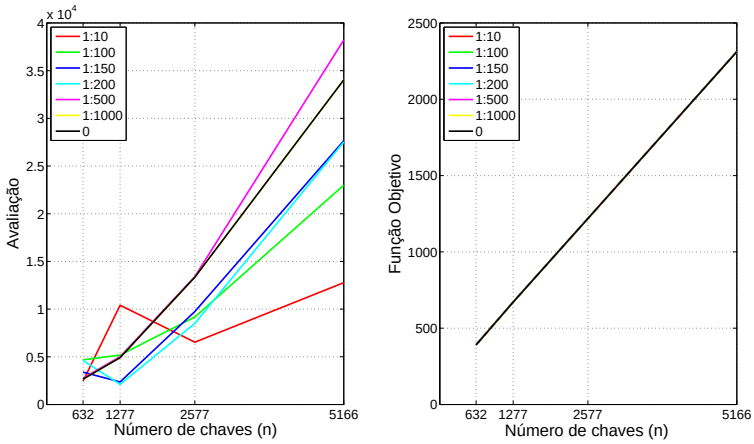
O tamanho do problema é determinado pelo número de chaves que existem no sistema. Cada sistema possui uma função objetivo diferente buscando tornar o problema o mais difícil possível, com o propósito de avaliar melhor a contribuição do EHR nos casos mais



(a) EHR com máximo de 4 modificações.



(b) EHR com máximo de 6 modificações.



(c) EHR com máximo de 10 modificações.

Figura 7.14: EAN com subpopulações de 5 indivíduos.

críticos: redes maiores e com funções de avaliação mais complicadas. Por exemplo, o aumento do peso para num_{ma} aumenta significativamente as descontinuidades na função, uma vez que o número de manobras é uma variável inteira.

Cada um dos casos de teste é aplicado para todas as combinações de parâmetros do EAN, cada instância foi executada 5 vezes, com o número máximo de novos indivíduos gerados de 120000. Em cada figura de resultados, o gráfico à esquerda apresenta a avaliação em que foi encontrado o melhor indivíduo e o gráfico à direita o valor obtido para a função objetivo. A abscissa de ambos os gráficos é o tamanho do sistema. A Figura 7.14 apresenta os gráficos para EAN com cada subpopulação contendo 5 indivíduos e o número máximo de modificações no EHR de 4, 6 e 10. Observa-se que, com os diferentes número de modificações todas as taxas de aplicação do EHR encontraram a mesma solução ótima. Assim, a melhor configuração de parâmetros foi determinada com base na convergência do EAN, ou seja, pela configuração que requer o menor número de avaliações. Analisando os gráficos de convergência da Figura 7.14 verifica-se que, em geral, a taxa de aplicação do EHR mais adequada é 1:10 pois, a solução ótima é encontrada com o menor número de avaliações.

Um fator importante para garantir a complexidade de tempo do EHR é o número máximo de modificações que são aplicadas a cada chamada desse operador. Comparando as Figuras 7.14(a), 7.14(b) e 7.14(c), têm-se que para o Sistema 4, que é o maior, quando são aplicadas 4 modificações, o melhor indivíduo é encontrado pouco depois da geração 20000, com 6 modificações, por volta da geração 25000 e com 10 modificações em torno da geração 12500. Assim, considerando somente o número de indivíduos para convergir, a escolha adequada seria o EHR com até 10 modificações.

No entanto, 10 modificações são mais do que o dobro de 4, como o tempo para aplicar cada modificação é constante, pois cada uma corresponde a uma aplicação do operador 1, dobrar o número de modificações consiste diretamente em dobrar o tempo utilizado pelo EHR. Assim, pode-se concluir que o EHR com 4 modificações apesar de necessitar de mais avaliações, possibilita EAN convergir mais rapidamente.

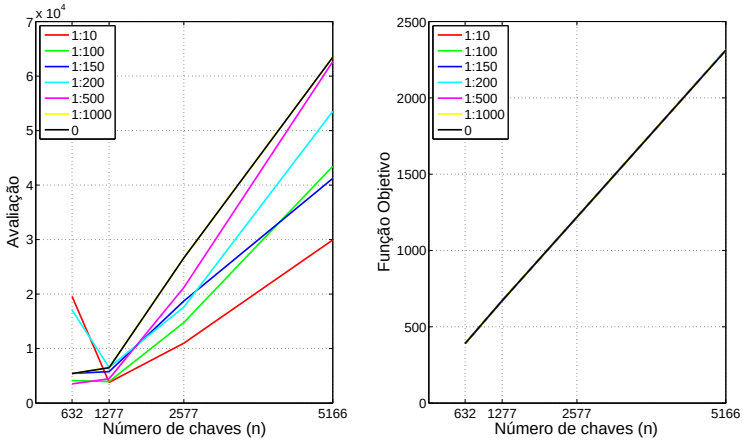


Figura 7.15: EAN com subpopulações de tamanho 10 e EHR com máximo de 4 modificações.

As Figuras 7.15 e 7.16 apresentam os resultados obtidos quando o tamanho de cada população aumenta para 10 e 25 indivíduos, respectivamente, com um máximo de 4 modificações a cada aplicação do EHR. Esses tamanhos de população também foram testados com as outras variações no número máximo de modificações de 6 e 10. No entanto, o comportamento do algoritmo é similar aos casos com subpopulações de 5 indivíduos. Assim, são apresentados somente os resultados com 4 modificações.

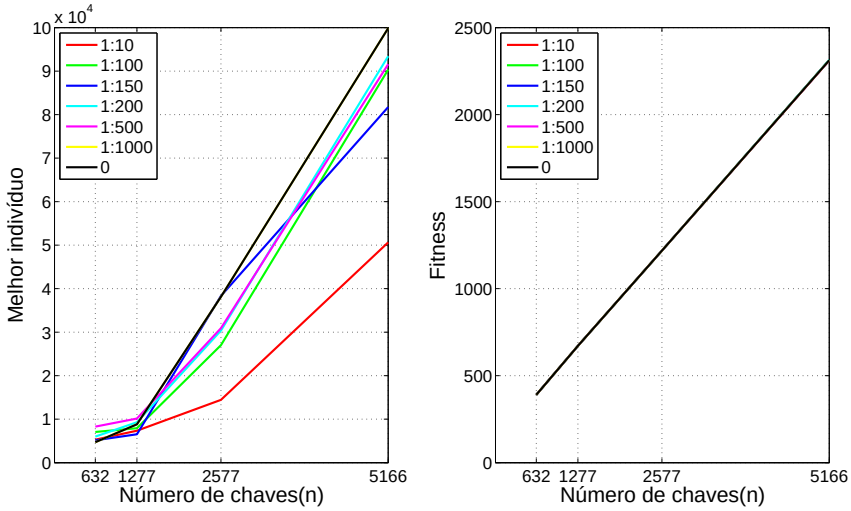


Figura 7.16: EAN com subpopulações de tamanho 25 e EHR com máximo de 4 modificações.

Por outro lado, com a subpopulações de 25 indivíduos, o comportamento do EAN foi similar ao de subpopulações com 5 indivíduos e 4 modificações. Porém, ocorre um aumento no número de avaliações necessárias para encontrar a solução ótima. Esse fato é esperado, pois como a mutação é a operação mais freqüentemente utilizada em EAN, é necessário um número maior de avaliações para atingir o mesmo resultado do que nos casos que utilizam uma subpopulação de tamanho menor. Sub-populações de tamanho menores, possibilitam que a mutação concentre mais a sua atuação sobre certos tipos de indivíduos.

A Figura 7.17 apresenta a melhor população final obtida após 30 execuções do EAN com subpopulações de 5 indivíduos e EHR com um máximo de 4 modificações e taxa de aplicação 1:10. Os objetivos considerados são perdas e número de manobras. A rede considerada corresponde ao Sistema 1. A curva tracejada representa a fronteira de Pareto dessas soluções. Essa fronteira possui 6 indivíduos. Um dos indivíduos é o melhor da população de ponderação dos objetivos. Outros três indivíduos são os melhores em termos de perdas com diferentes números de manobras. Outros dois indivíduos possuem valores de perdas e manobras entre os valores dos outros indivíduos da fronteira. No entanto, não é possível afirmar que eles são dominados pelos demais indivíduos da fronteira. Os demais indivíduos são dominados por esses em pelo menos um dos objetivos analisados.

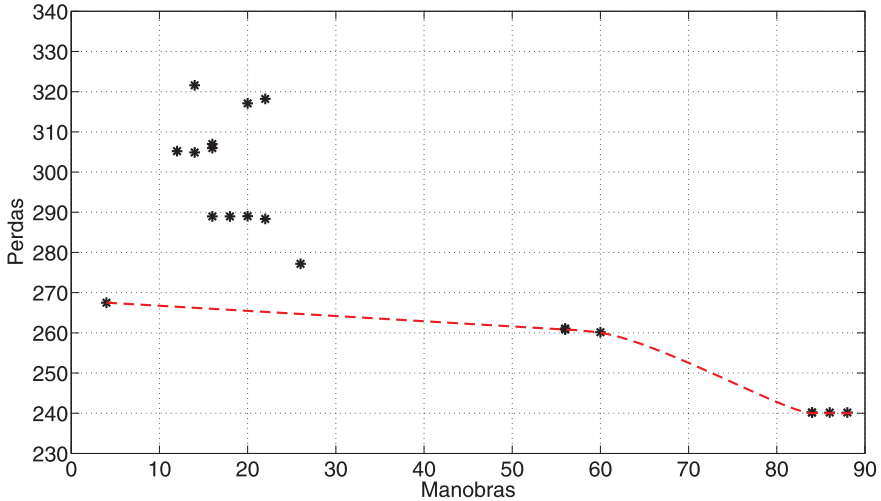


Figura 7.17: Resultados de perdas por número de manobras. Os valores são a melhor população final em 30 execuções do EAN com um máximo de 4 modificações na taxa de 1:10.

Na Figura 7.18 são apresentados o número de avaliações para obter o melhor indivíduo variando o número de manobras, para o EAN em duas configurações diferentes: 1) EAN

com EHR, máximo de 4 modificações e com taxa 1:10 e 2) EAN sem EHR, somente com o operador 1. Em ambos os casos usam-se subpopulações de 5 indivíduos. Utilizou-se o Sistema 1 com a função objetivo $10^{-3} * P_e + a * C_R + b * C_S + \alpha * num_{ma}$, com $a = 100$ se $C_R > 0.5$, caso contrário $a = 0$ e $b = 100$ se $C_S > 0.5$; caso contrário $b = 0$; com α variando entre os valores 1, 2, 3, 4 e 5. Conforme α aumenta a função objetivo torna-se mais descontínua, aumentando a dificuldade do problema.

Por outro lado, conforme num_{ma} reduz, também reduz-se o número de configurações capazes de reduzir o valor da função objetivo, uma vez que para reduzir as perdas significativamente são necessários num_{ma} relativamente altos. Assim, o espaço de soluções úteis na prática diminui rapidamente com o aumento do peso para num_{ma}. Por exemplo, para peso igual a 5, nenhuma nova configuração produz melhoria nos valores da função objetivo.

A Figura 7.18 mostra que o EAN com EHR explora mais eficientemente o espaço de busca nesse caso.

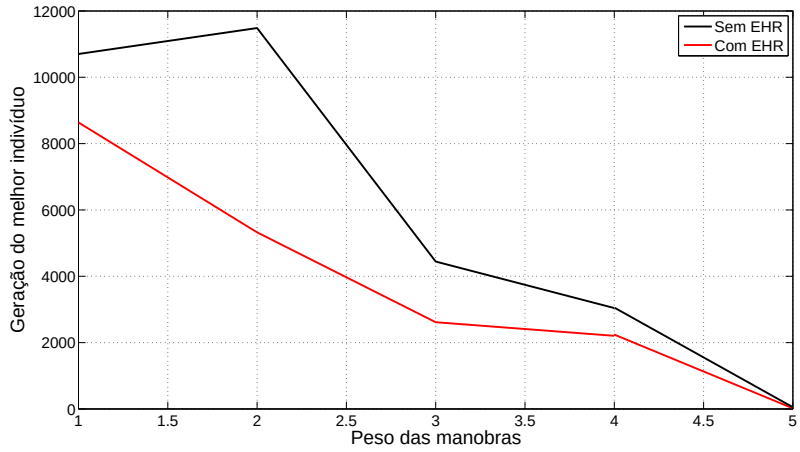


Figura 7.18: Avaliações para encontrar o melhor indivíduo versus peso para num_{ma}.

7.3 Considerações Finais

Este Capítulo apresentou os resultados da proposta dessa pesquisa aplicada a PPRs de floresta com uma árvore e PPRs de floresta com mais de uma árvore.

Os PPRs de floresta com uma única árvore considerados foram OMTP, dc-MSTP e OCSTP. Para esses problemas é utilizado um EA com a NDDE e os operadores de mutação PAO e CAO, bem como o operador de recombinação NOX. Os resultados mostram que

conforme aumenta o tamanho da população, ou cresce a dimensão do problema a utilização de NOX auxilia o EA a encontrar boas soluções com menor número de avaliações, principalmente para o OCSTP. É importante ressaltar também que o EAND, em geral, obtém melhores resultados para instâncias maiores dos problemas do que outras representações utilizadas em EAs para PPRs. Verificou-se também que, para os PPRs considerados, a operação de mutação é fundamental para que EAND obtenha resultados satisfatórios.

Para o PPR de floresta com mais de uma árvore é utilizado como base um EA com a NDE (EAN) ao qual é acrescentada uma implementação do EHR. Esse EA foi aplicado ao PPR de reconfiguração de sistemas de distribuição de energia elétrica. Dos resultados obtidos para esse problema, conclui-se que o uso de EHR melhora a convergência do EAN. Além disso, ao variar o peso de num_{ma} na função objetivo construindo um problema mais difícil, o número de avaliações necessárias para encontrar o ótimo diminui progressivamente com o uso de EHR.

Os testes mostraram que para esse problema e, possivelmente para outros de mesma complexidade, o número de modificações utilizadas por aplicação do EHR deve ser 4 e que, quanto maior o tamanho do problema, maior é a contribuição do EHR para encontrar a melhor solução com menos avaliações.

O Capítulo 8 apresenta as conclusões finais dessa tese.

Capítulo

8

Conclusões e Trabalhos Futuros

Este trabalho apresentou o problema de projeto de redes (PPRs), principal foco de aplicação dessa pesquisa, e algumas de suas variações. Para o entendimento da proposta desenvolvida nessa pesquisa foram introduzidos os principais conceitos sobre algoritmos evolutivos (EAs) e representações para EAs aplicados a PPRs.

Com destaque, a representação nó-profundidade (NDE), recentemente proposta na literatura de representações de algoritmos evolutivos, foi apresentada. Em seguida, descreveu-se a representação nó-profundidade-grau (NDDE), que consiste no aperfeiçoamento da NDE. Esses aperfeiçoamentos são compostos por melhorias nos operadores PAO e CAO da NDE e no desenvolvimento de operadores de recombinação, os quais são importantes para melhorar a convergência dos EAs. Foram desenvolvidas três propostas de operadores de recombinação. Dois operadores são baseados em permutação (NOX e NPBX), e um operador (EHR) que fundamenta-se no histórico de aplicações do operador PAO. Essa última proposta tem como motivação desenvolver um operador de recombinação que possa ser aplicado tanto para PPRs representados por grafos completos quando por grafos não-completos, visto que as duas primeiras propostas são específicas para PPRs representados por grafos completos.

Posteriormente à proposta dos operadores de recombinação, foi realizada uma análise das propriedades desejáveis de representação e seus operadores. Os resultados da investigação das propriedades mostraram que tanto a NDE quanto a NDDE possuem redundância. Com o objetivo de corrigir esse problema, foram propostas duas modificações na codificação da NDDE. A primeira modificação fixa o vértice raiz da árvore e a segunda

define uma ordem (crescente) para os vértices irmãos na NDDE. Para garantir essa ordem nos operadores da NDDE, PAO e CAO, é suficiente selecionar adequadamente a posição em que a subárvore podada será inserida na NDDE de $T_{destino}$. Para os operadores NOX e NPBX é proposto um algoritmo que verifica e corrige as soluções redundantes.

Os operadores PAO, CAO e EHR não possuem tendência para tipos específicos de árvores (árvore geradora mínima, árvore estrela). Por outro lado, o NOX e o NPBX apresentaram uma tendência em aproximar as soluções a árvores estrela, felizmente conforme aumenta o tamanho da rede esse efeito tende a diminuir. O uso de PAO e CAO combinado com NOX e NPBX auxilia a diminuir a tendência deles.

Com relação a hereditariedade, o NOX em geral produz novos indivíduos que possuem em torno de 20% de arestas que não estão nos pais, independente do tamanho do problema. Por outro lado, no NPBX, a hereditariedade piora conforme aumenta o tamanho do problema. Com relação ao EHR, estimativas mostram que esse operador possui alta hereditariedade, quase perfeita. Ao analisar os resultados de complexidade de tempo, o operador NPBX mostrou-se mais lento do que o NOX. Dadas as desvantagens apresentadas pelo NPBX, optou-se por utilizar somente NOX e EHR nos testes da NDDE aplicada a PPRs.

Um EA utilizando a NDDE com os operadores de mutação PAO e CAO e o operador de recombinação NOX foi aplicado aos seguintes PPRs com floresta composta de uma única árvore: uma árvore máxima (OMTP), árvore geradora mínima com restrição de grau (dc-MSTP) e árvore de comunicação ótima (OCSTP). Esses problemas foram utilizados devido a sua importância na literatura para avaliar o desempenho de representações. Além disso, esses problemas são considerados difíceis e similares a problemas do mundo real, principalmente o de árvore de comunicação ótima.

No OMTP, EAND consegue encontrar a solução para os grafos com maior número vértices em um número de gerações significativamente menor, principalmente para T_{opt} com topologia estrela. Quando aplicado ao dc-MSTP, os resultados obtidos pelo EAND também foram melhores em valores da função objetivo do que os resultados apresentados na literatura para as instâncias com maior número de vértices. Novamente, para o OCSTP os melhores resultados de EAND em valores da função objetivo são obtidos para as instâncias com maior número de vértices. Os resultados obtidos para esse conjunto de problemas mostram que o uso de operadores de recombinação em conjunto com o operadores de mutação, melhora a convergência do EA, principalmente em situações em que a dimensão do problema aumenta consideravelmente. Em síntese, EAND apresentou resultados superiores em problemas com maior dimensão do que outras representações

utilizadas em EAs para PPRs.

Adicionalmente, foi desenvolvido um EA com o operador de recombinação EHR com o objetivo de aplicação em PPR com floresta contendo mais de uma árvore. Como estudo de caso, o EA com o operador EHR foi aplicado ao problema de reconfiguração de sistemas de energia elétrica. Os casos de teste foram construídos com base em um sistema de energia elétrica real da cidade de São Carlos de 1994, além de outros 3 sistemas baseados no de São Carlos, mas com dimensões dobrada, quadruplicada e octuplicada. Esse sistema possui, naturalmente, da ordem de $\sqrt{n}$ árvores (alimentadores). Com isso, pode-se verificar que um EA usando a NDE (EAN) pode produzir novas soluções em tempo médio $O(\sqrt{n})$ mesmo se o sistema corresponde a um grafo não-completo. No caso, o sistema utilizado é um grafo esparso. Além disso, foi comparado o desempenho do EAN sem e com o EHR para os quatro sistemas. O uso do EHR produziu reduções significativas do número de iterações para convergência do EA, diminuindo significativamente esse número. O diferencial do EHR foi ainda salientado tornando a função objetivo mais complicada pelo aumento das descontinuidades variando o peso do objetivo referente ao número de manobras. Nesses casos, a contribuição do EHR é ainda maior.

Conclui-se desses resultados que o uso da recombinação na NDDE pode auxiliar o desempenhos de EAs para PPRs complexos, diminuindo o tempo de convergência necessário para encontrar a solução para o problema. Para problemas mais simples, o uso dos operadores de mutação, sem recombinação, mostrou-se suficiente para resolver os problemas.

As principais contribuições deste trabalho podem ser sintetizadas como segue:

1. Desenvolvimento da NDDE a partir da NDE, de modo que os operadores de mutação PAO e CAO demandem um tempo de execução médio $O(\sqrt{n})$, enquanto os operadores para PPRs mais eficientes requerem tempo $O(n)$;

2. Análise das propriedades dos operadores PAO e CAO;

3. Correção do problema de redundância nos operadores PAO e CAO;

4. Desenvolvimento de dois operadores de recombinação baseados em permutação, NOX e NPBX para florestas com uma única árvore em grafos completos;

5. Desenvolvimento do operador de recombinação baseado em mutação, o EHR para florestas com uma ou mais árvores em grafos completos e não-completos;

6. Obtenção de melhores soluções para PPRs clássicos para as instâncias de maior dimensão;

7. Proposição de uma técnica capaz de obter em tempo de computação aceitável soluções para PPRs clássicos com instâncias superiores a 100 vértices atingindo casos com até 5000 vértices (milhões de arestas);

8. Obtenção mais rapidamente (menos iterações) de soluções para reconfiguração de sistemas de energia elétrica de larga-escala. Essa redução de tempo é fundamental na reconfiguração em tempo real que é necessária em caso de apagões por exemplo.

8.1 Trabalhos Futuros

Como continuidade da pesquisa desenvolvida, propõem-se os seguintes trabalhos futuros:

1. Analisar outras estratégias para desenvolvimento de operadores de recombinação específicos para grafos completos que possam garantir uma maior hereditariedade. Idealmente deseja-se que o número de arestas dos filhos que não estão presentes nos pais seja bem pequeno. Uma sugestão é empregar os outros operadores de recombinação que existem na literatura para permutações, tais como crossover de ciclo e crossover baseado em ordem. Acredita-se que dada a característica desses operadores de preservar mais a ordem dos genes de um dos pais no filho obtenha melhor hereditariedade;

2. Desenvolver operadores de recombinação topológicos, ou seja, que recombinem somente os valores de profundidade da NDDE. Existem relatos na literatura de PPRs em que a topologia da rede é mais importante do que a posição do vértice. Como exemplo, é possível citar o caso de teste de Berry (ver Capítulo 7) com 35 vértices e matriz de custo unitária, onde a topologia é mais importante para definir o custo total da rede do que a posição dos vértices. Isso deve-se ao fato de que a distância de ligação entre todos os vértices é unitária, como resultado têm-se que a demanda é definida para todos os vértices. Além disso, tendo uma topologia adequada poderia ser aplicado um método de busca local para a disposição dos vértices na rede;

3. Aplicar a NDDE em algoritmos de estimação de distribuição (EDAs, ver Capítulo 3). Uma análise preliminar mostra que o histórico de aplicações dos operadores PAO e CAO (histórico que é a base do EHR) para os melhores indivíduos da população é aproximadamente um modelo probabilístico dos melhores genes e *building blocks*, a princípio, dos indivíduos da população atual. Esse modelo poderia, então, ser utilizado para a obtenção de novos indivíduos;

4. Desenvolvimento de métodos de busca local utilizando os operadores PAO e CAO;

5. Implementar o EHR utilizando aplicações do operador CAO;

6. Ampliar a investigação de ajustes de parâmetros. Variar das taxas de aplicação de mutação e recombinação durante o processo evolutivo bem como a taxa de aplicação entre os operadores PAO e CAO. Atualmente, essas taxas são fixadas no inicio do processo e mantém-se constantes. Usar número de modificações variável a cada aplicação de EHR, com um valor máximo estabelecido. Dependendo da distância (número de modificações entre o ancestral comum e o pai) entre os pais e o ancestral comum, em alguns casos pode ser relevante aplicar um conjunto maior de modificações;

7. Aplicar o EHR em problemas clássicos de PPRs e em outros problemas como reconstrução de filogenias e roteamento de múltiplos veículos;

8. Aplicar a NDDE em outros PPRs clássicos, tais como árvore geradora mínima com restrição de diâmetro e árvore geradora mínima probabilística;

9. Utilizar a NDDE e seus operadores com outros algoritmos de otimização, por exemplo, o GRASP (*Greedy Randomized Adaptive Search Procedure*).

Referências Bibliográficas

Abdalla, A.; Deo, N. Random-tree diameter and the diameter-constrained MST. *Int. J. Comput. Math.*, v. 79, n. 6, p. 651–663, 2002.

Abuali, F. N.; Schoenefeld, D. A.; Wainwright, R. L. Designing telecommunications networks using genetic algorithms and probabilistic minimum spanning trees. In: *SAC '94: Proceedings of the 1994 ACM symposium on Applied computing*, New York, NY, USA: ACM Press, p. 242–246, 1994.

Abuali, F. N.; Wainwright, R. L.; Schoenefeld, D. A. Determinant factorization: A new encoding scheme for spanning trees applied to the probabilistic minimum spanning tree problem. In: Eshelman, L. J., ed. *Proc. of the Sixth Int. Conf. on Genetic Algorithms*, San Francisco, CA: Morgan Kaufmann, p. 470–477, 1995.

Atallah, M. J.; Fox, S. *Algorithms and theory of computation handbook*. Boca Raton, FL, USA: CRC Press, Inc., produced By-Suzanne Lassandro, 1998.

Ausiello, G.; Crescenzi, P.; Kann, V.; Marchetti-sp; Gambosi, G.; Spaccamela, A. M. *Complexity and approximation: Combinatorial optimization problems and their approximability properties*. Heidlberg: Springer-Verlag, 2003.

Bäck, T.; Fogel, D. B.; (eds) *Evolutionary computation 1: Basic algorithms and operators*. IOP Publishing Ltd, 2000.

Bean, J. C. Genetic algorithms and random keys for sequencing and optimization. *ORSA JOURNAL ON COMPUTING*, v. 6, n. 2, p. 154–160, 1994.

Benayoun, R.; de Montgolfier, J.; Tergny, J.; Laritchev, O. Linear programming with multiple objective functions: Step method (stem). *Mathematical Programming*, v. 1, n. 1, p. 1436–4646, 1971.

Cahn, R. S. *Wide area network design: concepts and tools for optimization.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998.

Carrano, E.; Fonseca, C.; Takahashi, R.; Pimenta, L.; Neto, O. A preliminary comparison of tree encoding schemes for evolutionary algorithms. p. 1969–1974, 2007.

Carvalho, P. M. S.; Ferreira, L. A. F. M.; Barruncho, L. M. F. Solving radial topology constrained problems with evolutionary algorithms. In: *SEAL'98: Selected papers from the Second Asia-Pacific Conference on Simulated Evolution and Learning on Simulated Evolution and Learning*, London, UK: Springer-Verlag, p. 58–65, 1999.

Carvalho, P. M. S.; Ferreira, L. A. F. M.; Barruncho, L. M. F. On spanning-tree recombination in evolutionary large-scale network problems - application to electrical distribution planning. *IEEE Trans. Evolutionary Computation*, v. 5, n. 6, p. 623–630, 2001.

Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; Stein, C. *Introduction to algorithms, 2nd edition.* MIT Press, McGraw-Hill Book Company, 2000.

Current, I.; Marsh, M. Multiobjective transportation network design and routing-problems: Taxonomy and annotation. *European Journal of Operational Research*, v. 65, p. 4–19, 1993.

Darwin, C. *A origem das espécies.* 1859.

Darwin, C. *A origem das espécies - coleção a obra prima de cada autor.* Martin Claret, 2004.

Davis, L.; Orvosh, D.; Cox, A.; Qiu, Y. A genetic algorithm for survivable network design. In: *Proceedings of the 5th International Conference on Genetic Algorithms*, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., p. 408–415, 1993.

De Jong, K. A. *An analysis of the behavior of a class of genetic adaptive systems.* Tese de Doutoramento, 1975.

De Jong, K. A. *Evolutionary computation: A unified approach.* MIT Press, 2006.

Deb, K. *Multi-objective optimization using evolutionary algorithms.* New York, NY, USA: John Wiley & Sons, Inc., 2001.

Delbem, A.; de Carvalho, A.; Policastro, C. A.; Pinto, A. K.; Honda, K.; Garcia, A. C. Node-depth encoding for evolutionary algorithms applied to network design. In: Deb, K.; Poli, R.; Banzhaf, W.; Beyer, H.-G.; Burke, E.; Darwen, P.; Dasgupta, D.; Floreano, D.; Foster, J.; Harman, M.; Holland, O.; Lanzi, P. L.; Spector, L.; Tettamanzi, A.; Thierens, D.; Tyrrell, A., eds. *Genetic and Evolutionary Computation – GECCO-2004, Part I*, Seattle, WA, USA: Springer-Verlag, p. 678–687, 2004 (*Lecture Notes in Computer Science*, v.3102).

Delbem, A. C. B. *Restabelecimento de energia em sistemas de distribuição por algoritmo evolucionionário associado a representaçãoo por cadeias de grafos.* Tese de Doutoramento, Universidade de São Paulo, 2002.

Delbem, A. C. B.; de Carvalho, A. A forest encoding for evolutionary algorithms applied to design problems. *Genetic Algorithm and Evolutionary Computation Conference 20003, Lecture Notes in Computer Science*, v. 2723, p. 634–635, 2003.

Delbem, A. C. B.; de Carvalho, A.; Bretas, N. G. Optimal energy restoration in radial distribution systems using a genetic approach and graph chain representation. *Electric Power Systems Researchs*, v. 67/3, p. 197–205, 2003.

Delbem, A. C. B.; de Carvalho, A. C. P. L. F.; Bretas, N. G. Main chain representation for evolutionary algorithms applied to distribution system reconfiguration. *IEEE Transactions on Power Systems*, v. 20, n. 1, p. 425–436, 2005.

Even, S. *Graph algortithms.* Computer Science Press, 1979.

Fogel, D. B. *Evolutionary computation: Toward a new philosophy of machine intelligence.* IEEE Press, 2006.

Fogel, L. Autonomous automata. *Industrial Research*, v. 4, p. 14–19, 1962.

Fogel, L.; Owens, A.; Walsh, M. *Artificial intelligence through simulated evolution.* Willey and Sons, 1966.

Gaube, T.; Rothlauf, F. The link and node biased encoding revisited: Bias and adjustment of parameters. In: *Proceedings of the EvoWorkshops on Applications of Evolutionary Computing*, London, UK: Springer-Verlag, p. 1–10, 2001.

Gen, M.; Cheng, R. *Genetic algorithms and engineering design.* Wiley Series in Engineering Design and Automation. New York, USA: John Wiley & Sons (Sd), 1997.

Gen, M.; Zhou, G. *An approach to the degree-constrained minimum spanning tree problem using genetic algorithm.* Relatório Técnico, Ashikaga Intitute of Technology, 1995.

Goldberg, D.; Holland, J. Genetic algorithms and machine learning. *Mach. Learn.*, v. 3, n. 2-3, p. 95–99, 1988.

Goldberg, D. E. *Genetic algorithms in search, optimization and machine learning.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989.

Goldberg, D. E. *The design of innovation: Lessons from and for competent genetic algorithms.* Norwell, MA, USA: Kluwer Academic Publishers, 2002a.

Goldberg, D. E. *The design of innovation: Lessons from and for competent genetic algorithms.* Boston, MA, USA: Kluwer, 248 p., 2002b.

Gottlieb, J.; Julstrom, B. A.; Raidl, G. R.; Rothlauf, F. Prüfer numbers: A poor representation of spanning trees for evolutionary search. In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, San Francisco, California, USA: Morgan Kaufmann, p. 343–350, 2001.

Gross, J.; Yellen, J. *Graph theory and its applications.* Boca Raton, FL, USA: CRC Press, Inc., 1999.

Gross, J. L.; Yellen, J. *Handbook of graph theory.* CRC Press, 2004.

Gruber, M.; van Hemert, J.; Raidl, G. R. Neighborhood searches for the bounded diameter minimum spanning tree problem embedded in a vns, ea, and aco. Seattle, USA: ACM, p. 1187–1194, 2006.

Hamming, R. W. *Coding and information theory.* Prentice-Hall, 1980.

Han, L.-X.; Wang, Y.; Guo, F.-Y. A new genetic algorithm for the degree-constrained minimum spanning tree problem. *VLSI Design and Video Technology, 2005. Proceedings of 2005 IEEE International Workshop on*, p. 125–128, 28-30 May 2005.

Holland, J. *Adaptation in natural and artificial systems.* University of Michigan Press, 1975.

Johnson, D. S.; Lenstra, J. K.; Kan, A. H. G. R. The complexity of the network design problem. *Networks*, v. 9, p. 279–285, 1978.

Julstrom, B. A. The blob code: A better string coding of spanning trees for evolutionary search. In: Rothlauf, F., ed. *Representations and Operators for Network Problems (ROPNET 2001)*, San Francisco, California, USA, p. 256–261, 2001.

Julstrom, B. A. Codings and operators in two genetic algorithms for the leaf-constrained minimum spanning tree problem. *International Journal of Applied Mathematics and Computer Science*, v. 14, n. 3, p. 385–396, 2004a.

Julstrom, B. A. Encoding bounded-diameter spanning trees with permutations and with random keys. In: *Genetic and Evolutionary Computation – GECCO-2004, Part I*, Seattle, WA, USA: Springer-Verlag, p. 1272–1281, 2004b (*Lecture Notes in Computer Science*, v.3102).

Julstrom, B. A. The blob code is competitive with edge-sets in genetic algorithms for the minimum routing cost spanning tree problem. In: *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, New York, NY, USA: ACM Press, p. 585–590, 2005.

Jungnickel, D. *Graphs, networks and algorithms (algorithms and computation in mathematics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2004.

Kershenbaum, A. *Telecommunications network design algorithms*. New York, NY, USA: McGraw-Hill, Inc., 1993.

Knowles, J.; Corne, D. The Pareto Archived Evolution Strategy: A New Baseline Algorithm for Multiobjective Optimisation. In: *1999 Congress on Evolutionary Computation*, Washington, D.C.: IEEE Service Center, p. 98–105, 1999.

Knowles, J.; Corne, D.; Oates, M. A new evolutionary approach to the degree-constrained minimum spanning tree problem. In: *Proceedings of the Genetic and Evolutionary Computation Conference*, Orlando, Florida, USA: Morgan Kaufmann, 1999.

Knowles, J. D.; Corne, D. A new evolutionary approach to the degree-constrained minimum spanning tree problem. *IEEE Trans. Evolutionary Computation*, v. 4, n. 2, p. 125–134, 2000.

Koza, J. *Genetic programming: On the programming of computers by means of natural selection (complex adaptive systems)*. MIT Press, 1992.

Krishnamoorthy, M.; Ernst, A. T.; Sharaiha, Y. M. Comparison of algorithms for the degree constrained minimum spanning tree. *Journal of Heuristics*, v. 7, n. 6, p. 587–611, 2001.

Larraenaga, P.; Lozano, J. Estimation of distribution algorithms: A new tool for evolutionary computation. In: *Kluwer Academic Publishers.*, 2001.

Li, Y.; Bouchebaba, Y. A new genetic algorithm for the optimal communication spanning tree problem. *Artificial Evolution*, v. 1829, p. 162–173(1), 2000.

Libralao, G. L.; Pereira, F. C.; Lima, T. W.; Delbem, A. C. B. Node-depth encoding for evolutionary algorithms applied to multi-vehicle routing problem. In: *IEA/AIE'2005: Proceedings of the 18th international conference on Innovations in Applied Artificial Intelligence*, London, UK: Springer-Verlag, p. 557–559, 2005.

de Lima, T. W.; Rothlauf, F.; Delbem, A. C. The node-depth encoding: analysis and application to the bounded-diameter minimum spanning tree problem. In: *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, New York, NY, USA: ACM, p. 969–976, 2008.

Lin, L.; Gen, M. Node-Based Genetic Algorithm for Communication Spanning Tree Problem. *IEICE Trans Commun*, v. E89-B, n. 4, p. 1091–1098, 2006.

Mhlenbein, H.; Paab, G. From recombination of genes to the estimation of distributions i. binary parameters. In: *In Lecture Notes in Computer Science 1411: Parallel Problem Solving from Nature-PPSN IV*, p. 178–187, 1996.

Michalewicz, Z. *Genetic algorithms + data structures = evolution programs.* Springer, 1996.

Michalewicz, Z.; Fogel, D. *How to solve it: Modern heuristics.* Springer-Verlag New York, Inc., 2004.

Nemhauser, G. L.; Wolsey, L. A. *Integer and combinatorial optimization.* New York, NY, USA: Wiley-Interscience, 1988.

Palmer, C. C. *An approach to a problem in network design using genetic algorithms.* Tese de Doutoramento, Brooklyn, NY, USA, 1994.

Palmer, C. C.; Kershenbaum, A. An approach to a problem in network design using genetic algorithms. *Networks*, v. 26, p. 151–163, 1995.

Paulden, T.; Smith, D. From the dandelion code to the rainbow code: a class of bijective spanning tree representations with linear complexity and bounded locality. *IEEE Transactions Evolutionary Computation*, v. 10, n. 2, p. 108–123, 2006a.

Paulden, T.; Smith, D. Recent advances in the study of the dandelion code, happy code, and blob code spanning tree representations. In: *IEEE Congress on Evolutionary Computation*, p. 2111–2118, 2006b.

Pelikan, M.; Goldberg, D. E.; Cant-paz, E. Linkage problem, distribution estimation and bayesian networks. *Evolutionary Computation*, v. 3, n. 8, p. 311–340, 2000.

Picciotto, S. *How to encode a tree.* Tese de Doutoramento, University of California, 1999.

Pióro, M.; Medhi, D. *Routing, flow, and capacity design in communication and computer networks.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2004.

Prasad, P. V.; Sivanagaraju, S.; Sreenivasulu, N. Network reconfiguration for load balancing in radial distribution systems using genetic algorithm. *Electric Power Components and Systems*, v. 36, n. 1, p. 63–72, 2008.

Prüfer, H. Neuer beweis eines satzes uber permutationen. *Arch. Math. Phys.*, 1918.

Raidl, G. An efficient evolutionary algorithm for the degree-constrained minimum spanning tree problem. In: *Proceedings of the 2000 Congress on Evolutionary Computation CEC00*, La Jolla Marriott Hotel La Jolla, California, USA: IEEE Press, p. 104–111, 2000.

Raidl, G. Coleção de grafos para o problema de árvore geradora mínima com restrição de grau., comunicação Pessoal, 2008.

Raidl, G. Coleção de casos de teste para o problema de árvore geradora de comunicação ótima., comunicação Pessoal, 2009.

Raidl, G.; Drexel, C. A predecessor coding in an evolutionary algorithm for the capacitated minimum spanning tree problem. In: Armstrong, C., ed. *Proceeding of 2000 Genetic and Evolutionary Computation Conference*, p. 309–316, 2000.

Raidl, G.; Julstrom, B. A. Edge-sets: An effective evolutionary coding of spanning trees. 2001.

Raidl, G. R.; Julstrom, B. A. Edge sets: an effective evolutionary coding of spanning trees. *IEEE Trans. Evolutionary Computation*, v. 7, n. 3, p. 225–239, 2003.

Rechenberg, I. Cybernetc solution path of an experimental problem. *Library Translation*, v. 1122, 1965.

Ridley, M. *Evolution*. Phoenix, 2001.

Rothlauf, F. *Representations for genetic and evolutionary algorithms*. Springer-Verlag, 2006.

Rothlauf, F.; Gerstacker, J.; Heinzl, A. On the optimal communication spanning tree problem. 2003.

Rothlauf, F.; Goldberg, D. E.; Heinzl, A. Network random keys: A tree representation scheme for genetic and evolutionary algorithms. *Evolutionary Computation*, v. 10, n. 1, p. 75–97, 2002.

Rothlauf, F.; Tzschoppe, C. *On the bias and performance of the edge-set encoding*. Relatório Técnico, Fakultät für Betriebswirtschaftslehre - Uni-Mannheim, 2005.

dos Santos, A. C. *Algoritmo evolutivo computacionalmente eficiente para reconfiguração de sistemas de distribuição*. Tese de doutorado, Escola de Engenharia Elétrica de São Carlos - Universidade de São Paulo, São Carlos - SP - Brasil, 2009.

dos Santos, A. C.; Delbem, A. C. B.; Bretas, N. G. A multiobjective evolutionary algorithm with node-depth encoding for energy restoration. *International Conference on Natural Computation*, v. 6, p. 417–422, 2008.

Schindler, B.; Rothlauf, F.; Pesch, H.-J. Evolution strategies, network random keys, and the one-max tree problem. In: *Proceedings of the Applications of Evolutionary Computing on EvoWorkshops 2002*, London, UK: Springer-Verlag, p. 143–152, 2002.

Schwefel, H. *Kybernetische evolution als strategie der experimentellen forschung in der stramunngstechnik*. Relatório Técnico, Technical University of Berlin, 1965.

Singh, A.; Gupta, A. K. Improved heuristics for the bounded-diameter minimum spanning tree problem. *Soft Comput.*, v. 11, n. 10, p. 911–921, 2007.

Soak, S.-M. A new evolutionary approach for the optimal communication spanning tree problem. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, v. E89-A, n. 10, p. 2882–2893, 2006.

Soak, S.-M.; Corne, D.; Ahn, B.-H. A new evolutionary algorithm for spanning-tree based communication network design. *IEICE Trans Commun*, v. E88-B, n. 10, p. 4090–4093, 2005.

Soak, S.-M.; Corne, D.; Byung-Ha, A. A new encoding for the degree constrained minimum spanning tree problem. In: *KES*, p. 952–958, 2004.

Starkweather, T.; McDaniel, S.; Mathias, K.; Whitley, D.; Whitley, C. A comparison of genetic sequencing operators. In: Belew, R.; Booker, L., eds. *Proceedings of the Fourth International Conference on Genetic Algorithms*, San Mateo, CA: Morgan Kaufman, p. 69–76, 1991.

Thompson, E.; Paulden, T.; Smith, D. K. The dandelion code: A new coding of spanning trees for genetic algorithms. *IEEE Transactions on Evolutionary Computation*, v. 11, n. 1, p. 91–100, 2007.

Toth, P.; Vigo, D. *The vehicle routing problem.* Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2001.

Tzschoppe, C.; Rothlauf, F.; Pesch, H.-J. The edge-set encoding revisited: On the bias of a direct representation for trees. In: *Genetic and Evolutionary Computation – GECCO-2004, Part II*, Seattle, WA, USA: Springer-Verlag, p. 258–270, 2004 (*Lecture Notes in Computer Science*, v.3103).

Valiente, G. *Algorithms on trees and graphs.* Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2002.

Voss, S.; Osman, I. H.; Roucairol, C. *Meta-heuristics: Advances and trends in local search paradigms for optimization.* Norwell, MA, USA: Kluwer Academic Publishers, 1999.

Zhou, G.; Gen, M. A note on genetic algorithms for degree-constrained spanning tree problems. *Networks (USA)*, v. 30, n. 2, p. 91–95, 1997.

Zhou, G.; Gen, M. A genetic algorithm approach on tree-like telecommunication network design problem. *J. of the Operational Research Society*, v. 54, n. 3, p. 248–254, 2003.

Zhou, G.; Meng, Z.; Cao, Z.; Cao, J. A new tree encoding for the degree-constrained spanning tree problem. *Computational Intelligence and Security, 2007 International Conference on*, p. 85–90, 15-19 Dec. 2007.

Apêndice

A

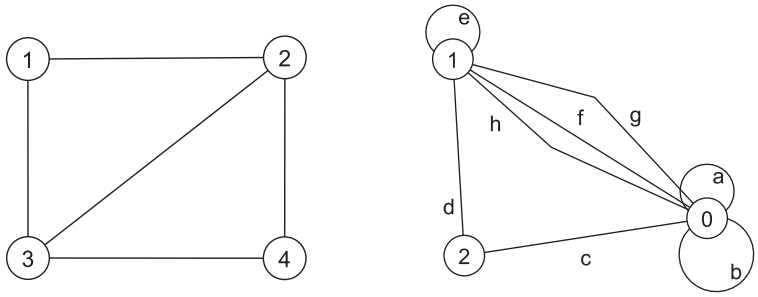
Conceitos e Algoritmos Básicos em Grafos

Os grafos estão presente em diversos problemas do cotidiano das pessoas, principalmente nos projetos de redes, como uma forma de modelagem dos mesmos. Um grafo consiste de um conjunto de pontos denominados vértices ou nós e de um conjunto de ligações entre esses pontos chamadas de arestas. Esse par de conjuntos é denominado grafo ou rede, o qual pode ser utilizado para representar uma grande diversidade de sistemas, tais como circuitos elétricos, estradas, moléculas orgânicas, relacionamentos sociais, bancos de dados e o fluxo de controle em programas de computador. As Figuras A.1(a) e A.1(b) são exemplos de grafos.

Formalmente, um grafo $G = (V, E)$ é uma estrutura formada por um conjunto finito e não vazio V denominado conjunto de vértices (ou nós) e um conjunto também finito de pares de vértices E denominado conjunto de arestas ou ligações (Gross e Yellen, 2004; Jungnickel, 2004; Valiente, 2002; Gross e Yellen, 1999). Um subgrafo $H = (V_H, E_H)$ de um grafo G é o grafo que no qual todos os seus vértices e arestas pertencem a V e E, ou seja $V_H \subseteq V$ e $E_H \subseteq E$.

Algumas importantes propriedades de um grafo são sua ordem e seu tamanho. A ordem é o número de vértices do grafo, representada por n, com $n = |V|$ (Gross e Yellen, 2004). Por outro lado, o tamanho é o número de arestas, representado por m, onde $m = |E|$ (Jungnickel, 2004).

Os vértices e as arestas de um grafo podem conter características associadas a eles.



(a) Grafo A. (b) Grafo B.

Figura A.1: Exemplos de grafos.

A característica mais comum associada a um vértice é chamada de rótulo ou nome do vértice (ver Figura A.1). As arestas também podem ter rótulos associados a elas, como na Figura A.1(b), mas em geral as arestas possuem como atributo um valor de custo.

A principal classificação de um grafo é de acordo com a existência ou não de orientação nas suas arestas. Na presença de orientação nas arestas o grafo é chamado de grafo orientado ou dígrafo. No caso de grafo orientado o conjunto das arestas é composto por pares ordenados e as arestas são chamadas de arcos. Os grafos que não possuem orientação nas arestas são chamados de grafos não orientados ou somente grafos e, nesse caso, os pares de arestas (v_i, v_j) e (v_j, v_i) representam a mesma aresta.

A relação entre os vértices e as arestas é chamada incidência. Uma aresta é incidente em um vértice v_i se este é um dos vértices que compõem o par ordenado que representa aresta (Gross e Yellen, 1999). O número de arestas incidentes em um vértice define a propriedade chamada grau do vértice representado por $deg(v_i)$. Em dígrafos o grau é dividido em grau de saída e grau de entrada. O grau de saída é o número de arcos que possuem o vértice v_i na primeira posição do par ordenado. O grau de entrada é o número de arcos que possuem v_i na segunda posição do par ordenado. O grau total de um vértice em um dígrafo é dado pela soma dos graus de saída e entrada (Gross e Yellen, 2004).

A existência de uma aresta ou arco entre dois vértices estabelece uma relação de adjacência ou vizinhança entre eles, ou seja, um vértice v_i é adjacente ou vizinho a um vértice v_j se existe a aresta ou arco (v_i, v_j) (Gross e Yellen, 2004).

Um caminho no grafo de um vértice v_i para um vértice v_j é uma seqüência de vértices $\{v_i, v_{i+1}, v_{i+2}, \ldots, v_{j-1}, v_j\}$ onde dois vértices consecutivos são adjacentes. Quando os vértices v_i e v_j de um caminho são iguais, este é chamado de ciclo. O comprimento de um caminho ou ciclo é determinado pelo número de arestas que formam o caminho (Valiente, 2002; Gross e Yellen, 1999).

Para que um grafo seja conexo, é preciso que para todos os pares de vértices $v_i, v_j \in V$

exista um caminho ou uma relação de adjacência entre eles. Uma componente de um grafo G é o maior subgrafo H de G que seja conexo. Além disso, um grafo é completo se todo par distinto de vértices v_i e v_j com $i \neq j$ possui uma relação de adjacência. Em um grafo completo não-orientado, o número de arestas do grafo pode ser calculado por $m = n(n-1)/2$.

Existem duas estruturas básicas para representar um grafo. A primeira é a matriz de adjacências, na qual um grafo é representado por uma matriz quadrada A de ordem n, onde cada linha e coluna é o índice de um vértice e cada aresta (v_i, v_j) presente no grafo é representada pela atribuição do valor 1 ou do seu custo à posição $a_{i,j}$ da matriz. A segunda estrutura utilizada para representar um grafo é a lista de adjacências que consiste em listas encadeadas (Atallah e Fox, 1998) para cada vértice contendo os vértices adjacentes a ele (Gross e Yellen, 2004).

Árvores e Florestas

Árvores e florestas são os tipos de grafos mais conhecidos e utilizados na área de projeto de redes. Uma árvore, $T = (V_T, E_T)$, é um grafo conexo minimal, ou seja a remoção de uma aresta torna o grafo desconexo. A Figura A.2 apresenta o exemplo de uma árvore com oito vértices. O número de arestas de uma árvore é $m = n - 1$ (Gross e Yellen, 2004; Jungnickel, 2004; Gross e Yellen, 1999).

Uma floresta $F = (V_F, E_F)$ é um grafo sem ciclos, não necessariamente conexo (Gross e Yellen, 2004), que pode conter uma árvore, nesse caso a floresta é conexa; ou mais de uma árvore, nesse caso o grafo é desconexo.

Quando uma árvore possui um vértice distinguível chamado de raiz (*root*) de T, essa árvore é dita enraizada. Por exemplo, na Figura A.2 o vértice 1 é a raiz da árvore, então esta é uma árvore enraizada. Em uma árvore, os vértices que possuem grau um são chamados de vértices folha, os demais vértices são conhecidos como vértices internos (Gross e Yellen, 1999). Os vértices no caminho de um vértice v_i até a raiz são chamados de predecessores de v_i. O primeiro predecessor nesse caminho é chamado de pai de v_i. Vértices que possuem o mesmo pai são chamados de irmãos. Na Figura A.2, os vértices 2, 6 e 7 são irmãos.

Em uma árvore enraizada a profundidade de um vértice é definida como o comprimento do caminho único da raiz da árvore até o vértice. A profundidade da árvore é a profundidade máxima encontrada na árvore. Por exemplo, na árvore da Figura A.2 a maior profundidade é 3 (vértice 8), sendo esta também a profundidade da árvore. Um vértice v_i é o pai de um vértice v_j, se existe a aresta (v_i, v_j) no caminho de *root* até v_j e

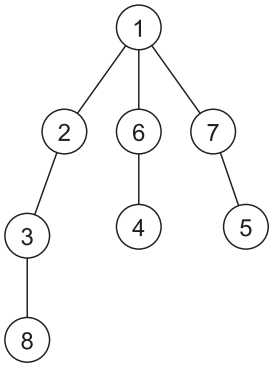


Figura A.2: Exemplo de árvores.

v_i aparece antes de v_j no caminho. Nesse caso, o vértice v_j é dito filho de v_i.

O grafo $T_H = (V_{T_H}, E_{T_H})$ denota uma sub-árvore de T onde, $V_{T_H} \subseteq V_T$ e $E_{T_H} \subseteq E_T$. Um tipo especial de árvore é a árvore geradora de um grafo. Uma árvore geradora de um grafo $G = (V, E)$ é a árvore $T = (V_T, E_T)$, onde $V_T = V$ e $E_T \subset E$, ou seja, a árvore geradora é um subgrafo de G que contém todos os vértices de G e um subconjunto de arestas que não forma ciclos (Valiente, 2002). Em um grafo G onde são definidos pesos para as arestas, uma árvore geradora cuja a soma dos pesos associados às arestas é o menor dentre todas as árvores geradoras de G é chamada de árvore geradora mínima. Uma floresta geradora mínima é uma floresta cuja soma dos pesos de todas as árvores que compõem a floresta é mínimo.

Algoritmos de árvore geradora mínima

Os algoritmos de Prim e Kruskal (Gross e Yellen, 2004; Valiente, 2002; Atallah e Fox, 1998) tem como objetivo encontrar a árvore geradora mínima $T = (V_T, E_T)$ de um grafo G. Ambos os algoritmos consideram os conjuntos V_T e E_T inicialmente vazios e obtém uma árvore geradora mínima por meio da seleção das arestas de menor custo para E_T e dos vértices correspondentes para V_T.

No algoritmo de Prim o processo para encontrar uma árvore geradora mínima começa pela seleção aleatória de um vértice v_i, o qual é inserido em V_T. No passo seguinte o algoritmo seleciona a aresta $e_i = (v_i, v_j)$ de menor custo entre o conjunto de arestas ligadas a v_i e adiciona a aresta e_i ao conjunto E_T de arestas da árvore. Obtida a aresta de menor custo, o outro vértice v_j que compõem a aresta é incluído no conjunto V_T de vértices na árvore. O processo repete-se até que sejam inseridos todos os vértices em V_T. A cada iteração, o algoritmo analisa as arestas dos vértices já inseridos na árvore e seleciona a aresta de menor peso desde que esta não forme um ciclo. Após todos os vértices terem

sido inseridos, o resultado é uma árvore geradora mínima do grafo. A complexidade de tempo do algoritmo de Prim é $O(mlogn)$ (Gross e Yellen, 2004).

Ao contrário do algoritmo de Prim, o algoritmo de Kruskal trabalha diretamente com as arestas. O primeiro passo do algoritmo ordena as arestas do grafo de forma crescente, obtendo uma seqüência em que as arestas são analisadas pelo algoritmo. No segundo passo, é selecionada uma aresta e_i de menor peso e esta é inserida no conjunto E_T de arestas da árvore. No processo iterativo do algoritmo de Kruskal as arestas são analisadas em seqüência. Se a aresta não formar ciclo, esta é inserida em E_T até que $|E_T| = n - 1$. Ao terminar o processo iterativo, obtém-se uma árvore geradora mínima do grafo. A complexidade de tempo do algoritmo de Kruskal é $O(mlogn)$ (Gross e Yellen, 2004).

Percursos em Grafos, Árvores e Florestas

Uma classe importante de algoritmos para resolver problemas em grafos são os algoritmos que determinam os percursos. Os principais percursos em grafos (florestas ou não) são busca em profundidade e busca em largura (Valiente, 2002). Os percursos específicos para árvores e florestas são pré-ordem e pós-ordem (Jungnickel, 2004; Valiente, 2002).

Busca em Profundidade

A busca em profundidade de um grafo $G = (V, E)$ com n vértices consiste em a partir de um vértice aleatório marcá-lo como visitado e visitar o primeiro de seus adjacentes, marcando-o. O processo segue recursivamente, sempre visitando o primeiro adjacente não visitado do último vértice visitado. Se o ultimo vértice visitado não possuir nenhum adjacente, retorna-se ao vértice anterior e verifica-se a existência de algum adjacente a ser visitado. O algoritmo repete este processo até que todos os vértices tenham sido visitados. O Algoritmo A.0.1 descreve os passos para efetuar uma busca em profundidade (Valiente, 2002). O algoritmo A.0.1 possui complexidade de tempo $O(n + m)$ (Valiente, 2002).

Algoritmo A.0.1: - Busca em Profundidade.

(1) Selecione um vértice inicial $z \epsilon V$
(2) Marque z
(3) Adicione z a pilha
(4) Enquanto a pilha for não vazia faça
  (5) $u \leftarrow$ copia o topo da pilha
  (6) Para cada adjacente v de u tal que v não é marcado
    (7) Marque v e insira v na pilha
    (8) Retorne ao passo (4)
  (9) Senão, remova u da pilha e retorne ao passo (4)

Busca em Largura

A busca em largura em um grafo $G = (V, E)$ começa visitando um vértice aleatório e, em seguida, percorre todos os seus adjacentes ainda não visitados. Recursivamente, percorre-se os adjacentes não visitados dos vértices já visitados na ordem em que estes foram visitados. A visitação de todos os vértices do grafo é utilizado como critério de parada. Os passos para efetuar uma busca em largura estão descritos no Algoritmo A.0.2 (Valiente, 2002). O algoritmo de busca em largura possui complexidade de tempo $O(n + m)$ (Valiente, 2002).

Algoritmo A.0.2: Busca em Largura.

(1) Selecione um vértice inicial $z \epsilon V$
(2) Marque z
(3) Marque todos adjacentes a z e adicione-os à fila
(4) Enquanto a fila não for vazia faça
    (5) $u \leftarrow$ copie o primeiro vértice da fila
    (6) Marque todos adjacentes a u não marcados e adicione-os a fila
    (9) Remova u da fila e retorne ao passo (4)

Percurso em Pré-ordem

No percurso em pré-ordem de uma árvore $T = (V_T, E_T)$, os vértices são percorridos começando pela raiz de T, em seguida é visitado o primeiro filho de *root*. O algoritmo segue visitando os filhos da última raiz visitada até chegar em um vértice folha. Ao chegar neste tipo de vértice, o processo retorna para o próximo filho do pai do vértice folha. Após todos os filhos da raiz da sub-árvore terem sido visitados, o algoritmo retorna para visitar os filhos da raiz da sub-árvore anterior, até que todos os vértices tenham sido percorridos. O Algoritmo A.0.3 descreve os passos do percurso em pré-ordem (Jungnickel, 2004; Valiente, 2002). O algoritmo recursivo em pré-ordem tem complexidade de tempo $O(n)$ para uma árvore de n vértices.

Algoritmo A.0.3: Percurso em Pré-ordem.

(1) Se a árvore for vazia pare
(2) Senão visite a raiz
(3) Percorra o filho mais a esquerda não visitado em pré-ordem
(4) Se a árvore não vazia volte ao passo (3)

Percurso em Pós-ordem

No percurso em pós-ordem de uma árvore $T = (V_T, E_T)$, os vértices são percorridos a partir dos vértices folha de cada sub-árvore da árvore sendo que a raiz da sub-árvore (árvore), só é percorrida depois que todos os seus filhos são visitados. O Algoritmo A.0.4 descreve os passos do percurso em pós-ordem (Jungnickel, 2004; Valiente, 2002). O algo-

ritmo recursivo em pós-ordem possui complexidade de tempo $O(n)$ para uma árvore de n vértices.

Algoritmo A.0.4: - Percurso em Pós-ordem.

(1) Se a árvore for vazia pare
(2) Percorra o filho mais a esquerda não visitado em pós-ordem
(3) Se árvore não vazia volte ao passo (2)
(4) Senão visite a raiz

Apêndice

B

Análise de Complexidade dos Operadores de Mutação da NDDE

A análise de complexidade apresentada nesse Apêndice foi desenvolvida pelos professores Alexandre C. B. Delbem e Guilherme P. Telles.

A complexidade de tempo para obter uma nova floresta na NDDE é avaliada segundo dois limitantes: s_{op} complexidade para aplicar PAO e CAO (ver Seção 5.1) e s_{tripla} a complexidade para encontrar os vértices p, r e a, ambos como função de $|T_{origem}|$ e $|T_{destino}|$ o tamanho das árvores (ver Seção 5.1.1).

Ao analisar os passos dos Algoritmos 5.1.1 e 5.1.2 de PAO e CAO, respectivamente, tem-se que os Passos 1, 2 e 4 de ambos podem ser efetuados percorrendo a árvore T_{origem}, com tempo de execução $O(|T_{origem}|)$. Para construir $T'_{destino}$ é necessário encontrar o ponto de inserção da sub-árvore em $T_{destino}$ e copiar os vértices de ambos os *arrays* para $T'_{destino}$. Novamente, é suficiente percorrer as árvores para efetuar essa tarefa o que resulta em tempo de execução $O(|T_{origem}| + |T_{destino}|)$.

Os Passos 5 e 6 dos algoritmos de PAO e CAO, respectivamente, são responsáveis por atualizar a estrutura da floresta e necessitam de um tempo $O(t)$, com t o número de árvores da floresta, pois essa etapa consiste em uma atualização de ponteiros. Assim, complexidade de s_{op} é $O(|T_{origem}| + |T_{destino}| + t)$. Para simplificar a notação, usa-se $s = |T_{origem}| + |T_{destino}|$, e obtém-se $s_{op} = O(s + t)$.

Resta analisar a complexidade s_{tripla} para encontrar os vértices p, r e a. No entanto, determinar a complexidade de s_{tripla} em função de $|T_{origem}|$, $|T_{destino}|$ e t é um pouco

mais difícil. A Seção B.1 apresenta o cálculo de s_{tripla}. A Seção B.2 mostra que a média de $s_{op} + s_{tripla}$ é $O(\sqrt{n})$ para florestas com $O(\sqrt{n})$ árvores. A Seção B.3 mostra como manipular florestas que não têm $O(\sqrt{n})$ árvores, preservando a complexidade de $s_{op}+s_{tripla}$ em $O(\sqrt{n})$.

B.1 Complexidade de s_{tripla}

Nos Algoritmos 5.1.3 e 5.1.4, o Passo 1 pode percorrer a NDDE da floresta F (o *array* com os ponteiros para a NDDE de cada árvore) inteira em busca de uma árvore que possua um vértice com adjacente em outra árvore. Assim, o Passo 1 custa $O(t)$, com t o número de árvores na floresta.

O Passo 2 (dos Algoritmos 5.1.3 e 5.1.4, substituindo p por r) pode necessitar a investigação de todos os vértices de T_{origem} (na NDDE correspondente) para encontrar p [1]. Portanto, o Passo 2 custa $O(|T_{origem}|)$.

No Passo 3(i) do Algoritmo 5.1.4, a seleção do vértice p é efetuada pelos passos do Algoritmo 5.1.6. Com base nesse algoritmo esse passo custa $O(|T_{origem}|)$, pois no pior caso é necessário percorrer a árvore para encontrar um vértice p adequado, se i_r é o último vértice na árvore.

Para implementar o Passo 3 do Algoritmo 5.1.3 e 3(ii) do Algoritmo 5.1.4, utiliza-se os *arrays*: L_G, L_O e L_T definidos na Seção 5.1.1 e o pseudo-código presente no Algoritmo 5.1.5. Os *arrays* L_G e L_O podem ser pré-processados. L_T é um *array* de rascunho que necessita ser inicializado somente uma vez. Pode-se verificar se um vértice é adjacente a p em G em tempo constante por meio de L_G. Assim, pode-se executar somente $|T_{origem}|$ verificações. Como consequencia, a marcação de L_T (Passo 1) necessita de tempo de execução $O(|T_{origem}|)$. O Passo 2 é $O(1)$. Como existem no máximo $|T_{origem}|$ marcas em L_T, o Passo 3 é repetido no máximo $|T_{origem}| + 1$ vezes. O processo de desmarcação do Passo 4 também necessita $|T_{origem}|$ passos. Portanto, a determinação de a requer $O(|T_{origem}|)$.

No caso de florestas com uma árvore e de grafos completos a complexidade é $O(|T|)$, pois no pior caso será necessário percorrer toda a permutação L_O dos índices de T para encontrar um vértice a adequado.

Como conseqüência os Passos 1, 2, e 3 dos Algoritmos 5.1.3 e 5.1.4 para determinação dos vértices p e a (p, a, e r para CAO) são $O(|T_{origem}| + t)$. Como $|T_{origem}|$ é limitado por $|T_{origem}| + |T_{destino}|$, ambas as operações são também $O(s + t)$.

[1] A existência de pelo menos um p adequado é garantida pelo Passo 1.

O Passo 4 determina a posição de a na $NDDE$, que requer uma busca em Π_a. O tempo de execução para essa busca é limitado pelo número de colunas em Π_a e o tamanho de π (ver Seção 4.3.2). Essas duas estruturas auxiliares são limitadas pelo número de florestas f geradas entre cada execução do procedimento de reinicialização. Como f pode ser arbitrariamente escolhido, faz-se $f = k\overline{s}$, onde k é uma constante positiva e $\overline{s}$ é o valor médio de s. Assim, o processo de busca resulta em $O(\overline{s})$.

O tempo de computação requisitado para reinicializar as matrizes Π_{v_x} é $O(bn)$, onde b é o número de florestas que serão preservadas após a reinicialização. Esse custo é amortizado por $f - 1$ florestas produzidas sem a reinicialização. Dessa forma, o custo amortizado é $\frac{bn}{f} = \frac{bn}{k\overline{s}}$, que resulta em $O(\frac{n}{\overline{s}})$ uma vez que $\frac{b}{k}$ é uma constante. Portanto, a determinação da posição de a requer tempo amortizado $O(\overline{s} + \frac{n}{\overline{s}})$.

Em resumo, os Passos 1, 2 e 3 são $O(s + t)$ e o Passo 4 $O(\overline{s} + \frac{n}{\overline{s}})$. Portanto, s_{tripla} necessita de tempo amortizado $c(s + t + \overline{s} + \frac{n}{\overline{s}})$, onde c é uma constante positiva. Seja $\overline{s_{tripla}}$ o valor médio de s_{tripla}. Verifica-se que $\overline{s_{tripla}} \leq c(\overline{s} + t + \frac{n}{\overline{s}})$. Portanto, $\overline{s_{tripla}}$ é $O(\overline{s} + t + \frac{n}{\overline{s}})$.

B.2 Complexidade de $\overline{s}$

Se os n vértices do grafo estiverem quase uniformemente distribuído nas t árvores da floresta, então o tamanho das árvores será $O(\frac{n}{t})$ e $\overline{s} = O(\frac{n}{t})$. Supondo que $t = O(\sqrt{n})$, lembre que $\overline{s}$ é o tamanho médio de $|T_{origem}| + |T_{destino}|$, então $\overline{s} = O(\sqrt{n})$.

Para os casos com árvores maiores que $O(\frac{n}{t})$, pode-se mostrar que $\overline{s}$ é $O(\sqrt{n})$ após diversas aplicações dos operadores, embora para alguns casos s seja maior que $O(\frac{n}{t})$. O $\overline{s}$ pode ser calculado por $\sum_{\{i,j\}\in\Omega, i\neq j} \frac{|T_i|+|T_j|}{|\Omega|}$. O resultado dessa somatória é $\frac{2n}{t}$.

Assim, $\overline{s} = O(\frac{n}{t})$ é, de fato, $\overline{s} = O(\sqrt{n})$ para $t = \sqrt{n}$. Se T_i e T_j $(\{i, j\} \in \Omega, i \neq j)$ não possuírem vértices adjacentes, nenhuma árvore é produzida por PAO e CAO. Nesse caso, $\overline{s}$ é sobrestimado pela somatória e, portanto, $\overline{s}$ mantém-se $O(\sqrt{n})$.

Como $\overline{s} = O(\sqrt{n})$, $\overline{s_{tripla}}$ é $O(\sqrt{n})$ (ver Seção B.1). Assim, como $s_{op} = s + t$, então $\overline{s_{op}} \leq \overline{s} + t$. Portanto, $\overline{s_{op}}$ é também $O(\sqrt{n})$.

B.3 Florestas que Não Têm $O(\sqrt{n})$ Árvores

Conforme mostrado na Seção B.2, o tempo médio necessário pelo algoritmo proposto é relativamente baixo se t é $O(\sqrt{n})$. As Seções seguintes mostram como manter $\overline{s_{op}}$ e $\overline{s_{tripla}}$ limitados por $O(\sqrt{n})$, respectivamente, para florestas com menos que $O(\sqrt{n})$ árvores e

para florestas com mais que $O(\sqrt{n})$ árvores.

B.3.1 Florestas com Menos que $O(\sqrt{n})$ Árvores

Esta Seção mostra que uma floresta com menos que $O(\sqrt{n})$ árvores pode ser manipulada pelos operadores da NDDE como se a floresta tivesse $O(\sqrt{n})$ árvores e, com isso, PAO e CAO mantém seu desempenho.

Seja $G = (V, E)$ um grafo não-orientado, onde $V = \{1, \ldots, n\}$ é o conjunto dos vértices e E é o conjunto das áreas. Suponha G conexo e $T = (V, E_T)$ uma árvore (ou floresta) geradora de G. Particione T em T_k sub-árvores conexas, $1 \leq k \leq \sqrt{n}$. Seja $G_k = (V_k, E_k)$ um subgrafo de G induzido por T_k e $E_r = E \setminus \bigcup_{k=1}^{\sqrt{n}} E_k$. O conjunto de arestas de T referentes a conexões entre sub-árvores que resultam do particionamento é denotado por E_{T*}, e o conjunto de vértices conectados pelas arestas de E_{T*} é denotada V_{T*}. O conjunto de arestas de E_r conectando vértices de V_{T*} é denotado de E^*. O par (V_{T*}, E^*) é o grafo G^* e o par(V_{T*}, E_{T*}) é a árvore geradora $T*$ de G^*.

Para cada T_k, uma NDDE N_k pode ser construída. Utilizando PAO e CAO, pode-se combinar as árvores $T_1, \ldots, T_{\sqrt{n}}$ em novas árvores, gerando novas florestas. Além disso, pode-se gerar novas florestas aplicando a abordagem para florestas de uma árvore para $T*$ ou trocando os vértices em $T*$, como descrito a seguir.

O particionamento de T resulta em uma árvore $T*$ que pode restringir os tipos de florestas que podem ser produzidas, não se obtendo todas as florestas geradoras de G. Para superar essa restrição, pode-se reparticionar T e obter uma nova $T*$ ou pode-se modificar $T*$ como segue. Seja a raiz r da sub-árvore T_k um vértice de T_k que também esteja em $T*$.

Inicialmente, selecione um sub-árvore T_k e um vértice a^* da sub-árvore adjacente a r. Inclua a^* em $T*$. Remova a raiz da sub-árvore em a^* de T_k e crie uma nova sub-árvore com seus vértices. Para manter o tamanho de $T*$ inalterado remova uma folha arbitrária x (diferente de a^*) de $T*$ e una as sub-árvores com raiz em x e com raiz no vértice adjacente a x em $T*$. .

É importante descrever o particionamento para casos especiais como para florestas com um, dois ou três vértices. A abordagem proposta não é aplicada para grafos de um ou dois vértices, uma vez que há apenas uma floresta para esses casos. Para grafos com três vértices, $\lceil\sqrt{3}\rceil = 2$, então $T*$ deve ter pelo menos 2 vértices e uma das sub-árvores correspondentes (T_1 ou T_2) deve ter dois vértices e a outra um vértice. Nesse caso, os vértices podem ser trocados entre as sub-árvores sempre que houver uma aresta conectando-as. Além disso, $T*$ pode ser alterada pela inserção e remoção de vértices como

descrito.

A remoção e a inserção de sub-árvores são passos definidos em PAO e CAO. Para modificar $T*$, pode-se aplicar o procedimento para florestas de uma árvore. Para isso, deve-se criar uma matriz Π_{a^*} para a^* e incluir a^* em T_{aux}. Então, $T*$ com diferentes vértices podem ser gerados usando PAO ou CAO.

Portanto, usando a estratégia de particionamento de árvores, qualquer árvore pode ser tratada pelos operadores PAO e CAO como uma floresta de pelo menos $\sqrt{n}$ árvores.

B.3.2 A Complexidade de Florestas com Menos que $O(\sqrt{n})$ Árvores

Para manipular florestas com $l < \sqrt{n}$ árvores, e $\sqrt{n} - l$ sub-árvores são criadas por reparticionamento e além da árvore $T*$ relativa a essas sub-árvores (ver Seção B.3.1).

Então, PAO e CAO são utilizados para gerar novas florestas a partir do conjunto das $T*$ sub-árvores. Assim, a produção de florestas a partir de sub-árvores alteradas requer $O(\sqrt{n})$ (ver Seção B.1).

Se $T*$ for modificada, mas seu conjunto de vértices não é, PAO e CAO podem ser aplicados a $T*$ para produção de novas florestas usando a abordagem para florestas de uma árvore. $T*$ pode também ser modificado por meio da alteração de seu conjunto de vértices. Como $|T*| = \sqrt{n}$ (ver Seção B.3.1) e a modificação de $T*$ necessita alguns passos de PAO e CAO, o mesmo limite de tempo de computação pode ser obtido para florestas com menos que $O(\sqrt{n})$.

B.3.3 Florestas com Mais que $O(\sqrt{n})$ Árvores

Uma floresta geradora F de um grafo G com mais que $O(\sqrt{n})$ árvores pode ser alterada de forma a obter uma outra floresta F' com $O(\sqrt{n})$ conforme segue. Pode-se modificar F criando um nó adicional v e conectando $t - \sqrt{n}$ vértices a v pela adição das arestas. Novas florestas podem ser produzidas pela aplicação de PAO e CAO a F', produzindo novas florestas descendentes de F' e assim por diante. Removendo v e as arestas adicionadas de F', pode-se obter uma nova floresta de G.

B.3.4 A Complexidade de Tempo para Florestas com Menos que $O(\sqrt{n})$ Árvores

Como F' tem $\sqrt{n}$ árvores, a abordagem proposta apresenta complexidade média $O(\sqrt{n})$. A inclusão de um vértice adicional e arestas para obter F' custa $O(n - \sqrt{n})$, uma vez que t é $O(n)$. De forma similar, a remoção de vértices e arestas adicionais da floresta é $O(n - \sqrt{n})$. Esse custo pode ser amortizado gerando $O(\sqrt{n})$ florestas sem a inserção ou remoção de vértices e arestas adicionais. Portanto, a abordagem proposta pode produzir florestas geradoras com mais que $O(\sqrt{n})$ árvores usando tempo médio $O(\sqrt{n})$.