UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

DANILO ALVES MARTINS DE FARIA

Operador de Recombinação EHR Aplicado ao Problema da Árvore Máxima

Goiânia
2013

DANILO ALVES MARTINS DE FARIA

Operador de Recombinação EHR Aplicado ao Problema da Árvore Máxima

Dissertação apresentada ao Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás, como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Ciência da Computação.

Orientador: Prof. Dra. Telma Woerle de Lima Soares

Goiânia
2013

Danilo Alves Martins de Faria

Graduou-se em Ciência da Computação na UFU - Universidade Federal de Uberlândia. Especializou-se pela FGV - Fundação Getúlio Vargas em (MBA) Tecnologia da Informação Aplicada à Gestão Estratégica de Negócios. Foi professor da Pontifícia Universidade Católica de Goiás e da Faculdade Cambury, entre 2002 a 2008.

Dedico esse trabalho a Deus, por me conceder todas as condições de iniciar e finalizar com sucesso este mestrado.

Aos meus pais pela paciência e companheirismo, mesmo eu ficando omisso à família neste período de estudos.

Agradecimentos

Agradeço a Deus pela graça de ter chegado até aqui.

À minha professora orientadora Dra. Telma Woerle de Lima Soares, por todo o empenho e prestatividade nas aulas, estudos dirigidos, orientações, correções, conselhos, incentivos e, sobretudo, pela paciência que teve durante este projeto.

Ao professor Dr. Anderson Soares, meu ex-aluno na PUC-GO, por todos os auxílios no decorrer deste mestrado.

Agradeço aos colegas do grupo de estudo de Teoria da Computação, Eduardo, Edjalma, Kelton e Ericsson, que foram importantíssimos para o crescimento de meus conhecimentos na disciplina de TC e para a minha readaptação à vida acadêmica 11 anos após minha conclusão de graduação.

Resumo

Faria, Danilo Alves Martins de. **Operador de Recombinação EHR Aplicado ao Problema da Árvore Máxima**. Goiânia, 2013. 81p. Dissertação de Mestrado. Instituto de Informática, Universidade Federal de Goiás.

Problemas de Projeto de Redes (PPRs) estão presentes em diversas áreas, tais como reconfiguração de sistemas de distribuição de energia elétrica, projetos de redes de comunicação, roteamento de veículos, reconstrução de árvores filogenéticas entre outros. Vários PPRs pertencem à classe de problemas NP-Difíceis. Dentre as técnicas utilizadas para resolvê-los, destacam-se os Algoritmos Evolutivos (AE), cujo processo de resolução de um problema simula a evolução natural das espécies. Entretanto, os AEs em sua forma padrão também possuem limitações quanto a PPRs de larga escala, ou com características muito específicas. Para solucionar esses problemas, diversas pesquisas têm estudado formas específicas de estruturas de dados dos PPRs. Dentre essas destaca-se a representação Nó-Profundidade-Grau (RNPG). Essa representação produz apenas soluções factíveis, independente da característica da rede. A RNPG possui dois operadores de mutação Preserve Ancestor Operator (PAO) e Change Ancestor Operator (CAO) e o operador de recombinação EHR (Evolutionary History Recombination Operator), que utiliza o histórico de aplicações dos operadores de mutação, o qual tem sido aplicado a PPRs com mais de uma árvore com bons resultados. Este trabalho propõem a adequação do EHR para PPRs clássicos de uma única árvore. Além disso, são desenvolvidos dois algoritmos evolutivos: o AE-RNPG, que utiliza a RNPG somente com os operadores de mutação; e o AE-EHR, que faz uso tanto dos operadores de mutação quanto do operador de recombinação EHR para o problema da Árvore máxima. Os resultados obtidos mostram que o AE-EHR obtém melhores soluções do que o AE-RNPG para a maioria das instâncias analisadas.

Palavras–chave

Nó-profundidade-grau, algoritmos evolutivos, projeto de rede, Problema da árvore máxima.

Abstract

Faria, Danilo Alves Martins de. **EHR recombination operator applied to One-Max Tree problem**. Goiânia, 2013. 81p. MSc. Dissertation. Instituto de Informática, Universidade Federal de Goiás.

Network Design Problems (NDPs) are present in many areas, such as electric power distribution, communication networks, vehicle routing, phylogenetic trees among others. Many NDPs are classified as NP-Hard problems. Among the techniques used to solve them, we highlight the Evolutionary Algorithms (EA). These algorithms simulate the natural evolution of the species. However, in its standard form EAs have limitations to solve large scale NDPs, or with very specific characteristics. To solve these problems, many researchers have studied specific forms of representation of NDPs. Among these stands we show Node-Depth-Degre Encoding (NDDE). This representation produces only feasible solutions, regardless of the network characteristics. NDDE has two mutation operators Preserve Ancestor Operator (PAO) and Ancestor Change Operator (CAO) and the recombination operator EHR (Evolutionary History Recombination Operator) that uses historical applications of mutation, and was applied to NDPs more than one tree and had good results. Thus, this work proposes adapt EHR for NDPs classics represented by a single tree. In addition, two evolutionary algorithms are developed: the AE-RNPG, which uses only NDDE, with mutation operators. And the AE-EHR, which makes use of mutation operators and recombination operator EHR to the One Max Tree Problem. The results showed that the AE-EHR obtained better solutions than the EA-RNPG for most instances analyzed.

Keywords

Node-deep-degree encoding, evolutionary algorithms, network design, one-max tree problem

Sumário

Lista de Figuras **9**

Lista de Tabelas **11**

1 Introdução **14**

2 Algoritmos Evolutivos **17**
- 2.1 Operações dos Algoritmos Evolutivos 18
  - 2.1.1 Seleção 19
  - 2.1.2 Mutação 20
  - 2.1.3 Recombinação 21
  - 2.1.4 Critério de Parada 21
- 2.2 Programação Evolutiva 21
- 2.3 Estratégias Evolutivas 22
- 2.4 Algoritmos Genéticos 22

3 Problemas de Projeto de Redes **24**
- 3.1 Árvore Geradora Mínima com Restrição de Grau 25
- 3.2 Árvore Máxima 26
- 3.3 Árvore Geradora de Comunicação Ótima 27

4 Representações **29**
- 4.1 Representações Diretas 30
- 4.2 Representações Indiretas 31
- 4.3 Mistas 32
  - 4.3.1 Nó-Profundidade 32
  - 4.3.2 Nó-Profundidade-Grau 34

5 Proposta de Algoritmo Evolutivo baseado na RNPG para PPR **36**
- 5.1 Operador EHR 37
- 5.2 Alterações do EHR para Árvore única 40
- 5.3 Formulação dos Algoritmos Evolutivos AE-PC e AE-EPC 41

6 Experimentos e Resultados **44**
- 6.1 Topologia Linha 45
  - 6.1.1 População 10 46
  - 6.1.2 População 20 48
  - 6.1.3 População 50 50
  - 6.1.4 População 100 52

6.1.5 Conclusões parciais 53
6.2 Topologia Estrela 54
6.2.1 População 10 55
6.2.2 População 20 57
6.2.3 População 50 59
6.2.4 População 100 61
6.2.5 Conclusões parciais 63
6.3 Topologia Aleatória 63
6.3.1 População 10 64
6.3.2 População 20 66
6.3.3 População 50 67
6.3.4 População 100 69
6.3.5 Conclusões parciais 71
6.4 Considerações finais 71

7 Conclusões **73**

Referências Bibliográficas **76**

Lista de Figuras

3.1 Um grafo representando um problema de projeto de redes e sua solução. 24
3.2 Grafo sua árvore geradora mínima. 25
3.3 Exemplo de árvores com 5 nós T_a e T_{opt}. 27
3.4 Exemplo de grafos de custo, demanda e possíveis soluções para uma OCSTP. 28

4.1 Árvore sobre a qual será aplicada a representação de Número de Prüfer. 31
4.2 Tabela que mostra os nós da árvore em cada estágio do processo. 31
4.3 Grafo e uma de suas árvores. 33
4.4 Representação nó-profundidade da árvore da Figura 4.3. 33

5.1 Histórico de aplicações utilizando o PAO. 37
5.2 Florestas para aplicação do EHR. 39
5.3 Histórico de aplicações das florestas com o ancestral comum. 39
5.4 Histórico de aplicação do subconjunto s sobre o $Ancestral\_comum$. 40

6.1 Exemplo de árvore geradora com topologia linha. 45
6.2 População 10 e N_Π 30. 46
6.3 População 10 e N_Π 50. 46
6.4 População 10 e N_Π 100. 47
6.5 População 20 e N_Π 30. 48
6.6 População 20 e N_Π 50. 48
6.7 População 20 e N_Π 100. 49
6.8 População 50 e N_Π 30. 50
6.9 População 50 e N_Π 50. 50
6.10 População 50 e N_Π 100. 51
6.11 População 100 e N_Π 30. 52
6.12 População 100 e N_Π 50. 52
6.13 População 100 e N_Π 100. 53
6.14 Topologia Estrela. 54
6.15 População 10 e N_Π 30. 55
6.16 População 10 e N_Π 50. 55
6.17 População 10 e N_Π 100. 56
6.18 População 20 e N_Π 30. 57
6.19 População 20 e N_Π 50. 57
6.20 População 20 e N_Π 100. 58
6.21 População 50 e N_Π 30. 59
6.22 População 50 e N_Π 50. 59
6.23 População 50 e N_Π 100. 60

6.24 População 100 e N_Π 30. 61
6.25 População 100 e N_Π 50. 61
6.26 População 100 e N_Π 100. 62
6.27 Topologia Aleatória. 63
6.28 População 10 e N_Π 30. 64
6.29 População 10 e N_Π 50. 64
6.30 População 10 e N_Π 100. 65
6.31 População 20 e N_Π 30. 66
6.32 População 20 e N_Π 50. 66
6.33 População 20 e N_Π 100. 67
6.34 População 50 e N_Π 30. 68
6.35 População 50 e N_Π 50. 68
6.36 População 50 e N_Π 100. 68
6.37 População 100 e N_Π 30. 69
6.38 População 100 e N_Π 50. 70
6.39 População 100 e N_Π 100. 70
6.40 *Fitness* das configurações indicadas: Linha, com população 20 e N_Π 50; Estrela, com população 10 e N_Π 30; Aleatória, com população 20 e N_Π 30. 72

Lista de Tabelas

6.1 Parâmetros analisados durante os testes para avaliação de desempenho do EHR na RNPG para o problema de árvore máxima 44

Lista de Abreviaturas e Siglas

AE – Algoritmo evolutivo

AE-EPC – Algoritmo evolutivo (com operadores EHR, PAO e CAO)

AE-PC – Algoritmo evolutivo (com operadores PAO e CAO)

AG – Algoritmo genético

CAO - *Change Ancestor Operator* (Operador Altera Ancestral)

dc-MSTP – Problema da Árvore Geradora Mínima com Restição de Grau

dd-MSTP – Problema da Árvore Geradora Mínima com Restição de diâmetro

EA - *Evolutive Algorithm* (Algoritmo Evolutivo)

EE – Estratégia evolutiva

EHR – *Evolutionary History Recombination* (Operador de recombinação histórico de evolução)

FITNESS – resultado, avaliação ou indicador de adaptabilidade

$Matriz\Pi$ – Estrutura de dados utilizada pela RNPG

MEF – Máquina de Estados Finitos

MSTP – *Minimum Spanning Tree Problem* (Problema da Árvore Geradora Mínima)

NDDE – *Node-Depth-Degree Encoding* (Representação nó-profundidade-grau)

NDE – *Node-Depth Encoding* (Representação nó-profundidade)

NDP – *Network Design Projects* (Problema de Projeto de Redes)

NOX – *NDDE OX* (Operador de recombinação de ordem da NDDE)

NP – *Non-Deterministic Polynomial time* (Tempo polinomial não determinístico)

N_Π – Tamanho da $matriz\Pi$

PG – Programação Genética

OCSTP – Problema de árvore geradora de comunicação ótima

OMTP – Problema de uma árvore máxima

PAO – *Preserve Ancestor Operator* (Operador Preserva Ancestral)

PE – Programação Evolutiva

PMX – *Partial Mapped Crossover* (Operador de recombinação parcialmente mapeado)

PPRs – Problemas de Projeto de Redes

RNP – Representação nó-profundidade

RNPG – Representação nó-profundidade-grau

CAPÍTULO 1

Introdução

Os Algoritmos Evolutivos (AEs) são técnicas computacionais inspiradas na Teoria da Evolução Natural das espécies, que podem ser aplicadas a problemas de diversas áreas do mundo moderno [41]. Os AEs têm mostrado ser ferramentas eficientes quando aplicados a diversos problemas de otimização, principalmente para problemas com espaços de solução não-contínuos e não-convexos, em que existe dificuldade para a utilização de técnicas convencionais de otimização, principalmente quando não se tem acesso à função a ser otimizada [11, 42]. Dentre esses problemas, os problemas de projeto de redes são representantes relevantes de aplicações no mundo real.

Os Problemas de Projeto de Rede (PPRs) envolvem uma série de aplicações do mundo real tais como, redes de comunicação, controle de tráfego, redes de distribuição, entre outras que podem ser representadas por grafos. Os PPRs também possuem algumas formulações teóricas utilizadas para validar os algoritmos propostos nessa área. Dentre esses destacamos o Problema da Árvore Máxima (OMTP) [37].

Os AEs com representações convencionais não têm apresentado resultados satisfatórios com um tempo computacional aceitável quando aplicados aos problemas de projeto de redes, em geral, de larga-escala [42]. Grande parte do esforço computacional é despendido na decodificação da solução e na sua validação e correção de infactibilidades, tais como, ciclos, árvore desconexas, arestas não pertencentes ao problema, entre outros. Além disso, quanto maior o problema, maior o esforço computacional para manter tal representação. Por esse motivo, diversas pesquisas têm sido dedicadas ao estudo e desenvolvimento de estruturas de dados especiais, denominadas representações, para codificar os PPRs em grafos nos AEs [33, 24, 51, 47, 37, 15, 26, 31].

Dentre as representações propostas recentemente, a representação Nó-Profundidade-Grau (RNPG) tem apresentado resultados com significativo avanço em relação à eficiência computacional, principalmente quando aplicada a problemas de larga-escala [9, 53]. A RNPG tem como base a representação nó-profundidade (RNP) [8] e sua eficiência computacional é devida, principalmente, aos operadores de mutação PAO (*Preserve Ancestor Operator*) e CAO (*Change Ancestor Operator*) da RNP e na proposta do operador de recombinação EHR (*Evolutionary History Recombination* ou

Operador de recombinação histórico de evoluç˜ão) [53]. A RNPG possui relativamente baixa complexidade de tempo, tanto de seus operadores quanto dos métodos de codificação e decodificação da mesma. Dada essa propriedade, AEs que usam a RNPG podem ser aplicados a PPRs de larga-escala com tempo computacional aceitável, inclusive para certas aplicações que exigem respostas em tempo real. Além disso, a RNPG pode ser aplicada não somente a problemas representados por grafos completos, mas também por grafos não-completos, esparsos, Euclidianos e com soluções na forma de árvores ou florestas geradoras [53].

Trabalhos anteriores aplicaram AEs com a RNPG para os problemas de formulação teórica de PPRs com os operadores de mutação PAO e CAO e os operadores de recombinação NOX e NPBX. Esses trabalhos obtiveram resultados satisfatórios na solução dos dc-MSTP e OCSTP. No entanto, o operador EHR foi aplicado somente para o problema de reconfiguração de sistemas de distribuição de energia elétrica, que tem a sua solução representada por florestas geradoras, ou seja, composta por mais de uma árvore [53]. Como existem diversos PPRs que possuem a sua solução modelada por uma única árvore geradora, e não por várias, é importante possibilitar a utilização do operador EHR para esses problemas, visto que para os casos com diversas árvores sua eficiência já foi comprovada.

Assim, este trabalho propõem a adequação do operador EHR da RNPG para PPRs teóricos, nos quais a solução é representada por uma única árvore geradora. O uso de operadores de recombinação é importante para reduzir o tempo de convergência dos AEs. Com o objetivo de analisar o desempenho do operador EHR é utilizado o PPR teórico da OMTP. Apesar desse problema ser fácil para a montagem dos blocos construtivos (*building blocks*), a solução é composta por um único bloco, que aumenta de tamanho conforme aumenta o tamanho do problema, tornando o problema complexo para AEs, principalmente os baseados em recombinação [42]. Assim, propõem-se também o desenvolvimento de dois algoritmos evolutivos: o AE-RNPG, que utiliza a RNPG somente com os operadores de mutação e o AE-EHR que se diferencia pelo uso do operador EHR. Durante os ensaios, foram avaliados diferentes valores de parâmetros de configuração tanto do AE quanto específicos da RNPG e do operador EHR de forma a identificar as opções que refletem em um melhor desempenho da proposta desenvolvida. Além disso, foram utilizados como casos de teste para o PPR da OMTP diferentes árvores de referência para cada uma das topologias indicadas na formulação do problema com número de vértices significativamente diferentes. Todas as configurações de parâmetros foram aplicadas no dois algoritmos propostos e os resultados mostraram que o desempenho do AE-EHR é superior na maioria dos experimentos. No decorrer do trabalho serão detalhados o problema, a proposta, os tipos e variações dos testes aplicados, bem como os resultados que mostram que o uso do operador de recombinação EHR

mostra-se também como uma solução interessante para a categoria dos PPRs teóricos.

O trabalho está estruturado da seguinte forma. O capítulo 2 apresenta uma introdução aos AEs. O capítulo 3 discute os principais PPRs que são alvo de estudo deste trabalho. O capítulo 4 aborda as representações de PPRs para AEs. O capítulo 5 propõe os métodos propostos neste trabalho. O capítulo 6 apresenta as configurações dos métodos utilizados nos experimentos realizados e os resultados obtidos. Por fim, o Capítulo 7 sintetiza as principais conclusões e apresenta propostas de trabalhos futuros.

CAPÍTULO 2

Algoritmos Evolutivos

Inspirados na teoria da evolução das espécies proposta por Charles Darwin [4], os Algoritmos Evolutivos (AE) são técnicas computacionais que por meio de variações de soluções existentes podem obter outras mais eficientes. As áreas de Biologia e Ciências Naturais têm utilizado os AEs como ferramentas para a simulação de evolução de populações. Em computação e engenharias, esses algoritmos são utilizados para buscar a solução útil em sistemas de otimização complexos.

Neste último caso, os AEs são modelados conforme descrito a seguir. Uma solução candidata a solução de um problema é achamada de indivíduo. Um conjunto de indivíduos é denominado população. Cada interação do algoritmo é chamada de geração, em que se constrói uma nova população. Novas soluções são obtidas por meio de cruzamentos ou recombinações de indivíduos que combinam as características de duas ou mais das soluções já existentes (reprodução sexuada), e de mutações que obtém novas soluções a partir de uma ou mais alterações em um elemento já existente (reprodução assexuada). Para simular o processo da seleção natural, é calculado o *fitness* dos indivíduos, que é um indicador de qualidade utilizado para classificar os indivíduos. Por fim, selecionam-se os melhores indivíduos (soluções), dentre os da população geradora (pais) e/ou os da nova população obtida (filhos), com base no seu *fitness*, para formar a nova população. O processo de seleção, também é utilizado para escolher os indivíduos que produzirão descendentes para as próximas gerações e prioriza aqueles com maior *fitness*, ou seja, os mais adaptados. Esse processo é repetido diversas vezes até que se alcance uma solução (indivíduo) que seja considerada satisfatória ou até que se atinja algum critério de parada estabelecido.

Na Biologia, as características dos indivíduos ficam armazenadas nos cromossomos, que são estruturas compostas por vários genes. As informações armazenadas nos genes são os alelos. De forma similar, na representação computacional, os cromossomos podem ser modelados como vetores, onde cada posição equivale a um gene. O tipo de dados usado em cada gene deve ser compatível com a quantidade de variações dos alelos. Os processos de evolução natural são simulados por aplicações de funções que recebem populações e, após a realização de mutações ou recombinações, retornam novos indivíduos

ou populações.

Por volta de 1930, já existiam pesquisas abordando AE. Eles eram empregados na exploração de múltiplos picos de funções objetivo. Na segunda metade do século XX, a disponibilidade de computadores com maior eficiência permitiu intensificar os estudos nesta área [6]. Com o crescimento das pesquisas, por volta de 1970 os AE foram divididos em três linhas de pesquisas: programação evolutiva, estratégias evolutivas e algoritmos genéticos. Na década seguinte, surgiu a programação genética, possibilitando a criação de programas de computadores [28]. A Seção 2.1 descreve as principais operações que compõem os algoritmos evolutivos. As Seções 2.2, 2.3 e 2.4 abordam os principais tipos de algoritmos evolutivos.

Os AEs têm sido principalmente usados para solucionar problemas de otimização global [6, 7], que se caracterizam por formulações com função objetivo simples ou múltiplas, multimodais ou descontínuas, que geram resposta para cada entrada, isto é, em uma abordagem te otimização tipo caixa-preta. Seu principal diferencial é a busca simultânea em vários pontos do espaço de soluções, não se restringindo a um único caminho de exploração no espaço de busca. Esta técnica também pode ser aplicada a problemas de classificação, redes neurais, elaboração de código LISP, entre outras [21].

O algoritmo 2.1 apresenta o pseudocódigo de um AE [12].

Algoritmo 2.1: Pseudocódigo de um AE típico.

Entrada: Parâmetros típicos
Saída: População final de soluções

```
1 INICIALIZA população com soluções candidatas aleatórias;
2 AVALIA cada solução candidata
3 repita
4     SELECIONA pais
5     RECOMBINA pares de pais
6     MUTA os indivíduos descendentes resultantes
7     AVALIA novas candidatas (indivíduos após a mutação)
8     SELECIONA indivíduos para a nova geração
9 até CONDIÇÃO DE PARADA satisfeita
```

2.1 Operações dos Algoritmos Evolutivos

As principais etapas de um AE que podem ser organizadas como segue: seleção, avaliação, recombinação, mutação e (critério de) parada.

2.1.1 Seleção

A operação de seleção é responsável por realizar o processo de seleção natural da teoria da evolução. Essa operação é, geralmente, utilizada para a escolha de soluções para reprodução (pais). Além disso, pode ser utilizada para definir os indivíduos sobreviventes da próxima geração. As principais formas de realizar a seleção são descritas a seguir.

1. Seleção uniforme ou neutra: como o próprio nome diz, não há prioridade ou preferência no critério de escolha, logo quaisquer indivíduos podem ser selecionados independente de seu *fitness*. Este mecanismo é comumente usado para escolher indivíduos que irão gerar descendentes;
2. Proporcional ao *fitness*: este mecanismo prioriza quem tem o maior *fitness*. A probabilidade é calculada a partir do valor do *fitness* de cada indivíduo e dividido pela soma dos demais *fitness*. Como o valor da soma dos *fitness* é alterado a cada novo indivíduo gerado ou eliminado, torna-se um critério dinamicamente atualizado. Como no processo evolutivo os primeiros indivíduos são menos parecidos e no final do processo são mais homogêneos, este método apresenta um comportamento mais elitista no início e mais uniforme no final. Ele pode ser usado para escolher tanto os indivíduos que produzirão descendentes quanto os que irão sobreviver;
3. Classificação linear e torneio de dois: da mesma forma que em um campeonato de futebol, pares de indivíduos são colocados em comparação (torneio) com base no valor de *fitness*. Ao final, tem-se uma classificação linear por ordem decrescente de resultados. Em [6], prova-se esta característica. Este processo pode ser utilizado para seleção dos pais e dos indivíduos para sobreviver;
4. Classificação não-linear e torneio de $K > 2$: é semelhante ao torneio de dois, porém os torneios são feitos com K indivíduos, sendo $K > 2$. Isso gera uma maior complexidade da comparação e seu resultado final é uma lista não-linear. Em [6], também prova-se esta característica. Assim, como o método de torneio de dois, esse método pode ser utilizado para selecionar os indivíduos que produzirão descendentes e os indivíduos para a próxima geração;
5. Truncamento: este método seleciona os T indivíduos de maior *fitness* de uma população, desprezando os demais. Tem sido mais utilizado para selecionar os indivíduos que sobreviverão para a próxima geração.

Ainda considerando [6], pode-se dizer que em geral se utiliza a seleção linear para escolher os indivíduos para a reprodução, enquanto que, para escolher os indivíduos que sobreviverão, a seleção por truncamento pode ser mais adequada. Além disso, os métodos baseados em torneio têm sido mais utilizados para escolher os indivíduos para a reprodução.

2.1.2 Mutação

Em comparação com a natureza, a mutação é um processo de reprodução assexuada, no qual um indivíduo pai gera um filho igual a si. Em seguida, o filho sofre uma alteração. Esse processo pode modificar um ou mais genes do seu cromossomo. A operação de mutação é aplicada de acordo com uma taxa probabilística que, em geral, indica probabilidade de alteração de cada gene do cromossomo. Como essas alterações podem gerar indivíduos com *fitness* pior que o dos seus ancestrais, deve-se utilizar uma taxa de mutação pequena.

O operador de mutação depende do tipo de representação utilizada no cromossomo. O princípio da mutação é escolher, por seleção aleatória com distribuição uniforme, dada a taxa de mutação, alguns genes que serão alterados e, em seguida, aplicar a modificação dos valores. Um tipo simples de representação de cromossomo é a binária. Nela o cromossomo é um vetor binário e cada gene pode ser representado por 0 ou 1. Desta forma, pode-se fazer mutação apenas invertendo os valores dos seus genes [2] ou escolhendo um trecho do cromossomo e trocando-os de lugar, no mesmo cromossomo. Para genes não binários, outras regras podem ser elaboradas para a mudança dos valores. Outro ponto que merece destaque é que a escolha dos genes a serem modificados, que pode ser feita em blocos, ou seja, escolhe-se 2, 3 ou k trechos do cromossomo e faz-se uma permutação entre eles; porém é preciso verificar se os genes contidos nos blocos podem ser alterados (possuem alelos compatíveis) e se as novas soluções são factíveis permitidos para aqueles genes. Verificações de factibilidade em geral consomem muito tempo de computação para instâncias de larga-escala de um problema com proporção relativamente pequena de soluções factíveis no espaço de busca. Para as representações mais comuns já existem vários operadores de mutação definidos na literatura [6, 13, 30].

Em resumo, tanto a escolha dos genes a serem alterados como a forma de alterar os seus valores devem considerar as peculiaridades do problema a ser abordado. Como exemplo, o problema do Caixeiro Viajante exige que sempre se tenha um Circuito Hamiltoniano, ou seja, deve-se percorrer todos os vértices do grafo sem repetições.

Outro ponto que merece atenção é a intensidade das modificações, pois ela determina se o novo indivíduo será parecido ou não com o seu antecessor. Isso pode influenciar na convergência (ou não) para ótimos locais e, também, no tempo de resposta do algoritmo. Em geral, para cromossomos binários, opta-se por variação intensa (complemento) no valor do gene e um intervalo de genes não muito grande. Para problemas com ponto flutuante, opta-se por mudanças mais leves nos valores dos genes, porém em um trecho maior. Estas duas opções possuem forte relação com o tamanho do espaço de busca a ser explorado [6].

2.1.3 Recombinação

Nos AE, a recombinação é a técnica pela qual se obtém novos indivíduos pela troca informações de dois ou mais indivíduos existentes em uma população [6]. Esta troca pode ser feita em um único ou mais trechos de cromossomo de uma vez. As recombinações em cromossomos de tamanho fixo são também conhecidas por *Crossover* [6].

O caso mais simples de crossover é o de 1-ponto. Nele, escolhe-se um ponto do cromossomo (posição do vetor de genes) e faz-se a quebra do cromossomo em dois pontos. Em seguida, faz-se a quebra de outro cromossomo na mesma posição. Por fim, faz-se a combinação dos quatro trechos gerados pelas quebras, produzindo novos cromossomos filhos.

Além do *crossover* de 1-ponto, existem: o *crossover* de dois pontos e o multipontos. Esses operadores atuam de forma semelhante ao anterior, porém quebrando o cromossomo em três ou mais trechos. É importante ressaltar que um gene armazena uma característica do indivíduo, porém, cada característica pode ter uma de várias opções de valores. Desta forma, a estrutura de dados utilizada necessita comportar tais variações. Nos casos de genes representados por ponto flutuante, algumas técnicas utilizam a média aritmética dos genes dos pais, a média ponderada, a média geométrica, entre outras [6].

2.1.4 Critério de Parada

Os algoritmos evolutivos são executados repetidamente buscando uma solução com melhor *fitness*. Dependendo do problema, da implementação do algoritmo e dos parâmetros de entrada, o AE pode entrar em uma busca que requeira tempo computacional que não se dispõe. Mesmo que aplicação esteja gerando resultados melhores, se as melhorias forem muito discretas, pode ser que uma solução satisfatória não seja alcançada em tempo aceitável. Para lidar com isso, pode-se usar dois critérios de parada. O primeiro é o cumprimento do objetivo proposto. O segundo, é limitar o número de tentativas sem evoluções nas soluções.

2.2 Programação Evolutiva

Em 1962, com o objetivo de utilizar os conceitos de evolução em projetos de inteligência artificial, Fogel [14] desenvolveu os conceitos da programação evolutiva (PE). Nessa técnica, os indivíduos equivalem a uma máquina de estados finitos (MEF) que recebem uma sequência de símbolos e comparam o resultado obtido com uma sequência "satisfatória". O resultado desse cálculo é a função de aptidão do indivíduo, *fitness* [14].

Na PE, a reprodução dos indivíduos ocorre apenas por meio do operador de mutação, o que na natureza é conhecido por reprodução assexuada. Sendo assim, se a população começa com N elementos, cada um deles produz um filho. Após obter a nova população de filhos, o processo de seleção elimina os N piores indivíduos entre as duas populações $(N+N)$.

O problema dessa técnica é que existe a possibilidade de os resultados convergirem para pontos de ótimos locais, trabalhando com pouca ou nenhuma evolução das soluções encontradas [6].

2.3 Estratégias Evolutivas

As Estratégias Evolutivas (EE) são uma evolução das PEs . Nelas, os cromossomos são formados por dois vetores, sendo que um armazena os valores dos alelos para cada gene e o segundo vetor o desvio padrão desses alelos [40, 48].

Nessa técnica, ao aplicar o operador de mutação, é utilizada a distribuição de probabilidade Gaussiana, com média zero e o desvio padrão correspondente a cada gene para modificar o valor do mesmo no pai e assim produzir uma nova solução. O primeiro modelo proposto para as EEs é da forma (1+1)-EE desenvolvido por Rechenberg e Schwefel, na década de 1960, e nesse caso, a população é composta por um único indivíduo que a cada geração produz somente um novo indivíduo. Ambos competem diretamente pela sobrevivência para a próxima geração. Nesse modelo, também existe o problema de convergência em ótimos locais e aumento do tempo de convergência, das PEs [40, 48].

Visto que o modelo (1+1)-EE possui problemas, foram propostas algumas variações do mesmo, tais como, a (μ,λ)-EE e ($\mu+\lambda$)-EE. No primeiro caso não há competição entre pais e filhos, ou seja, os filhos substituem os pais, e, no segundo, sobrevivem apenas os μ melhores indivíduos entre as duas populações [6].

2.4 Algoritmos Genéticos

Na década de 70, Holland [19] e seus colaboradores propuseram os Algoritmos Genéticos (AGs). Essa classe de algoritmos evolutivos tem como inspiração, não somente a reprodução assexuada utilizada pelas PEs e EEs, mas, também, a reprodução sexuada [19]. Outra diferença em relação as técnicas anteriores é que nos AGs é utilizada a representação binária para codificar as soluções nos cromossomos [6].

O processo de reprodução sexuada é simulada nos AGs por meio de operadores de recombinação denominados crossover. Os principais tipos de crossover são: o de um ponto, o de dois pontos e multipontos (ou uniforme) [16]. Normalmente, nos AGs, a

operação de mutação é aplicada aos indivíduos obtidos na aplicação do crossover e não diretamente sobre os indivíduos pais.

Com essas modificações, os resultados não seguem a mesma tendência de convergir para ótimos locais, como nos métodos anteriores. Segundo Goldberg, os AGs conseguem gerar resultados satisfatórios mesmo que os parâmetros iniciais não sejam muito eficientes [18, 16].

Nos AGs um grupo de pais é selecionado para produzir os descendentes por meio do crossover e da mutação. Obtidos os novos indivíduos esses substituem inteiramente a geração anterior e constituem a nova população de indivíduos [19, 18]. Assim, como nos demais algoritmos o processo é repetido até que o critério de parada seja atingido.

Os AGs são a base para o desenvolvimento deste trabalho.

CAPÍTULO 3

Problemas de Projeto de Redes

Diversos problemas do mundo moderno podem ser representados por PPR. Alguns exemplos são árvores filogenéticas, problemas de logísticas, redes de comunicação, entre outros. Esses problemas podem ser modelados por grafos não-orientados com pesos associados as arestas. E quanto maior o grafo, ou seja, quanto maior o número de vértices (elementos do sistema) maior a dificuldade para se encontrar soluções satisfatórias.

A solução de um PPR, representado por grafos, consiste em buscar um subgrafo que obedeça as restrições do problema e otimize os critérios utilizados para avaliar a rede. Em geral, esse subgrafo consiste em uma árvore geradora do grafo sujeita a restrições.

Por exemplo, a Figura 3.1(a) apresenta um grafo que representa um PPR. A Figura 3.1(b) indica, com arestas em negrito, uma árvore geradora para o mesmo grafo, como solução para o problema apresentado.

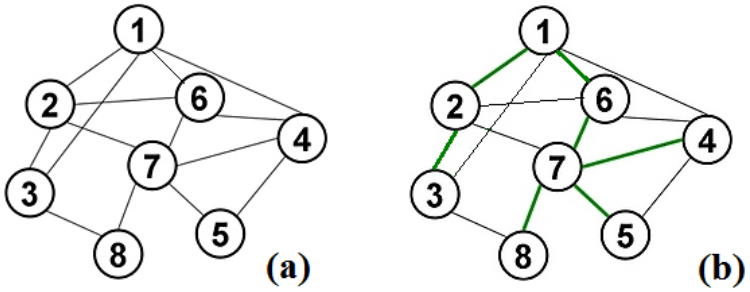


Figura 3.1: *Um grafo representando um problema de projeto de redes e sua solução.*

O PPR clássico é um problema $NP-Completo$ [20], ou seja, resolvível em tempo polinomial. Matematicamente um PPR genérico pode ser definido como segue. Dado um grafo $G = (V, E)$, um conjunto de pesos $L \in \mathbb{N}$, e B o orçamento da rede, o PPR consiste em encontrar o subgrafo $H = (V, E_H)$ de G com pesos tais que $\sum_{(i,j)\in E_H} L(i,j) \leq B$.

Dentre os diversos problemas estudados da área de projeto de redes destacam-se a árvore geradora de comunicação ótima [43, 31], dc-MSTP [38], dd-MSTP [49, 23, 1] e roteamento de veículos [55].

Diversas soluções tem sido propostas para esses problemas tendo como base os algoritmos clássicos de árvore geradora mínima de Prim e Kruskal. No entanto, quando aplicados para redes de larga-escala esses algoritmos podem obter soluçõesq que não são adequados para vários PPRs.

As Seções a seguir apresentam alguns dos principais problemas de projeto de redes utilizados na literatura como referência nas pesquisas dessa área.

3.1 Árvore Geradora Mínima com Restrição de Grau

O problema da árvore geradora mínima com restrição de grau (dc-MSTP, Degree-Constrained Minimum Spanning Tree Problem) é definido para um grafo com pesos positivos associados as arestas. A solução do problema consiste em encontrar a árvore gerador mínima, com restrição na quantidade de arestas que cada vértice possui, ou seja, no grau do mesmo [27]. Uma árvore geradora mínima é uma árvore que contem todos os vértices de um grafo, tal que a soma dos pesos das arestas seja o menor possível [29]. Os pesos das arestas são valores que podem indicar distância, fluxo, atrasos, dificuldade, entre outras características relacionadas ao problema aplicado.

A Figura 3.2 apresenta o exemplo de um grafo com pesos nas arestas e a sua árvore geradora mínima representada por meio das arestas em negrito que possui o peso total igual a 25, ou seja, 1+12+6+2+4 = 25.

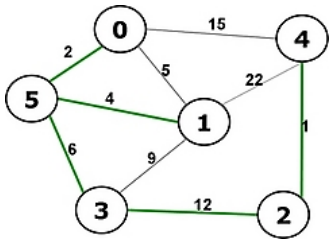


Figura 3.2: *Grafo sua árvore geradora mínima.*

No mesmo exemplo, na Figura 3.2, dc-MSTP 2 teria o peso total 26, ou seja, 1+12+6+2+5 = 26. Nesse caso, o vértice 5 teria apenas 2 arestas, tendo uma de suas arestas removidas e o vértice 0 passaria a ter duas arestas, ligando-se ao vértice 1, que no caso da árvore gerador mínima era ligado ao vértice 5.

Formalmente, o dc-MSTP pode ser definido como segue. Dado $G = (V, E)$ um grafo não-orientado, e W a matriz de pesos associados as arestas e $d(v_i)$ 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:

$$min_T \sum_{(i,j)\in T} w_{v_i,v_j} \tag{3-1}$$

sujeito a restrição

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

3.2 Árvore Máxima

O Problema da Árvore Máxima é um clássico problema de otimização, no qual se busca identificar a distância entre uma árvore geradora escolhida aleatoriamente (ou não) T_a e uma árvore geradora reconhecida como uma solução ótima T_{opt}. Por sua vez, a distância entre duas árvores geradoras $d_{opt,a}$ é a quantidade de arestas que a árvore ótima T_{opt} possui e que não estão contidas na outra árvore T_a. A Equação 3-2 calcula a distância entre as duas árvores.

$$d_{opt,a} = \frac{1}{2} \sum_{i=1}^{n-1} \sum_{j=0}^{i-1} |l_{ij}^{opt} - l_{ij}^{a}|. \tag{3-2}$$

Na equação 3-2, l_{ij}^{opt} é 1 se existe uma aresta que conecta o vértice i ao vértice j na árvore T_{opt} e 0 caso contrário; l_{ij}^{a}, segue o mesmo princípio; n representa o número de vértices. A distância entre as duas árvores é baseada nos conceitos de distância de Hamming e $d_{opt,a} \in \{0, 1, ..., n-1\}$ [47].

A topologia das árvores pode ser estrela, lista ou qualquer outra estrutura que contenha n vértices,, denominada aleatória [47]. A topologia estrela é caracterizada por possuir um vértice central, de grau $n-1$ com todos os demais vértices ligados a ele e com grau 1. Nesse tipo de rede se houver uma falha no vértice central todos os vértices deixam de ser atendidos. Na topologia do tipo linha tem-se dois vértices com grau 1, extremos da rede, e os $n-2$ vértices restantes com grau 2. Quando ocorre uma falha em um dos vértices internos a rede fica dividida em duas sub-redes nessa topologia. Já a topologia não possui nenhum característica específica para o grau dos vértices. Nessa topologia o grau pode variar de $1 \ldots (n-2)$.

O OMTP pode ser visto como problema de minimização ou maximização. No primeiro caso, o fitness f_a^{min} é igual a distância para solução ótima ($d_{a,opt}$), ou seja, se um indivíduo T_a tem fitness igual 2, então a árvore geradora de T_{opt} tem apenas 2 arestas que não existem em T_a. Se o fitness é 0, elas são iguais ($T_a = T_{opt}$).

Definido como problema de maximização, o fitness f_a^{max} de uma árvore T_a é definido pelo número de arestas que ela tem em comum com árvore ótima T_{opt}, ou seja, $f_a^{max} = n - 1 - d_{a,opt}$. Considere como exemplo a Figura 3.3, para o problema de maximizar o fitness é 3, pois as árvores possuem 5 vértices e 3 arestas em comum. Logo, $f_a^{max} = 5 - 1 - 1 => f_a^{max} = 3$.

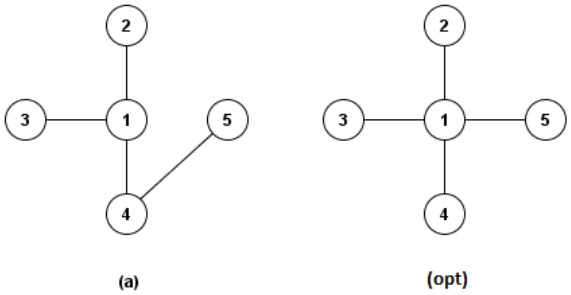


Figura 3.3: *Exemplo de árvores com 5 nós T_a e T_{opt}.*

Com base na teoria de esquemas e blocos construtivos de AE o problema da árvore máxima é considerado um problema fácil. Essa afirmação tem como fundamento que qualquer esquema que compreenda a solução ótima possui fitness médio superior ao dos esquemas que não possuem tal solução [42]. No entanto, na prática não se verifica essa afirmação ao fazer uso de algoritmos evolutivos como método de busca para a solução do problema [42].

O OMTP é similar ao problema um máximo (*one-max*) para representações binárias [42]. Segundo [17], o problema um máximo é fácil para ser resolvido por algoritmos evolutivos baseados em mutações, porém, são, por algum motivo, difíceis para os algoritmos baseados em recombinação. Como os problemas são similares a afirmativa também é valida para o problema da árvore máxima.

3.3 Árvore Geradora de Comunicação Ótima

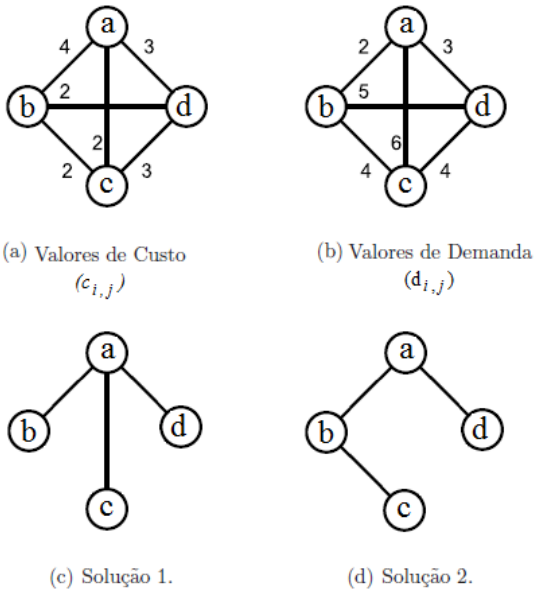
O problema da Árvore Geradora de Comunicação Ótima (OCSTP, *Optimal Communication Spanning Tree Problem*) é um caso especial de problema de árvore geradora mínima, o qual contempla os custos e as demandas de comunicação entre cada par de vértices do grafo. Para esse problema é associado além do peso as arestas, que representa o custo de comunicação, uma demanda de comunicação para cada par de vértices que deve ser atendida. As soluções desse problema são avaliadas multiplicando-se a demanda de comunicação pelo custo de transmissão entre cada par de vértices. O objetivo desse problema é encontrar a árvore geradora que minimize o custo total de comunicação [31]. A Equação 3-3 define o cálculo do custo para o OCSTP.

$$D(T) = \sum_{s,d \in T} p_{s,d} d_{s,d}, \tag{3-3}$$

em que T é uma árvore geradora, $d_{s,d}$ é a demanda entre os vértices s e d de T e

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

em que c_{v_i,v_j} é um elemento da matriz $C_{n,n}$ de custos, 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 $p_{s,d}$ [31].



(a) Valores de Custo ($c_{i,j}$)

(b) Valores de Demanda ($d_{i,j}$)

(c) Solução 1.

(d) Solução 2.

Figura 3.4: *Exemplo de grafos de custo, demanda e possíveis soluções para uma OCSTP.*

A Figura 3.4 apresenta um exemplo do OCSTP. As Figuras 3.4(a) e 3.4(b) representam, respectivamente, os grafos de custo e de demanda. As Figuras 3.4(c) e 3.4(d), representam duas possíveis soluções para o problema. Conforme a Equação 3-3, o custo de comunicação do exemplo proposto é dado por:

$$\mathrm{D} = c_{a,b} * d_{a,b} + c_{a,c} * d_{a,c} + c_{a,d} * d_{a,d} + c_{b,c} * d_{b,c} + c_{b,d} * d_{b,d} + c_{c,d} * d_{c,d}.$$

Percorrendo o grafo 3.4(c), obtém-se o seguinte custo para a solução:

$$\mathrm{D} = 4*2+2*6+3*3+6*4+7*5+5*4 = 108.$$

O custo da solução em 3.4(d), é dado por:

$$\mathrm{D} = 4*2+6*6+3*3+2*4+7*5+9*4 = 130.$$

Para encontrar a solução ótima, será necessário gerar todas as árvores geradoras, calcular o resultado de cada uma e depois identificar o menor entre eles. Segundo [43, 50], a OCSTP é um problema NP-Difícil e é mais complexo que os outros problemas de árvore geradora com restrição.

CAPÍTULO 4

Representações

Os PPRs são, na maioria dos casos, representados por meio de grafos, e as soluções dos mesmos são uma árvore geradora desse grafo. Encontrar representações adequadas para PPRs e que consumam menor esforço de busca têm sido o objetivo de diversas pesquisas na área de algoritmos evolutivos [31, 51, 35, 26, 36, 47, 22, 15, 44].

A escolha da representação é muito importante para o desempenho dos algoritmos evolutivos. O uso de uma representação inadequada pode fazer com que o algoritmo não encontre resultados ou até mesmo obtenha soluções infactiveis para o problema. Em alguns casos, um algoritmo evolutivo pode ter o mesmo comportamento de métodos de buscas aleatórias [42]. Uma representação adequada para AE aplicados a PPRs deve possuir algumas características que são descritas a seguir.

1. Espaço: a memória necessária para representar uma árvore deve ser o mínimo necessário;
2. Tempo: a complexidade de tempo para avaliar, recombinar e mutar as soluções deve ser baixa;
3. Factibilidade: todas as codificações, principalmente as obtidas por recombinação e mutação, devem representar soluções factíveis do problema;
4. Cobertura: a representação deve ser capaz todas as soluções presentes no espaço de busca;
5. Tendência: todas as soluções devem possuir a mesma probabilidade de serem representadas;
6. Localidade: pequenas alterações na codificação, ou seja, no genótipo, devem resultar em pequenas alterações na árvore representada, em outras palavras no fenótipo;
7. Hereditariedade: as operações de recombinação devem preservar a maioria das arestas dos pais nos filhos;
8. Restrições: a representações devem possibilitar a inclusão de restrições dos problemas sem a necessidade de grandes modificações;
9. Hibridismo: os operadores da representação devem possibilitar a inserção de heurísticas do problema;

Além dessas características espera-se que uma representação adequada de PPRs possua uma função de codificação bijetora, ou seja, cada codificação representa uma única árvore e cada árvore deve possuir uma cromossomo único. Quando a representação possui essa característica, diz-se que ela não possui redundância [42].

As representações para PPRs em algoritmos evolutivos ser divididas em três categorias: diretas, indiretas ou mistas. As Seções 4.1, 4.2 e 4.3 abordam as características de cada categoria e as principais representações para cada uma delas.

4.1 Representações Diretas

As representações diretas utilizam o próprio conjunto de arestas para representar as árvores. Além disso, aplicam os operadores de mutação e recombinação diretamente sobre o conjunto de arestas do grafo [42]. Assim, essas representações não necessitam de um tratamento ou conversão de modelo antes da aplicação dos operadores. Essa manipulação de arestas exige o desenvolvimento de operadores de reprodução específicos [42]. Os principais exemplos de representações diretas são: Conjunto de Arestas [35] e NetDir [42].

A representação conjunto de arestas utiliza um vetor ou uma tabela de dispersão (*hash*) para montar o cromossomo com o conjunto de arestas. As entradas dos cromossomos são os pares de vértices que que indicam uma aresta [35]. Quando é utilizada a tabela de dispersão, as operações de inserção, remoção e busca de arestas de um vértice possuem tempo constante. As duas formas de codificação demandam tempo na ordem de $O(n)$ para executar as operações de reprodução e avaliação das soluções, sendo n o número de vértices do grafo [39].

Em [35, 38] são propostas três formas de inicializar a população de indivíduos do AE, KruskalRst, PrimRst e RandomWalk. Os dois primeiros tem como princípio os operadores de Kruskal e Prim para construção de árvores geradoras em grafos, respectivamente. O terceiro operador utiliza o processo de caminhos aleatórios para construir as árvores.

O operador de recombinação proposto por [35] começa inserindo no filho as arestas em comum entre os dois pais. O segundo passo do algoritmo seleciona aleatoriamente arestas que pertencem a somente um dos pais e insere no filho até completar as $n-1$ arestas necessárias. A cada aresta escolhida é verificado se não ocorre a formação de ciclos e somente nesse caso a aresta é inserida na solução.

A mutação em conjunto de arestas consiste em inserir uma aresta aleatória de forma a obter um ciclo. No segundo passo, escolhe-se uma outra aresta do ciclo formado e remove para que a solução volte a representar uma árvore [35].

4.2 Representações Indiretas

As representações indiretas codificam uma árvore para um vetor de caracteres, normalmente binário ou de números inteiros. Com esse processo de codificação, essas representações permitem a utilização dos operadores de mutação e recombinação definidos para as codificações convencionais de algoritmos evolutivos [46, 45]. Os principais exemplos de representações indiretas são: Número de Prüfer [56], Vetor de Características [31, 5], Tendência de Ligação e Nó [32], Chaves Aleatórias para Redes [47], Blob Code [24], Dandelion Code [54].

A representação mais conhecida dessa classe é o Número de Prüfer. Nessa codificação cada árvore é representada por uma *string* de $n-2$ dígitos inteiros, onde n é o número de vértices do grafo. O processo de codificação consiste basicamente em duas etapas, como descrito a seguir.

Dado um grafo G e uma árvore $T \subset G$, o processo de codificação inicia excluindo o vértice folha de T com o menor rótulo, em seguida é armazenado em um lista, o valor do seu vértice pai. Esses passos são repetidos até que a árvore tenha apenas dois vértices. A Figura 4.1 mostra uma árvore geradora que será utilizada para exemplificar a codificação por Número de Prüfer. Na Figura 4.2 é apresentado o passo a passo do processo de codificação.

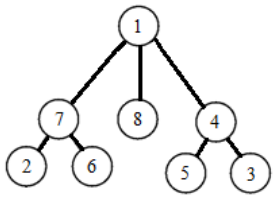


Figura 4.1: *Árvore sobre a qual será aplicada a representação de Número de Prüfer.*

Iteração	Nós da árvore	Excluídos	Pai dos excluídos
0	$\{1,2,3,4,5,6,7,8\}$	$\emptyset$	$\emptyset$
1	$\{1,3,4,5,6,7,8\}$	$\{2\}$	$\{7\}$
2	$\{1,4,5,6,7,8\}$	$\{2,3\}$	$\{7,4\}$
3	$\{1,4,6,7,8\}$	$\{2,3,5\}$	$\{7,4,4\}$
4	$\{1,6,7,8\}$	$\{2,3,5,4\}$	$\{7,4,4,1\}$
5	$\{1,7,8\}$	$\{2,3,5,4,6\}$	$\{7,4,4,1,7\}$
6	$\{1,8\}$	$\{2,3,5,4,6,7\}$	$\{7,4,4,1,7,1\}$

O código Prüfer da árvore acima é (7,4,4,1,7,1)

Figura 4.2: *Tabela que mostra os nós da árvore em cada estágio do processo.*

Em um grafo completo de n vértices existem n^{n-2} árvores distintas. Segundo [34] cada árvore pode ser codificada por uma única *string* de comprimento $n-2$. E cada

string representa uma única árvore [34]. Assim, é dito que a representação de problemas de projeto de redes por meio do Número de Prüfer é não tendenciosa a nenhum tipo de topologia de árvore [42]. Além de não ser tendenciosa a grande vantagem dessa representação é que não existe a necessidade de mecanismos de correção das soluções ao aplicar os operadores usuais de recombinação e mutação, quando o grafo é completo, pois todo cromossomo sempre representa uma árvore. O problema da representação por Número de Prüfer é a baixa localidade das operações de mutação, pois a alteração de um número da *string* de inteiros resulta em grandes mudanças na estrutura da árvore, dada a forma de codificação e decodificação da representação [42].

4.3 Mistas

Como o próprio nome diz, esse grupo de representações compreende características de mapeamento genótipo-fenótipo das representações indiretas e o uso de operadores especiais de reprodução, como as representações diretas. O uso de operadores específicos visa eliminar a necessidade de mecanismos de correção das soluções nos indivíduos resultantes dos operadores de reprodução, o que ocorre com frequência nas representações indiretas. Seus principais exemplos são: Nó-Profundidade [8, 10], Precedentes Diretos [3], Ajuste Adaptativo das Ligações [51], Permutação Baseada em Árvore [57], D-Based [52], Sub-Conjunto de Comprimento Fixo [23], Predecessores ou Codificação Determinante [32] e mais recentemente a representação Nó-Profundidade-Grau [53, 9].

A seguir são apresentadas as representações nó-profundidade e nó-profundidade-grau objeto de estudo deste trabalho.

4.3.1 Nó-Profundidade

A representação Nó-Profundidade (RNP) utiliza os conceitos de vértice e profundidade para representar uma árvore. A representação consiste em mapear uma árvore em um vetor de pares do tipo (vértice,profundidade). Para isso considera-se a árvore como sendo enraizada em um vértice, sendo esse o primeiro par do cromossomo. A profundidade de cada vértice na representação é dado pelo comprimento do caminho do vértice até a raiz da árvore, e a ordem dos vértices no cromossomo é obtida por meio de uma busca em profundidade [10].

As Figuras 4.3 e 4.4 mostram, respectivamente uma árvore, destacada pelas arestas em negrito, a sua codificação na RNP por meio do vetor de pares. Para construir a codificação apresentada na Figura 4.4 considerou-se a árvore enraizada no vértice 1.

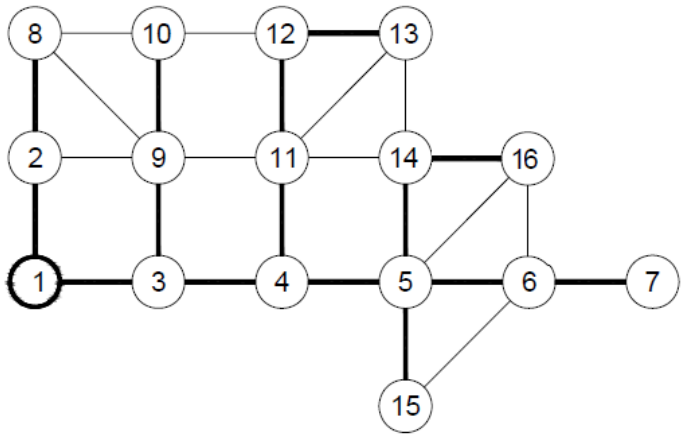


Figura 4.3: *Grafo e uma de suas árvores.*

0	1	2	1	2	3	2	3	4	5	3	4	5	4	5	4
1	2	8	3	9	10	4	11	12	13	5	14	16	6	7	15

Figura 4.4: *Representação nó-profundidade da árvore da Figura 4.3.*

A RNP possui como operadores de reprodução, somente operadores de mutação. No caso, são utilizados dois operadores que possuem funcionamento similar. Ambos fazem a poda de um trecho do cromossomo, que representa uma sub-árvore e reinserem esse trecho em outra parte do cromossomo. Ou seja, é feita a poda de uma sub-árvore em um vértice chamado de p e em seguida o seu enxerto em um vértice chamado de a.

O primeiro operador é conhecido como $Operador1$ e faz a poda e enxerto simples, ou seja, a sub-árvore mantém a sua raiz no vértice de poda. Para isso, o $Operador1$ utiliza somente os vértices p e a. O outro operador, conhecido como $Operador2$, executa modificações mais complexas na árvore do que o $Operador1$. Ao fazer o enxerto da sub-árvore, a mesma passa a possuir uma nova raiz chamada de vértice r que pertence a sub-árvore do vértice p. Assim é necessário um processo de reconstrução do cromossomo da sub-árvore para contemplar a nova raiz [8, 10].

A RNP permite a representação tanto de problemas cuja a solução é composta por uma única quanto de problemas de solução desconexa, ou seja que possui várias árvores. Quando a solução possui várias árvores essa é representa por um conjunto de RNPs, uma para cada árvore da solução, armazenadas em um vetor de ponteiros para as mesmas. Nesse casos os operadores da RNP fazem a transferência de sub-árvores entre as árvores do problema. Em outras palavras, o vértice a onde ocorre o enxerto pertence a uma árvore diferente da árvore onde ocorre a poda [8, 10].

A RNP possui duas estruturas auxiliares: a matriz Π_x e o vetor π [10]. Essas estruturas são utilizadas para acelerar o processo de escolha e determinação da localização

dos vértices p, r e a. O vetor π é utilizado para armazenar o ancestral de cada indivíduos. Para cada indivíduo gerado existe uma posição correspondente no vetor π que contém o índice correspondente ao indivíduo que deu origem a ele. Isso é possível pois cada indivíduo é obtido a partir de um outro por meio de um dos operadores de mutação da RNP. Já a matriz Π_x mantém as informações relativas ao vértice x, e existe uma matriz para cada vértice do grafo. As informações contidas na matriz Π_x são o índice do indivíduo no vetor π, o índice da árvore a qual o vértice pertence, no caso de uma única árvore todos os vértice possuem o mesmo valor, e a posição no vértice na RNP da árvore. Sempre que um vértice tem sua posição alterada na RNP é inserida uma nova coluna na sua matriz para atualizar os seus dados [10].

Os vértices p e a são escolhidos de forma aleatório dentro do grafo. Após a escolha do vértice procura-se na matriz Π_x dele pela posição com base no índice do indivíduo selecionado para aplicar a operação. Se a coluna para o indivíduo existe na matriz as informações são retornados, caso contrário utiliza-se o vetor π para localizar o ancestral do indivíduo, até que seja possível recuperar as informações relativas ao vértice. A não existência da coluna na matriz Π_x para um indivíduo indica que na obtenção do mesmo aquele vértice não foi modificado e por isso deve-se buscar na matriz pelo índice dos ancestrais. De posse das informações contidas na matriz Π_x os vértices escolhidos aleatoriamente são validados para verificar a factibilidade da utilização deles pelos operadores 1 e 2.

4.3.2 Nó-Profundidade-Grau

A representação Nó-Profundidade-Grau (RNPG) é uma melhoria da RNP apresentada na Seção 4.3.1. Essas melhorias consistem na proposta de operadores de recombinação e melhoramentos no $Operador1$ e no $Operador2$, da RNP, de forma a garantir a complexidade de tempo da representação. Os operadores *PAO* (*Preserve Ancestor Operator*) e *CAO* (*Change Ancestor Operator*) são os novos nomes dos operadores de mutação [9].

Além das informações vértice e profundidade (da RNP), na RNPG foi adicionada a informação do grau do vértice na árvore. Assim, uma árvore é codificada por meio de um vetor de triplas de valores (v_i, de_i, deg_i), onde v_i representa o vértice, de_i a profundidade e deg_i o grau do vértice v_i ($i \in 1, 2, ..., n | n$ é a quantidade de vértices da árvore). Assim como na sua antecessora, um dos vértices é escolhido como raiz da árvore terá a profundidade 0. A ordem dos demais vértices no vetor é determinada utilizando algum método de busca em grafos, como, por exemplo, a busca em profundidade [9].

Semelhantes aos operadores da RNP, PAO e CAO obtém novos indivíduos a partir da poda e enxerto de sub-árvores recortadas de uma árvore (origem) e implantadas

em outra árvore (destino), ou na mesma árvore. A modificação em relação a RNP ocorre na forma como são escolhidos os vértices p, a e r que os operadores necessitam. Os novos métodos para escolha dos vértices garante a complexidade de tempo computacional da RNPG, que já era obtida experimentalmente pela RNP [9].

Uma das estratégias da RNPG para garantir a complexidade de tempo é manter as matrizes Π_x e vetor π com o tamanho fixo e proporcional a complexidade de tempo da representação. Isso é importante pois ambas as estruturas são utilizados pelo métodos de escolha dos vértice p, a e r. Assim, quando o vetor π atinge o seu tamanho máximo as estruturas tem seus dados removidos e são preenchidas somente com os dados relativos aos indivíduos presentes na população atual [9].

A RNPG possui três operadores de recombinação NOX (*Node-depth-encoding Order Crossover*), NPBX (*Node-depth-encoding Position Based Crossover*) e EHR (*Evolutionary History Recombination Operator*) [53], porém os dois primeiros não serão detalhados neste trabalho.

O objetivo do EHR é combinar o histórico de aplicações dos operadores PAO e CAO, que produziram bons indivíduos e aplicá-las a partir de um ancestral comum dos pais de forma a obter um novo indivíduo. Em [53] o operador EHR foi desenvolvido para a versão de PAO e CAO em problemas com múltiplas árvores e apresentou bom desempenho. Os detalhes de funcionamento do operador EHR serão descritos juntamente com a proposta desse trabalho.

As melhorias da RNPG possibilitam redução da complexidade de tempo por meio dos novos mecanismos de PAO e CAO e a redução do tempo de convergência com o uso dos operadores de recombinação.

CAPÍTULO 5

Proposta de Algoritmo Evolutivo baseado na RNPG para PPR

Operadores de recombinação são muito importantes para o desempenho dos EAs. A vantagem do uso desse operadores é que ao mesmo tempo em que preservam informações presentes nas gerações anteriores, eles permitem a exploração do espaço de busca por meio da combinação de características dos pais. Além disso, o uso de operadores de recombinação diminui o tempo de convergência necessário para o EA, quando comparado aos algoritmos que não usam desses operadores no processo evolutivo [6].

No entanto, o desenvolvimento de operadores de recombinação para os PPR é uma tarefa complexa [44]. As soluções para PPR são, em sua maioria, representadas por árvores em grafos. Assim, ao desenvolver a operação de recombinação é necessário garantir que os indivíduos resultantes sejam factíveis, ou seja, que, pelo menos, representam uma árvore conexa. Além disso, também é necessário considerar as restrições do problema em estudo. Com o propósito de garantir essas propriedades, cada uma das representações desenvolvidas para PPR têm proposto operadores de recombinação específicos [36, 37, 25, 42]. Além disso, esses estudos também mostram que o uso de operadores de recombinação convencionais, tais como crossover de ponto, em geral, necessitam de mecanismos de correção para garantir a factibilidade das soluções [31, 47, 44].

Este trabalho usa a representação nó-profundidade-grau (RNPG) para codificar o problema de projeto de redes da Árvore Máxima. Em [53], foram propostos alguns operadores de recombinação para a representação nó-profundidade-grau que apresentaram resultados satisfatórios. Dentre os operadores propostos, destaca-se o Operador de Recombinação com Base em Histórico Evolutivo (EHR), que foi desenvolvido para PPR cuja solução é representada por mais de uma árvore, ou seja, por uma floresta geradora. O operador teve seu desempenho avaliado em [53] para o problema de reconfiguração de sistemas de distribuição de energia elétrica apresentando resultados considerados interessantes para a área.

Com base no desempenho do EHR para problemas cuja solução é uma floresta,

este trabalho propõem alterar operador de recombinação para PPR, cuja solução é de apenas uma árvore. Com o intuito de avaliar o desempenho do EHR para soluções representadas por uma única árvore, propõem a utilização do Problema da Árvore Máxima (ver Seção 3.2).

A Seção 5.1 apresenta a formulação do operador EHR de acordo com [53]. As alterações necessárias para que ele possa ser aplicado a problemas com uma única árvore são descritas na Seção 5.2. Por fim, a Seção 5.3 descreve o funcionamento dos AEs proposta neste trabalho AE-PC, que usa a RNPG e os operadores de mutação PAO e CAO, e do AE-EPC que faz uso da RNPG, dos operadores de mutação PAO e CAO e o operador de recombinação EHR.

5.1 Operador EHR

O EHR é um operador de recombinação que foi desenvolvido com base nos operadores de mutação PAO e CAO da RNPG. Esse operador pode ser aplicado em problemas que envolvem grafos completos ou não completos, sem a necessidade de mecanismos de correção para soluções infactíveis. O funcionamento do EHR utiliza o histórico de aplicações dos operadores de mutação da RNPG para obter um novo indivíduo. Em outras palavras, o EHR utiliza a sequência de vértices p e a (e r, no caso do CAO), que foram aplicadas para gerar os indivíduos pais, a partir de um ancestral em comum, com o objetivo de obter um novo indivíduo [53]. A Figura 5.1 ilustra o histórico de aplicação do operador PAO para indivíduos com uma árvore. Os valores entre parênteses representam um par de vértices p e a.

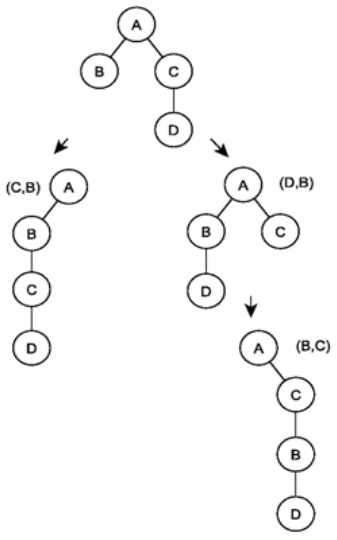


Figura 5.1: *Histórico de aplicações utilizando o PAO.*

Para realizar a operação de recombinação o EHR, primeiramente obtém-se os dois indivíduos que serão os novos pais. Depois, procura-se o ancestral comum aos dois pais, ou seja, o indivíduo em comum no histórico de evolução. Obtido esse ancestral, é recuperado do histórico de aplicações de PAO e CAO que deu origem a cada um dos pais, a partir do indivíduo comum. Para auxiliar no recuperação do histórico de aplicações o vetor π da RNPG foi alterado para incluir além do índice do ancestral o conjunto de vértices que foi utilizado para obter o novo indivíduo. A seguir, de posse do conjunto total de aplicações (históricos), o EHR seleciona um subconjunto de K aplicações que serão utilizadas para obter o novo indivíduo. A recombinação por meio do EHR resulta em somente um novo indivíduo, obtido pela aplicação do subconjunto de aplicações ao ancestral comum. O Algoritmo 5.1 descreve os passos de funcionamento do operador EHR.

Algoritmo 5.1: Pseudocódigo do EHR.

Entrada: População do Algoritmo Evolutivo
Saída: Novo Indivíduo

1 Selecionar na população dois indivíduos que representam os pais;
2 Determinar o ancestral comum aos indivíduos pais. Caso não haja ancestral comum entre eles, escolher um dos pais e aplicar PAO ou CAO (assim, ele será o ancestral comum).
3 Recuperar a sequência de aplicações (histórico) de PAO e CAO no ancestral para gerar cada um dos pais. ou seja, os pares de vértices p e a, para PAO e, no caso do CAO, a tripla p, a e r.
4 Selecionar aleatoriamente um subconjunto s de sequências de pares (triplas no caso do CAO) de vértices do conjunto de aplicações do Passo 3, tal que $|s| \leq K$, onde $|s|$ é o tamanho do subconjunto e K é uma constante inteira positiva.
5 Aplicar as sequências de s no ancestral para obter um novo indivíduo.

Para realizar o passo 5 do Algoritmo 5.1 é utilizado o mesmo processo para aplicação dos operadores PAO e CAO, com a diferença de que os vértices já estão escolhidos. Porém, antes de chamar a rotina dos operadores de mutação, é necessário verificar se o par de vértices, para PAO, ou a trinca de vértices, para CAO, continuam factíveis. Ou seja, 1) o vértice a encontra-se em outra árvore da floresta, tanto para PAO quanto para CAO; e 2) no caso de CAO, o vértice r pertence à subárvore do vértice p.

Com o propósito de ilustrar o funcionamento do EHR, considere as duas florestas apresentadas na Figura 5.2. *Individuo*_1 e *Individuo*_3, representam o primeiro e o segundo pai, respectivamente.

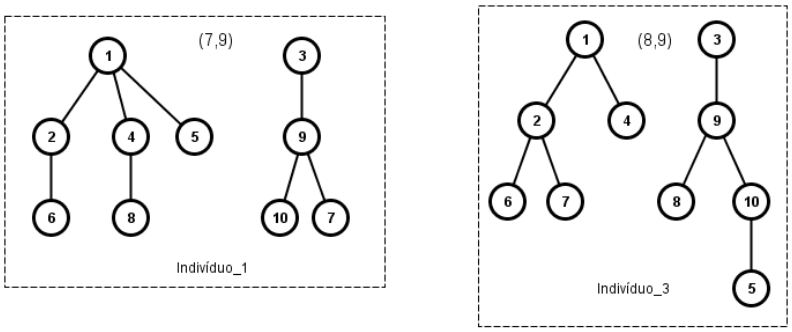


Figura 5.2: *Florestas para aplicação do EHR.*

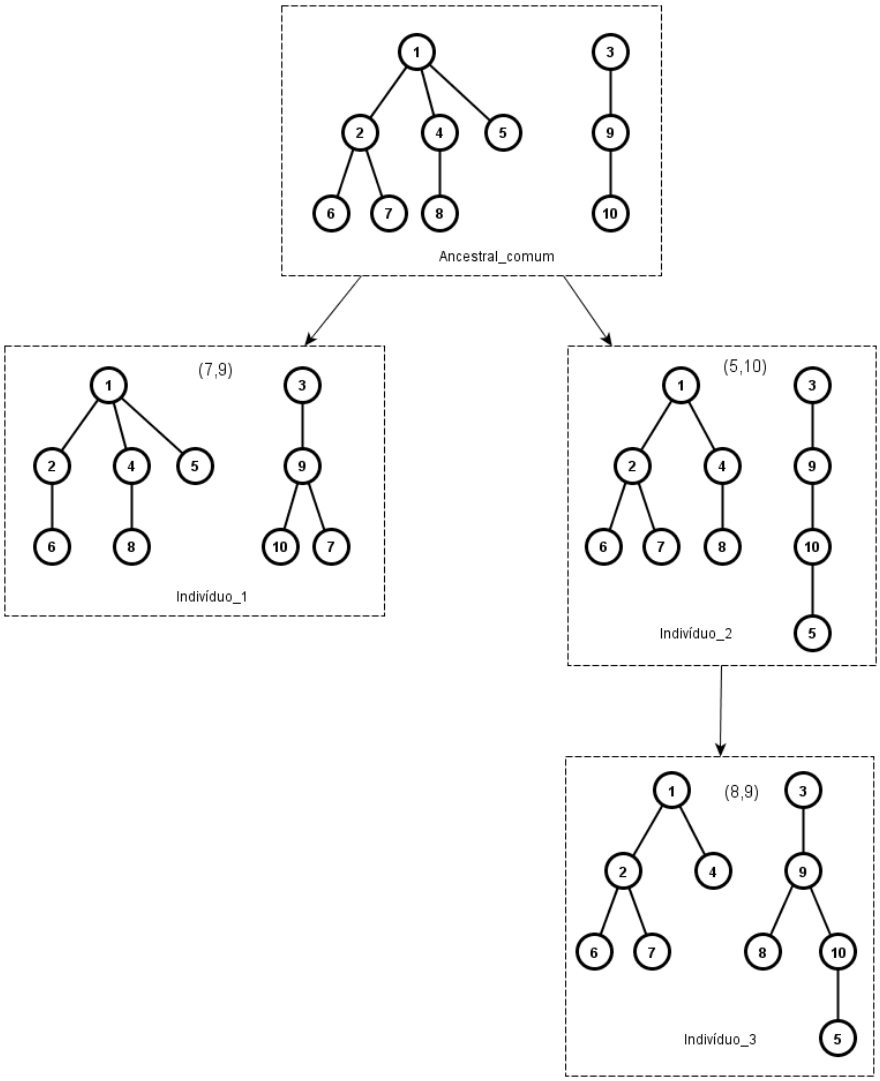


Figura 5.3: *Histórico de aplicações das florestas com o ancestral comum.*

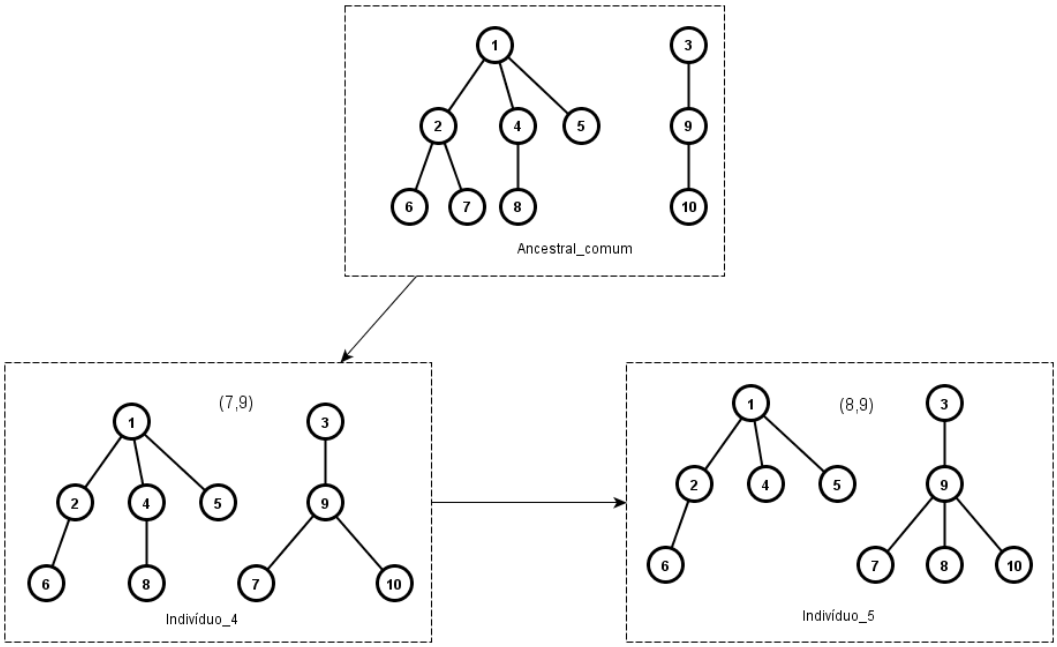


Figura 5.4: *Histórico de aplicação do subconjunto s sobre o Ancestral_comum.*

A Figura 5.2 mostra o passo (1), do algoritmo 5.1. Na Figura 5.3 é apresentado o *ancestral_comum* de *Individuo*_1 e *Individuo*_3 , passo (2) do algoritmo, juntamente com o histórico de aplicações que dá origem aos pais (ver passo (3)). Por exemplo, (7,9) indica que a subárvore iniciada no vértice 7 foi implantada no vértice 9. Por fim, na Figura 5.4, o subconjunto *s* de aplicações do passo (4) e a sua aplicação ao *Ancestral_comum*, que resulta na floresta *Indivduo*_5, obtida conforme descrito no passo (5) do Algoritmo 5.1.

5.2 Alterações do EHR para Árvore única

Analisando os passos do Algoritmo 5.1 observa-se que a execução dos passos (1), (2), (3) e (4) independe da estrutura da solução do problema, ou seja, os indivíduos podem conter uma única árvore ou uma floresta com mais de uma árvore, que não modifica o processo de execução desses passos. No entanto, para o passo (5), a estrutura da solução interfere na sua execução e é esse o ponto onde devem ser feitas as adaptações do operador EHR.

A primeira parte para executar o passo (5) do Algoritmo 5.1 é validar os pares (p,a) ou triplas (p,a,r) antes de efetuar as operações. Ao contrário do processo de validação, quando utiliza-se mais de uma árvore, o vértice a obrigatoriamente pertence à mesma árvore do vértice p, pois o indivíduo possui uma única árvore. Assim, para poder aplicar uma modificação de PAO ou CAO, o vértice a não pode estar na subárvore

de p. No caso dos vértices da aplicação indicarem uma modificação de CAO, a validação do vértice r obedece a mesma regra para a aplicação com mais de uma árvore, ou seja, vértice r necessariamente precisa estar na subárvore de p. Quando for aplicado o PAO, o vértice r tem valor -1. O Algoritmo 5.2 apresenta o processo de validação dos vértices para a utilização do operador EHR com PPRs representados por uma única árvore.

Algoritmo 5.2: Pseudocódigo de validação dos vértices antes da aplicação do EHR.

Entrada: Floresta, Indice_p, Indice_a, Indice_r
Saída: verdadeiro/falso

```
1  aux := Índice do último elemento da sub-árvore de p
2  if (Indice_a ⩾ Indice_p) E (Indice_a ⩽ aux) then
3  |  retorna falso
4  end
5  if (Indice_r ≠ −1) then
6  |  if (Indice_r ⩽ Indice_p) OU (Indice_r > aux) then
7  |  |  retorna falso
8  |  end
9  end
10 retorna verdadeiro
```

Após fazer a validação dos vértices, para continuar a execução do passo (5) do algoritmo do EHR, é utilizada a definição dos operadores PAO e CAO. A única diferença do uso normal de PAO e CAO para o uso no EHR, é que, ao serem executados pelo EHR, os operadores recebem como entrada os vértices já validados. Em [53], além da formulação dos operadores de recombinação, também é apresentado e validado como utilizar os operadores PAO e CAO para PPR com uma única árvore. Assim, para possibilitar a aplicação do operador EHR nesses problemas, a alteração necessária é a da validação dos vértices, que é realizada seguindo o Algoritmo 5.2 apresentado e, após isso, utilizar a implementação dos operadores PAO e CAO da RNPG para problemas de uma única árvore.

5.3 Formulação dos Algoritmos Evolutivos AE-PC e AE-EPC

Como parte deste trabalho foram desenvolvidos dois AE para o OMTP que serão utilizados para a validação da proposta. O primeiro algoritmo, denominado AE-PC usa a RNPG e seus operadores de mutação PAO e CAO. O segundo algoritmo, além

da representação e dos operadores do AE-PC, inclui em sua implementação o operador EHR, sendo denominado AE-EPC. A maioria das características de ambos os algoritmos é a mesma e serão descritas a seguir.

No desenvolvimento de AE diversos parâmetros, operações e taxas precisam ser definidos. Primeiramente, foi escolhido que os algoritmos teriam como processo geracional a formulação *steady-state*, em outras palavras, a cada geração só é obtido um novo indivíduo. Assim, o número de gerações e avaliações é o mesmo.

Para o processo de seleção dos indivíduos para reprodução, tanto o AE-PC quanto o AE-EPC utilizam a operação de torneio com participação de 3 indivíduos. A diferença entre os algoritmos é que para o AE-PC sempre é necessário escolher somente um indivíduo, mas no AE-EPC quando for aplicada a operação de recombinação devem ser selecionados dois indivíduos. Vale ressaltar que quando o AE-EPC utiliza as operações de mutação também é selecionado somente um indivíduo.

Diferentemente de outros AEs com a operação de recombinação seguida da operação de mutação, no AE-EPC, as operações são aplicadas de forma disjunta, pois o EHR necessita de aplicações de PAO ou CAO independentes. Assim, ou utiliza-se o operador EHR ou PAO e CAO. Esses operadores são chamados de mutação pois consistem em modificações de uma solução existente e não necessariamente por modificarem uma solução com origem na recombinação, como ocorre em situações convencionais.

Ambos os algoritmos utilizam taxas de aplicação dinâmicas para definir qual operador será utilizado. No caso do AE-PC a taxa dinâmica determina se o PAO ou CAO dará origem à nova solução. Já no AE-EPC, a taxa dinâmica é utilizada para escolher se será aplicado o EHR ou os operadores de mutação. No caso em que a operação de mutação é escolhida, então é utilizada uma probabilidade de 50% de aplicar o PAO ou o CAO. O dinamismo da taxa de aplicação é realizado com base no sucesso dos filhos obtidos: se o filho resultante apresenta *fitness* melhor ou igual ao dos pais, então, o operador obteve sucesso e sua probabilidade de aplicação é incrementada; caso contrário, ela é decrementada. A taxa de cada operador é complementar à da outra operação. Além disso, como o AE-EPC necessita de um histórico de evolução (conjunto de aplicações de PAO e CAO) para realizar a operação do EHR, o EHR só é utilizado após um número pré-definido de gerações. Durante essas gerações, o AE-EPC só aplica PAO e CAO.

A população, tanto do AE-PC quanto do AE-EPC, é mantida de forma totalmente elitista, ou seja, um indivíduo só é inserido na população se possui *fitness* melhor que o pior indivíduo da população. Além disso, o indivíduo precisa ter cromossomo diferente dos indivíduos já inseridos na população, de forma a manter a diversidade de características. O processo de verificação de igualdade ou equivalência dos cromossomos considera somente as arestas dos indivíduos que também estão presentes na árvore de referência analisada - modelo a ser alcançado. Em outras palavras, se o novo indivíduo

possui, como únicas arestas diferentes dos outros indivíduos da população, arestas que não estão presentes na árvore de referência, então os indivíduos são considerados equivalentes. Essa forma de cálculo de equivalência produziu uma melhora no desempenho dos AEs em relação aos testes preliminares que não utilizavam tal comparação.

Um parâmetro importante dos AEs é o critério de parada das iterações do mesmo. Para os algoritmos propostos serão utilizados dois critérios de parada independentes. A primeira condição de parada é diretamente relacionada à característica do problema da árvore máxima. Nesse critério é verificado se a solução ótima foi encontrada, ou seja, a árvore de referência foi obtida e, em caso afirmativo, encerra-se a execução. O segundo critério de parada visa estabelecer uma condição de convergência do algoritmo. Esse critério será utilizado como parada principalmente nas execuções para as quais a solução ótima não é obtida. Essa condição avalia o número de gerações que o AE não obtém um *fitness* superior ao do melhor indivíduo. Para os ensaios realizados, o número de gerações sem melhora do fitness é de 100.000.

Em [9], monstra que a $matriz\Pi$ deve ter seus valores reiniciados após um número pré-definido de gerações denominado N_Π. Esse processo é aplicado para garantir a complexidade de tempo da representação nó-profundidade-grau. N_Π é da ordem do número de vértices do grafo, assim, calcula-se como $N_\Pi = round(c * \sqrt{n})$, em que c é uma constante positiva inteira (um dos parâmetros de entrada da execução do AE) e n é o tamanho do grafo. Ambos os algoritmos fazem usam esse processo e foram executados com diferentes valores para o parâmetro c que será chamado de tamanho da $matriz\Pi$, na apresentação dos resultados. Nesse processo de reinicialização da $matriz\Pi$, o histórico de evolução é perdido, então utilizou-se para o AE-EPC que, a cada vez que a matriz é reinicializada, o algoritmo aguarda o número pré-definido de gerações para começar a utilizar o EHR, como abordado durante a descrição da taxa de aplicação. Esse número de gerações foi definido como 10% de N_Π.

Para a aplicação do EHR no AE-EPC, é preciso definir K, o tamanho máximo do subconjunto s. O valor de K será um dos parâmetros analisados durante os testes de desempenho do algoritmo. Além dos dois parâmetros já especificados, que serão avaliados durante os ensaios, o valor de K também será analisado para verificar o desempenho do operador EHR.

Por fim, o tamanho da população é outro parâmetro importante de configuração dos AE. Neste trabalho, o tamanho da população será um parâmetro variável a cada execução do AE-PC e do AE-EPC. Assim, durante os testes serão analisados alguns tamanhos de população diferentes para verificar sua influência no desempenho, de ambos os algoritmos.

CAPÍTULO 6

Experimentos e Resultados

Este capítulo avalia o desempenho do operador de recombinação EHR da representação Nó-Profundidade-Grau (RNPG) em um Algoritmo Evolutivo (AE), denominado AE-EPC, aplicado à diversas instâncias do problema de árvore máxima. Os resultados obtidos com a utilização do EHR serão comparados com a versão do AE que utiliza a RNPG em sua formulação padrão, denominado AE-PC, que faz uso somente dos operadores de mutação PAO e CAO.

O conjunto de instâncias composto foi dividido em três diferentes tipos de topologia da árvore geradora de referência: linha, estrela e aleatória. Para cada topologia foram utilizados grafos completos com a ordem variando de 10 a 200 vértices. Além disso, para cada caso, foram propostas seis árvores de referência diferentes. Nos ensaios, também foram realizadas variações nos parâmetros de entrada do EHR, para que a sua eficiência fosse analisada em situações distintas. A Tabela 6.1 sumariza os parâmetros utilizados durante os testes e os valores correspondentes, de forma a facilitar a reprodução da análise. Os experimentos foram realizados em um computador com processador i7, 32GB de RAM e sistema operacional Linux Ubuntu 64.

Fatores variáveis	**Valores utilizados nos testes**
Topologias de árvore	Linha, estrela e aleatória
Quantidades de vértices	10, 20, 30, 50, 100 e 200
Número de árvores de referência	6
N_{Π}(1)	30, 50 e 100
Constante K(2)	0(3), 4, 6 e 10
População	10, 20, 50 e 100
Números de execuções por teste	50
Critério de parada	Encontrou a solução ótima ou até 100.000 avaliações sem melhora

(1) N_{Π} é um dos parâmetros dos AE-EPC que indica em quantas avaliações a $matriz\Pi$ será reinicializada.

(2) A constante K é a quantidade de aplicações dos operadores a partir do ancestral comum.

(3) $K = 0$ indica os testes realizados sem a utilização do operador EHR.

Tabela 6.1: *Parâmetros analisados durante os testes para avaliação de desempenho do EHR na RNPG para o problema de árvore máxima*

A análise de desempenho do operador EHR será feita considerando a média dos resultados para as seis árvores de referência de cada grafo de cada tipo de topologia. Além disso, os valores apresentados nos gráficos referem-se à média das 50 execuções realizadas. Para a apresentação dos resultados, os valores de *fitness* foram normalizados entre 0 e 1, equivalentes a 0,00% e 100,00% (das arestas da árvore de referência foram encontradas). Esse processo foi utilizado para indicar o quão próximas as soluções encontradas estão da árvore de referência. Além disso, a normalização dos valores mantém-se na mesma ordem de grandeza conforme aumenta-se o número de vértices do grafo. Para cada uma das topologias analisadas, a escala dos gráficos foram adaptadas para evidenciar as diferenças pontuais de cada uma.

Nos gráficos de números de gerações, é apresentada a geração em que ocorreu a última alteração no *fitness* do melhor indivíduo. Quando essa alteração não representa a obtenção do valor ótimo, o número de gerações final é decrescido do valor 100.000.

Considerando que o grau de dificuldade e complexidade é alterado consideravelmente conforme a topologia da árvore, as análises de desempenho serão realizadas separadamente. A Seção 6.1 apresenta os resultados para a topologia do tipo linha com as diferentes configurações. A análise para a topologia do tipo estrela é descrita na Seção 6.2. Por fim, a seção 6.3 descreve o desempenho com a topologia do tipo aleatória.

6.1 Topologia Linha

As árvores geradoras com topologia linha são caracterizadas por possuírem dois vértices folha, ou seja, de grau 1 e os demais vértices com grau 2. A Figura 6.1 apresenta um exemplo desta topologia para a árvore geradora de um grafo com 6 vértices.

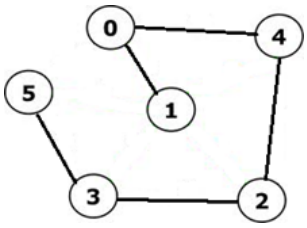


Figura 6.1: *Exemplo de árvore geradora com topologia linha.*

Com o objetivo de facilitar a análise comparativa dos resultados obtidos, esses serão agrupados de acordo com o parâmetro tamanho da população. Para os valores de *fitness*, os gráficos dessa topologia utilizam escala entre 0,88 e 1,00.

6.1.1 População 10

Os resultados obtidos pelo AE-EPC com população de 10 indivíduos para a topologia do tipo linha são discutidos nesta Seção. As Figuras 6.2, 6.3 e 6.4 apresentam o número de gerações, o tempo de execução em segundos e os valores de *fitness* normalizados para a média de cada um dos grafos analisados, com variação no N_Π e K, conforme destacam as legendas das curvas EHR-K (K = 4, 6 e 10) e sem EHR (K = 0).

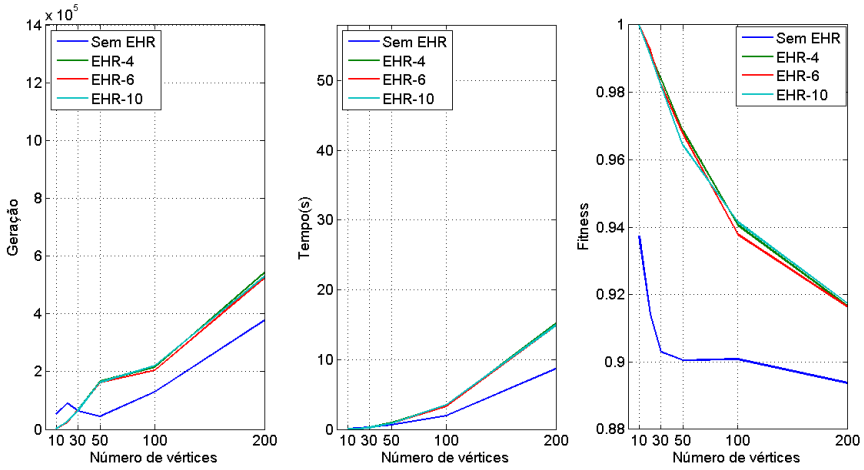


Figura 6.2: *População 10 e N_Π 30.*

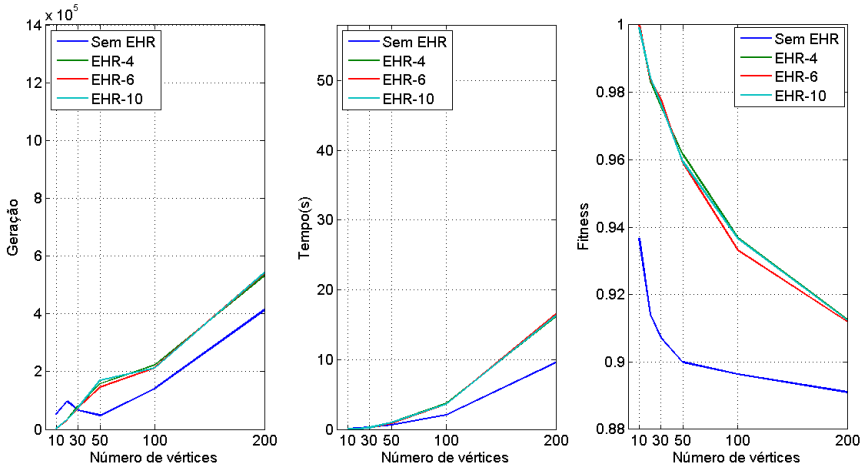


Figura 6.3: *População 10 e N_Π 50.*

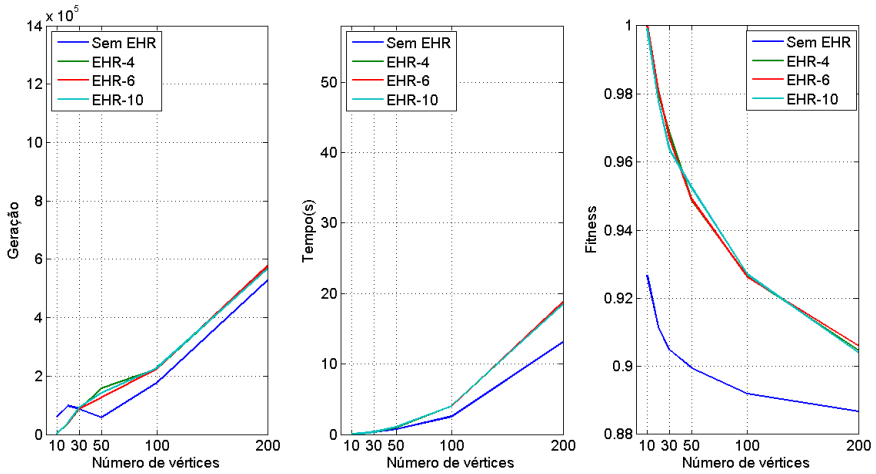


Figura 6.4: *População 10 e N_Π 100.*

Analisando os resultados, os valores de *fitness* com o AE-EPC são melhores em todos os testes com população de tamanho 10 que os do AE-PC. Vale destacar que o AE-EPC teve 100,00% de eficácia nos testes com 10 vértices, ou seja, em todas as execuções foi encontrada a árvore de referência, independentemente do N_Π. No melhor caso, o ganho foi de cerca de 9,00%, nos testes com AE-EPC, para grafos com 30 vértices e $N_\Pi = 100$, em relação ao AE-PC. No pior caso, houve cerca de 2,20% de ganho do AE-EPC em relação ao AE-PC, com grafo de 200 vértices e $N_\Pi = 100$. A variação dos valores de K (4, 6 e 10), no AE-EPC, resultou em uma diferença máxima de 0,50% nos valores de *fitness* obtidos. Essa diferença ocorreu quando foi utilizada $N_\Pi = 100$ aplicada ao grafo com 50 vértices. Observa-se dos gráficos que a variação do parâmetro K não produz melhora significativa no desempenho do AE-EPC.

O número de gerações manteve-se estável nos testes com a variação do N_Π de 30 para 50. No entanto, ao aumentar o N_Π para 100, houve um ligeiro aumento do número de gerações utilizado. Também é importante ressaltar que, embora tenha ocorrido aumento do número de gerações com o aumento do número de vértices e do N_Π, no AE-EPC, o aumento foi proporcionalmente menor que nos testes com o AE-PC. Além disso, ressalta-se que o aumento no número de gerações não reflete em um aumento substancial no tempo execução.

Quanto ao tempo de execução, para os grafos com até 30 vértices, o AE-EPC teve melhor resultado, apesar do maior número de gerações, que o AE-PC. No entanto, para os grafos maiores que 30 vértices, o AE-EPC teve um tempo de execução maior que o AE-PC. Com a expansão do N_Π de 50 para 100, embora todos tenham aumentado, a diferença diminuiu entre os dois algoritmos analisados.

Uma característica comum em todos os critérios avaliados (número de gerações, tempo de execução e *fitness*), é que o AE-EPC foi melhor ou igual para os grafos com até 30 vértices. Outro fato importante é que os valores de *fitness* sempre foram superiores,

embora tenha demandado maior tempo de execução e maior número de gerações, com o AE-EPC em relação ao AE-PC. Apesar de que o esperado era que o AE-EPC diminui-se o número de avaliações em relação ao AE-PC, ressalta-se que a qualidade das soluções encontradas com o AE-EPC foi superior. Se considerarmos o número de gerações para encontrar a mesma solução que o AE-PC obteve, o AE-EPC demanda menor número de gerações.

6.1.2 População 20

A seguir são descritos os resultados obtidos pelo AE-EPC e o AE-PC com população de 20 indivíduos e com variação dos demais parâmetros. As Figuras 6.5, 6.6 e 6.7, apresentam os gráficos de número de gerações, tempo de execução em segundos e valores do *fitness* normalizados para os tamanhos 30, 50 e 100 da $Matriz\Pi$.

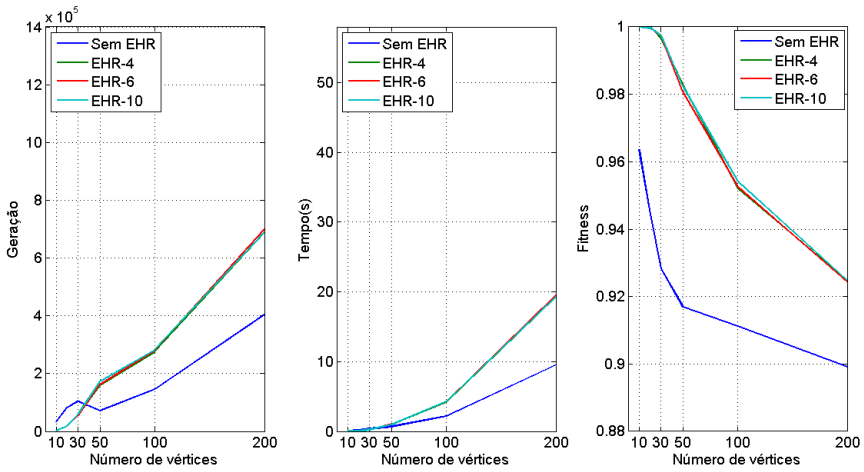


Figura 6.5: *População 20 e N_Π 30.*

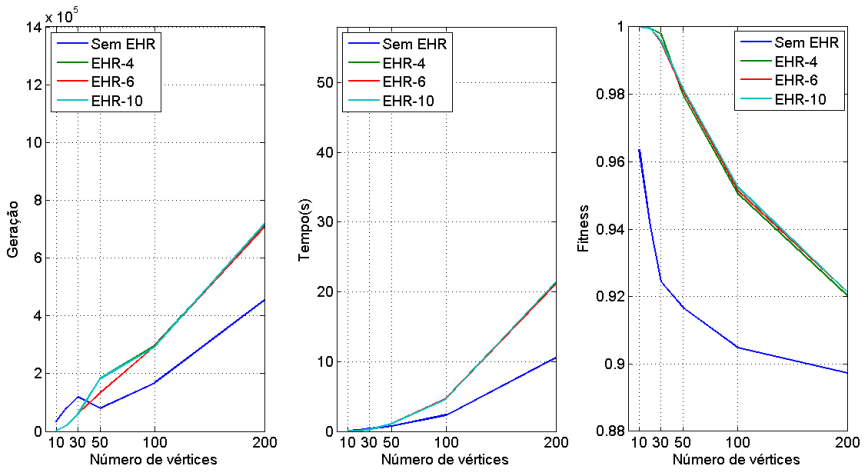


Figura 6.6: *População 20 e N_Π 50.*

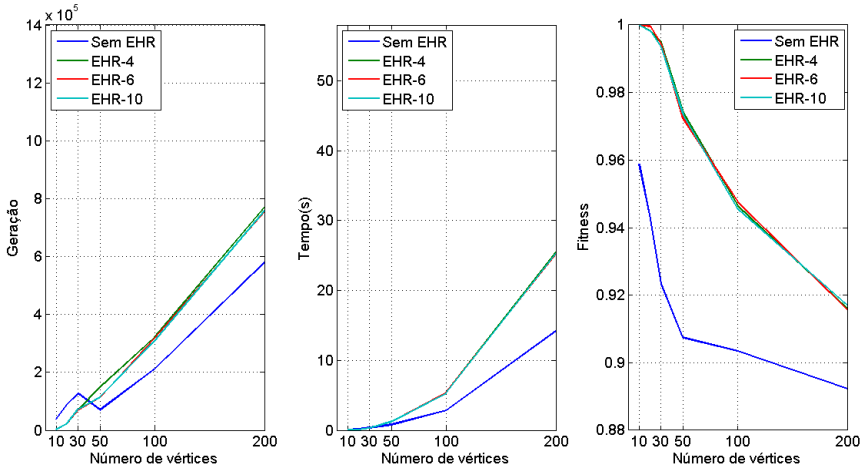


Figura 6.7: *População 20 e N_Π 100.*

Analisando os resultados de *fitness*, o AE-EPC obteve melhor desempenho em todos os testes com população de tamanho 20, em comparação com o AE-RNP, assim como ocorreu para os testes com população de tamanho 10. No entanto, com o aumento no tamanho da população, além de obter 100,00% de eficácia nos testes com 10 vértices, o AE-EPC teve eficácia acima de 99,30% com até 20 vértices. Isto significa que a solução ótima foi encontrada na grande maioria das execuções. No melhor caso, o ganho foi de cerca de 7,50%, do AE-EPC comparado ao AE-PC, nos testes com grafos de 30 vértices e $N_\Pi = 50$. No pior caso, o ganho foi cerca de 2,50%, com 200 vértices e $N_\Pi = 50$. Nos demais casos o ganho do AE-EPC sobre o AE-PC ficou nesse intervalo. Novamente, a variação do parâmetro K (4, 6 e 10) do AE-EPC não apresentou diferença significativa de desempenho no algoritmo.

Ao contrário do que aconteceu com o AE-EPC de população 10, com o aumento da população para 20 indivíduos, para os grafos com até 30 vértices obteve-se uma redução do número gerações em relação ao AE-PC. Esse comportamento repetiu-se mesmo com variações do parâmetro K e do N_Π. No entanto, a partir de 50 vértices, o AE-EPC continuou a utilizar um maior número de gerações que o seu opositor, porém com melhores valores de *fitness*. Com grafo de 50 vértices e $N_\Pi = 30$, a variação no parâmetro K do AE-EPC não apresentou diferença no número de gerações. Porém, com o aumento do $N_\Pi = 50$, houve redução de 19,00% no número de gerações para $K = 6$ e discreta variação nos demais. Com $N_\Pi = 100$, houve nova redução de 14,00% para $K = 6$ e $K = 4$ e de 35,00% para $K = 10$.

Quanto ao tempo de execução, com até 30 vértices, o AE-EPC foi melhor que o AE-PC, assim como ocorreu com o número de gerações. Além disso, a variação do N_Π contribuiu para o aumento do tempo em todos os casos, tanto para o AE-EPC quanto para o AE-PC.

Semelhante a Seção anterior, uma característica comum é que, em todos os

critérios avaliados (número de gerações, tempo de resposta e *fitness*), o AE-EPC foi melhor ou igual para os grafos com até 30 vértices. Outro fato importante é que os valores de *fitness* obtidos pelo AE-EPC foram sempre superiores, embora tenha demandado mais tempo e número de gerações.

6.1.3 População 50

Nesta Seção são discutidos os resultados obtido com população de 50 indivíduos. Assim como nos casos anteriores as Figuras 6.8, 6.9 e 6.10 apresentam os resultados com os $N_{\Pi} = 30, 50 e 100$.

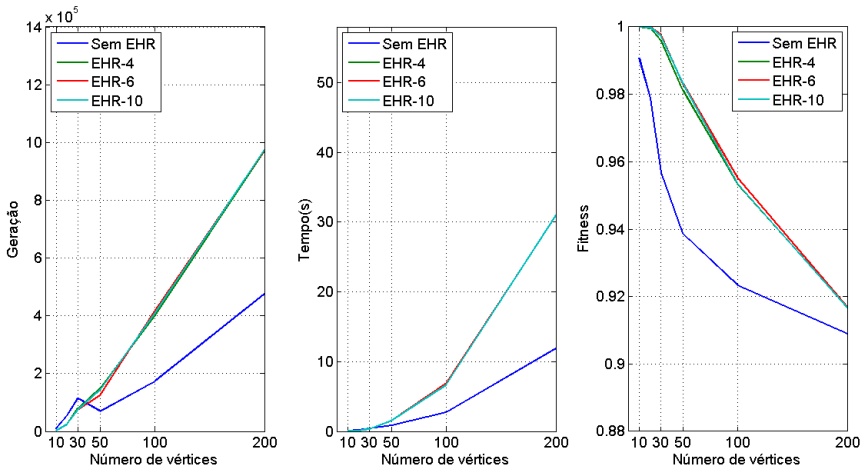


Figura 6.8: *População 50 e N_{Π} 30.*

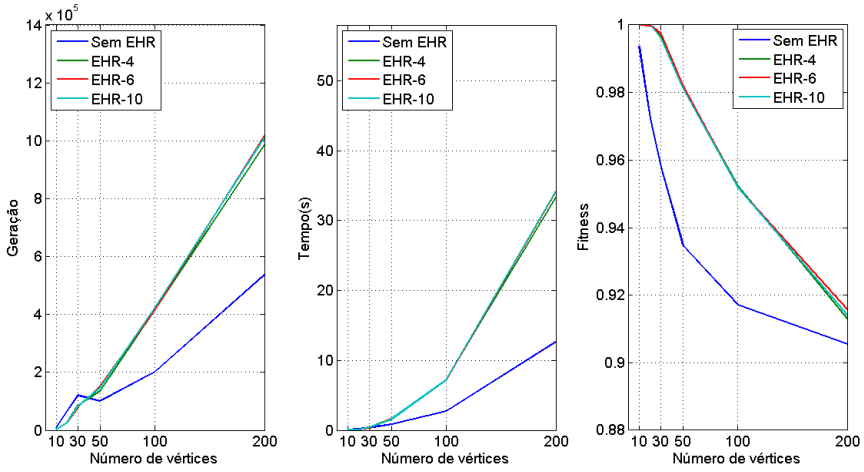


Figura 6.9: *População 50 e N_{Π} 50.*

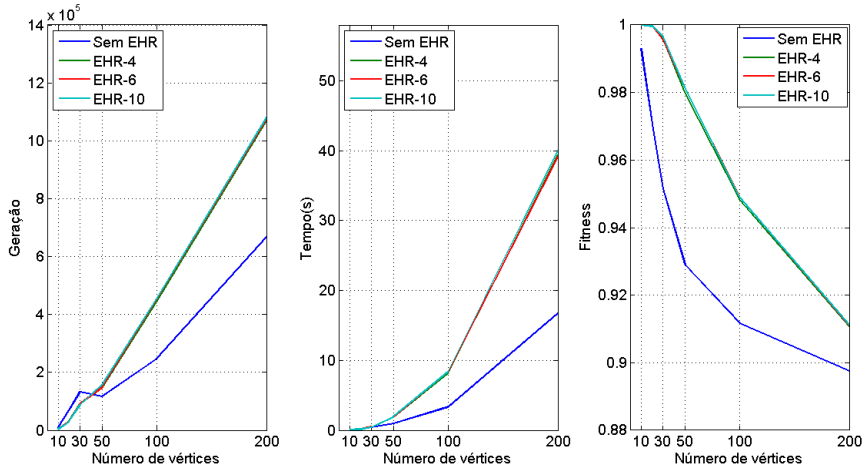


Figura 6.10: *População 50 e N_Π 100.*

Analisando as figuras temos que os valores de *fitness* obtidos pelo AE-EPC são melhores que os obtidos pelo AE-PC. Assim como ocorreu com as populações de tamanho 10 e 20, o AE-EPC obteve 100,00% de eficácia com 10 vértices e acima de 99,96% com até 20 vértices. Com o aumento do tamanho da população o AE-PC teve uma melhora no seu desempenho, com isso a porcentagem de ganho obtida pelo AE-EPC diminuiu para 5,30% no melhor caso e 0,60% no pior caso. A variação dos valores de K (4, 6 e 10) afetou em menos de 0,30% os resultados.

A quantidade de gerações ficou praticamente estável com a variação o N_Π nos testes com 10, 20 e 30 vértices. Neste intervalo, o AE-EPC obteve resultados melhores em todos os testes, em comparação ao AE-PC, como esperado, e a variação do fator K praticamente não afetou os resultados. A partir de 50 vértices, o AE-EPC demandou mais gerações que o seu opositor, porém com melhor valor de *fitness*. Assim, como nos teste anteriores o aumento do N_Π implicou em um aumento no número de gerações.

Analisando o tempo de execução, tem-se que com até 30 vértices, o AE-EPC utilizou o mesmo tempo ou foi discretamente melhor. Com o aumento do número de vértices para 100, o tempo com EHR ficou cerca de 5 segundos (144,00%) acima. Ao alcançar 200 vértices, a diferença chegou a 24 segundos (149,00%), em relação ao AE-PC. Essa diferença no tempo de execução deve-se à diferença no número de gerações que para esses grafos é superior no AE-EPC. A variação do N_Π contribuiu diretamente para o aumento do tempo em todos os casos.

Semelhante aos demais testes com topologia linha já apresentados, uma característica comum é que, em todos os critérios avaliados (número de gerações, tempo de execução e *fitness*), o AE-EPC foi melhor ou igual ao AE-PC para os grafos com até 30 vértices. Outro fato importante é que o *fitness* sempre foi superior, embora tenha demandado mais tempo e número de gerações.

6.1.4 População 100

Finalmente, esta Seção aborda o desempenho do AE-EPC com o parâmetro tamanho da população de 100 indivíduos. Os resultados com a variação nos outros parâmetros são apresentados nas Figuras 6.11, 6.12 e 6.13.

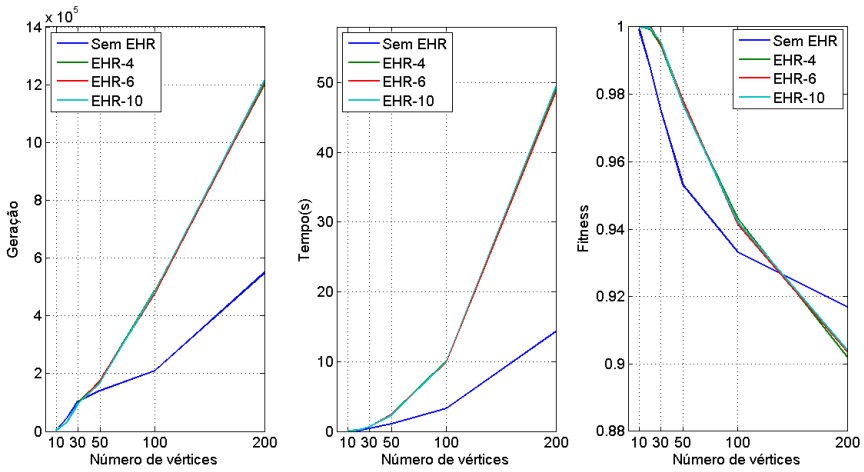


Figura 6.11: *População 100 e N_Π 30.*

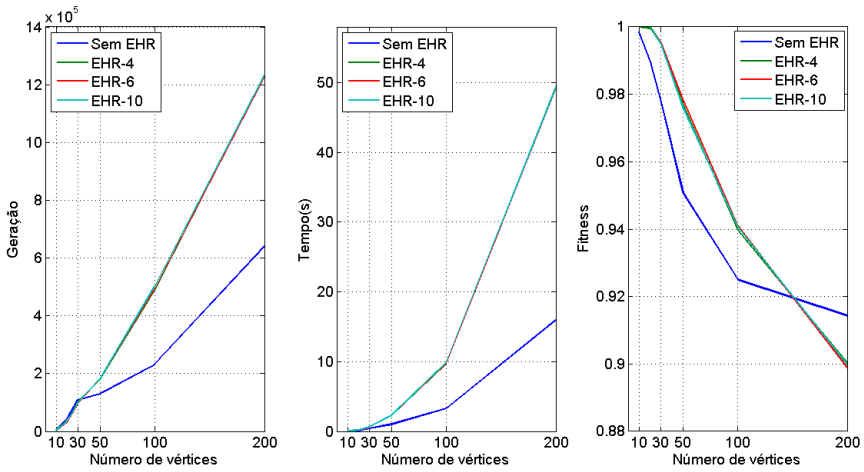


Figura 6.12: *População 100 e N_Π 50.*

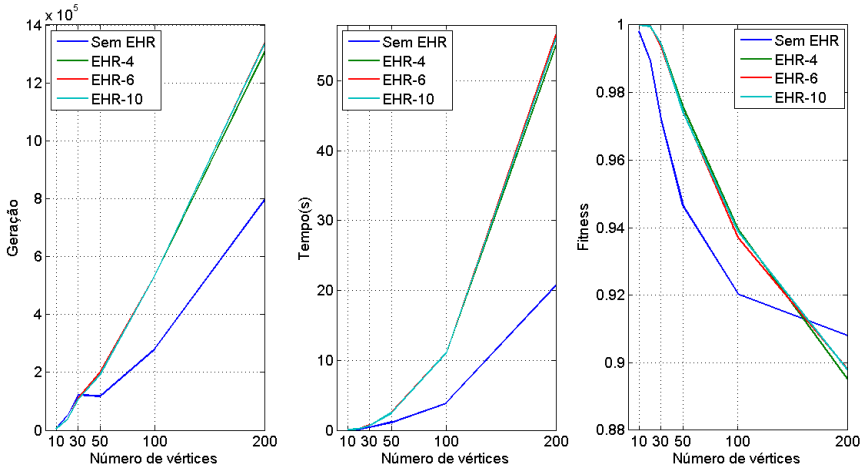


Figura 6.13: *População 100 e N_Π 100.*

Com base nos gráficos de valor de *fitness* observa-se que, diferentemente dos demais testes, o AE-EPC foi superior ao AE-PC, somente para os grafos com até 100 vértices, para as diferentes variações do N_Π. Para os grafos de 200 vértices o desempenho do AE-PC foi superior. No entanto, a diferença não é muito grande, cerca de 1,67%. Para os demais grafos, onde o AE-EPC é superior, seu desempenho foi similar ao dos demais testes realizados. A variação dos valores de K (4, 6 e 10) afetou em menos de 0,33% os resultados.

A quantidade de gerações ficou praticamente estável com a variação do N_Π nos testes com até 50 vértices. Até 30 vértices, o AE-EPC obteve resultados iguais ou melhores em todos os testes e a variação do fator K praticamente não afetou o desempenho do AE-EPC de 10 a 200 vértices. A partir de 50 vértices, o AE-EPC demandou mais gerações que o seu opositor. Com o aumento do N_Π, o número de gerações aumentou, contudo, a diferença entre os resultados com e sem EHR diminuiu, com 200 vértices, de 121,00% para 68,00%.

O tempo de execução de ambos os algoritmos aumentou com o aumento do N_Π, assim como o número de gerações.

6.1.5 Conclusões parciais

Com base nas análises feitas para os testes com topologia linha, pode-se destacar as seguinte conclusões parciais sobre o desempenho do AE-EPC:

1. Pelos testes realizados, o AE-EPC conseguiu melhorar os valores de *fitness* em relação ao AE-PC, com exceção ao teste realizado com população de tamanho 100 e número de vértices igual a 200;
2. Os pontos onde houve maior vantagem do *fitness* sempre ocorreram com 30 ou 50 vértices;

3. Com 10 vértices, o algoritmo sempre alcançou a solução ótima;
4. No caso mais complexo, de 200 vértices, a configuração que obteve o melhor valor de *fitness*, para ambos os algoritmos, foi com $N_{\Pi} = 30$ e população de 20 indivíduos;
5. O desempenho do AE-PC melhora com o aumento do tamanho da população e diminui com o aumento do N_{Π}. No entanto para o AE-EPC, a partir da população de 20 indivíduos, ocorre o contrário;
6. As variações do parâmetro K para 4, 6 e 10, ou seja, a quantidade de mutações aplicadas a partir do ancestral comum alterou discretamente os resultados. Assim, o parâmetro K não influenciou o desempenho do algoritmo;
7. O tempo em segundos de execução do AE-EPC foi melhor ou igual em quase todos os casos com até 30 vértices, em relação ao do AE-PC. A partir de 50 vértices, o AE-EPC passou a ser significativamente mais lento, devido ao aumento no número de gerações;
8. Até 30 vértices, o EHR demandou menos gerações em quase todos os testes.

Pelos testes realizados, avaliando os critérios de qualidade do resultado (*fitness*), tempo de execução e número de gerações, destaca-se que o AE-EPC, sempre encontrou a solução ótima para o caso mais simples, com 10 vértices. Assim, para casos menos complexos opta-se pela configuração de parâmetros que demanda menos tempo de execução. Para os demais casos recomenda-se a execução do AE-EPC com população de 20 indivíduos e $N_{\Pi} = 50$, pois as soluções encontradas possuem eficácia superior a 92,00% com tempo de execução, no pior caso, em torno de 20 segundos.

6.2 Topologia Estrela

As árvore geradoras que caracterizam a topologia estrela possuem um vértice de grau $n-1$, onde n é o número de vértices do grafo, e os demais vértices possuem grau 1. A Figura 6.14 apresenta um exemplo de árvore geradora com topologia estrela.

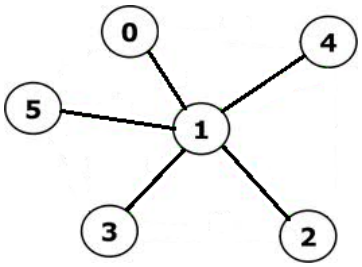


Figura 6.14: *Topologia Estrela.*

Ressalta-se que a topologia estrela é considerada a mais fácil de ser encontrada pelos AEs aplicados ao problema de árvore máxima. Novamente, com o intuito de facilitar a análise dos resultados, os mesmos serão agrupados pelo parâmetro tamanho

da população. Para os gráficos de valores de *fitness* é utilizada a escala com valores entre 0,97 e 1,00.

6.2.1 População 10

Esta Seção analisa os resultados obtidos pelo AE-EPC com população de 10 indivíduos aplicado a topologia estrela. As Figuras 6.15, 6.16 e 6.17, apresentam os valores de número de gerações, tempo de execução e *fitness* normalizados para a $Matriz\Pi$ com tamanho 30, 50 e 100.

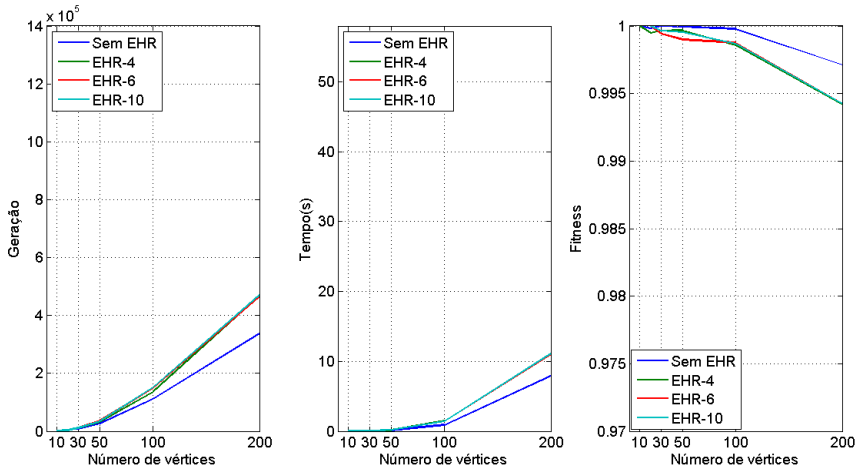


Figura 6.15: *População 10 e N_Π 30.*

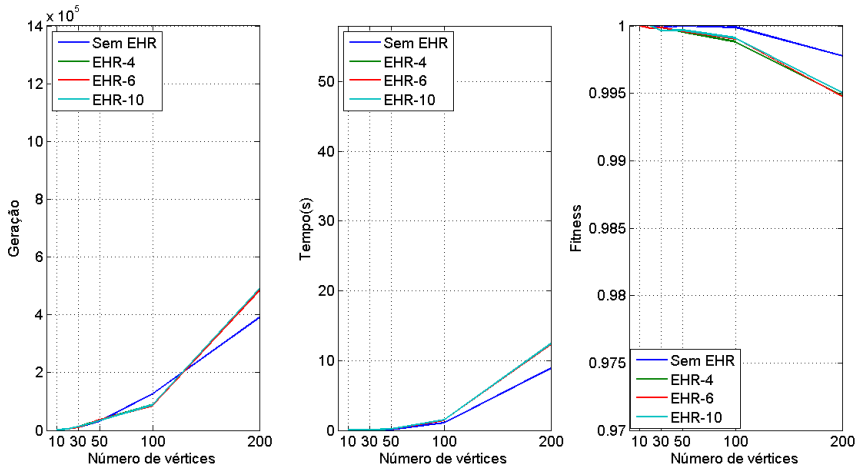


Figura 6.16: *População 10 e N_Π 50.*

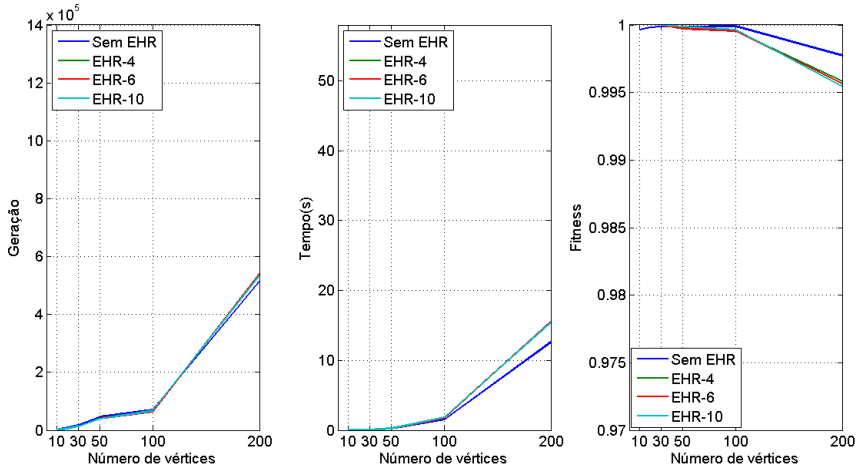


Figura 6.17: *População 10 e N_Π 100.*

Analisando os gráficos de *fitness* no pior caso, o AE-EPC obteve soluções com 99,41% de eficácia e o AE-PC 99,71%, ambos para grafos com 200 vértices e $N_\Pi = 30$. Essa taxa de eficácia indica que na maioria das 50 execuções a solução ótima foi encontrada. Com o aumento do N_Π para 50 e depois para 100, obteve-se uma melhora nos valores de *fitness* para todos os grafos onde a eficácia não havia atingido 100,00%. A variação da constante K afetou em, no máximo, 0,07% (50 vértices e PI de tamanho 30) o desempenho do AE-EPC. Observa-se que para essa topologia o uso o operador EHR não resultou em melhora nos valores de *fitness* obtidos. De fato o AE-PC obtém resultados com no máximo 0,33% (200 vértices e PI de tamanho 30) melhores que o AE-EPC. Ressalta-se que essa diferença é muito pequena, e que ambos os algoritmos são capazes de encontrar a solução ótima, no entanto o AE-PC obteve esse sucesso um maior número de vezes que o AE-EPC.

Com relação ao número de gerações, os resultados de ambos os algoritmos são similares para os grafos com até 50 vértices. Para os grafos com mais de 100 vértices e com N_Π superior a 50, o AE-EPC necessitou de um número maior de gerações que o AE-PC, isso deve-se ao fato de que o AE-EPC encontrou a solução ótima em um número menor de execuções e valeu-se do critério do número de gerações sem melhora como parada, enquanto que o AE-PC encerrou sua execução na solução ótima.

Visto que o tempo de execução está associado ao número de gerações, nos grafos com até 50 vértices, ambos os algoritmos tem executaram em tempos semelhantes, de 0 a 0,2 segundos em todos os casos e praticamente estável com a mudança do tamanho da PI. Já para os grafos maiores, novamente o AE-EPC demandou um pouco mais de tempo que o AE-PC.

6.2.2 População 20

Nesta Seção são descritos os resultados obtidos pelo AE-EPC com população de 20 indivíduos. Nas Figuras 6.18, 6.19 e 6.20, são apresentados gráficos dos valores de número de gerações, tempo de execução e *fitness* normalizados para os diferentes N_{Π}.

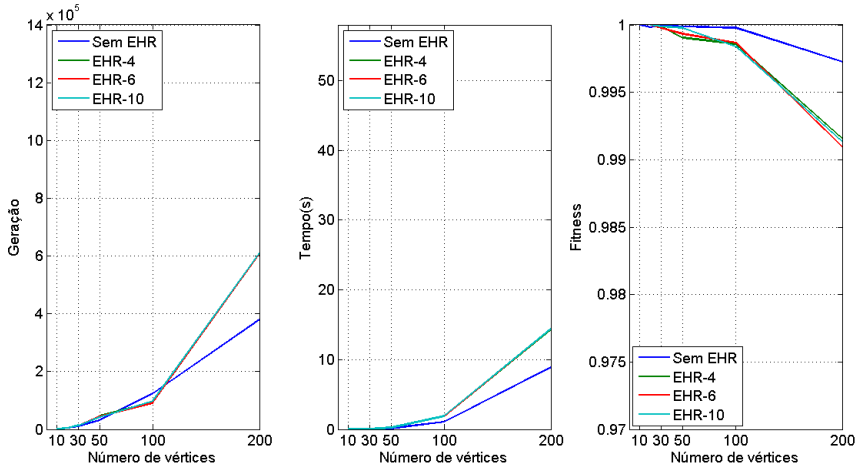


Figura 6.18: *População 20 e N_{Π} 30.*

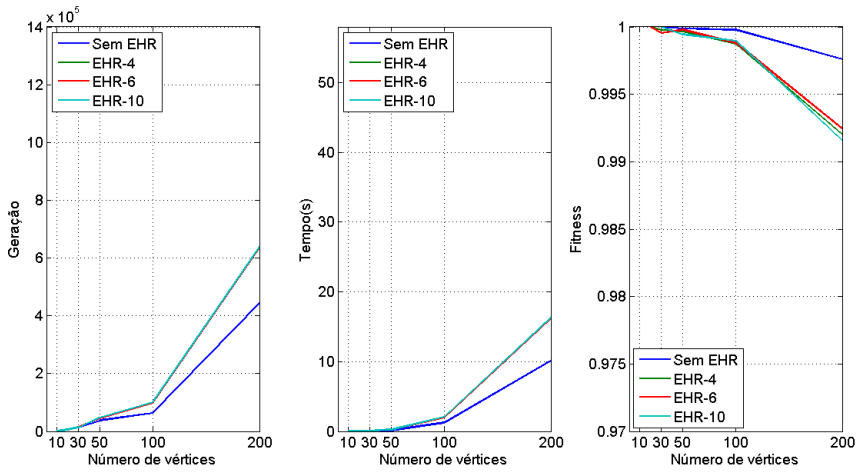


Figura 6.19: *População 20 e N_{Π} 50.*

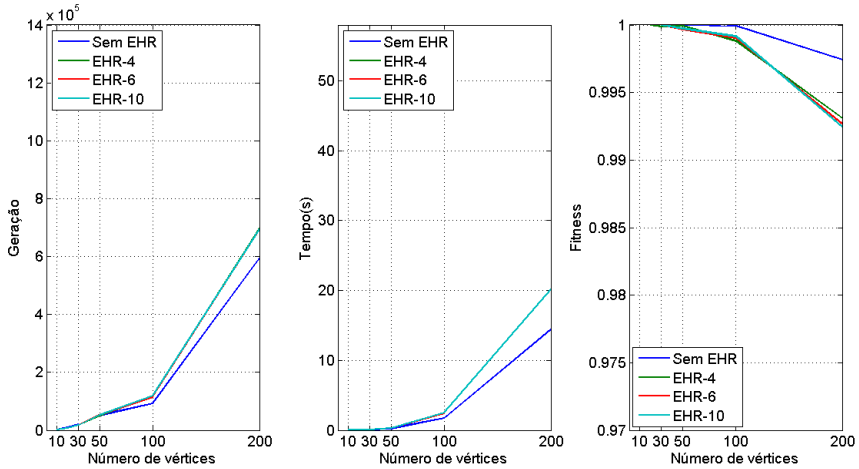


Figura 6.20: *População 20 e N_Π 100.*

Os gráficos dos valores de *fitness* apresentam resultados que, no pior caso, o AE-EPC obteve soluções que são 99,09% da solução ótima e o AE-PC que são 99,72%. Esses resultados foram obtidos para o maior grafo analisado com 200 vértices e $N_\Pi = 30$. Esses valores de *fitness* indicam que a solução ótima foi obtida na maioria das execuções. Com o aumento do N_Π para 50, houve discreta melhora na maioria dos casos onde a solução ótima não foi obtida em todas as execuções. Ao aumentar N_Π para tamanho 100, os resultados com o AE-EPC melhoraram ainda mais, em relação ao AE-PC que se manteve estável. A variação da constante K afetou em, no máximo, 0,08% o desempenho do AE-EPC. A maior vantagem do AE-EPC em relação ao AE-PC foi de 0,02% (20 vértices e PI de tamanho 30). Já o AE-PC teve um melhor resultado de 0,63% (200 vértices e PI de tamanho 30) em relação ao AE-EPC. Observa-se que apesar do melhor desempenho do AE-PC na maioria das instâncias analisadas, os resultados com o AE-EPC estão muito próximos do ótimo. De fato, ambos os algoritmos encontram a solução ótima, mas o AE-PC atingiu esse resultado em um número maior de execuções.

Para os grafos com até 50 vértices ambos os algoritmos obtiveram o mesmo número de gerações, em todas as configurações de parâmetros analisadas. No entanto, para os grafos com mais de 100 vértices o AE-PC precisou, em média, de menos gerações durante as suas execuções. Esse fato é um resultado direto do número de vezes em que a solução ótima é obtida. Como o AE-EPC encontra a solução ótima em menos execuções, o seu número de gerações é maior devido ao critério de parada. A variação do parâmetro K nos testes do AE-EPC não produziu alteração significativa no número de gerações. Com relação ao tempo de execução, observa-se que a análise comparativa entre os dois algoritmos é similar à análise feita para o número de gerações, uma vez que o tempo em segundos é uma consequência direta desse. Além disso, ressalta-se que ambos os algoritmos são eficientes, necessitando de poucos segundos para realizar a sua execução. Por exemplo, para os grafos com até 50 vértices o tempo de execução foi de

aproximadamente 0,3 segundos.

Em todos os casos até 100 vértices, o AE-EPC obteve *fitness* superior a 99,84% e superior a 99,09% com 200 vértices. Mesmo com o *fitness* abaixo do oponente, o resultado continua sendo de alto desempenho. As variações do parâmetro K e do N_{Π} influenciaram pouco nos resultados absolutos de tempo e *fitness*.

6.2.3 População 50

Nos parágrafos a seguir serão realizadas as análises dos resultados obtidos pelo AE-EPC e pelo AE-PC para a topologia de árvore estrela com população de 50 indivíduos. As Figuras 6.21, 6.22 e 6.23 ilustram os resultados na forma de gráficos para as variações testadas do parâmetro N_{Π}.

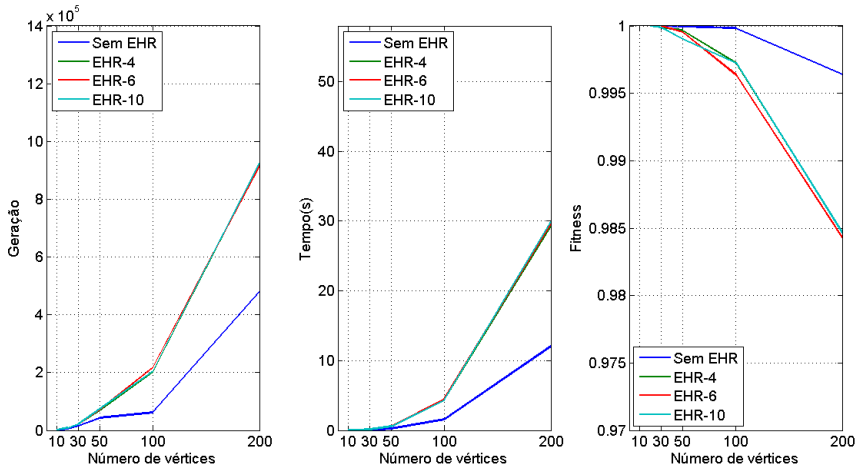


Figura 6.21: *População 50 e N_{Π} 30.*

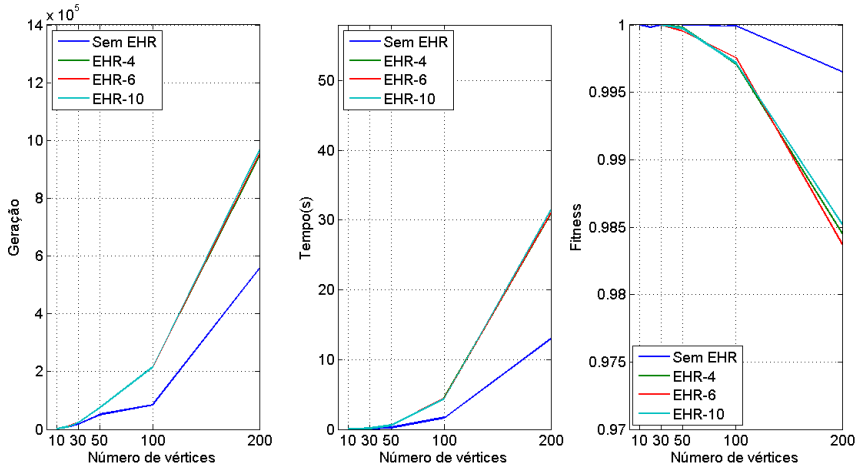


Figura 6.22: *População 50 e N_{Π} 50.*

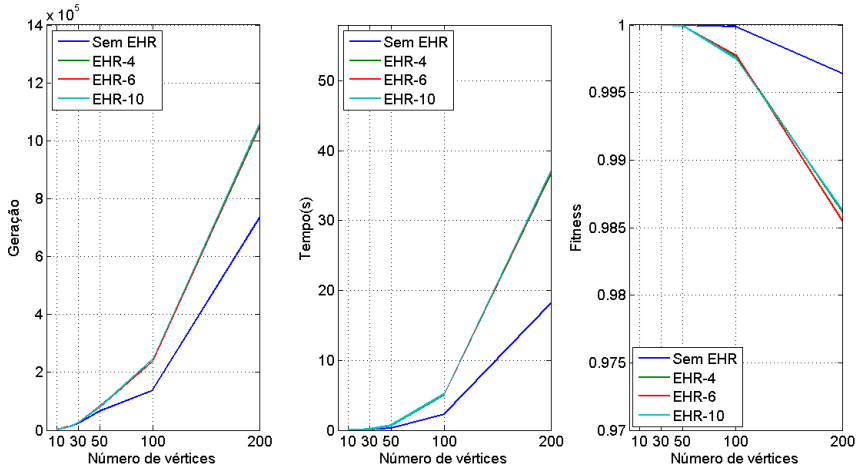


Figura 6.23: *População 50 e N_Π 100.*

Observa-se dos resultados apresentados para os valores de *fitness* que o aumento do tamanho da população para 50 indivíduos piorou o desempenho do AE-EPC e manteve o desempenho do AE-PC, quando comparado aos testes com população de 10 e 20 indivíduos. As soluções obtidas ainda podem ser consideradas boas, pois o ótimo continua sendo encontrado na maioria das execuções, mas o resultado reflete que nesse caso a acurácia do algoritmo é menor, principalmente para os grafos com mais de 50 vértices. O aumento do N_Π gerou discreto aumento de desempenho na maioria dos testes.

A variação da constante K no AE-EPC afetou o *fitness* em, no máximo, 0,10%. A maior vantagem do AE-EPC em relação ao AE-PC foi de 0,02% (20 vértices e PI de tamanho 50). Já o AE-PC teve como maior vantagem em relação ao AE-EPC 1,30% (200 vértices e PI de tamanho 50).

O número de gerações é similar para ambos os algoritmos em todos os casos com até 30 vértices. Em todos os testes, os aumentos do N_Π implicaram em leve crescimento do número de gerações, enquanto que o crescimento do número de vértices determinou um crescimento mais intenso do número de gerações. Novamente, vale ressaltar que esse crescimento deve-se à diminuição do número de execuções que obtém a solução ótima, que pode ser comprovado pela qualidade do *fitness*. A partir dos 100 vértices, o AE-PC, mesmo tendo menor número de gerações, foi mais influenciado pelos aumentos do N_Π do que o AE-EPC. Consequentemente, as diferenças entre eles caíram de 72,00% para 30,00%. O parâmetro K praticamente não influenciou no número de gerações.

Para os grafos com até 50 vértices, o tempo de execução máximo foi inferior a 0,6 segundos em todos os casos. Com 100 vértices, o AE-EPC demandou 2,9 segundos mais lento que o AE-PC em todos os testes. Com 200 vértices, houve aumento de tempo em todos os casos, porém, a diferença caiu de 58,00% para 51,00%, favorável ao AE-PC. As variações no tempo de execução têm relação direta com as variações no número de

gerações que são também consequência do número de vezes em que a solução ótima é obtida pelo algoritmo.

6.2.4 População 100

Finalmente, esta Seção aborda o desempenho do AE-EPC com o parâmetro tamanho da população de 100 indivíduos. Os resultados com a variação nos outros parâmetros são apresentados nas Figuras 6.24, 6.25 e 6.26.

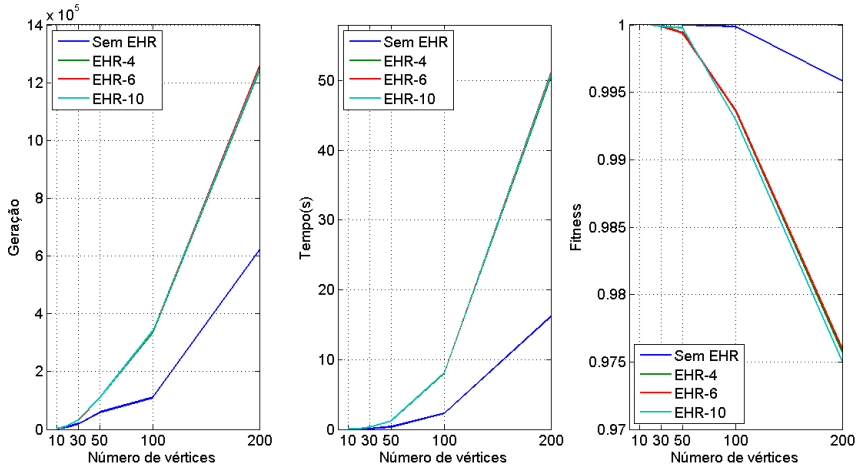


Figura 6.24: *População 100 e N_Π 30.*

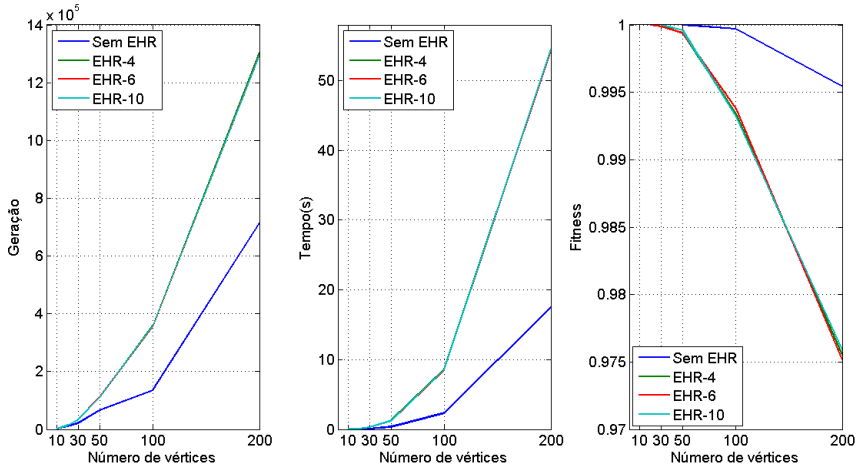


Figura 6.25: *População 100 e N_Π 50.*

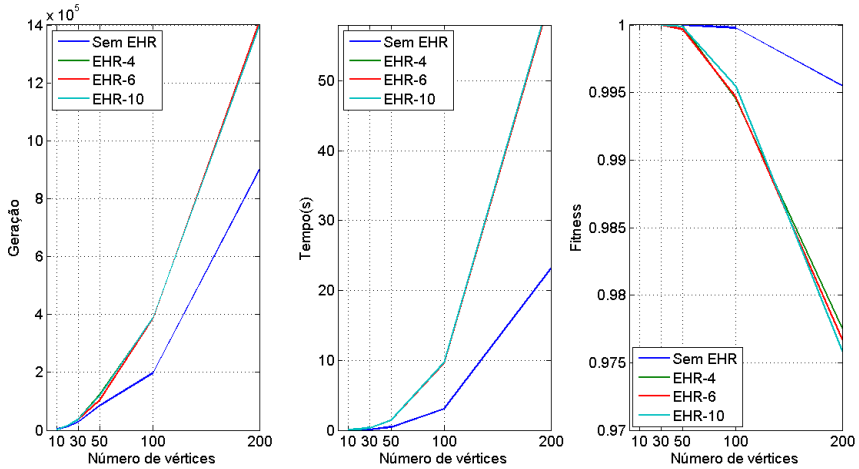


Figura 6.26: *População 100 e N_Π 100.*

O aumento da população para 100 indivíduos resultou em uma piora significativa no desempenho do AE-EPC em todos os critérios analisados. Por exemplo, o *fitness* com o AE-EPC foi de 97,50% para o pior caso e para o AE-PC foi de 99,54%. Isso mostra o AE-PC manteve seu desempenho constante, enquanto que AE-EPC piorou seu resultado, quando comparado aos outros tamanhos de população. O aumento do N_Π gerou discreto aumento de desempenho (máximo de 0,20%) em quase todos os testes. O AE-EPC sofreu mais o impacto do aumento do número de vértices que o AE-PC.

A variação da constante K do AE-EPC afetou o *fitness* em, no máximo, 0,10% (com 100 vértices e PI de tamanho 30). O parâmetro K também praticamente não influenciou no número de gerações.

O número de gerações utilizado pelos algoritmos é similar em todos as instâncias de grafos com até 30 vértices. Em todos os testes, o aumento do N_Π implicou em leve crescimento do número de gerações, enquanto que o aumento do número de vértices determinou um crescimento mais intenso do número de gerações. Isso deve-se ao fato de que ao aumentar o número de vértices os algoritmos encontraram a solução ótima em um número menor de execuções. A partir dos 100 vértices, o AE-PC, mesmo tendo menor número de gerações, foi mais influenciado pelos 2 tipos de aumento, N_Π e número de vértices, do que o AE-EPC. Ou seja, em ambos os casos, o AE-PC teve maior crescimento. Neste contexto, a diferença entre eles caiu de 72,00% para 30,00%.

Para os grafos com até 50 vértices, o tempo de execução máximo foi inferior a 0,6 segundos em todos os casos. Com 100 vértices, o AE-EPC foi cerca de 2,9 segundos mais lento que seu opositor em todos os testes. Com 200 vértices, houve aumento de tempo em todos os casos, porém, a diferença caiu de 58,00% para 51,00%, favorável ao AE-PC.

6.2.5 Conclusões parciais

As conclusões parciais acerca dos testes com topologia estrela estão descritas a seguir:

1. Pelos testes realizados, o *fitness* com o AE-EPC foi superior em apenas alguns testes com grafos menores do que 50 vértices;
2. No caso mais complexo (200 vértices), os melhores *fitness* para ambos os algoritmos ocorreram com $N_{\Pi} = 100$ (99,58%, AE-EPC e 99,78%, AE-PC);
3. O pior desempenho do AE-EPC foi com população de tamanho 100 e $N_{\Pi} = 30$. Contudo, ainda são resultados satisfatórios;
4. Em geral, o desempenho dos AE-EPC e AE-PC aumentou com o crescimento do N_{Π} para uma mesma população e diminuiu com o aumento da população;
5. As variações do parâmetro K (4, 6 e 10) afetaram discretamente os resultados e não foi possível observar uma relevância da alteração desse parâmetro;
6. O tempo de resposta (em segundos) do novo algoritmo foi igual ou próximo nos testes com até 100 vértices e população até 20 indivíduos;
7. Nas mesmas condições acima, o AE-EPC mostrou-se competitivo, sendo, em alguns pontos, até melhor que o AE-PC.

Pelos testes realizados, avaliando os critérios de qualidade do resultado (*fitness*), tempo de execução e número de gerações, podemos concluir que o uso do AE-EPC para topologia estrela só é indicado com população inferior a 20 indivíduos e para grafos com até 50 vértices. Para grafos com mais de 50 vértices o AE-PC é o mais indicado, por ter melhores valores de *fitness* com tempo inferior ao do AE-EPC.

6.3 Topologia Aleatória

A topologia aleatória de árvores geradoras, diferentemente das topologias linha e estrela, não possui um padrão no grau dos vértices. Assim, o conjunto é composto por qualquer árvore geradora que não segue os padrões de grau dos vértices das topologias anteriores. A Figura 6.27 apresenta um exemplo de árvore geradora que pode ser classificada como aleatória.

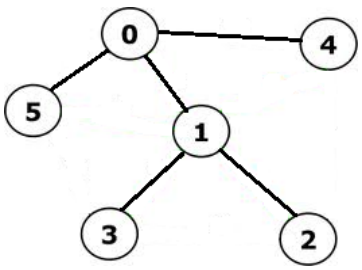


Figura 6.27: *Topologia Aleatória.*

Novamente, com o intuito de facilitar a análise dos resultados, os mesmos serão agrupados pelo parâmetro tamanho da população. Para os gráficos de valores de *fitness* é utilizada a escala com valores entre $0,915$ e $1,00$.

6.3.1 População 10

Esta Seção analisa os resultados obtidos pelo AE-EPC com população de 10 indivíduos aplicado a topologia aleatória. As Figuras 6.28, 6.29 e 6.30, apresentam os valores de número de gerações, tempo de execução e *fitness* normalizado para a $Matriz\Pi$ com tamanho 30, 50 e 100, respectivamente.

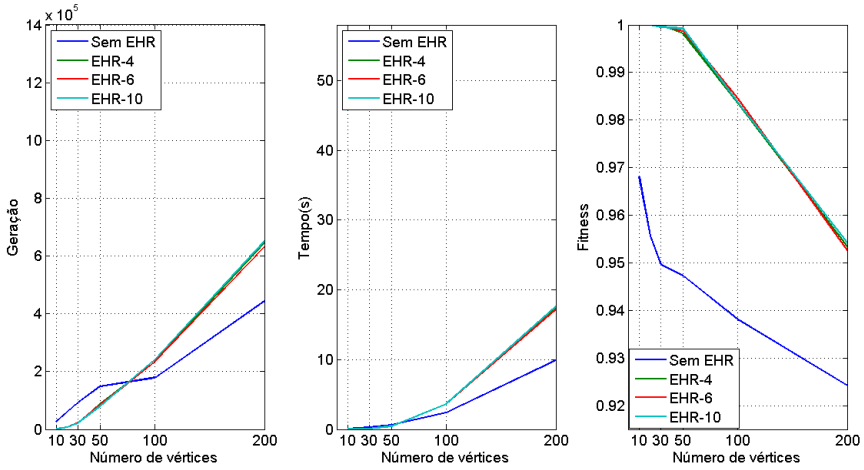


Figura 6.28: *População 10 e N_Π 30.*

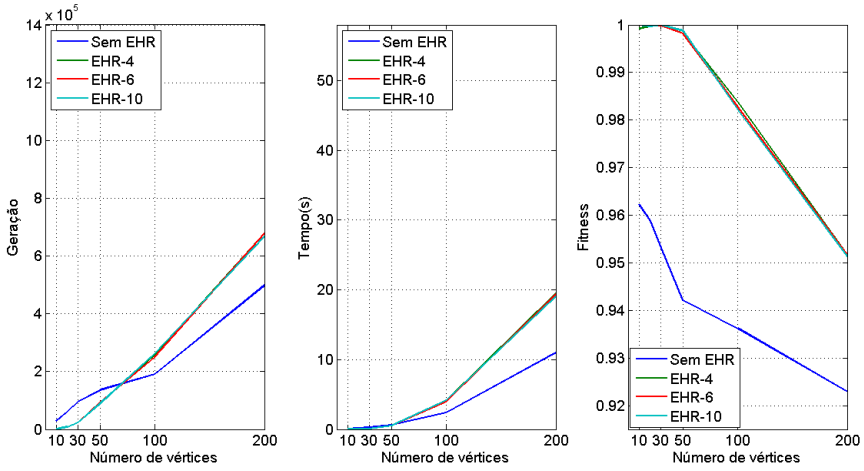


Figura 6.29: *População 10 e N_Π 50.*

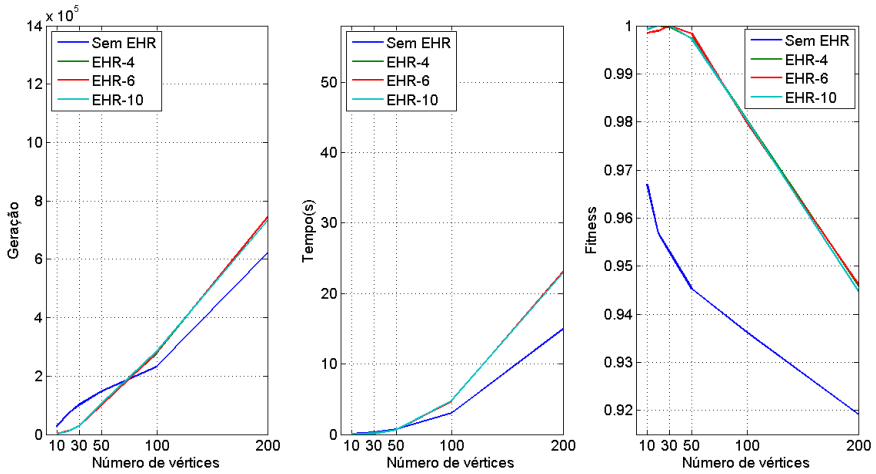


Figura 6.30: *População 10 e N_Π 100.*

De acordo com os gráficos apresentados, os valores de *fitness* com o AE-EPC são melhores do que os obtidos com o AE-PC em todos os experimentos e com eficácia superior a $99,70\%$ para os grafos com até 50 vértices. A maior vantagem de *fitness* do AE-EPC foi de $6,00\%$, para o grafo com 50 vértices e $N_\Pi = 50$. No pior caso, o resultado do AE-EPC ficou $2,80\%$ acima do obtido pelo AE-PC, com 200 vértices e $N_\Pi = 100$. A variação dos valores de K gerou uma diferença máxima de $0,10\%$ nos testes com o AE-EPC, isso mostra que esse parâmetro não influenciou o desempenho do algoritmo.

Para os grafos com até 50 vértices, observa-se que o AE-EPC demandou até $76,00\%$ menos gerações que o AE-PC, equivalente a aproximadamente 71.000 gerações a menos. No entanto, para os grafos de 100 e 200 vértices, a situação se inverteu, com o AE-EPC demandando até $46,80\%$ gerações a mais do que o AE-PC, nesses casos ambos os algoritmos utilizaram como critério de parada a o número de gerações sem melhora na maioria das execuções. Analisando conjuntamente com os valores de *fitness*, isso mostra que o AE-PC estabiliza com menos gerações do que o AE-EPC em um ótimo local inferior. O aumento do N_Π resultou em pequenos acréscimos no número de gerações para o AE-EPC e acréscimos um pouco mais acentuados para o AE-PC, principalmente para os grafos com mais de 50 vértices.

Quanto ao tempo de execução, o AE-EPC para grafos com até 50 vértices não apresentou diferença significativa em relação ao AE-PC. No entanto, para os grafos com mais de 50 vértices obtém-se o mesmo comportamento do número de gerações.

Uma característica comum é que, em todos os critérios avaliados, o AE-EPC foi melhor ou igual do que o AE-PC para os grafos com até 50 vértices. Além disso, para os demais grafos o AE-EPC obteve *fitness* superior em todos os experimentos.

6.3.2 População 20

Esta Seção analisa os resultados obtidos pelo AE-EPC com população de 20 indivíduos aplicado a topologia aleatória. As Figuras 6.31, 6.32 e 6.33, apresentam os valores de número de gerações, tempo de execução e *fitness* normalizados para a $Matriz\Pi$ com tamanhos 30, 50 e 100.

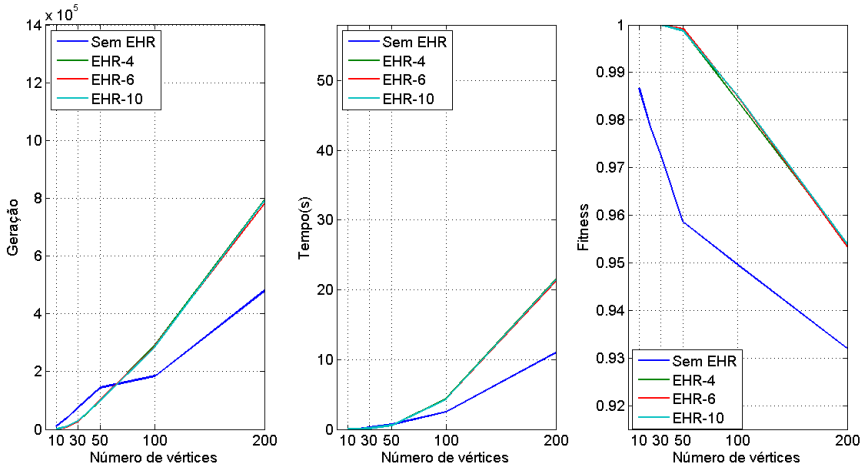


Figura 6.31: *População 20 e* N_Π *30.*

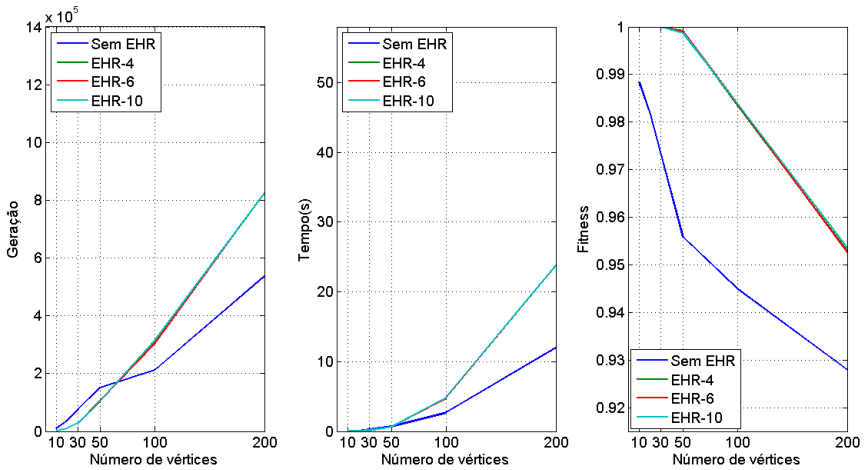


Figura 6.32: *População 20 e* N_Π *50.*

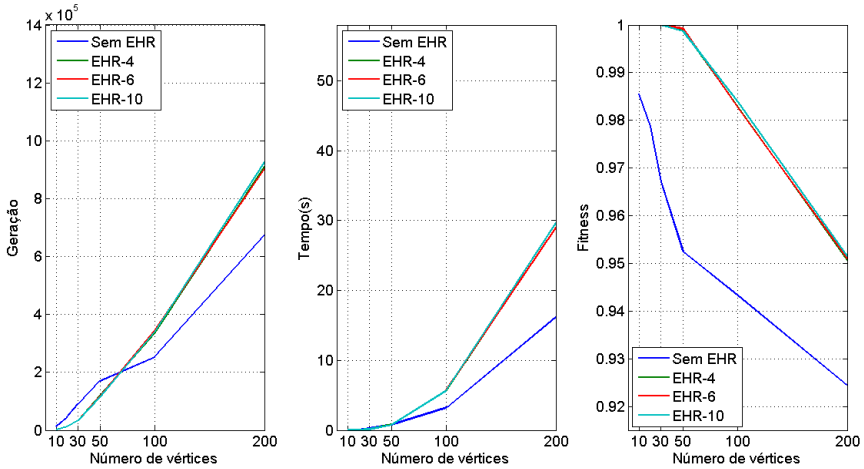


Figura 6.33: *População 20 e N_Π 100.*

Com o aumento do tamanho da população de 10 para 20 indivíduos, o AE-EPC encontrou a solução ótima em todos os testes para os grafos com até 30 vértices e para os grafos com 50 vértices a eficácia foi superior a 99,86%. A maior vantagem do AE-EPC em relação ao AE-PC foi de 4,90%, para o grafo com 50 vértices e $N_\Pi = 100$. No pior caso, o AE-EPC ficou 2,30% acima do AE-PC, com 200 vértices e $N_\Pi = 30$. A variação do parâmetro K não resultou em diferença significativa no desempenho do AE-EPC.

Assim como para a população de 10 indivíduos, o AE-EPC demandou menos gerações que o AE-PC para os grafos com até 50 vértices, sendo essa diferença de aproximadamente 50.000 gerações a menos. E, novamente, para os grafos com 100 e 200 vértices, a situação se inverteu. O aumento do N_Π gerou acréscimos no número de gerações para o AE-EPC e acréscimos um pouco mais acentuados para o AE-PC, principalmente após 50 vértices. Por consequência, a desvantagem máxima do primeiro caiu dos 65,00% para 37,00%. Observa-se que apesar do aumento no número de gerações, os valores de *fitness* do AE-EPC são superiores ao dos AE-PC.

Quanto ao tempo de execução, o AE-PC foi discretamente melhor com diferença de aproximadamente 0,12s para os grafos com até 50 vértices. Assim como ocorre para o número de gerações, o AE-EPC necessita de um tempo de execução superior ao do AE-PC para os grafos de 100 e 200 vértices.

6.3.3 População 50

Esta Seção analisa os resultados obtidos pelo AE-EPC com população de 50 indivíduos aplicado a topologia aleatória. As Figuras 6.34, 6.35 e 6.36, apresentam os valores de número de gerações, tempo de execução e *fitness* normalizados para a $Matriz\Pi$ com tamanhos 30, 50 e 100.

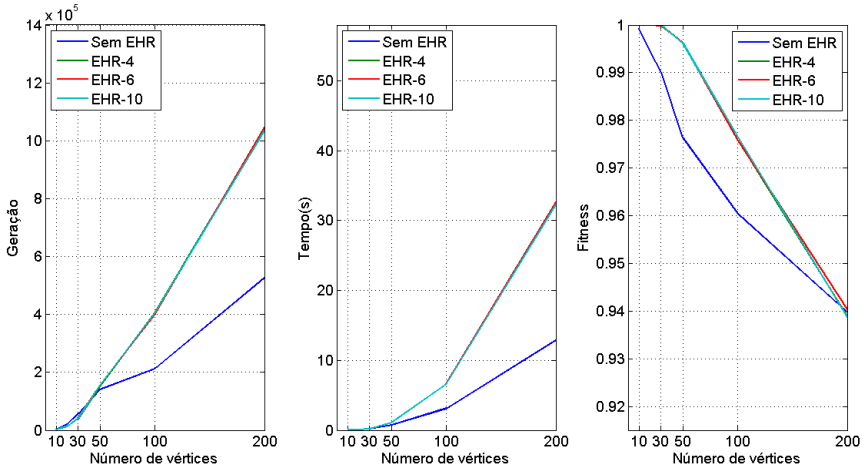


Figura 6.34: *População 50 e N_Π 30.*

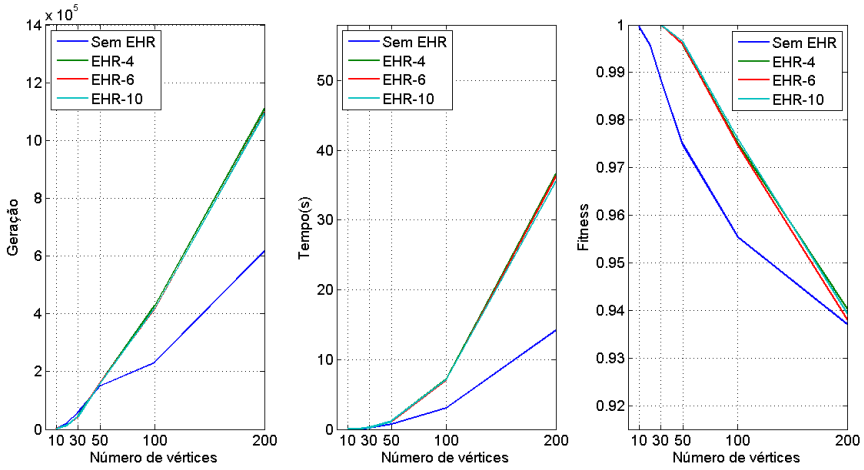


Figura 6.35: *População 50 e N_Π 50.*

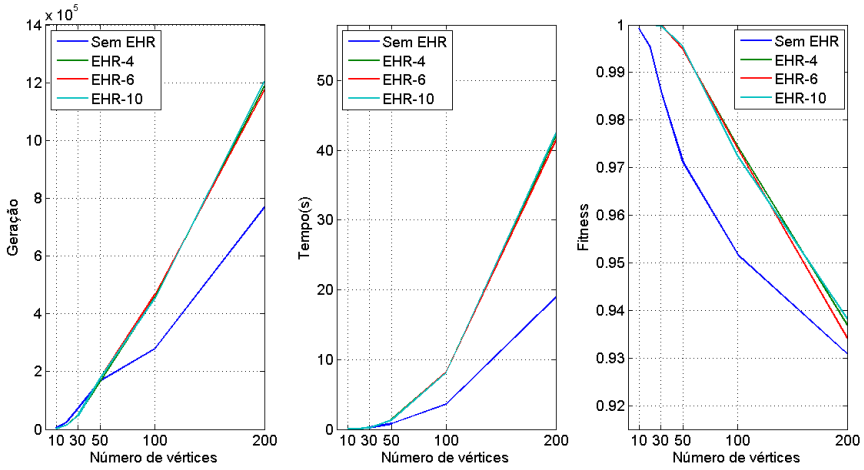


Figura 6.36: *População 50 e N_Π 100.*

Com o aumento da população para 50 indivíduos o AE-EPC obteve *fitness* superior ao AE-PC na maioria dos testes realizados, com exceção dos valores de K=4 e 10 para o grafo com 200 vértices e $N_{\Pi} = 30$, de acordo com os gráficos apresentados. A variação do parâmetro K do AE-EPC não influenciou significativamente os valores de *fitness* obtidos. Novamente o AE-EPC encontrou a solução ótima em todos os casos para grafos com até 30 vértices.

Analisando os gráficos de número de gerações observa-se um comportamento similar aos resultados obtidos com população de 10 e 20 indivíduos. No entanto, houve um aumento no número de gerações de ambos os algoritmos. A principal diferença em relação aos resultados anteriores é que o AE-EPC demandou menos gerações somente para os grafos com até 30 vértices, enquanto que nos testes anteriores esse resultado foi obtido para grafos com até 50 vértices.

Quanto ao tempo de execução, o AE-EPC foi semelhante ao AE-PC para os grafos com até 20 vértices. Com 30 vértices, ele começou a demandar maior tempo de execução. O aumento do N_{Π}, aumentou o tempo de execução de ambos os algoritmos, assim como também aumento o número de gerações.

6.3.4 População 100

Esta Seção analisa os resultados obtidos pelo AE-EPC com população de 100 indivíduos aplicado a topologia aleatória. As Figuras 6.37, 6.38 e 6.39, apresentam os valores de número de gerações, tempo de execução e *fitness* normalizados para a $Matriz\Pi$ com tamanhos 30, 50 e 100.

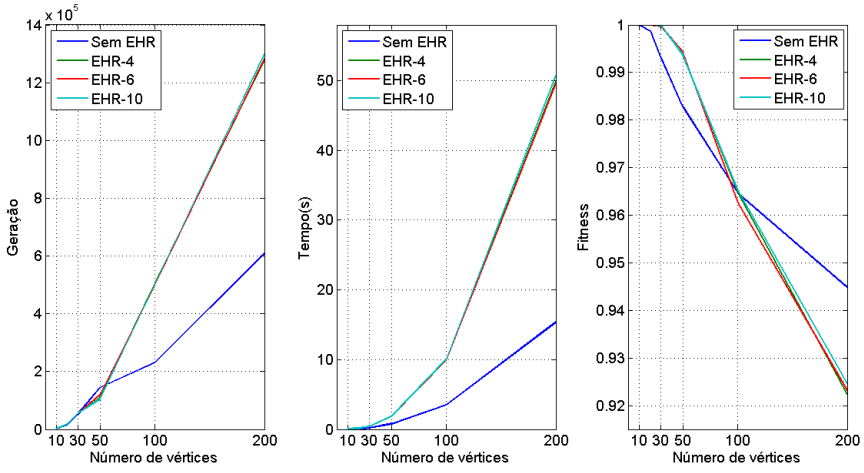


Figura 6.37: *População 100 e* N_{Π} *30.*

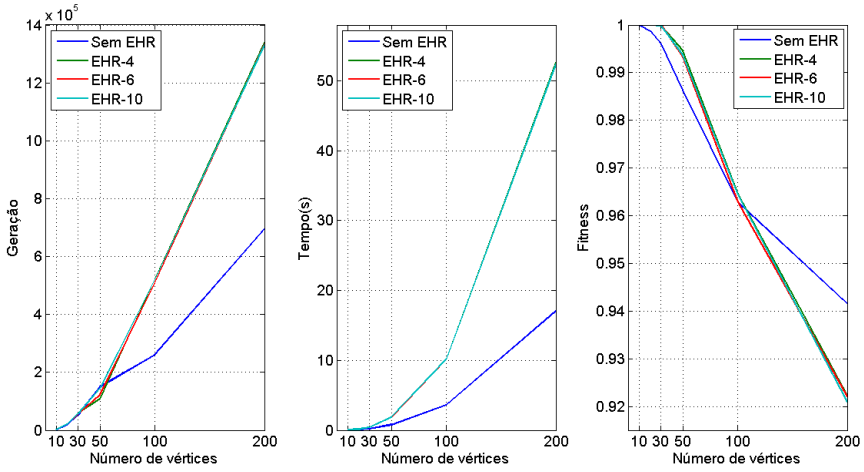


Figura 6.38: *População 100 e* N_{Π} *50.*

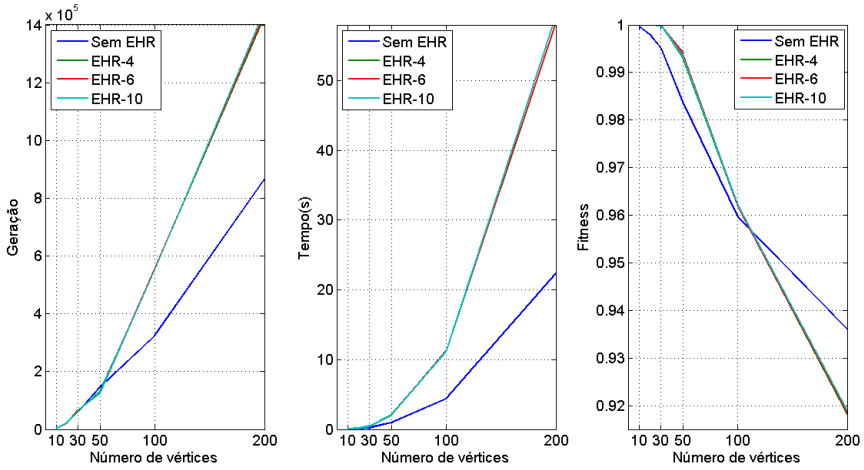


Figura 6.39: *População 100 e* N_{Π} *100.*

Com o aumento da população para 100 indivíduos, observa-se dos gráficos que o AE-PC teve uma melhora nos valores de *fitness* em relação aos valores obtidos com os outros tamanhos de população utilizados. Com essa melhora os resultados do AE-PC aproximaram-se dos obtidos pelo AE-EPC, porém esse continua com melhor desempenho na maioria dos testes até 100 vértices. O AE-EPC, manteve sua eficácia para os grafos com até 30 vértices, encontrando a solução ótima em todas os casos. Novamente, o parâmetro K não apresentou alteração significativa nos valores de *fitness* do AE-PC. Embora tanto o aumento do tamanho da matriz quanto o do número de vértices tenham diminuído os valores de *fitness* obtidos pelo AE-EPC, a segunda variável teve um impacto muito mais significativo que a primeira, pois aumenta a dificuldade do problema.

Nota-se que os gráficos de número de gerações para a população com 100 indivíduos apresentam três realidades distintas. Com grafos de até 30 vértices, ambos os algoritmos utilizaram número de gerações similares. Para os grafos com 50 vértices, o

AE-EPC utilizou um número menor de gerações, principalmente com N_Π de tamanho 30. E, para os grafos de 100 e 200 vértices, o AE-PC necessitou de menos gerações do que o AE-EPC. O aumento do tamanho da N_Π, influencia o número de gerações de ambos os algoritmos, porém para o AE-PC a diferença nos resultados é proporcionalmente superior à do AE-EPC.

O tempo de execução é diretamente dependente do número de gerações utilizado pelos algoritmos. Com isso, observa-se que o comportamento desse critério, ao variar os parâmetros dos algoritmos, é o mesmo obtido para o número de gerações. Vale ressaltar que, apesar do número gerações aumentar consideravelmente, o tempo de execução cresce mais lentamente. Assim como para o *fitness*, o parâmetro K do AE-EPC não influenciou os critérios de número de gerações e tempo de execução.

6.3.5 Conclusões parciais

As conclusões parciais acerca dos testes com topologia aleatória estão descritas a seguir:

1. Pelos testes realizados, os valores de *fitness* obtidos pelo AE-EPC foram superiores aos do AE-PC em todos os experimentos com população de até 20 indivíduos, exceto 2 casos. O aumento do tamanho da população após esse valor resultou em uma queda de desempenho do novo algoritmo;
2. No caso mais complexo, com 200 vértices, os melhores valores de *fitness* de ambos os algoritmos ocorreram com $N_\Pi = 30$;
3. Os valores de *fitness* do AE-PC melhoram com o aumento do tamanho da população e pioram com o aumento do N_Π;
4. A variação dos valores do parâmetro K do AE-EPC não resultou em variação significativa dos critérios analisados;
5. Além de obter melhores soluções, com população de até 20 indivíduos e para grafos com 50 vértices, o AE-EPC utilizou menor número de gerações ou foi equivalente ao AE-PC. Esse resultado também ocorreu para o critério de tempo de execução.

De acordo com os resultados obtidos para a topologia do tipo aleatório sugere-se para a aplicação do AE-EPC a utilização da população com 20 indivíduos e $N_\Pi = 30$. Essa sugestão tem como base a qualidade das soluções obtidas pelo algoritmo e o menor tempo de execução.

6.4 Considerações finais

Nas seções anteriores os gráficos contendo os valores de *fitness* foram apresentados com escala específica para cada topologia, a fim de mostrar em detalhes as diferenças

entre seus respectivos os resultados. A Figura 6.40 apresenta os gráficos com valores de *fitness* de cada uma das topologias para a configuração de parâmetros indicados nas conclusões parciais das respectivas seções. Observa-se da Figura 6.40 que, mesmo quando o AE-EPC não apresenta desempenho superior ao AE-PC, a diferença é muito pequena. Assim, concluí-se que o uso do operador EHR resultou em melhor desempenho quando foi adicionado a representação Nó-Profundidade-Grau.

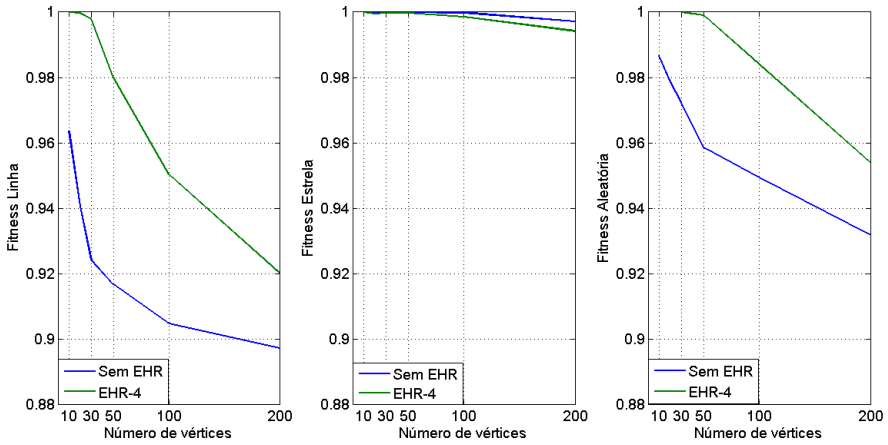


Figura 6.40: *Fitness das configurações indicadas: Linha, com população 20 e N_Π 50; Estrela, com população 10 e N_Π 30; Aleatória, com população 20 e N_Π 30.*

CAPÍTULO 7

Conclusões

Este trabalho abordou sobre algoritmos evolutivos e as principais linhas de pesquisa nesta área, apresentou algumas classes de problemas muito relevantes para os dias atuais, com destaque para os problemas de projetos de redes. Essa categoria de problemas envolve controle de tráfego, de rotas, de distribuição de dados, energia ou mercadorias ou outros problemas de otimização que podem ser modelados em forma de grafos.

Com base na complexidade dos problemas de projeto de redes e dos algoritmos para resolvê-los em tempo satisfatório, muitas pesquisas vêm sendo desenvolvidas sobre esse assunto. Uma delas aborda a utilização da Representação Nó-Profundidade-Grau (RNPG) aliada a mecanismos de recombinação, como, por exemplo, o EHR. As representações são estruturas de dados especiais que possibilitam uma codificação mais eficiente do problema de forma a reduzir o custo de tempo computacional para a manipulação das soluções pelo algoritmo. Ao modelar o Problema da Árvore Máxima para RNPG com EHR (AE-EPC), foi necessário fazer vários ajustes no algoritmo que havia sido elaborado para trabalhar com florestas e não com uma só árvore. Essas modificações tem por objetivo garantir a factibilidade das soluções e a complexidade de tempo em problemas com uma única árvore geradora.

Um grande diferencial da RNPG e do algoritmo do operador EHR é que os vértices envolvidos nos processos de poda e enxerto das subárvore são validados antes de aplicar as mutações ou as recombinações. Assim, os operadores não produzem árvores infactíveis. Por conseguinte, pode-se assegurar que todas as soluções são válidas, sem ter que testá-las posteriormente. Além do ganho em confiabilidade, esse processo de validação prévia evita gasto de tempo computacional com a aplicação de operações desnecessárias e posterior validação da solução.

Neste trabalho foram desenvolvidos dois algoritmos evolutivos para o OMTP. O AE-PC que usa somente da RNPG com os operadores de mutação PAO e CAO. O AE-EPC, além dos operadores do AE-PC, inclui o operador EHR.

Entre as melhorias do AE-EPC, vale destacar:

1. Os operadores PAO, CAO e EHR foram modificados para trabalharem com problemas cuja solução é representada por apenas (01) uma árovre;
2. Foi implementada uma rotina que identifica soluções (indivíduos) equivalentes, a qual os despreza por não agregarem melhorias nos resultados.

Durante a fase de testes, o AE-PC e o AE-EPC passaram por centenas de experimentos até que todas as partes dos algoritmos fossem ajustadas e fossem definidas as configurações a serem utilizadas nos testes finais. Finalizado o AE-EPC, foi feita uma nova e longa bateria de testes que contemplou: 3 topologias de árvores, 6 quantidades de vértices, 5 variedades de árvores de referência (soluções ótimas), 3 tamanhos para a *matrizPI*, 5 variações da constante K e 3 tamanhos de populações iniciais. Cada configuração foi executada 50 vezes. Considerando o produto cartesiano destas configurações, o algoritmo foi executado no mínimo 270.000 vezes, sem contar os testes refeitos para chegar no segundo critério de parada, ou seja, quando não foi encontrada uma solução melhor nas últimas 100.000 tentativas.

Os testes mostraram resultados otimistas em quase todas as situações avaliadas. Para todas as topologias e tamanhos da *matrizPI*, o AE-EPC alcançou 100,00% de aproveitamento (árvore de referência) em, pelo menos, um ponto. Seu opositor não alcançou tal resultado nem na metade dos testes. Na topologia linha, que é a mais complexa, o AE-EPC obteve até $9,00\%$ de vantagem em relação ao AE-PC (População de 10 indivíduos, *matrizPI* de tamanho 30 e 30 vértices). No pior caso, o AE-EPC teve desempenho $2,40\%$ inferior ao seu opositor, com mínimo de $92,45\%$ de aproveitamento (População de 100 indivíduos, *matrizPI* de tamanho 30 e 200 vértices). Nas topologias aleatória, o AE-EPC também teve resultados superiores em quase todos os testes. Já na estrela, ele teve resultados inferiores ao AE-PC em até $2,10\%$, contudo, sempre teve acima de $97,50\%$ de sucesso. Os maiores resultados de ambos os algoritmos sempre ocorreram entre 10 e 20 vértices; os maiores picos de vantagem do AE-EPC sempre ocorreram entre 30 e 50 vértices; e os piores resultados de ambos ocorreram com 200 vértices.

Quanto ao número de gerações e ao tempo de resposta, o AE-EPC foi mais competitivo até 50 vértices. Após 100 vértices a desvantagem passou a ser mais latente.

Diante de tudo exposto neste trabalho e dos resultados dos experimentos realizados, dos quais centenas de detalhes não foram incluídos nesta dissertação, as modificações implementadas na RNPG e no EHR mostraram-se como uma boa solução para o problema de projeto de rede proposto, ou seja, o problema da árvore máxima.

Como trabalhos futuros resultados desta pesquisa propõem-se:

1. Aplicação do AE-EPC para os demais problemas de projeto de redes representados por uma única árvore;

2. Adaptação do EHR para a RNPG com a divisão do problema em $O(\sqrt{n})$ árvores (ver [9]);
3. Estudo de métodos para a construção do subconjunto s de aplicações do EHR;
4. Aplicação de heurísticas do problema na seleção das modificações que formam o subconjunto s;
5. Aumentar o conjunto de exemplos das árvores de referência e repetição da bateria de testes realizados.

Referências Bibliográficas

[1] ABDALLA, A.; DEO, N. **Random-tree diameter and the diameter-constrained mst.** *Int. J. Comput. Math.*, 79(6):651–663, 2002.

[2] BÄCK, T.; FOGEL, D. B.; (EDS). **Evolutionary Computation 1: Basic Algorithms and Operators**. IOP Publishing Ltd, 2000.

[3] 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*, 5(6):623–630, 2001.

[4] DARWIN, C. **A Origem das Espécies**. teste, 1859.

[5] 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*, p. 408–415, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

[6] DE, JONG, K. A. **Evolutionary Computation: A Unified Approach**. MIT Press, 2006.

[7] DEB, K.; AGRAWAL, S.; PRATAB, A.; MEYARIVAN, T. **A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II**. KanGAL report 200001, Indian Institute of Technology, Kanpur, India, 2000.

[8] 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*, 2723:634–635, 2003.

[9] DELBEM, A. C. B.; LIMA, T. W.; TELLES, G. P. **Efficient forest data structure for evolutionary algorithms applied to network design**. *IEEE Transactions on Evolutionary Computation*, 99:1–28, 2012.

[10] 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., editors, *Genetic and Evolutionary Computation – GECCO-2004, Part I*, volume 3102 de **Lecture Notes in Computer Science**, p. 678–687, Seattle, WA, USA, 26-30 June 2004. Springer-Verlag.

[11] DELBEM, A. C. B.; CARVALHO, A. C. P. L. F.; BRETAS, N. G. **A fast algorithm for generation of forests: application to distribution system reconfiguration**. *IEEE Porto Powertech*, 3:1–7, 2001.

[12] EIBEN, A. E.; SMITH, J. E. **Introduction to evolutionary computing.** Natural Computing Series. Springer, 2003.

[13] FOGEL, D. B. **Evolutionary Computation: Toward a New Philosophy of Machine Intelligence**. IEEE Press, 2006.

[14] FOGEL, L. **Autonomous automata**. *Industrial Research*, 4:14–19, 1962.

[15] 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*, p. 1–10, London, UK, 2001. Springer-Verlag.

[16] GOLDBERG, D. E. **The Design of Innovation: Lessons from and for Competent Genetic Algorithms**. Kluwer, Boston, MA, USA, 2002.

[17] GOLDBERG, D. E.; DEB, K.; THIERENS, D. **Toward a better understanding of mixing in genetic algorithms**. *Journal of the Society of Instrument and Control Engineers*, 32(1):10–16, 1993.

[18] GOLDBERG, D. E. **Genetic Algorithms in Search, Optimization and Machine Learning**. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[19] HOLLAND, J. **Adaptation in natural and artificial systems**. University of Michigan Press, 1975.

[20] JOHNSON, D. S.; LENSTRA, J. K.; KAN, A. H. G. R. **The complexity of the network design problem**. *Networks*, 9:279–285, 1978.

[21] JONG, K. A. D. **An analysis of the behavior of a class of genetic adaptive systems.** PhD thesis, University of Michigan, Ann Arbor, MI, USA, 1975.

[22] JULSTROM, B. A. **The blob code: A better string coding of spanning trees for evolutionary search**. In: Rothlauf, F., editor, *Representations and Operators for Network Problems (ROPNET 2001)*, p. 256–261, San Francisco, California, USA, 7 July 2001.

[23] JULSTROM, B. A. **Encoding bounded-diameter spanning trees with permutations and with random keys**. 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., editors, *Genetic and Evolutionary Computation – GECCO-2004, Part I*, volume 3102 de **Lecture Notes in Computer Science**, p. 1272–1281, Seattle, WA, USA, 26-30 June 2004. Springer-Verlag.

[24] 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*, p. 585–590, New York, NY, USA, 2005. ACM Press.

[25] JULSTROM, B. A.; RAIDL, G. R. **A permutation-coded evolutionary algorithm for the bounded-diameter minimum spanning tree problem**. In: Barry, A. M., editor, *GECCO 2003: Proceedings of the Bird of a FeatherWorkshops, Genetic and Evolutionary Computation Conference*, p. 2–7, Chigaco, 11 July 2003. AAAI.

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

[27] KNOWLES, J. D.; CORNE, D. **A new evolutionary approach to the degree-constrained minimum spanning tree problem.** *IEEE Trans. Evolutionary Computation*, 4(2):125–134, 2000.

[28] KOZA, J. **Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems)**. MIT Press, 1992.

[29] KRUSKAL, J. B. **On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem**. In: *Proceedings of the American Mathematical Society, 7*, 1956.

[30] MICHALEWICZ, Z.; FOGEL, D. **How to Solve It: Modern Heuristics**. Springer-Verlag New York, Inc., 2004.

[31] PALMER, C. C. **An approach to a problem in network design using genetic algorithms**. PhD thesis, Polytechnic University, Brooklyn, NY, USA, 1994.

[32] PALMER, C. C.; KERSHENBAUM, A. **An approach to a problem in network design using genetic algorithms**. *Networks*, 26:151–163, 1995.

[33] 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*, 10(2):108–123, April 2006.

[34] PRÜFER, H. **Neuer beweis eines satzes uber permutationen**. *Arch. Math. Phys.*, 1918.

[35] RAIDL, G. **An efficient evolutionary algorithm for the degree-constrained minimum spanning tree problem**. In: *Proceedings of the 2000 Congress on Evolutionary Computation CEC00*, p. 104–111, La Jolla Marriott Hotel La Jolla, California, USA, 6-9 2000. IEEE Press.

[36] RAIDL, G.; DREXEL, C. **A predecessor coding in an evolutionary algorithm for the capacitated minimum spanning tree problem**. In: Armstrong, C., editor, *Proceeding of 2000 Genetic and Evolutionary Computation Conference*, p. 309–316, 2000.

[37] RAIDL, G.; JULSTROM, B. A. **Edge-sets: An effective evolutionary coding of spanning trees**, 2001.

[38] RAIDL, G. R.; JULSTROM, B. A. **Edge sets: an effective evolutionary coding of spanning trees.** *IEEE Trans. Evolutionary Computation*, 7(3):225–239, 2003.

[39] RAIDL, G. R.; JULSTROM, B. A. **Greedy heuristics and an evolutionary algorithm for the bounded-diameter minimum spanning tree problem**. In: *SAC '03: Proceedings of the 2003 ACM symposium on Applied computing*, p. 747–752, New York, NY, USA, 2003. ACM.

[40] RECHENBERG, I. **Cybernetc solution path of an experimental problem**. *Library Translation*, 1122, 1965.

[41] RIDLEY, M. **Evolution**. Phoenix, 2001.

[42] ROTHLAUF, F. **Representations for Genetic and Evolutionary Algorithms**. Springer-Verlag, 2006.

[43] ROTHLAUF, F.; GERSTACKER, J.; HEINZL, A. **On the optimal communication spanning tree problem**, 2003.

[44] ROTHLAUF, F.; GOLDBERG, D. E.; HEINZL, A. **Network random keys: A tree representation scheme for genetic and evolutionary algorithms**. *Evolutionary Computation*, 10(1):75–97, 2002.

[45] ROTHLAUF, F.; TZSCHOPPE, C. **Making the edge-set encoding fly by controlling the bias of its crossover operator**. In: Raidl, G. R.; Gottlieb, J., editors, *Evolutionary Computation in Combinatorial Optimization – EvoCOP 2005*, volume 3448 de **LNCS**, p. 202–211, Lausanne, Switzerland, 30 March-1 April 2005. Springer Verlag.

[46] ROTHLAUF, F.; TZSCHOPPE, C. **On the bias and performance of the edge-set encoding**. Technical report, Fakultät für Betriebswirtschaftslehre - Uni-Mannheim, 2005.

[47] 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*, p. 143–152, London, UK, 2002. Springer-Verlag.

[48] SCHWEFEL, H.-P. **Evolutionsstrategie und numerische optimierung**. Technical report, Technische Universitat, 1975.

[49] SINGH, A.; GUPTA, A. K. **Improved heuristics for the bounded-diameter minimum spanning tree problem**. *Soft Comput.*, 11(10):911–921, 2007.

[50] SOAK, S.-M. **A new evolutionary approach for the optimal communication spanning tree problem**. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, E89-A(10):2882–2893, 2006.

[51] SOAK, S.-M.; CORNE, D.; AHN, B.-H. **A new evolutionary algorithm for spanning-tree based communication network design**. *IEICE Trans Commun*, E88-B(10):4090–4093, 2005.

[52] 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.

[53] SOARES, T. W. D. L. **Estruturas de dados eficientes para algoritmos evolutivos aplicados a projeto de redes**. Tese, Instituto de Ciencias Matematicas e de Computacao, Universidade de Sao Paulo, 2009.

[54] THOMPSON, E.; PAULDEN, T.; SMITH, D. K. **The dandelion code: A new coding of spanning trees for genetic algorithms**. *IEEE Transactions on Evolutionary Computation*, 11(1):91–100, February 2007.

[55] VOSS, S.; OSMAN, I. H.; ROUCAIROL, C. **Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization**. Kluwer Academic Publishers, Norwell, MA, USA, 1999.

[56] ZHOU, G.; GEN, M. **A note on genetic algorithms for degree-constrained spanning tree problems**. *Networks (USA)*, 30(2):91–95, 1997.

[57] ZHOU, G.; GEN, M. **A genetic algorithm approach on tree-like telecommunication network design problem**. *J. of the Operational Research Society*, 54(3):248–254, 2003.