Algoritmo de otimização bayesiano com detecção de comunidades

Marcio Kassouf Crocomo

Algoritmo de otimização bayesiano com detecção de comunidades

Marcio Kassouf Crocomo

Orientador: **Prof. Dr. Alexandre Cláudio Botazzo Delbem**

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

USP – São Carlos
Novembro de 2012

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados fornecidos pelo(a) autor(a)

Crocomo, Marcio Kassouf
C937a Algoritmo de otimização bayesiano com detecção de comunidades / Marcio Kassouf Crocomo; orientador Alexandre Cláudio Botazzo Delbem. -- São Carlos, 2012.
148 p.

Tese (Doutorado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2012.

1. Algoritmos de Estimação de Distribuição. 2. Computação Evolutiva. 3. Algoritmo de Otimização Bayesiano. I. Delbem, Alexandre Cláudio Botazzo, orient. II. Título.

À minha família, sempre presente.

Agradecimentos

A Francisco, Faridi, Lucas e Luziana, pelo carinho e apoio sempre disponíveis.

Ao Prof. Dr. Eduardo do Valle Simões, por ter me incentivado a ingressar no doutorado e, assim, seguir no caminho acadêmico.

Ao Prof. Dr. Zhao Liang pelas aulas sobre Redes Complexas que muito contribuíram para as ideias desenvolvidas durante o projeto.

Ao Programa de Pós-graduação de Ciências de Computação e Matemática Computacional (PG-CCMC) do Instituto de Ciências Matemáticas e de Computação (ICMC), pela oportunidade e pela estrutura oferecida para a realização do curso de doutorado. Agradeço aos professores e funcionários, dentre os quais destaco Ana Paula Sampaio Fregona, da Seção Técnico Acadêmica, e Diego Jesus Talarico Ferreira, do Serviço de Convênios, Bolsas e Auxílios.

Ao Prof. Dr. Cláudio Fabiano Motta Toledo e aos professores que participaram de minha banca de qualificação: Prof. Dr. Renato Tinós, Prof. Dr. Dorival Leão Pinto Junior e Prof. Dr. Leandro Nunes de Castro Silva, agradeço pelas importantes orientações e discussões sobre o trabalho que muito auxiliaram para a qualidade final do mesmo.

A todos os colegas do laboratório LCR pela companhia nestes últimos anos, pelas produtivas conversas sobre os tópicos da tese, e com os quais tive o prazer de trabalhar, em especial Jean Paulo Martins, Danilo Vasconcellos Vargas, Marcilyanne Moreira Gois, Antonio Helson Mineiro Soares, Daniel Rodrigo Ferraz Bonetti e Prof. Dr. Vinícius Veloso de Melo.

Ao grande amigo Paulo Henrique Ribeiro Gabriel, por sua companhia, pelas interessantes conversas sobre assuntos relacionados ao trabalho, recomendações importantes sobre as referências bibliográficas e pela imprescindível ajuda na utilização do programa LaTeX utilizado na escrita da tese. Ao também amigo Eduardo Alves da Silva Ticianelli, pela companhia e presença de espírito nas horas necessárias.

Ao meu orientador Prof. Dr. Alexandre Cláudio Botazzo Delbem, com quem tive o prazer de trabalhar nestes últimos anos. Agradeço a confiança em mim depositada e toda atenção despendida a este projeto. Agradeço também pelos valiosos ensinamentos e pelas longas e produtivas conversas e planejamentos que conduziram aos resultados aqui apresentados. Por sua amizade e dedicação, muito obrigado.

Aos meus tios Sylvio e Emelie e à minha prima Julia, pela ajuda com as revisões de textos em inglês, resultantes deste trabalho. Agradeço também, pelo mesmo motivo, a Profa. Angela Cristina Pregnolato Giampedro, do Centro Cultural da USP em São Carlos.

À Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), pela concessão da bolsa de doutorado e pelo apoio financeiro para a realização desta pesquisa (Processo número 2010/01634-5), e também à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pela concessão da bolsa durante os 10 meses anteriores à bolsa FAPESP.

Resumo

ALGORITMOS de Estimação de Distribuição (EDAs) compõem uma frente de pesquisa em Computação Evolutiva que tem apresentado resultados promissores para lidar com problemas complexos de larga escala. Nesse contexto, destaca-se o Algoritmo de Otimização Bayesiano (BOA) que usa um modelo probabilístico multivariado (representado por uma rede Bayesiana) para gerar novas soluções a cada iteração. Baseado no BOA e na investigação de algoritmos de detecção de estrutura de comunidades (para melhorar os modelos multivariados construídos), propõe-se dois novos algoritmos denominados CD-BOA e StrOp. Mostra-se que ambos apresentam vantagens significativas em relação ao BOA. O CD-BOA mostra-se mais flexível que o BOA, ao apresentar uma maior robustez a variações dos valores de parâmetros de entrada, facilitando o tratamento de uma maior diversidade de problemas do mundo real. Diferentemente do CD-BOA e BOA, o StrOp mostra que a detecção de comunidades a partir de uma rede Bayesiana pode modelar mais adequadamente problemas decomponíveis, reestruturando-os em subproblemas mais simples, que podem ser resolvidos por uma busca gulosa, resultando em uma solução para o problema original que pode ser ótima no caso de problemas perfeitamente decomponíveis, ou uma aproximação, caso contrário. Também é proposta uma nova técnica de reamostragens para EDAs (denominada REDA). Essa técnica possibilita a obtenção de modelos probabilísticos mais representativos, aumentando significativamente o desempenho do CD-BOA e StrOp. De uma forma geral, é demonstrado que, para os casos testados, CD-BOA e StrOp necessitam de um menor tempo de execução do que o BOA. Tal comprovação é feita tanto experimentalmente quanto por análise das complexidades dos algoritmos. As características principais desses algoritmos são avaliadas para a resolução de diferentes problemas, mapeando assim suas contribuições para a área de Computação Evolutiva.

Abstract

ESTIMATION of Distribution Algorithms represent a research area which is showing promising results, especially in dealing with complex large scale problems. In this context, the Bayesian Optimization Algorithm (BOA) uses a multivariate model (represented by a Bayesian network) to find new solutions at each iteration. Based on BOA and in the study of community detection algorithms (to improve the constructed multivariate models), two new algorithms are proposed, named CD-BOA and StrOp. This paper indicates that both algorithms have significant advantages when compared to BOA. The CD-BOA is shown to be more flexible, being more robust when using different input parameters, what makes it easier to deal with a greater diversity of real-world problems. Unlike CD-BOA and BOA, StrOp shows that the detection of communities on a Bayesian network more adequately models decomposable problems, resulting in simpler subproblems that can be solved by a greedy search, resulting in a solution to the original problem which may be optimal in the case of perfectly decomposable problems, or a fair approximation if not. Another proposal is a new resampling technique for EDAs (called REDA). This technique results in multivariate models that are more representative, significantly improving the performance of CD-BOA and StrOp. In general, it is shown that, for the scenarios tested, CD-BOA and StrOp require lower running time than BOA. This indication is done experimentally and by the analysis of the computational complexity of the algorithms. The main features of these algorithms are evaluated for solving various problems, thus identifying their contributions to the field of Evolutionary Computation.

Conteúdo

Lista de Figuras

Lista de Tabelas

Dicionário de Siglas

AdClust	Agrupamento Adaptativo, do inglês *Adaptive Clustering*
aFA	Algoritmo Rápido adaptado, do inglês *adapted Fast Algorithm*
ASDP	Problemas Deceptivos Aditivamente Separáveis, do inglês *Additively Separable Deceptive Problems*
BB	Blocos Construtivos, do inglês *Building Blocks*
BBv	grupo de Variáveis do Bloco Construtivo, do inglês *Building Blocks variables*
BGS	Busca Gulosa do BOA, do inglês *BOA Greedy Search*
BN	Rede Bayesiana, do inglês *Bayesian Network*
BOA	Algoritmo de Otimização Bayesiano, do inglês *Bayesian Optimization Algorithm*
CD-BOA	Algoritmo de Otimização Bayesiana com Detecção de Comunidades, do inglês *Community Detection BOA*
CDA	Algoritmo de Detecção de Estrutura de Comunidades, do inglês *Community Detection Algorithm*
CGA	Algoritmo Genético Compacto, do inglês *Compact Genetic Algorithm*
CNF	Forma Normal Conjuntiva, do inglês: *Conjunctive Normal Form*
DE	Evolução Diferencial, do inglês *Differential Evolution*

DecOA Algoritmo de Otimização por Decomposição, do inglês *Decomposition Optimization Algorithm*

DOA Algoritmo de Otimização de Domínio, do inglês *Domain Optimization Algorithm*

EA Algoritmos Evolutivos, do inglês *Evolutionary Algorithms*

ECGA Algoritmo Genético Compacto Estendido, do inglês *Extended Compact Genetic Algorithm*

EDA Algoritmos de Estimação de Distribuição, do inglês *Estimation of Distribution Algorithm*

EO Otimização Extrema, do inglês *Extreme Optimization*

FA Algoritmo Rápido, do inglês *Fast Algorithm*

GA Algoritmos Genéticos, do inglês *Genetic Algorithms*

hBOA BOA hierárquico, do inglês *hierarchical BOA*

MAXSAT Problema de Satisfatibilidade Máxima, do inglês *Maximum Satisfiability Problem*

mohBOA BOA hierárquico, do inglês *multi-objective hierarchical BOA*

moStrOp Otimização Direta Multiobjetivo, do inglês *multi-objective Straight Optimization*

NA Número de Avaliações

NMP Número Máximo de vértices Pais

PhyDE Algoritmo Filogenético com Evolução Diferencial, do inglês *Phylogenetic Differential Evolution*

PhyGA Algoritmo Filogenético, do inglês *Phylogenetic Algorithm*

PSM *Population Size Management*

REDA Reamostragem para EDAs, do inglês *Resampling for EDAs*

RTR Torneio de Substituição Restrita, do inglês *Restricted Tournment Replacement*

SRA Algoritmo de Redução de Espaço de Busca, do inglês *Search-space Reduction Algorithm*

SRA Algoritmo de Redução de Espaço de busca, do inglês *Search-space Reduction Algorithm*

SS Amostragem Inteligente, do inglês *Smart Sampling*

StrOp Otimização Direta, do inglês *Straight Optimization*

TBO Taxa de Blocos Ótimos

TE Tempo de Execução

CAPÍTULO

1

Introdução

"Existe uma coisa que uma longa existência me ensinou: toda a nossa ciência, comparada à realidade, é primitiva e inocente; e, portanto, é o que temos de mais valioso"

Albert Einstein

Com o passar dos anos, os seres humanos desenvolvem novas tecnologias, sendo que parte delas é produzida com base em outras já existentes, criadas por seus antepassados. Esse processo permite que tecnologias mais condizentes com as necessidades humanas correntes sejam obtidas. Em computação, essa mesma ideia reflete-se no desenvolvimento de algoritmos, em que parte das propostas inovadoras é inspirada em ideias desenvolvidas por outros pesquisadores.

Por outro lado, um ramo da computação, chamado de Computação Bioinspirada (Castro, 2010; Floreano e Mattiussi, 2008), busca sua inspiração em mecanismos que existem até mesmo antes do surgimento da raça humana. A Computação Bioinspirada desenvolve técnicas para problemas de grande complexidade, inspirando-se em modelos biológicos como o cérebro humano, a evolução natural das espécies e o comportamento encontrado em colônias de animais,

como formigueiros. Exemplos de áreas da Computação Bioinspirada são Redes Neurais Artificiais (Braga *et al.*, 2007), Inteligência de Enxames (Engelbrecht, 2007) e Computação Evolutiva (De Jong, 2006). A Computação Evolutiva é a área de pesquisa que estuda o desenvolvimento dos Algoritmos Evolutivos (EAs, do inglês *Evolutionary Algorithms*). Esses algoritmos baseiam-se na teoria da evolução natural das espécies.

Na natureza, as espécies existentes possuem características herdadas de seus antepassados por meio do código genético. Em geral, os indivíduos mais adaptados ao meio ambiente deixam mais descendentes. Dessa forma, as características que contribuíram para que o indivíduo fosse mais apto são encontradas na nova geração de indivíduos com uma frequência maior do que as características de indivíduos que não conseguiram deixar muitos descendentes. Com o passar das gerações, e supondo um ambiente estacionário ou sujeito a poucas variações, existe uma tendência de que a população esteja cada vez mais adaptada ao ambiente. Essa é, resumidamente, a ideia central da teoria da evolução natural das espécies (Darwin, 1859).

Com o objetivo de encontrar soluções adequadas para problemas da computação, os EAs utilizam uma população composta por indivíduos que representam soluções candidatas a resolverem o problema. Uma função de avaliação – capaz de retornar o quão adequada é uma solução – é utilizada para selecionar os indivíduos mais promissores. Pela combinação das soluções selecionadas e de outras ações, que são chamadas de operações de reprodução, geram-se novos indivíduos. Esses compõem uma nova população, completando uma etapa evolutiva chamada de geração. A cada nova geração, existe uma tendência de que soluções melhores sejam encontradas.

Embora baseados nessa ideia simples, EAs têm sido desenvolvidos em diversas áreas da engenharia e ciências por se mostrarem capazes de resolver de forma eficiente problemas considerados complexos. Como exemplo das diversas áreas de aplicação nas quais EAs obtiveram sucesso, encontram-se sistemas de redes elétricas (Hadi e Rashidi, 2005; Santos *et al.*, 2010), robótica (Simões e Barone, 2002; Teo *et al.*, 2008), projetos de redes (Delbem *et al.*, 2012; Rajagopalan *et al.*, 2008), jogos (Baba e Handa, 2007; Crocomo, 2008), predição de estruturas de proteínas (Bonetti *et al.*, 2012; Brasil, 2012; Cotta, 2003) entre outras.

Embora EAs apresentem sucesso para uma grande variedade de problemas, pesquisadores começaram a deparar-se com casos em que o desempenho atingido pelos mesmos não era o esperado (Harik *et al.*, 1999; Rana e Whitley, 1998). Com o objetivo de melhorar tal desempenho, uma nova classe de EAs tem sido investigada: os Algoritmos de Estimação de Distribuição (EDAs, do inglês *Estimation of Distribution Algorithms*). Segundo estudos, os EDAs podem ser mais eficientes ou eficazes que EAs convencionais para vários tipos de problemas complexos (Goldberg, 2002). Diferente dos EAs convencionais, que em geral utilizam operadores de reprodução que sintetizam fenômenos genéticos (como crossover e mutação), a ideia

principal dos EDAs é a construção de um modelo probabilístico baseado nas melhores soluções existentes na população para, então, gerar as novas soluções a cada geração que são amostradas a partir das informações captadas pelo modelo. O uso do modelo possibilita focar nas formas de combinação de soluções que são mais promissoras, aumentando significativamente a capacidade de solução de um EDA em relação a um EA convencional. Dessa forma, o desempenho de um EDA é diretamente dependente da qualidade do modelo probabilístico gerado.

O Algoritmo de Otimização Bayesiana (BOA, do inglês *Bayesian Optimization Algorithm*) (Pelikan *et al.*, 1999) é um exemplo de um EDA bem sucedido, talvez o mais bem sucedido considerando-se a quantidade de problemas complexos que tem resolvido. Esse algoritmo utiliza Redes Bayesianas (Niedermayer, 2009) (BN, do inglês *Bayesian Networks*) para montar o modelo probabilístico utilizado. Proposto em (Pelikan *et al.*, 1999), o BOA é capaz de resolver problemas deceptivos de larga escala. Extensões desse algoritmo têm sido utilizadas para resolver um conjunto maior de problemas, como no domínio dos números reais (Ahn *et al.*, 2004), problemas hierárquicos (Pelikan, 2005; Pelikan e Goldberg, 2003) e multiobjetivos (Pelikan *et al.*, 2005).

No entanto, o BOA possui limitações, como por exemplo: i) a necessidade de parâmetro de entrada relativo ao conhecimento a priori sobre o problema sendo tratado, de forma a garantir sua eficiência; e ii) o alto custo computacional para a construção do modelo probabilístico utilizado, como apontado em (Aporntewan e Chongstitvatana, 2007). Este trabalho, por meio do uso de um Algoritmo de Detecção de Estruturas de Comunidades (CDA, do inglês *Community Detection Algorithm*) (Newman e Girvan, 2004), propõe um novo método que utiliza BN como modelo probabilístico, assim como o BOA, mas é capaz de superar limitações de desempenho do BOA. Dessa forma, o objetivo principal deste trabalho é enunciado como:

> Mostrar como CDAs podem contribuir no aumento de desempenho de EDAs baseados em Redes Bayesianas.

Para tanto, tornou-se necessário atingir objetivos específicos, dentre eles:

1. Desenvolvimento de um CDA recursivo, baseado na técnica *Fast Algorithm* (FA) (Newman e Girvan, 2004);
2. Proposta e implementação do Algoritmo de Otimização Bayesiano com Detecção de Comunidades (CD-BOA, do inglês: *Community Detection BOA*);
3. Extensão do CD-BOA para operar somente com uma amostragem do espaço de busca (apenas uma geração), desenvolvendo o Algoritmo de Otimização Direta (StrOp, do inglês *Straight Optimization*);

4. Criação de uma técnica de Reamostragem para EDAs (REDA, do inglês *Resampling for EDAs*), baseada em BNs e CDAs, capaz de melhorar a eficiência de EDAs em geral;

5. Realização de experimentos extensivos visando verificar: i) número de avaliações exigidas pelos algoritmos para a resolução de problemas computacionalmente complexos, ii) tempo de execução, iii) a robustez dos mesmos com relação a valores de parâmetros de entrada, além de iv) particularidades dos algoritmos para casos de testes específicos de forma a mapear adequadamente as propriedades de cada técnica.

A execução do último objetivo apresentado permitiu identificar as principais contribuições dos métodos desenvolvidos. Como exemplo, comparações das complexidades de tempo de etapas dos algoritmos possibilitaram a constatação de que o CD-BOA e o StrOp apresentam um menor tempo de computação que o BOA. Tal vantagem também foi verificada experimentalmente, chegando o StrOp a se mostrar até 30 vezes mais rápido que o BOA em alguns problemas de *benchmark* (Seção 5.4) que modelam características de problemas para os quais EAs convencionais não apresentam desempenho aceitável (Goldberg, 2002). Outros experimentos mostram uma maior eficiência do CD-BOA quando comparado ao BOA em problemas de *benchmark* que modelam características importantes de problemas hierárquicos (Seção 5.5), os quais formalizam aspectos presentes em problemas complexos do mundo real. Tal resultado indica que as estratégias utilizadas no CD-BOA também podem ser adaptadas à extensão do BOA dedicada a problemas hierárquicos (Pelikan e Goldberg, 2003), aumentando o desempenho na solução de problemas do mundo real. Por sua vez, nos experimentos aplicando o REDA, o CD-BOA e o StrOp apresentam melhorias significativas de seus desempenhos. O REDA possibilita que os EDAs resolvam os problemas utilizando um menor número de avaliações.

Os demais capítulos deste documento estão organizados como descrito a seguir: Os Capítulos 2, 3 e 5 reúnem a revisão bibliográfica realizada, apresentando os principais conceitos relacionados a EDAs (Capítulo 2); técnicas de detecção de comunidades e de reamostragem (Capítulo 3), cujo estudo é de grande importância na elaboração dos algoritmos propostos; além de problemas do mundo real e funções de *benchmark* para testes de EDAs (Capítulo 5). O Capítulo 4 descreve os novos métodos propostos neste trabalho. O Capítulo 6 mostra os experimentos e as análises de desempenho realizadas. Por fim, o Capítulo 7 registra as principais conclusões da tese e destaca perspectivas de extensões relevantes deste trabalho.

CAPÍTULO

2

Algoritmos de Estimação de Distribuição

"Já ouvi falar em cachorro de estimação, gato de estimação, cobra de estimação... Mas, algoritmo de estimação é a primeira vez!"

Eduardo Ticianelli, amigo curioso

Este Capítulo apresenta os principais conceitos relacionados aos EDAs, bem como o estado da arte sobre esses algoritmos. A Seção 2.1 introduz conceitos fundamentais sobre EAs. A Seção 2.2 descreve estratégias relacionadas a criação de soluções promissoras para um problema. A Seção 2.3 apresenta alguns conceitos sobre estruturas presentes em conjuntos de soluções. A Seção 2.4 aborda aspectos da dificuldade de um problema, considerando a existência de Blocos Construtivos (BBs, do inglês *Building Blocks*), além de ideias importantes para compreender o funcionamento dos EDAs. Por sua vez, os EDAs são explicados na Seção 2.5. A Seção 2.6 explora o EDA no qual se baseia a proposta: o Algoritmo de Otimização Bayesiano (BOA, do inglês *Bayesian Optimization Algorithm*). Por fim, a Seção 2.7 apresenta variações existentes do BOA desenvolvidas para tratar uma diversidade maior de problemas.

2.1 Algoritmos Evolutivos

Os EAs são baseados em aspectos presentes na Teoria da Evolução Natural das Espécies (Darwin, 1859). Basicamente, dado um conjunto inicial de soluções, chamado de população, este tende a evoluir até convergir para soluções aceitáveis dentro do espaço de busca considerado (conjunto formado por todas as potenciais soluções para o problema) (Gaspar-Cunha *et al.*, 2012). Tal tarefa é realizada por operadores de evolução que, em geral, correspondem a dois procedimentos: recombinação e mutação (Goldberg, 1989). Esses operadores são aplicados a cada elemento da população, os quais são denominados de cromossomos (também conhecidos como indivíduos). Um cromossomo pode ser representado por uma *string* de símbolos (por exemplo: *string* binária). Cada *string* é tratada como se fosse o próprio indivíduo. Em geral, cada elemento do cromossomo, chamado de gene, corresponde a uma característica (ou variável) do problema. Quando uma *string* é decodificada ("traduzida"), o resultado é uma possível solução (ou solução candidata)[1] para um problema. Assim, cada nova *string* equivale a uma nova solução proposta. Em linhas gerais, um EA possui os seguintes passos:

1. Uma população inicial é criada;
2. As soluções da população atual são avaliadas por meio de uma função de aptidão, que atribui a cada solução uma pontuação (*fitness*) indicando o quão adequada é a mesma;
3. Um procedimento de seleção é aplicado para escolher os indivíduos a serem usados na criação das novas soluções;
4. Operadores de reprodução são utilizados para criar uma nova população de soluções a partir das soluções selecionadas no Passo 3;
5. Retorno ao Passo 2, executando o algoritmo repetidamente (de forma iterativa) até que alguma condição de parada seja atendida.

No Passo 1, a população criada inicialmente é normalmente aleatória. No entanto, é possível incluir soluções com tendência (*bias*) de características que supostamente são mais adequadas, caso exista conhecimento prévio sobre o problema sendo otimizado. No Passo 2, é utilizada uma função de aptidão para avaliar cada solução existente na população. Dessa forma, a cada solução é atribuída uma pontuação chamada de *fitness*. No Passo 3, um mecanismo de seleção é utilizado

[1]Para facilitar a leitura, a palavra *solução* é utilizada neste trabalho significando *solução candidata*. Para se referir a uma boa solução do problema, são usados termos específicos como *solução ótima, solução encontrada*, entre outros.

para escolher os indivíduos usados na criação de novas soluções que irão compor a próxima geração. Abaixo, encontram-se explicados alguns mecanismos alternativos de seleção(De Jong, 2006):

- **Seleção Proporcional**: Cada indivíduo possui uma probabilidade de ser selecionado relativa a seu próprio *fitness*, definida por f_i / f_{pop}, em que f_i é o *fitness* da própria solução e f_{pop} é a soma de todos os *fitness* da população. Com base nessa probabilidade, são selecionados todos os indivíduos necessários para o próximo passo do algoritmo;
- **Torneio de j indivíduos**: São sorteados, com igual probabilidade, j indivíduos da população. O indivíduo selecionado é o que possui maior *fitness* entre os sorteados. O processo é repetido, sempre tomando toda a população, até que todos os indivíduos necessários para o próximo passo sejam selecionados;
- **Seleção por truncamento**: São selecionados os w indivíduos de maior *fitness*, em que w é o número de indivíduos necessários para o próximo passo do algoritmo.

O Passo 4 aplica mecanismos de reprodução que garantem que qualquer ponto do espaço de busca possa ser atingido. A seguir, encontram-se explicados dois dos mecanismos mais comuns: operadores de recombinação e de mutação.

Operadores de recombinação: São responsáveis pela criação de novas soluções a partir da combinação de dois ou mais indivíduos selecionados. Um exemplo de operador de recombinação é o crossover de 1 ponto, que gera novas soluções a partir de dois indivíduos selecionados e da definição aleatória de um ponto de corte, quebrando o cromossomo (*string*) em duas subcadeias de *bits*. As soluções resultantes são a permutação das subcadeias definidas por este ponto entre os indivíduos selecionados. A Figura 2.1 ilustra esse processo para cromossomos de 6 *bits*.

Outros operadores de recombinação podem ser encontrados. Dentre os mais comumente usados, podem ser destacados: o crossover de dois pontos, em que dois pontos de corte são definidos ao invés de um, trocando-se a subcadeia entre esses pontos; e o crossover uniforme, em que cada gene possui uma dada probabilidade de provir de uma das duas soluções selecionadas (Bäck *et al.*, 2000).

Mutação: A mutação é responsável por inserir modificações em um ou mais genes de uma solução que irá compor a próxima geração. Diversos operadores de mutação podem ser encontrados na literatura (De Jong, 2006; Fogel, 2005). Cada gene possui uma probabilidade, de ter seu valor alterado. Essa probabilidade é chamada de taxa de mutação. A forma como o valor é modificado depende da representação do cromossomo. Se os genes são representados por

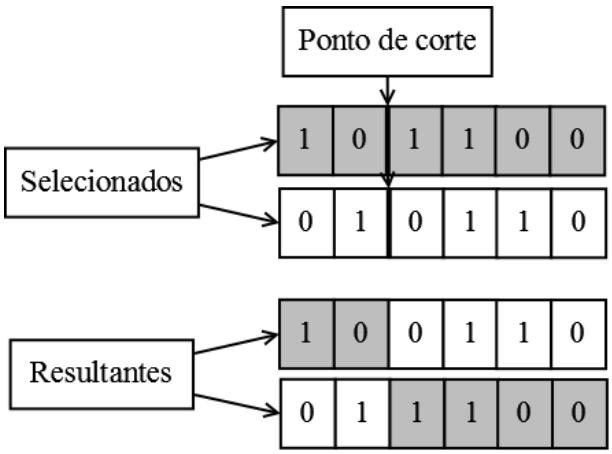


Figura 2.1: Exemplo de crossover de 1 ponto.

valores reais, um possível operador de mutação pode adicionar ao gene um valor obtido com base em uma distribuição probabilística (Bäck *et al.*, 2000). Para soluções representadas por valores binários, um operador de mutação pode inverter o valor de um *bit*. A Figura 2.2 ilustra um exemplo de mutação em uma solução representada por uma *string* binária.

O Algoritmo 2.1 mostra o pseudocódigo típico de um EA. Diversos critérios de parada (Passo 3 do Algoritmo 2.1) podem ser utilizados, por exemplo, a convergência dos valores das soluções [2], ou porcentagem de soluções que supostamente correspondem a valores ótimos (Pelikan *et al.*, 1999).

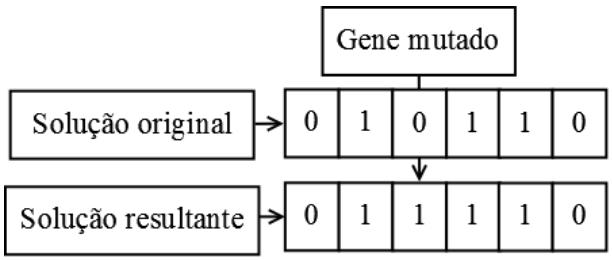


Figura 2.2: Exemplo de mutação em uma *string* binária.

Existem diversos tipos de EAs, entre os primeiros desenvolvidos estão os chamados EAs canônicos, como os Algoritmos Genéticos (GAs, do inglês *Genetic Algorithms*) (Goldberg, 1989), Estratégias Evolutivas (Beyer e Arnold, 2001) e Programação Evolutiva (Bäck, 1996; Fogel, 2005). Desses, estima-se que os GAs sejam os mais difundidos, com resultados relevantes em diversas áreas de aplicação na literatura (Cole *et al.*, 2004; Moura *et al.*, 2010; Ogata,

[2] Ao utilizar o critério de convergência, o algoritmo se repete até que cada variável possua o mesmo valor em pelo menos uma determinada quantidade de soluções.

Algoritmo 2.1: Um Algoritmo Evolutivo típico.

Entrada: Uma população P_0 vazia;
Saída: Uma população P_g contendo soluções promissoras;

```
1 g = 0;
2 InicializaPopulação(P_g);
  // avalia a aptidão dos indivíduos da população P
3 AvaliaPopulacao(P_g);
4 enquanto critério de parada não atingido faça
  |  // incrementa o contador da geração
5 |  g = g+1;
  |  // seleciona os indivíduos para a geração dos
  |     descendentes
6 |  S = Seleção(P_{g-1});
  |  // são aplicados os operadores reprodução sobre S,
  |     gerando a nova população
7 |  P_g = Reprodução(S);
  |  // avalia a aptidão dos indivíduos da população
8 |  AvaliaPopulacao(P_g);
9 fim
```

2006; Pessin *et al.*, 2010). Estudos mais recentes têm mostrado que EDAs podem apresentar resultados significativamente melhores que os GAs convencionais para diversos problemas complexos (Goldberg, 2002). A Seção 2.4 introduz conceitos importantes para se entender a questão central que motiva o desenvolvimento de EDAs e, em seguida, a Seção 2.5 explica o funcionamento desses algoritmos.

2.2 Gerando soluções promissoras

Para melhor entender o porque de se desenvolver EDAs, é interessante uma melhor compreensão da tarefa que se espera que um EA faça. Para essa finalidade, Goldberg (2002) propõe, de uma maneira didática, observar o problema sendo otimizado do ponto de vista de um EA.

Considere o problema de encontrar soluções com maiores valores de *fitness*, a partir de uma amostra de soluções avaliadas, dada pela Tabela 2.1. Como gerar novas soluções que tenham melhores valores de *fitness*? É importante lembrar que as informações da Tabela 2.1 são geralmente, toda a informação fornecida sobre o problema para o EA.

Uma abordagem, que utiliza operadores de crossover, pode ser a de combinar partes das boas soluções. Observe que utiliza-se o termo "boa" no sentido de promissora, sem garantias de que é a melhor ou que seja melhor que as soluções candidatas ainda não avaliadas. Por

Tabela 2.1: População de um EA.

solução	x_1	x_2	x_3	x_4	*fitness*
s_1	0	0	0	1	15
s_2	1	0	1	1	13
s_3	0	0	0	0	10
s_4	1	1	1	1	8

exemplo, pode-se argumentar que uma nova solução 0011 é uma boa candidata, por combinar os dois primeiros valores da solução s_1 com os dois últimos valores da solução s_2 (ambas com bons valores de *fitness*). Alternativamente, várias soluções candidatas podem ser propostas da mesma forma, selecionando diferentes trechos de diferentes soluções. No entanto, as soluções geradas só serão boas se os trechos selecionados de cada solução forem realmente responsáveis por boas contribuições aos valores de *fitness*.

Essa linha de raciocínio considera a existência de subestruturas, responsáveis por contribuições positivas ou negativas para o *fitness* da solução. Para melhor se referir a essas subestruturas, a Seção 2.3 apresenta algumas definições da literatura.

2.3 Esquemas, partições e blocos construtivos

Um esquema (Holland, 1975) representa um subconjunto de *strings* que possuem valores iguais em algumas posições. É possível representar um esquema por meio de um modelo de similaridade, que é uma *string* formada pelo alfabeto utilizado mais o caractere "*", responsável por representar qualquer caractere existente no alfabeto. Como exemplo, o esquema $11**$ representa o conjunto de soluções 1100, 1101, 1110, 1111.

A Seção 2.2 apresentou uma linha de raciocínio que supõe subestruturas responsáveis por contribuições ao *fitness* de uma solução. São chamadas partições, os subconjuntos de variáveis que identificam tais subestruturas. Como exemplo, considere o problema composto por quatro variáveis (x_1, x_2, x_3, x_4). Se os valores das duas primeiras variáveis são uma subestrutura e os valores das duas últimas variáveis são outra subestrutura, pode-se dizer que as partições desse problema são dadas pelos conjuntos x_1, x_2 e x_3, x_4. Dessa forma, é possível identificar agora dois subproblemas. O primeiro é dado pelos esquemas $00**$, $01**$, $10**$ e $11**$, e o segundo pelos esquemas $**00$, $**01$, $**10$ e $**11$.

É importante ressaltar que a partição ideal de um problema é geralmente desconhecida. Assim, um método de busca por uma solução poderia se concentrar em identificar tais partições para, então, resolver os subproblemas identificados. Isso será abordado na Seção 4.2.

Por outro lado, a definição de blocos construtivos (BBs, do inglês *Building Blocks*) está ambígua na literatura. Em (Goldberg, 2002), duas definições são utilizadas. A primeira diz que BB é qualquer esquema de uma solução ótima. Por exemplo, o esquema $1 * **$ é um BB da solução 1110, mas não é um BB da solução 0111. No entanto, o termo BB é redefinido para se referir aos esquemas que representam instâncias das partições do problema. Por exemplo, se uma partição, de um problema composto por quatro variáveis, é dada pelas duas primeiras variáveis, tem-se que $11 * *$ é um BB da solução 1111, mas, $1 * **$ e $111*$ não são. Neste trabalho, é utilizada esta última definição.

Para facilitar a argumentação no restante deste texto, usa-se BBv para se referir às variáveis dos blocos construtivos, ou partição (BBv, do inglês *Building Block variables*). O termo "instância de uma partição" também é utilizado como uma combinação possível de valores das BBv's. Além disso, usa-se k para se referir ao tamanho de um BB, que é número de variáveis na partição correspondente.

É importante observar que diferentes definições de esquemas e BBs são encontradas na literatura (Goldberg, 2002; Holland, 1975; Van Veldhuizen, 1999; Zydallis, 2003).[3] As definições utilizadas neste trabalho tornar-se-ão mais claras na Seção 2.4, que apresenta exemplos de partições e BBs, mostrando como o tamanho e a quantidade de BBs influenciam na dificuldade de um problema.

2.4 Blocos Construtivos e Problemas Difíceis

Ao se analisar a dificuldade de problemas, é preciso verificar como essa varia em escala, isto é, com o crescimento do tamanho do problema. É importante caracterizar também o tamanho do espaço de busca. Isso depende não somente do número de variáveis do problema, mas também da cardinalidade das variáveis. Por exemplo, ao trabalhar com variáveis binárias, tem-se um espaço de busca de tamanho 2^n, em que n é o número de variáveis. Assim, para variáveis χ-árias (cardinalidade χ), o espaço de busca possui tamanho χ^n.

Embora o tamanho do espaço de busca seja um fator muito importante para se determinar a dificuldade de um problema, limitantes superiores de complexidade significativamente menores que χ^n podem ser encontrados para diversos problemas. Em (Goldberg, 2002) é verificado que dois fatores muito importantes são: a quantidade m de BBs e o número k de variáveis que compõem cada BB.

[3]Como exemplo, um esquema em (Goldberg, 2002) é o conjunto de soluções definido pelo modelo de similaridade, enquanto que em (Holland, 1975) o esquema é o próprio modelo de similaridade. Outro exemplo é a referência a BBs como sendo tipos específicos de esquemas em (Goldberg, 2002); enquanto trabalhos como (Van Veldhuizen, 1999; Zydallis, 2003) tratam BBs e esquemas como sinônimos.

Para ilustrar a importância desses fatores, considere três problemas distintos, representados na Figura 2.3, todos referentes a um conjunto de cinco variáveis binárias (5 *bits*). Os problemas ilustrados são dados por funções, que possuem como argumento um valor u que corresponde à quantidade de variáveis binárias com valor '1' em uma *string*. Os problemas são os seguintes:

1. Problema UmMax (Goldberg, 2002): consiste em maximizar a quantidade de variáveis iguais a um. A função $f(u)$ é a quantidade de variáveis com valor um;

2. Problema de senha: $f(u)$ é zero para todas as soluções, exceto para o caso em que todas as variáveis são um, para o qual $f(u)$ possui valor máximo;

3. Problema armadilha (Goldberg, 2002): $f(u)$ cai com o aumento de u, exceto para o maior u, para o qual $f(u)$ é máximo. Como exemplo, considere a seguinte função armadilha de cinco *bits*:

$$ftrap5(u) = \begin{cases} 4-u & se\ u < 5, \\ 5 & caso\ contrário. \end{cases} \tag{2.1}$$

Ao modificar o valor de um único *bit* de entrada no problema UmMax, pode-se saber se a alteração gerou uma solução melhor apenas verificando se a alteração aumentou ou diminuiu $f(u)$. Observe que isso independe do valor de qualquer uma das outras variáveis. Devido a esse fato, pode-se considerar que o valor de cada variável define, isoladamente, um BB, isto é, uma subestrutura relevante na composição da solução ótima. No caso da função da Figura 2.3 (a), o problema pode ser decomposto em cinco partições, que definem cinco BBs de tamanho um.

No problema de senha ou no problema de armadilha, para se saber se a modificação em um *bit* gera uma solução mais próxima do ótimo global, é preciso saber os valores dos outros quatro *bits* da *string*. Em outras palavras, os cinco *bits* precisam ser analisados em conjunto, formando uma única partição. Portanto, esses dois problemas possuem um único BB de tamanho cinco em cada solução. A partir desse ponto, é possível observar que, embora todos esses problemas tenham um espaço de busca de mesma dimensão: (2^5); os problemas com BBs de tamanho maior são mais difíceis.

No problema UmMax, algoritmos de busca local (Hoos e Stützle, 2004) são suficientes para encontrar o ótimo global do problema (11111). No entanto, para o problema de senha, qualquer metaheurística (Talbi, 2009) é no máximo tão eficiente quanto uma busca exaustiva ou aleatória, uma vez que não existe informação em nenhum local do espaço de busca que possa orientar o algoritmo para o ótimo global (com exceção do próprio ótimo global). Por outro lado, no problema armadilha, há informação. Com isso, metaheurísticas que utilizem alguma informação local para orientar a busca, tendem a encontrar soluções com poucos 1s, podendo a solução 00000 (ótimo local da $ftrap5$) dominar a população e impedir a descoberta do ótimo

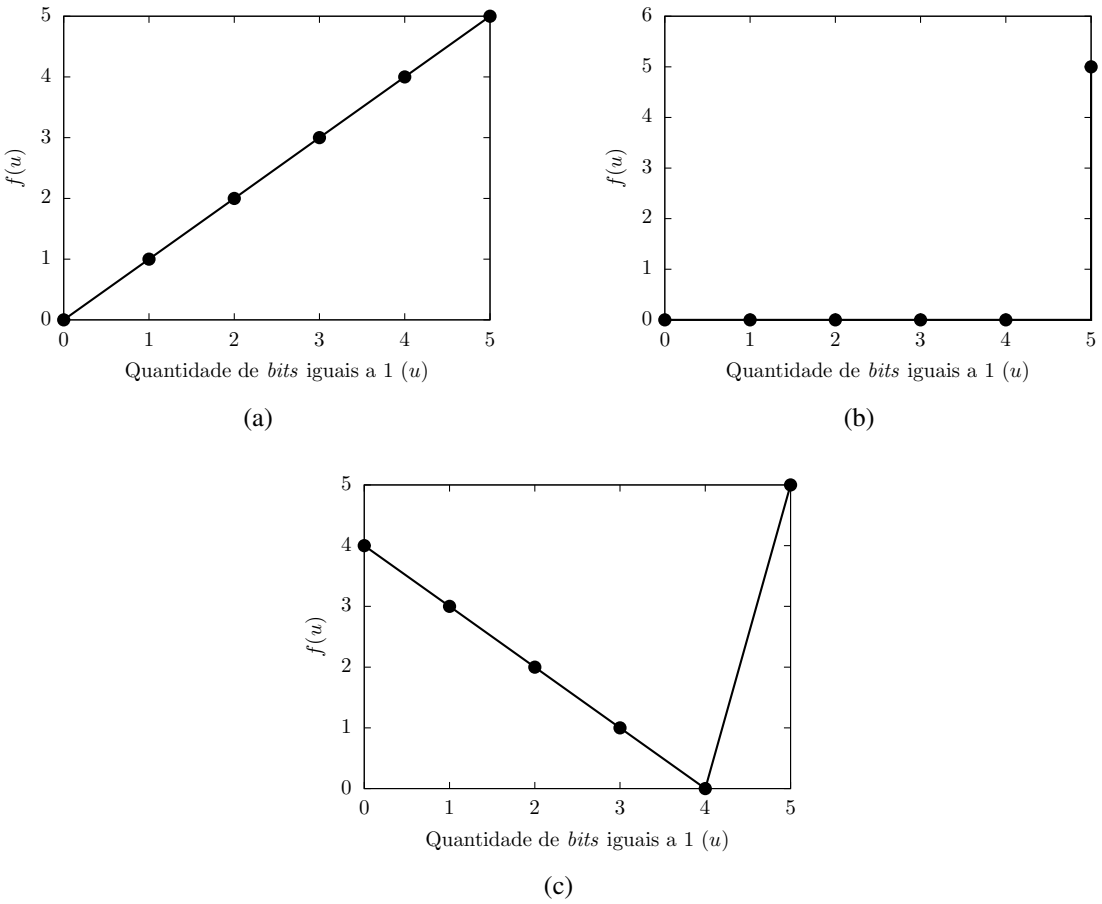


Figura 2.3: Funções: (a) UmMax, (b) Senha e (c) Armadilha.

global (11111). Portanto, para essas metaheurísticas, a informação existente é, na verdade, uma armadilha.

Uma forma de construir problemas mais difíceis que o problema armadilha é simplesmente pelo somatório de subproblemas em que cada termo da adição é uma função armadilha. Como exemplo, considere um problema composto pelo somatório de três funções $ftrap5$. Nesse exemplo, uma *string* é formada então por um conjunto de quinze *bits*, os quais são particionados em três subconjuntos de cinco *bits* e uma $ftrap5$ é aplicada a cada subconjunto. O *fitness* da solução representada pela *string* é a soma dos resultados obtidos para cada $ftrap5$. Dessa forma, uma solução do problema é formada por três BBs de tamanho cinco ($k = 5$ e $m = 3$), em que k é o tamanho dos BBs e m é o número de BBs no problema. Ilustrando, a melhor solução desse problema (o ótimo global) seria formada pelos BBs $11111 * * * * * * * * * *$, $* * * * *11111 * * * * *$ e $* * * * * * * * * * 11111$.

Como discutido anteriormente, em funções armadilha, a busca dentro de uma partição é induzida para ótimos locais diferentes do ótimo global. Dessa forma, caso um operador de

crossover seja utilizado de forma a "quebrar" um BB, como ilustrado pela Figura 2.4, BBs de ótimos globais podem estar sendo eliminados por completo da população. Com isso, a busca dentro da partição será provavelmente conduzida para um ótimo local. Analogamente, uma mutação que ocorra em um BB também pode ser responsável pela destruição de BBs promissores.

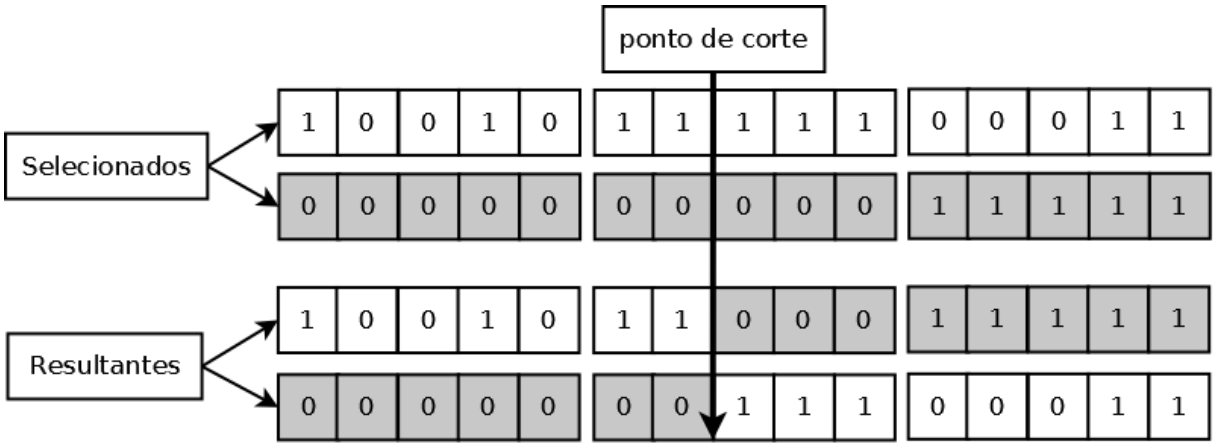


Figura 2.4: Exemplo de crossover com ponto de corte dentro de uma partição. O BB *****11111***** da primeira solução, que é um BB do ótimo global, não se encontra presente em nenhuma das soluções resultantes.

Uma alternativa apresentada em (Goldberg, 2002) para evitar as armadilhas é que o processo de busca não ocorra dentro dos BBs, mas somente entre os BBs. Para isso, parte-se do princípio de que a população inicial do algoritmo possua, pelo menos, uma representação de cada instância possível de uma partição. Com isso, o operador de recombinação utilizado não mais tem como objetivo combinar diretamente os valores das variáveis do problema, mas sim combinar os BBs existentes na população. Reforça-se que, para esse fim, é necessário que a população inicial tenha uma grande diversidade de soluções provendo pelo menos uma instância ótima de cada partição na população, de forma que essas instâncias possam ser combinadas gerando a solução ótima do problema.

Foi demonstrado em (Goldberg, 2002) que a Equação 2.2 estima adequadamente o tamanho da população inicial de forma que essa amostre, com alta probabilidade, todas as possíveis instâncias de cada partição.

$$n = \chi^k(k \ln\ \chi + \ln\ m), \tag{2.2}$$

em que n é o tamanho da população, χ a cardinalidade das variáveis, k o tamanho do BB e m a quantidade de BBs ou partições. A dedução da Equação 2.2 encontra-se no Apêndice A. Como

k é expoente positivo na Equação 2.2 e os demais termos não são exponenciais, χ^k domina a estimativa de n conforme k cresce.

A Figura 2.5 ilustra como o tamanho da população inicial, necessária para que todas as possíveis instâncias dos BBs sejam representadas na população, varia com k e m. As linhas contínuas são obtidas pela Equação 2.2; enquanto os pontos marcados foram obtidos experimentalmente pelo procedimento descrito no Apêndice B. Dessa forma, ilustra-se teórica e experimentalmente como a complexidade de um problema aumenta conforme k e m aumentam. Além disso, a Figura 2.5 mostra claramente que o tamanho do BB é o fator mais influente a ser considerado.

No entanto, uma amostra inicial grande não é o suficiente para que o ótimo global do problema seja construído ao longo do processo evolutivo, dado que o operador de crossover ainda pode quebrar os BBs encontrados (ou destruir instâncias de partições promissoras) caso as partições do problema não sejam conhecidas. A Seção 2.5 introduz os EDAs, que são uma alternativa para tratar problemas como os apresentados, uma vez que esses algoritmos utilizam uma abordagem diferente da de EAs convencionais para a criação de novas soluções.

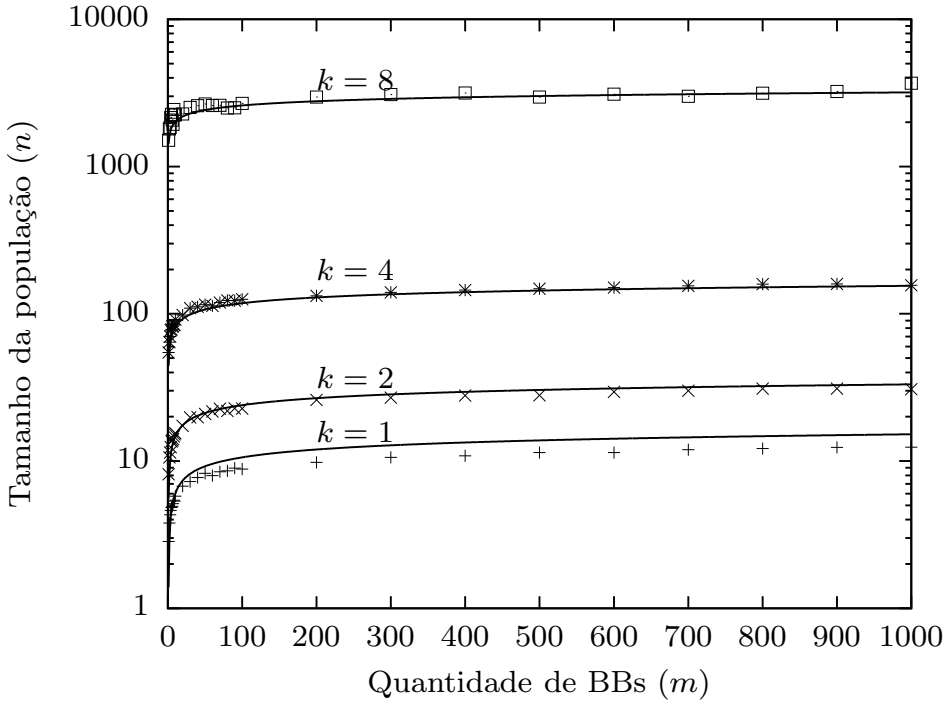


Figura 2.5: Tamanho médio da população inicial, n, necessária para que exista pelo menos uma de cada possível instância de uma partição: curvas teóricas (Apêndice A) e resultados experimentais (Apêndice B).

2.5 Características dos EDAs

Retornando ao assunto da Seção 2.2 de se analisar uma população e criar novas soluções promissoras, mas agora pelo ponto de vista de um EDA (Larrañaga e Lozano, 2001; Pelikan *et al.*, 2002), considere a população representada pela Tabela 2.2.

Tabela 2.2: População de um EDA.

solução	x_1	x_2	x_3	x_4	*fitness*
s_1	0	0	0	1	15
s_2	1	0	1	1	13
s_3	0	0	0	0	10
s_4	1	1	1	1	8
s_5	1	0	1	0	5
s_6	0	1	1	1	5
s_7	1	0	0	0	4
s_8	1	0	0	0	4

Uma forma de se criar novas soluções promissoras, sem utilizar o crossover, é observar o que as melhores soluções possuem em comum e, então, utilizar essa informação para se gerar novas soluções. Como exemplo, considere as quatro melhores soluções da população representada (s_1, s_2, s_3 e s_4). É possível calcular a distribuição dos valores de cada variável para esse conjunto selecionado. Dessa forma, pode-se dizer que, na metade mais promissora da população, 25% dos indivíduos possuem $x_2 = 1$, enquanto os 75% restantes possuem $x_2 = 0$. Utilizando esses valores, é possível construir um modelo informando a distribuição dos valores das variáveis, mostrado pela Tabela 2.3.

Tabela 2.3: Modelo probabilístico.

$P(x_1 = 0)$	$0,50$
$P(x_1 = 1)$	$0,50$
$P(x_2 = 0)$	$0,75$
$P(x_2 = 1)$	$0,25$
$P(x_3 = 0)$	$0,50$
$P(x_3 = 1)$	$0,50$
$P(x_4 = 0)$	$0,25$
$P(x_4 = 1)$	$0,75$

Tal modelo apresenta uma estimação da distribuição dos valores das variáveis nas melhores soluções. Ao utilizar esse modelo, novas soluções promissoras podem ser criadas por serem

similares às melhores soluções da população atual. Cada nova solução gerada usando o modelo apresentado na Tabela 2.3 teria, por exemplo, 75% de chance de possuir $x_2 = 0$, uma vez que $P(x_2 = 0) = 0,75$. Esse tipo de modelo probabilístico, em específico, é utilizado pelo EDA chamado Algoritmo Genético Compacto[4] (CGA, do inglês *Compact Genetic Algorithm*) (Harik *et al.*, 1999).

Embora não use operadores de crossover e mutação, os modelos do CGA e da Tabela 2.3 apresentam o mesmo problema de quebra de BBs discutido anteriormente. Isso deve-se ao fato deles serem modelos probabilísticos univariados, isto é, que não consideram a existência de variáveis correlacionadas. Se a relação entre as variáveis for conhecida, é possível estender o modelo para representar tal relação, possibilitando evitar a quebra dos BBs. Como exemplo, se for conhecido que duas partições do problema são dadas pelos conjuntos de variáveis $\{x_1, x_2\}$ e $\{x_3, x_4\}$, é possível estimar a distribuição dos valores para o conjunto desses pares de variáveis, conforme mostra a Tabela 2.4 (valores calculados considerando as quatro melhores soluções da população apresentada na Tabela 2.2).

Tabela 2.4: Modelo considerando variáveis correlacionadas.

$P(x_1 = 0, x_2 = 0)$	$0,50$
$P(x_1 = 0, x_2 = 1)$	$0,00$
$P(x_1 = 1, x_2 = 0)$	$0,25$
$P(x_1 = 1, x_2 = 1)$	$0,25$
$P(x_3 = 0, x_4 = 0)$	$0,25$
$P(x_3 = 0, x_4 = 1)$	$0,25$
$P(x_3 = 1, x_4 = 0)$	$0,00$
$P(x_3 = 1, x_4 = 1)$	$0,50$

Utilizando a Tabela 2.4, o problema de quebra de BBs pode ser evitado. Novas soluções criadas utilizando esta tabela terão 50% de chance de possuir o BB $**11$ e zero de possuir o BB $01**$ ou $**10$. Supondo que este seja um BB de uma solução ótima, é esperado que nas próximas gerações as tabelas possuam $p(x_3 = 1, x_4 = 1) > 0,50$, ou seja, é aumentada a ocorrência deste BB na população.

Para o último exemplo, foi admitido que as partições do problema já eram conhecidas. No entanto, essas informações normalmente não são conhecidas a priori. Dessa forma, cabe ao EDA encontrar a relação existente entre as variáveis em tempo de execução. Como exemplo, o Algoritmo Genético Compacto Estendido (ECGA, do inglês *Extended Compact Genetic*

[4]Embora este modelo seja o utilizado pelo CGA, a geração de novas soluções não é feita da forma como descrita. Mais informações sobre o funcionamento deste algoritmo encontra-se em (Harik *et al.*, 1999)

Algorithm) (Harik *et al.*, 2006) utiliza um método chamado *linkage learning* para encontrar as partições do problema para, então, construir o modelo da Tabela 2.4.

A partir dos exemplos apresentados até o momento, torna-se mais fácil a compreensão do princípio dos EDAs. Um conjunto de soluções selecionadas (promissoras) de uma população pode ser visto como uma amostra de dados que pode ser modelada por uma função de distribuição de probabilidades. Essa função pode modelar as correlações entre variáveis presentes nas soluções mais promissoras até então encontradas, possibilitando a amostragem de novos indivíduos com grande probabilidade de terem altos valores de *fitness*.

O Algoritmo 2.2 mostra o pseudocódigo de um EDA. Tanto a inicialização da população quanto a seleção de soluções ocorrem da mesma forma que em um EA comum (Seção 2.1). No entanto, ao invés de gerar novas soluções por meio de operadores de reprodução, como crossover e mutação, o EDA estima uma distribuição baseada nos indivíduos selecionados e, a partir dessa distribuição, gera (amostra) novos indivíduos. Posteriormente, os novos indivíduos são adicionados à população, substituindo indivíduos antigos. O processo de seleção, estimação e amostragem são repetidos até que algum critério de parada seja satisfeito.

Algoritmo 2.2: Algoritmo de Estimação de Distribuição.

Entrada: Uma população P_0 vazia;
Saída: Uma população P_g contendo soluções promissoras;

```
// Inicializa o contador de gerações e a população inicial.
g=0;
InicializaPopulação(P_g);
enquanto critério de parada não atingido faça
    // Avalia a aptidão dos indivíduos da população P_g.
    AvaliaPopulacao(P_g);
    // Incrementa o contador da geração.
    g = g+1;
    // Seleciona os indivíduos para a geração do modelo.
    S = Seleção(P_g-1);
    // Tendo os indivíduos selecionados como entrada,
       constrói uma modelo probabilístico M.
    M = CriaModelo(S);
    // Gera a nova população utilizando o modelo construído.
    P_g = NovaPopulacao(M);
fim
```

A diferença básica com relação a um EA típico é que os EDAs substituem a aplicação de operadores reprodutivos pelos seguintes passos:

1. Construção de um modelo probabilístico que representa de forma sintética os indivíduos selecionados;

2. Geração de novos indivíduos a partir do modelo construído.

Note que os EDAs são um tipo particular de EA com um procedimento de reprodução diferenciado. Assim, sob certas condições, um EDA poderia ter comportamento equivalente ao de um EA convencional. Um exemplo é o CGA, que constrói modelos univariados (supõe BBs de tamanho 1) e possui desempenho similar ao de um GA simples (Harik *et al.*, 1999; Mühlenbein, 1997).

É importante observar que a construção de um modelo probabilístico pode não ser uma tarefa trivial. Em geral, existe um compromisso entre a qualidade do modelo construído e a eficiência do método de construção do modelo (Pelikan *et al.*, 2002). Assim, pode-se dizer que um dos grandes desafios de pesquisa na área de EDA é determinar o método que possui a melhor relação entre o custo computacional de construção do modelo e a qualidade do mesmo.

A seguir são apresentadas duas linhas importantes de desenvolvimento de EDAs: o Algoritmo Genético Compacto Estendido (ECGA, do inglês *Extended Compact Genetic Algorithm*) e o BOA. Também são apresentados dois EDAs desenvolvidos pelo grupo de pesquisa do Laboratório de Computação Reconfigurável (LCR)[5]: O Algoritmo Filogenético (PhyGA, do inglês *Phylogenetic Algorithm*) (Vargas e Delbem, 2009), e o Filogenético com Evolução Diferencial (PhyDE, do inglês *Phylogenetic Differential Evolution*) (Melo *et al.*, 2011).

Algoritmo Genético Compacto Estendido

O ECGA (Harik *et al.*, 2006) busca encontrar as variáveis correlacionadas por meio de um processo chamado de *linkage learning*. Esse método considera que as variáveis relacionam-se em grupos (partições) sem que esses sejam sobrepostos, isto é, supõe que não haja uma ou mais variáveis pertencentes a duas partições distintas. Uma vez obtido um modelo que representa os BBs para um problema, é possível usar operadores reprodutivos competentes (Goldberg, 2002), isto é, operadores que não misturam valores de variáveis de instâncias de uma mesma partição de indivíduos diferentes ao combinar duas ou mais soluções. Observe que, sem o modelo de partições, o operador de recombinação pode desagrupar combinações promissoras de valores de variáveis se o ponto de corte for interno a uma partição, ou seja, pode destruir uma solução ótima de um subproblema (ver Seção 2.4).

[5]O LCR encontra-se no Instituto de Ciências Matemáticas e da Computação (ICMC), na Universidade de São Paulo (USP). Página do laboratório: `http://lcr.icmc.usp.br/` (último acesso em 12/08/2012).

Algoritmo de Otimização Bayesiano

A ideia central do BOA (Pelikan, 2005; Pelikan *et al.*, 1999) é a utilização de Redes Bayesianas (BNs, do inglês *Bayesian Networks*) (Niedermayer, 2009) para a representação de informações a respeito das dependências entre variáveis. As variáveis são representadas como vértices em uma rede (um grafo (Diestel, 2005)) e a existência de uma aresta entre dois vértices significa que o valor de uma dessas variáveis depende do valor da outra. Em outras palavras, as componentes conexas (Diestel, 2005) da rede podem ser vistas como BBs. Diferentemente do ECGA, a BN é capaz de representar sobreposição de BBs. A partir de uma BN, é possível gerar novos indivíduos que irão compor a nova geração. O BOA é explicado em mais detalhes na Seção 2.6.

Algoritmo Filogenético

O PhyGA (Vargas e Delbem, 2009) é um EDA cuja ideia é identificar as partições do problema sendo tratado uma única vez e, então, realizar uma busca em cada BB separadamente. Este algoritmo encaixa-se na categoria de algoritmos de Otimização por Decomposição (DecOA, do inglês *Decomposition Optimization Algorithm*), proposta na Seção 4.2 deste trabalho. Para a identificação das partições, é utilizado o algoritmo *Neighbor Joining* (Saitou e Nei, 1987). Este algoritmo é utilizado na Biologia para a construção de árvores filogenéticas (Felsenstein, 2003) e pode ser visto como um método de agrupamento hierárquico (Jain *et al.*, 1999). Após a identificação das partições (chamadas de clados em Filogenia), o PhyGA realiza uma busca exaustiva em cada partição, para definir as instâncias ótimas das mesmas (BBs ótimos) e, então, compor a solução final.

Filogenético com Evolução Diferencial

O PhyDE (Melo *et al.*, 2011) é uma extensão do PhyGA que utiliza o EA chamado Evolução Diferencial (DE, do inglês *Differential Evolution*) (Storn e Price, 1997) para encontrar os valores ótimos para os BBs, ao invés da busca exaustiva utilizada pelo PhyGA. O algoritmo em questão tem apresentado resultados interessantes para problemas com BBs relativamente grandes, tendo resolvido problemas com BBs de tamanho 8 ($k = 8$); enquanto EAs da literatura normalmente resolvem de forma eficiente problemas com $k < 6$ (Harik *et al.*, 2006; Pelikan *et al.*, 1999).

2.6 Algoritmo de Otimização Bayesiano

Considere o problema de gerar novas soluções promissoras a partir da população representada pela Tabela 2.5, desta vez pelo ponto de vista do BOA. Suponha conhecidas as seguintes relações entre as variáveis: o valor de x_2 depende do valor de x_1, e o valor de x_4 depende do valor de x_3. Essas relações são representadas pela BN da Figura 2.6.

Tabela 2.5: População de um EDA.

solução	x_1	x_2	x_3	x_4	*fitness*
s_1	0	0	0	1	15
s_2	1	0	1	1	13
s_3	0	0	0	0	10
s_4	1	1	1	1	8
s_5	1	0	1	0	5
s_6	0	1	1	1	5
s_7	1	0	0	0	4
s_8	1	0	0	0	4

Conhecidas as relações apresentadas, o BOA utiliza as 50% melhores soluções da população (s_1, s_2, s_3 e s_4) para calcular as probabilidades condicionais, dada pela Tabela 2.6. Para melhor compreender a construção dessa tabela, verifique que apenas duas das quatro melhores soluções da Tabela 2.5 possuem $x_1 = 1$, e dessas duas soluções, apenas uma apresenta $x_2 = 0$. Assim, pode-se dizer que dentre as quatro melhores soluções, tem-se que 50% das soluções com $x_1 = 1$ apresentam $x_2 = 0$, isto é, $P(x_2 = 0|x_1 = 1) = 0,50$, como indicado pela quarta linha da Tabela 2.6. Essas probabilidades são utilizadas para amostrar novas soluções promissoras, que substituem as piores soluções da população atual (s_5, s_6, s_7 e s_8) gerando a nova população.

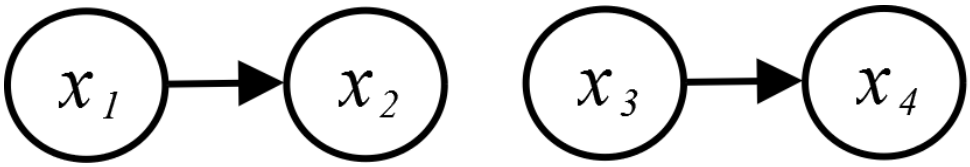


Figura 2.6: BN que mostra a relação entre quatro variáveis (x_2 é condicional a x_1, e x_4 é condicional a x_3), definindo as probabilidades condicionais da Tabela tab:probboa.

Na geração de uma nova solução a partir a Tabela 2.6, primeiro determinam-se os valores das variáveis que não dependem de outra segundo a BN (x_1 e x_3), a partir das respectivas probabilidades dadas por $P(x_1)$ e $P(x_3)$. Em seguida, valores são atribuídos para as outras variáveis, considerando as respectivas probabilidades condicionais ($P(x_2|x_1)$ e $P(x_4|x_3)$).

Tabela 2.6: Modelo considerando variáveis correlacionadas.

$P(x_1 = 0)$	$0,50$
$P(x_1 = 1)$	$0,50$
$P(x_2 = 0 \mid x_1 = 0)$	$1,00$
$P(x_2 = 1 \mid x_1 = 0)$	$0,00$
$P(x_2 = 0 \mid x_1 = 1)$	$0,50$
$P(x_2 = 1 \mid x_1 = 1)$	$0,50$
$P(x_3 = 0)$	$0,50$
$P(x_3 = 1)$	$0,50$
$P(x_4 = 0 \mid x_3 = 0)$	$0,50$
$P(x_4 = 1 \mid x_3 = 0)$	$0,50$
$P(x_4 = 0 \mid x_3 = 1)$	$0,00$
$P(x_4 = 1 \mid x_3 = 1)$	$1,00$

Embora as correlações entre as variáveis representadas pela BN da Figura 2.6 tenham sido dadas no exemplo acima, é papel do BOA estimar essas relações e montar uma BN em cada geração. A Seção 2.6.2 explica procedimentos para a criação de BNs, que utilizam uma amostra de dados como entrada (no caso do BOA, o conjunto de soluções selecionadas). Um grande desafio do BOA é justamente encontrar o melhor modelo que represente o conjunto de indivíduos selecionados da população, com o menor custo computacional possível (Emmendorfer, 2007).

Como visto no exemplo anterior, o BOA, proposto em (Pelikan *et al.*, 1999), utiliza BNs para modelar as soluções promissoras e, consequentemente, orientar a exploração do espaço de busca. De maneira simplificada, a Figura 2.7 e o Algoritmo 2.3 ilustram o pseudocódigo do BOA. As setas numeradas da Figura 2.7 representam os passos do BOA, que são explicados na sequência. Até que algum critério de parada escolhido (Seção 2.1) seja alcançado, os seguintes passos repetem-se:

1. Seleção de um conjunto de soluções. Em (Pelikan *et al.*, 1999) é utilizado o critério de seleção de truncamento, em que as 50% melhores soluções são selecionadas;

2. A partir das soluções selecionadas, uma busca é realizada para encontrar uma BN representativa dos dados. Essa busca é explicada na Seção 2.6.2;

3. A partir das probabilidades condicionais definidas pela BN (Tabela 2.6), novas soluções são criadas. Tais soluções substituem as 50% piores soluções da população atual, formando assim uma nova população.

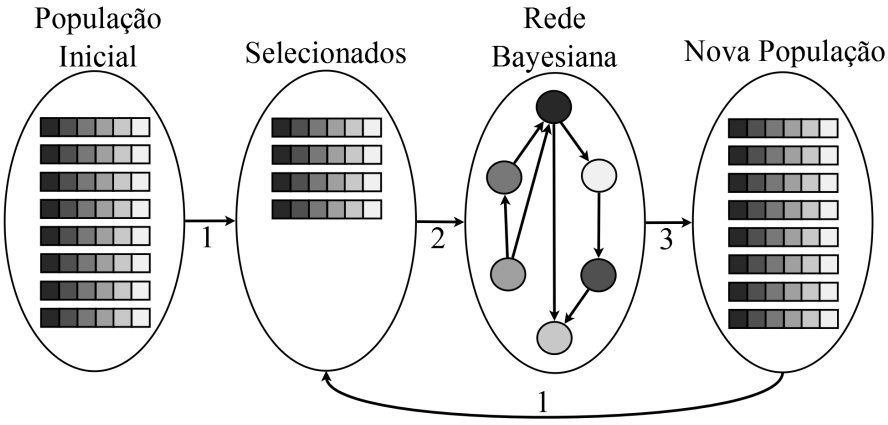


Figura 2.7: Princípio de funcionamento do BOA.

As BNs utilizadas no BOA são modelos complexos que podem identificar relacionamentos de ordem ℓ entre os genes (variáveis) da população, em que ℓ é o número de variáveis do problema. A Seção 2.6.1 apresenta uma explicação detalhada desses modelos.

2.6.1 Redes Bayesianas

Para se descrever formalmente uma BN (Niedermayer, 2009; Pelikan *et al.*, 1999; Santos, 2007), é preciso de conceitos básicos de grafos. Em (Diestel, 2005), grafos são definidos formalmente como o par $G = (V, E)$, em que V é um conjunto de vértices e E é o conjunto de arestas, em que $E \subseteq V \times V$, isto é, elementos de E são subconjuntos de dois elementos de V. A Figura 2.8 ilustra o exemplo de um grafo $G = (V, E)$, com $V = \{1, 2, 3\}$ e $E = \{\{1, 2\}, \{1, 3\}\}$, em que, a representação visual de um vértice é feita por uma circunferência, e uma aresta é representada por uma linha conectando dois vértices.

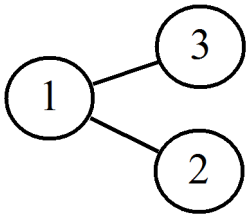


Figura 2.8: Exemplo de um grafo $G = (V, E)$, com $V = \{1, 2, 3\}$ e $E = \{\{1, 2\}, \{1, 3\}\}$.

Uma BN (Niedermayer, 2009; Pelikan *et al.*, 1999; Santos, 2007) é um modelo probabilístico multivariado baseado em grafos, o qual é usado para representar as relações existentes entre as variáveis. Cada possível grafo modelando tais relações é chamado de rede. Essas redes são usadas para representar dados multivariados em vários domínios de aplicação, incluindo situações

Algoritmo 2.3: Pseudocódigo do BOA.

Entrada: Uma população P_0 vazia;
Saída: Uma população P_g contendo soluções promissoras;

```
// Inicializa o contador de gerações e a população inicial.
```
1 g=0;
2 InicializaPopulação(P_g);
3 **enquanto** *critério de parada não atingido* **faça**

```
    // Avalia a aptidão dos indivíduos da população P_g.
```
4 AvaliaPopulacao(P_g);

```
    // Incrementa o contador da geração.
```
5 g = g+1;

```
    // Seleciona os indivíduos para a geração do modelo.
```
6 S = Seleção(P_{g-1});

```
    // Tendo os indivíduos selecionados como entrada,
       encontra uma BN apropriada usando uma métrica para
       avaliar cada rede considerada.
```
7 BN = RedeBayesiana(S);

```
    // Gera a nova população utilizando a BN construída.
```
8 P_g = NovaPopulacao(BN);
9 **fim**

nas quais a estrutura de correlações entre as variáveis é desconhecida ou apenas parcialmente conhecida a priori.

Cada vértice na rede corresponde a uma variável[6]. Tanto a variável quanto o vértice correspondente são denotados por x_i. Cada variável x_i corresponde a um valor na posição de uma *string* que representa uma solução. O relacionamento entre duas variáveis é representado por uma aresta entre os dois vértices correspondentes. Formalmente, uma BN é um grafo acíclico com arestas orientadas (Diestel, 2005) às quais se associam à distribuições de probabilidades conjuntas (Niedermayer, 2009). Tais probabilidades, denominadas $P(X)$, podem ser calculadas a partir das probabilidades condicionais, conforme mostra a Equação 2.3.

$$P(X) = \prod_{i=1}^{n} P(x_i|\pi_i), \tag{2.3}$$

em que:

- $X = (x_0, ..., x_{n-1})$ é um vetor de variáveis e n é o tamanho do vetor;

[6]Este trabalho lida com variáveis discretas, assim como em (Pelikan *et al.*, 1999). No entanto, uma extensão do BOA que trata número reais é apresentada em (Ahn *et al.*, 2004).

- π_i é o conjunto das variáveis pais de x_i na rede (vértices aos quais se conectam arestas orientadas apontando para x_i);
- $P(x_i|\pi_i)$ é a probabilidade de x_i condicional aos valores de seus pais (π_i).

A Equação 2.3 pode ser melhor compreendida a partir do exemplo descrito na sequência.

Exemplo de uma BN

Este exemplo baseia-se no texto disponível em (Niedermayer, 2009). Considere as duas variáveis binárias relativas ao problema de prever chuva em uma região, em que:

- $x_1 = 1$ indica que "chove hoje"; caso contrário $x_1 = 0$;
- $x_2 = 1$ indica que "chove amanhã"; caso contrário $x_2 = 0$.

Para esse problema, sabe-se que a probabilidade de chover em um dia é condicional ao fato de ter ou não chovido no dia anterior. Suponha que a chance de chover hoje seja de 20% ($P(x_1 = 1) = 0,20$). Caso tenha chovido no dia anterior, a probabilidade de chover é de 70% ($P(x_{i+1} = 1|x_i = 1) = 0,70$). A relação entre as variáveis x_1 e x_2 pode ser ilustrada pela Figura 2.9, destacando a relação de que "chover hoje" depende do fato de ter ou não chovido ontem.

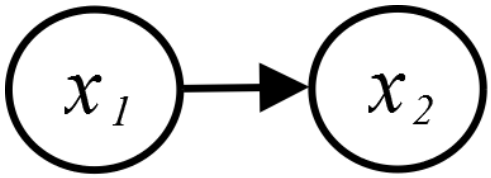


Figura 2.9: BN que mostra a relação entre duas variáveis (x_2 é condicional a x_1).

Supondo que ontem não choveu, então pode-se calcular a probabilidade de que chova hoje (x_1=1) e chova amanhã (x_2=1) utilizando o cálculo da probabilidade condicional e a informação dada pela BN da Figura 2.9, isto é, de que $\pi_2 = \{x_1\}$, obtém-se:

$P(x_1 = 1, x_2 = 1) = P(x_1 = 1) \times P(x_2 = 1|x_1 = 1) = 0,20 \times 0,70 = 0,14$

A Tabela de Probabilidades

No exemplo apresentado, as probabilidades de "chover hoje" e "chover amanhã" foram dadas como conhecidas. No entanto, é possível estimar tais probabilidades a partir de um conjunto de amostras de eventos H, representados por pares de dias consecutivos selecionados aleatoriamente de um mês. Para exemplificar, considere:

1. O problema de chover no dia x_1 e chover no dia consecutivo x_2, representado por uma *string* binária: $X = [x_1, x_2], x_i \in \{0, 1\}$;

2. Um conjunto de oito eventos selecionados aleatoriamente, mostrados na Tabela 2.7, em que cada evento H_i é uma instância de X;

3. Uma BN como a mostrada na Figura 2.9, que indica que a probabilidade de chover no dia x_2 é influenciada pelo fato de ter ou não chovido no dia x_1 ($P(x_2)$ é condicional a x_1);

4. A função $P(H)$, que retorna a frequência de ocorrências de um evento H (ver Equação 2.3).

Tabela 2.7: Conjunto de eventos H_i amostrados, referentes a chuva no dia d, representado por x_1 e chuva no dia $d + 1$, representado por x_2.

evento	x_1	x_2
H_1	0	0
H_2	0	0
H_3	0	1
H_4	1	1
H_5	1	1
H_6	1	0
H_7	0	0
H_8	0	0

Considerando que a chance de chover em um dia $(d + 1)$ é condicional ao fato de ter ou não chovido em um dia anterior (d), pode-se estimar as probabilidades de todos os possíveis eventos envolvendo x_1 e x_2 com base nas oito amostras da Tabela 2.7. Como exemplo, na Tabela 2.7 existem cinco eventos (H_1, H_2, H_3, H_7 *e* H_8) em que não houve chuva no dia d ($x_1 = 0$), dentre os quais em apenas um (H_3) houve chuva no dia $d+1$ ($x_2 = 1$). Dessa forma, pode-se estimar a chance de chover em um dia dado que não houve chuva no dia anterior como sendo um quinto, isto é, $P(x_2 = 1|x_1 = 0) = 0,20$. A Tabela 2.8 mostra tais estimativas.

É possível utilizar as probabilidades da Tabela 2.8 para criar eventos fictícios, similares aos amostrados na Tabela 2.7. Em outras palavras, utilizando as probabilidades calculadas, é possível estimar pares de dias consecutivos que mantenham características dos pares de dias amostrados do mundo real. Analogamente, o BOA utiliza o mesmo princípio ao selecionar soluções promissoras para construir o modelo utilizado, uma vez que por meio dos modelos probabilísticos do BOA é possível gerar novas soluções que compartilhem características promissoras.

Tabela 2.8: Exemplo de uma Tabela de Probabilidades obtida a partir de uma BN (Figura 2.9) e de uma amostra de eventos (Tabela 2.7).

$P(x_1 = 0)$	$0,63$
$P(x_1 = 1)$	$0,37$
$P(x_2 = 0 \mid x_1 = 0)$	$0,80$
$P(x_2 = 1 \mid x_1 = 0)$	$0,20$
$P(x_2 = 0 \mid x_1 = 1)$	$0,33$
$P(x_2 = 1 \mid x_1 = 1)$	$0,67$

No exemplo anterior, foi suposta conhecida a BN que representa as relações entre as variáveis. No entanto, tal informação normalmente não se encontra disponível para o EDA. O BOA utiliza o conjunto de soluções promissoras (como as da Tabela 2.7) para gerar a própria BN. Para isso, é necessário um algoritmo para a criação de uma BN dado um conjunto de dados de entrada. A Seção 2.6.2 apresenta métodos para a construção de BNs.

2.6.2 Construindo Redes Bayesianas

Há dois componentes básicos no algoritmo que constrói a topologia de uma BN (Pearl, 1988):

1. A métrica de pontuação;
2. O procedimento de busca.

A métrica de pontuação avalia a qualidade com que a rede modela os dados. O conhecimento prévio sobre o problema também pode ser incorporado na métrica. É importante destacar que encontrar a melhor BN (que possui a melhor pontuação dada pela métrica) para um conjunto de variáveis é um problema NP (Santos, 2007). Dessa forma, busca-se na prática um procedimento que construa uma rede com o valor de métrica de pontuação tão alto quanto possível em um tempo computacional aceitável. Como uma nova BN deve ser construída a cada geração do BOA, erros em um modelo podem ser corrigidos por modelos futuros.

A métrica utilizada pelo algoritmo K2, responsável por avaliar o quão bem uma BN B_s representa um conjunto de dados D, é dada por $P(B_s, D)$, calculada pela Equação 2.4. Um exemplo de uma rede B_s e de um conjunto de dados D a ser modelado, são apresentados pela Figura 2.9 e pela Tabela 2.7, respectivamente.

$$P(B_s, D) = P(B_s) \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!)} \prod_{k=1}^{r_i} N_{ijk}!, \tag{2.4}$$

em que:

- $P(B_s)$ é um fator utilizado quando se tem uma informação prévia sobre a qualidade da rede. Quando não há tal informação, o valor é 1 (valor usado neste trabalho);
- n é o número de variáveis x_i;
- q_i é o número de possíveis combinações, encontradas em D, dos valores das variáveis pais de x_i;
- r_i é o número de possíveis valores associados à variável x_i;
- N_{ijk} é o número de casos em D, com $x_i = v_{ik}$ e $\pi_i = w_{ij}$, em que v_{ik} representa o k-ésimo valor possível (entre os r_i valores existentes) da variável x_i, e w_{ij} a j-ésima combinação possível (das q_i combinações existentes em D) para os valores das variáveis pais de x_i (π_i);
- N_{ij} é o número de ocorrências que possuem $\pi_i = w_{ij}$. Dado por: $N_{ij} = \sum_{k=1}^{r_i} N_{ijk}$.

O algoritmo K2 utiliza um método de busca gulosa (*greedy search* (Cook *et al.*, 1997)) para maximizar $P(B_s, D)$. Assim, a contribuição de um vértice i da rede no valor da pontuação $P(B_s, D)$ (efeito local) depende do conjunto de pais (π_i) do vértice i, isto é, dos pares (i, π_i). Assim, pode-se reescrever a Equação 2.4 da seguinte forma: $P(B_s, D) = P(B_s) \prod_{i=1}^{n} g(i, \pi_i)$, em que $g(i, \pi_i)$ é chamado de ganho e definido como:

$$g(i, \pi_i) = \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} N_{ijk}!. \tag{2.5}$$

O pseudocódigo do Algoritmo K2 (descrito no Algoritmo 2.4, adaptado de (Cooper e Herskovits, 1992)) pode ser explicado como segue. Partindo de uma rede sem arestas e com n vértices (um para cada variável do problema), uma busca gulosa é utilizada para, a cada etapa, escolher uma nova aresta que será inserida na rede. A modificação na pontuação da BN pela inserção desta aresta, é dada pela Equação 2.5.

É possível utilizar a forma logarítmica da Equação 2.5 para sua implementação, como sugerido em (Cooper e Herskovits, 1992), resultando na Equação 2.6. A forma logarítmica apresentada é computacionalmente menos custosa que a versão anterior, por utilizar somas e subtrações ao invés de multiplicações e divisões. A utilização da forma logarítmica também pode evitar a ocorrência de erros devidos aos fatoriais que podem gerar números que esgotam a capacidade de representação do computador.

O cálculo da Equação 2.5 é uma complicação, pois o uso de fatoriais produz números muito grandes, podendo esgotar a capacidade de representação do computador (*overflow*) quando o algoritmo é executado. Portanto, deve-se utilizar a forma logarítmica da expressão, substituindo $g(i, \pi_i)$ por $ln(g(i, \pi_i))$, conforme mostrado pela Equação 2.6.

$$ln(g(i, \pi_i)) = \sum_{j=1}^{q_i} ln((r_i - 1)!) - ln((N_{ij} + r_i - 1)!) + \sum_{k=1}^{r_i} ln(N_{ijk}!). \tag{2.6}$$

Algoritmo 2.4: Pseudocódigo do algoritmo K2.

Entrada: Um conjunto de n vértices ordenados, limite superior v (número máximo de pais por vértice), base de dados D, contendo t casos.
Saída: Para cada vértice, a identificação de seus pais.

```
1  para i = 1 até t faça
2      π_i = {};
       // Função calculada usando a fórmula logarítmica, dada
          pela Equação 2.6
3      P_old = g(i,π_i);
4      OKparaProceder = sim;
5      enquanto OKparaProceder e |π|<v faça
           // z aponta para o vértice que, entre todos os
              vértices que antecedem x_i dada a ordenação de
              entrada, maximiza a Equação 2.6
6          z = maxPred(x_i);
7          P_new = g(i,π_i ∪ {z});
8          se P_new > P_old então
9              P_old = P_new;
10             π_i = π_i ∪ z;
11         fim
12         senão OKparaProceder = não;
13     fim
14     Escreva("Vértice:"x_i, "Pais deste vértice:", π_i);
15 fim
```

No BOA implementado em (Pelikan, 1999), a métrica do algoritmo K2 é utilizada em um outro algoritmo de busca gulosa, que será chamado neste trabalho de busca gulosa do BOA (BGS, do inglês *BOA Greedy Search*). Assim como no algoritmo K2, o BGS parte de uma BN vazia (sem arestas) e, então, trabalha apenas adicionando arestas. Diferente do algoritmo K2, o BGS não parte de um conjunto ordenado de vértices e, portanto, trabalha em um espaço de busca maior (considera uma quantidade maior de possíveis BNs). O Algoritmo 2.5 mostra o

pseudocódigo do BGS apresentado em um alto nível de abstração[7]. Neste algoritmo, o cálculo dos ganhos para possíveis arestas a serem inseridas é realizado a partir da Equação 2.6. É possível inserir uma aresta quando esta respeita as condições de: i) não criar ciclos na rede, e ii) o vértice destino não possui mais do que $v - 1$ vértices pais, em que v é um parâmetro de entrada do algoritmo que define o número máximo de pais por vértice. Caso a aresta não respeite essas condições, o ganho da mesma é -1.

Algoritmo 2.5: Pseudocódigo do algoritmo BGS.

Entrada: Um conjunto de n vértices, limite superior v (número máximo de pais por vértice), conjunto de dados D, BN vazia.
Saída: BN que representa os dados D.

```
1  númeroMáximoArestas=v×n;
2  encerrar=0;
3  númeroArestas=0;
4  Calcule o ganho a ser obtido para cada possível aresta a ser inserida;
5  enquanto encerrar!=1 e numeroArestas<númeroMáximoArestas faça
6  |  Selecione a aresta com o maior ganho calculado (ganhoMaximo);
7  |  se ganhoMaximo>0 então
8  |  |  insere aresta na BN;
9  |  |  numeroArestas=númeroArestas+1;
10 |  fim
11 |  senão encerrar=1;
12 |  Atualize os ganhos a serem obtidos para cada possível aresta a ser inserida;
13 fim
```

2.7 Variações do BOA

Variações do BOA foram propostas para tratar diferentes classes de problemas. A seguir, são brevemente apresentadas algumas dessas variações capazes de resolver problemas multiobjetivos (Apêndice C) e hierárquicos (Seção 5.5).

BOA para problemas hierárquicos

Alguns EAs utilizam métodos chamados *niching* (Mahfoud, 1995), que possuem o objetivo de descobrir múltiplas soluções para um problema e preservar soluções alternativas até que seja possível decidir qual solução é a melhor. O BOA Hierárquico (Pelikan e Goldberg,

[7]Este pseudocódigo foi obtido analisando a implementação do algoritmo disponível em (Pelikan, 1999). Para mais detalhes de implementação, consultar a função "constructTheNetwork" dentro do arquivo "bayesian.cc".

2001) (hBOA, do inglês *hierarchical BOA*) é uma extensão do BOA que utiliza Grafos de Decisão (Jaeger, 2004) no lugar das Tabelas de Probabilidade (apresentadas na Seção 2.6.1). Além disso, o hBOA emprega uma técnica de *niching* chamada Torneio de Substituição Restrita (RTR, do inglês *Restricted Tournment Replacement*).

O RTR funciona durante a etapa de substituição de soluções da população. Para cada nova solução gerada r, um subconjunto da população é selecionado. Deste, é selecionada a solução s mais similar à r. Caso a solução s tenha pior *fitness* que r, s é substituída por r; caso contrário, r é descartada.

Como medida de similaridade, foi utilizada em (Pelikan e Goldberg, 2001) a distância de *Hamming* (número de posições em que duas *strings* diferem entre si). No entanto, é apontado pelos autores do algoritmo, que outras medidas de similaridade podem ser utilizadas. O tamanho do subconjunto selecionado (chamado de *window size*) deve ser proporcional ao tamanho do problema.

O relativo sucesso do hBOA na solução de problemas deceptivos hierárquicos (Seção 5.5) apresentados na literatura (Pelikan e Goldberg, 2001; Pelikan *et al.*, 2005) mostra além da competência do hBOA, a importância do desenvolvimento de algoritmos eficientes para resolver problemas separáveis (Seção 5.4). Observe que não foi necessária grande modificação do BOA (que resolve problemas separáveis) para o capacitar a resolver também problemas hierárquicos, como feito pelo hBOA (Pelikan e Goldberg, 2001). Assim, extensões similares a partir de EDAs desenvolvidos para serem eficientes em problemas separáveis podem ser obtidos para problemas hierárquicos.

hBOA para problemas multiobjetivo

O algoritmo hBOA multiobjetivo (mohBOA, do inglês *multi-objective hBOA*) (Pelikan *et al.*, 2005) consiste de uma extensão do hBOA apresentado anteriormente. As principais modificação são: i) utilização da técnica chamada *nondominated crowding* (Deb *et al.*, 2002), para somente então aplicar operadores tradicionais de seleção multiobjetivo; ii) utilização de um algoritmo de *clustering* (MacQueen, 1967) para separar o conjunto de soluções selecionadas em subconjuntos; e iii) construção de um modelo para cada subconjunto encontrado.

Todos os modelos obtidos são então utilizados para se gerar diferentes novas soluções candidatas, que podem ser adicionadas na nova população por meio do RTR utilizado no hBOA. Cada modelo obtido gera um número igual de soluções candidatas, a fim de manter a variedade das novas soluções geradas. A diversidade das soluções é importante na exploração do espaço de objetivos em problemas multiobjetivo (Apêndice C).

CAPÍTULO

3

Detecção de Estruturas de Comunidades e Técnicas de Reamostragem

Este trabalho propõe a elaboração de um novo EDA que utiliza técnicas de detecção de estruturas de comunidades, da área de Redes Complexas, para a identificação das partições de um problema. Este Capítulo apresenta tais técnicas na Seção 3.1. Adicionalmente, a Seção 3.2 introduz técnicas de reamostragem, que servem como ponto de partida para o desenvolvimento de uma técnica de reamostragem para EDAs (apresentada na Seção 4.3), que beneficia a detecção de comunidades, isto é, a identificação das partições (Seção 2.3) do problema.

3.1 Detecção de Estruturas de Comunidades

Devido ao fato de pessoas, em geral, organizarem-se em grupos de acordo com seus interesses, trabalho, idade e outras características, redes sociais normalmente encontram-se subdivididas em uma estrutura de comunidades (Newman, 2004). Essas podem ser formalmente representadas por grupos de vértices de uma rede que, em geral, são densamente conectados entre si e, esparsamente conectados com o restante da rede (que pode conter outras comunidades). A Figura 3.1 ilustra uma possível estrutura de comunidades para uma rede.

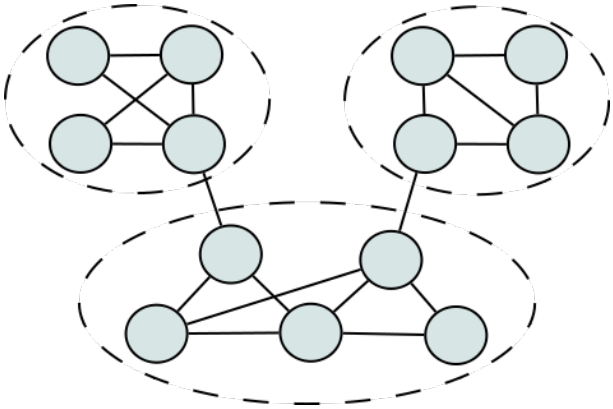


Figura 3.1: Exemplo de estrutura com três comunidades (destacadas por linhas tracejadas) em uma rede.

Comunidades são de grande interesse, pois geralmente correspondem a unidades comportamentais e funcionais. Devido a isso, ferramentas matemáticas e algoritmos têm sido pesquisados para detectar estruturas de comunidades na área de Redes Complexas (Donetti e Muñoz, 2004; Duch e Arenas, 2005). É importante ressaltar que algoritmos que verificam todas as possíveis estruturas de comunidade em uma dada rede são NP (Duch e Arenas, 2005) e, portanto, inviáveis de serem utilizados para redes grandes. Por isso, a literatura recente apresenta diversas novas técnicas que buscam encontrar a melhor estrutura de comunidade possível em um tempo aceitável.

Dessas técnicas, podem-se destacar as baseadas na métrica Q, definida por Newman (Newman, 2004) que representa a modularidade de uma divisão em comunidades. Quanto maior o valor de Q, melhor a estrutura de comunidades encontrada. A partir da modularidade, diversos pesquisadores têm proposto o uso de algoritmos de otimização para encontrar estruturas de comunidades. Entre esses, podem-se destacar o Algoritmo Rápido (FA, do inglês *Fast Algorithm*) (Newman, 2004) (Seção 3.1.1), o Agrupamento Adaptativo (AdClust, do inglês *Adaptive Clustering*) (Ye *et al.*, 2008) (Seção 3.1.2) e o Otimização Extrema (EO, do inglês *Extreme Optimization*) (Duch e Arenas, 2005) (Seção 3.1.3).

3.1.1 Detecção de Comunidades pelo FA

O FA (Newman, 2004) é baseado na métrica de modularidade, que qualifica o quão adequada é uma estrutura de comunidades para uma dada rede. Para isso, define-se o termo e_{ij} como a metade do número de arestas que conectam as comunidades i e j em relação ao total de arestas da rede. Dessa forma, $e_{ij}+e_{ji}$ corresponde ao total de arestas que conectam os vértices de ambas as comunidades; enquanto que e_{ii} corresponde ao número de arestas dentro da comunidade i em

relação ao total de arestas. Um algoritmo de detecção de comunidades deve encontrar a estrutura que maximiza a fração de arestas que conectam vértices de uma mesma comunidade, ou seja, maximizar $\sum_i e_{ii}$. Entretanto, essa medida não avalia bem a qualidade das comunidades, uma vez que seu valor máximo é facilmente atingido quando todos os vértices da rede pertencem à mesma comunidade. Para isso, mais um componente é utilizado: $a_i = \sum_j e_{ij}$, a fração das arestas conectadas a pelo menos um vértice da comunidade i. Com isso, define-se o índice de modularidade Q, ponderando entre ambas as frações, conforme mostra a Equação 3.1.

$$Q = \sum_i (e_{ii} - a_i^2). \tag{3.1}$$

De posse de um índice para quantificar a qualidade das comunidades, pode-se construir um algoritmo que otimize o valor de Q sobre todas as possíveis divisões da rede em comunidades. Foi desenvolvido em (Newman, 2004) um algoritmo guloso para esse fim. Inicialmente, esse algoritmo considera cada vértice da rede como uma comunidade. As comunidades são agrupadas em pares considerando todos os pares possíveis (comunidades que possuem pelo menos uma aresta as conectando) e, então, o agrupamento (estrutura de comunidades) é salvo, assim como o valor de Q para o mesmo. Esse processo de agrupar repete-se até que haja apenas uma comunidade. Por fim, o agrupamento com maior valor de Q é preservado. O Algoritmo 3.1 apresenta o funcionamento do FA.

Algoritmo 3.1: *Pseudocódigo do FA.*

Entrada: Um grafo com n vértices;
Saída: Uma estrutura de comunidades;
1 Separe os n vértices em n comunidades;
2 **enquanto** *número de comunidades > 1* **faça**
3 Junte as duas comunidades i e j cuja aglomeração forneça o maior acréscimo (ou menor decréscimo) no valor da modularidade;
4 Salve a divisão resultante e sua modularidade;
5 **fim**
6 Escolha a divisão com a melhor modularidade encontrada.

É importante ressaltar que apenas comunidades que possuam pelo menos uma aresta ligando seus vértices podem ser possivelmente unidas pelo algoritmo. Isso limita a um máximo de p pares de comunidades, em que p é o número de arestas da rede. A variação em Q, ao se unir duas comunidades, é dada pela Equação 3.2.

$$\Delta Q = e_{ij} + e_{ji} - 2a_i a_j = 2(e_{ij} - a_i a_j). \tag{3.2}$$

O valor inicial de e_{ij} é igual à metade do grau de cada vértice, uma vez que inicialmente cada comunidade é formada por apenas um vértice. Após a junção de duas comunidades, os valores de e_{ij} devem ser atualizados.

3.1.2 Detecção de Comunidades pelo AdClust

O princípio do AdClust (Ye *et al.*, 2008) é o de que cada vértice adapte-se, podendo ir para comunidades vizinhas que exerçam uma maior atração do que sua comunidade atual. Esse procedimento é realizado até que um equilíbrio seja atingido, naturalmente obtendo um melhor valor para a modularidade Q proposta em (Newman, 2004). Forças de atração de um vértice, são definidas pelas Equações 3.3 e 3.4:

$$F_{in}^{(v)} = e_{in}^{(v)} - \frac{k(d_{in}^{(v)} - k)}{2E}, \tag{3.3}$$

$$F_{out}^{(u)} = \max_{u \in V} \left\{ e_{out}^{(u)} - \frac{k d_{out}^{(u)}}{2E} \right\}, \tag{3.4}$$

em que:

- $F_{in}^{(v)}$ é a força de atração para a comunidade v em que o vértice está;
- $F_{out}^{(u)}$ é a força de atração sobre o vértice exercida pela comunidade vizinha u;
- $e_{in}^{(v)}$ é o número de arestas que conectam o vértice à sua atual comunidade v;
- $d_{in}^{(v)}$ é o grau total dos vértices dentro da comunidade v;
- $e_{out}^{(u)}$ é o número de arestas que conectam o vértice à comunidade vizinha u;
- $d_{out}^{(u)}$ é o grau total dos vértices dentro da comunidade u;
- k é o grau do vértice;
- E é o número total de arestas da rede;
- V representa o conjunto de comunidades com pelo menos uma aresta conectada ao vértice, com exceção da comunidade ao qual o vértice pertence.

Em outras palavras, as Equações 3.3 e 3.4 representam o número real de arestas que ligam certo vértice à comunidade, menos o número esperado de arestas ligadas à comunidade que um

vértice de mesmo grau teria. Se existir F_{out} maior do que F_{in}, o vértice muda de comunidade (vai para a comunidade que exerce maior atração). A Figura 3.2 ilustra esse comportamento. O Algoritmo 3.2 descreve o pseudocódigo do AdClust.

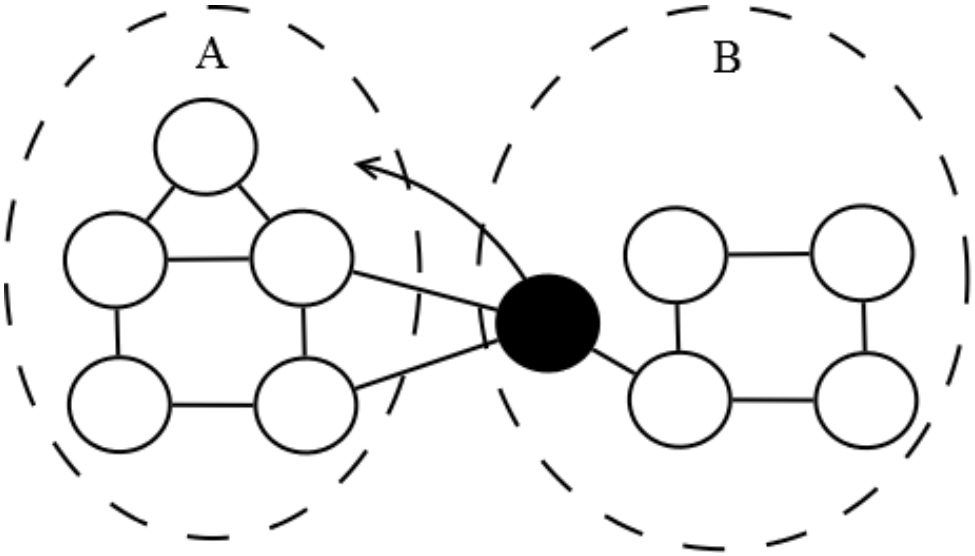


Figura 3.2: Princípio de funcionamento do AdClust: vértice em preto possui $F_{out} > F_{in}$ e, portanto, passa a pertencer à comunidade A.

Algoritmo 3.2: Pseudocódigo do AdClust.

Entrada: Um grafo com n vértices;
Saída: Uma estrutura de comunidades;
1 Separe os vértices em n comunidades;
2 Calcule F_{in} e F_{out} para cada vértice. Movimente os vértices para a comunidade com maior força de atração;
3 Repita 2 até que todos os vértices não mudem de comunidade. Salve a estrutura de comunidades encontrada e o valor de Q para a mesma;
4 Calcule a variação de Q para todos os possíveis pares de comunidades encontrados no Passo 3;
5 Escolha o par com o maior Q e una as respectivas comunidades;
6 Repita os Passos de 2 até 5 até que todos os vértices formem uma única comunidade;
7 Escolha a estrutura de comunidades com o maior Q encontrado.

É importante destacar que a ordenação dos vértices não influencia no desempenho do AdClust, sendo que no Passo 2 do Algoritmo 3.2, as forças são calculadas para todos os vértices da rede e, somente após isso, os vértices são deslocados para a comunidade de maior atração. Em comparação com o FA, esse método introduz uma rotina adicional (representada pelos Passos 2 e 3 do Algoritmo 3.2) de ordem $O(n)$. No artigo em que a técnica é apresentada (Ye *et al.*,

2008), os autores afirmam que as estruturas de comunidade detectadas pelo AdClust, em geral apresentam melhor modularidade Q (ver Seção 2.4.1) do que as estruturas encontradas pelo FA.

3.1.3 Detecção de Comunidades pelo EO

O algoritmo EO proposto em (Duch e Arenas, 2005) busca determinar melhores estruturas dividindo as comunidades existentes. Essa técnica baseia-se em uma métrica que verifica a contribuição de cada vértice para o valor de modularidade total. Tal valor é dado pela Equação 3.5.

$$q_i = k_{r(i)} - k_i a_{r(i)}, \tag{3.5}$$

em que:

- q_i é a contribuição do vértice i para a modularidade Q;
- $r(i)$ é a comunidade que contém o vértice i;
- $k_{r(i)}$ é o número de arestas que o vértice i possui para vértices da mesma comunidade;
- k_i é o grau do vértice i;
- $a_{r(i)}$ é a fração das arestas com pelo menos um vértice pertencente à comunidade $r(i)$.

A Equação 3.5 é calculada seguindo a ideia de que a modularidade Q (Newman, 2004) pode ser reescrita como um somatório das contribuições individuais de cada vértice, como mostra a Equação 3.6.

$$Q = \frac{1}{2L} \sum_i q_i. \tag{3.6}$$

Para obter a contribuição relativa de cada vértice para o valor de modularidade, a contribuição individual q_i é dividida pelo grau do vértice i, representado por k_i, resultando na contribuição relativa do vértice i, conforme a Equação 3.7. A partir dessa métrica, foi proposta uma busca heurística (Duch e Arenas, 2005) para encontrar uma estrutura de comunidades bem representativa, conforme mostra o Algoritmo 3.3.

$$\lambda_i = \frac{q_i}{k_i} = \frac{k_{r(i)}}{k_i} - a_{r(i)}. \tag{3.7}$$

Algoritmo 3.3: Pseudocódigo do EO.

Entrada: Um grafo com n vértices;
Saída: Uma estrutura de comunidades;
1 Divida os vértices em duas comunidades aleatórias;
`// cada grupo de vértices conexos em uma partição, representa uma comunidade`
2 Enquanto Q ainda pode ser melhorado, mude o vértice com menor contribuição relativa para o outro grupo;
3 Salve Q para a divisão atual; Remova as arestas entre todas as comunidades;
4 **se** Q *pode ser melhorado* **então**
5 | repita os Passos de 1 a 4 para cada grupo conectado;
6 **fim**

Os desenvolvedores do EO propõem substituir a seleção do vértice com menor contribuição relativa (ver Passo 2 do Algoritmo 3.3) por uma probabilidade de selecionar tal vértice (Boettcher e Percus, 2002). Essa probabilidade é dada pela Equação 3.8:

$$P(q)\alpha q^{-\tau}, \tag{3.8}$$

em que:

- q é o ranque do vértice, sua posição em um vetor dos λ_i's ordenados de forma decrescente;
- $\tau = 1 + \frac{1}{ln(n)}$, em que n é o número de vértices da rede.

Os autores afirmam que a técnica é a que encontra os melhores valores de modularidade para problemas de *benchmark* testados até a data de publicação do artigo (2005). No entanto, em (Ye *et al.*, 2008), o AdClust apresenta melhores valores de modularidade para alguns desses mesmos problemas. Comparações entre os algoritmos apresentados nas Seções 3.1.1, 3.1.2 e 3.1.3 são realizados na Seção 3.1.4, justificando o CDA escolhido para ser implementado neste trabalho: o FA.

3.1.4 Adequação dos algoritmos de detecção de estruturas de comunidades no desenvolvimento de EDAs

Os algoritmos apresentados podem ser comparados considerando dois diferentes fatores: (i) o valor de modularidade encontrado e (ii) o tempo de execução. Com relação ao primeiro item, os artigos que apresentam os algoritmos desenvolvidos após o FA realizam comparações com o mesmo, em que conseguem obter melhores resultados. Dentre os três algoritmos estudados, os melhores resultados referentes à modularidade são obtidos pelo AdClust, como mostra a

Tabela 3.1, que apresenta os valores de modularidade obtidos pelos algoritmos para dois problemas de *benchmark* (Newman, 2001; Zachary, 1977) (resultados extraídos de (Duch e Arenas, 2005) e (Ye *et al.*, 2008)).

Tabela 3.1: Comparação dos valores de modularidade Q obtidos pelos algoritmos EO, AdClust e FA para duas redes diferentes.

Redes	Q_{FA}	Q_{EO}	$Q_{AdClust}$
Zachary	0.3810	0.4188	0.4198
Rede de Cientistas	0.6683	0.6790	0.7610

Com relação ao tempo de execução, tanto o AdClust quanto o EO exigem mais tempo do que o necessário pelo FA, como apontado pelos autores das técnicas (Duch e Arenas, 2005; Ye *et al.*, 2008). No algoritmo CD-BOA desenvolvido neste projeto, Seção 4.1, é necessário executar um algoritmo de detecção de comunidades a cada nova geração. Dessa forma, é de interesse que o algoritmo escolhido seja de rápida execução. Devido a esse fato, o FA foi escolhido para implementação.

Testes foram realizados com uma versão do CD-BOA (Seção 4.1) utilizando o AdClust implementado, uma vez que este é o que apresentou melhores valores de modularidade, segundo a Tabela 3.1. No entanto, tal sucesso não foi refletido no EDA implementado com o AdClust, como mostrado nos experimentos preliminares do Apêndice D. Por esse motivo, o algoritmo escolhido para ser utilizado nas versões do CD-BOA (Seção 4.1) e StrOp (Seção 4.2) utilizadas neste trabalho, foi o FA.

Deve-se observar ainda que outras técnicas têm sido desenvolvidas para detecção de estruturas comunidades (Pujol *et al.*, 2006; Reichardt e Bornholdt, 2004). No entanto, devido aos resultados interessantes do FA e, ao fato de seu tempo de execução não ser o termo dominante da complexidade dos algoritmos CD-BOA e StrOp propostos neste trabalho (demonstrado nas Seções 6.4.1 e 6.4.2), tornou-se de menor importância para o desenvolvimento dos algoritmos propostos o estudo de outras técnicas mais eficientes que o FA. Além disso, o fato da implementação do AdClust não ter contribuído na melhoria das soluções encontradas pelo EDA desmotivou a investigação de outras técnicas capazes de encontrar melhores valores de modularidade.

3.2 *Técnicas de Reamostragem*

Nesta seção, são apresentadas duas técnicas de reamostragem para inferir propriedades de um estimador estatístico. O estudo dessas técnicas mostrou-se útil para o desenvolvimento de EDAs melhores quando se consideram: i) uma população como sendo uma amostra do espaço de busca e ii) as partições estimadas pelos EDAs em cada geração, como as variáveis estatísticas. Observe que nenhuma dessas duas técnicas são diretamente usadas nos algoritmos propostos neste trabalho, no entanto, inspiram o desenvolvimento do método de reamostragem proposto na Seção 4.3.

As Seções 3.2.1 e 3.2.2 a seguir, apresentam as técnicas de reamostragem *Bootstrap* e *Jackknife*, respectivamente. A Seção 3.2.3 apresenta uma discussão sobre como essas técnicas podem ser utilizadas no desenvolvimento de EDAs e discute, sucintamente, outros algoritmos desenvolvidos no grupo de pesquisa do SEER[1] do ICMC-USP que utilizam estratégias similares.

3.2.1 *Bootstrap*

Bootstrap[2] (Efron e Gong, 1983) é um método de reamostragem para inferir propriedades de um estimador estatístico. Para entender a utilidade do *bootstrap*, será abordado o problema de estimar o erro padrão da média de uma amostra. Sendo $X = \{x_1, x_2, x_3, ..., x_n\}$ uma amostra de um conjunto de dados, pode-se calcular a média amostral utilizando a equação:

$$\overline{x} = \sum_{i=1}^{n} \frac{x_i}{n}. \tag{3.9}$$

Da amostra também pode-se estimar a acurácia da média calculada por meio do erro padrão estimado ($\widehat{\sigma}$), dado pela Equação 3.10.

$$\widehat{\sigma} = \sqrt{\frac{1}{n(n-1)} \sum_{i=1}^{n} (x_i - \overline{x})^2}; \tag{3.10}$$

Uma limitação do uso da Equação 3.10 é que esta não se estende para outros estimadores como, por exemplo, a mediana. A técnica *bootstrap* possibilita realizar tal extensão, como explicado a seguir.

[1] Página do SEER: `http://www.icmc.usp.br/~seerweb/`, acessado em 27/08/2012

[2] *Bootstrap*, em inglês, é o nome de um acessório para ajudar a calçar botas. O nome é mantido na literatura em português.

Considere o conjunto $X^* = \{x_1^*, x_2^*, ..., x_n^*\}$, em que cada valor x_i^* é obtido aleatoriamente (com probabilidade $1/n$) da amostra inicial X. Dessa forma, X^* pode conter elementos de X repetidos e, por conseguinte, não conter alguns outros elementos de X. Em outras palavras, X^* é uma reamostragem de X. Seja $\widehat{\rho}^* = \widehat{\rho}(x_1^*, x_2^*, ..., x_n^*)$ o valor da estatística sendo calculada (exemplo: média ou mediana) é possível calcular uma estimativa por *bootstrap* para o erro padrão de tal estatística, por meio de B reamostragens e da Equação 3.11 (Efron e Gong, 1983):

$$\widehat{\sigma}_B = \sqrt{\left(\sum_{b=1}^{B}(\widehat{\rho}^{*b} - \widehat{\rho}^{*\bullet})^2\right)/(B-1)}, \; em \; que \; \widehat{\rho}^{*\bullet} \equiv \frac{\sum_{b=1}^{B}\widehat{\rho}^{*b}}{B}. \tag{3.11}$$

É possível observar que a Equação 3.11 é mais genérica que a Equação 3.10, pois é um cálculo possível de ser realizado para qualquer estatística $\widehat{\rho}$, não estando restrito à média.

3.2.2 *Jackknife*

Assim como o *Bootstrap* (Seção 3.2.1), a técnica *Jackknife*[3] (Efron e Gong, 1983) busca inferir propriedades de um estimador estatístico por meio de reamostragens. Com o objetivo de estimar o erro padrão para outras estatísticas além da média, o *Jackknife* considera $\overline{x}_{(i)}$ como sendo a média de todos os valores da amostra, com exceção do i-ésimo elemento, calculada pela Equação 3.12.

$$\overline{x}_{(i)} = \frac{1}{n-1}\sum_{j \neq i} x_j. \tag{3.12}$$

A partir da Equação 3.12, outra expressão para a média dos valores de X que pode então ser calculada como $\overline{x}_{(\cdot)}$, dada pela Equação 3.13.

$$\overline{x}_{(\cdot)} = \sum_{i=1}^{n}\frac{\overline{x}_{(i)}}{n}. \tag{3.13}$$

A partir dos valores de $\overline{x}_{(i)}$ e $\overline{x}_{(\cdot)}$ calcula-se a estimativa do erro padrão do *Jackknife* como mostra a Equação 3.14.

$$\widehat{\sigma}_J = \sqrt{\frac{n-1}{n}\sum_{i=1}^{n}(\overline{x}_{(i)} - \overline{x}_{(\cdot)})^2}, \tag{3.14}$$

[3] *Jacknife* significa canivete em inglês. O próprio nome em inglês é mantido na literatura em português.

Embora a Equação 3.10 seja equivalente à Equação 3.14, a última pode ser generalizada para outro estimador dado por $\widehat{\theta} = \widehat{\theta}(x_1, x_2, ..., x_n)$. Para esse fim, deve-se substituir $\overline{x}_{(i)}$ por $\widehat{\theta}_{(i)} = \widehat{\theta}(x_1, ..., x_{i-1}, x_{i+1}, ..., x_n)$ e $\overline{x}_{(\cdot)}$ por $\widehat{\theta}_{(\cdot)} = \sum_{i=1}^{n} \frac{\widehat{\theta}_{(i)}}{n}$.

3.2.3 Reamostragem no Desenvolvimento de Novos EDAs

As técnicas de reamostragem apresentadas podem ser exploradas neste projeto da seguinte forma: a partir de reamostragens das melhores soluções de uma população (Seção 2.1), é possível construir vários modelos probabilísticos a partir de um mesmo conjunto de soluções selecionadas. Então, uma comparação entre os modelos obtidos pode ser realizada, visando obter um único modelo mais verossímil para o problema.

Outra possível utilização é expandir um conjunto de selecionados (soluções promissoras) por meio de reamostragem da população. Com isso, poder-se-ia reduzir significativamente o tamanho da população inicial a partir da qual se escolhe um conjunto de selecionados. Esse resultado é especialmente interessante para os DecOA (Seção 4.2), uma vez que usam apenas uma população. Assim, o impacto dessa redução no número de soluções avaliadas pode ser mais significativo.

A estratégia de construção de um modelo probabilístico por reamostragem realizada poderia se mostrar inviável ao se utilizar um grande número de reamostragens, uma vez que os algoritmos para a construção desses modelos exigem tempo computacional relativamente grande (Capítulo2). Como o *Jackknife* realiza n reamostragens e EDAs normalmente trabalham com grandes populações (n grande), essa técnica não é adequada para se utilizar neste projeto (a complexidade do EDA seria multiplicada por n). Por outro lado, a forma como o número de reamostragens é controlado no *Bootstrap* torna essa técnica viável de ser utilizada no desenvolvimento de um novo EDA. Além disso, resultados da literatura mostram desempenho superior da técnica *Bootstrap* sobre o *Jackknife* (Efron e Gong, 1983).

Outros trabalhos do grupo de pesquisa do Laboratório de Computação Reconfigurável (LCR)[4] também investigaram aspectos relacionados a reamostragens na eficiência e eficácia de EDAs. Dentre esses trabalhos, destacam-se as técnicas de redução de espaço de busca, que trabalham com estratégias semelhantes a de reamostragem para encontrar regiões promissoras: o Algoritmo de Otimização de Domínio (DOA, do inglês *Domain Optimization Algorithm*) e a Amostragem Inteligente (SS, do inglês *Smart Sampling*). Além dessas, uma estratégia simples para aumento de eficiência de algoritmos de busca também foi desenvolvida: o Gerenciamento de Tamanho da População (PSM, do inglês *Population Size Management*). Na sequência, essas

[4]Página do laboratório: `http://lcr.icmc.usp.br`, acessado em 06/08/2012

técnicas são sucintamente descritas, por motivarem a técnica de reamostragem proposta para EDAs na Seção 4.3.

Algoritmo de Otimização de Domínio

O DOA (Melo *et al.*, 2007) é um Algoritmo de Redução de Espaço de Busca (SRA, do inglês *Search-space Reduction Algorithm*) (Barkat Ullah *et al.*, 2008; Barolli *et al.*, 2007; Melo *et al.*, 2007). SRAs são técnicas de pré-processamento, que determinam regiões promissoras antes de se utilizar um algoritmo de otimização. Nesse contexto, o DOA trabalha com modelos simples (regressão linear (Yong e Sannomiya, 2000)) que identificam as tendências em cada variável que geram soluções de qualidade superior. A partir desses modelos, pode-se eliminar regiões não promissoras que correspondem a valores de variáveis que tendem a gerar soluções de baixa qualidade. Com isso, realizam-se reamostragens apenas em regiões promissoras. Esse processo possibilita a diminuição do número de avaliações necessárias para que um algoritmo de otimização, utilizado ao final do processo, refine as soluções com o intuito de encontrar o ótimo global.

Amostragem Inteligente

O SS (Melo e Delbem, 2009) é uma técnica com o mesmo propósito do DOA: encontrar regiões promissoras no espaço de busca. Porém, ao invés de utilizar regressão linear para guiar o processo de reamostragem, o SS utiliza: i) uma técnica de aprendizagem de máquina para separar soluções de alta qualidade das de baixa qualidade; ii) um procedimento relativamente simples de geração de pontos (reamostragem) próximos aos melhores até então encontrados é utilizado para obter soluções cada vez melhores; e iii) outra técnica de aprendizagem de máquina para separar as regiões finais determinadas pelas melhores soluções. Por fim, um algoritmo de otimização é inicializado com as soluções de alta qualidade dentro de cada uma das regiões promissoras encontradas.

Gerenciamento de Tamanho de População

O PSM (Melo *et al.*, 2009) é uma estratégia desenvolvida para aumentar a eficiência do ECGA, para determinar o ótimo global com menos avaliações da função objetivo. A técnica é baseada na ideia de que ao se utilizar uma população de tamanho grande na primeira geração do ECGA, realiza-se uma melhor amostragem inicial do espaço de busca. A seleção de melhores indivíduos dessa população gera um conjunto de amostras mais representativas, possibilitando a construção de um modelo probabilístico significativamente melhor. Em outras palavras, a

qualidade do modelo encontrado aumenta com o tamanho da amostra inicial. Nas próximas gerações do ECGA, o PSM mostra que se pode utilizar populações de tamanho bem menores (amostradas em uma região promissora do problema indicada pelo modelo de alta qualidade) que a de primeira geração sem acarretar prejuízos na qualidade do modelo. Com isso, o número de avaliações requeridas pelo algoritmo diminui significativamente, com reduções de 30% a 70% conforme o tamanho e tipo de problema. Apesar de originalmente investigado com base apenas no ECGA, o PSM é simples e fácil de ser adaptado para outros algoritmos populacionais, com potencial para proporcionar ganhos similares.

3.3 Considerações Parciais

Neste capítulo foram apresentadas técnicas de detecção de estruturas de comunidades (Seção 3.1) e de reamostragem (Seção 3.2). Tais técnicas motivaram o desenvolvimento dos algoritmos propostos neste trabalho (Capítulo 4).

A técnica de reamostragem para EDAs proposta na Seção 4.3 tem como princípio construir vários modelos (um para cada reamostragem da população). A partir desses vários modelos, é possível construir um modelo consenso, mais representativo do problema. Isso resulta em duas contribuições para os EDAs que utilizam tal técnica: i) a necessidade de uma população de tamanho menor para se obter modelos representativos, e ii) um menor número de avaliações torna-se necessário para encontrar soluções de interesse, devido à qualidade dos melhores modelos construídos.

Os CDAs podem contribuir na construção de EDAs, como o CD-BOA (Seção 4.1) e StrOp (Seção 4.2), ao identificarem conjuntos de variáveis relacionadas em problemas como os explicados no Capítulo 5. Como visto nas Seções 2.2 e 2.4, a identificação das partições do problema pode auxiliar significativamente na geração de novas soluções promissoras. A determinação das partições é especialmente relevante para problemas com comportamentos similares a armadilhas (Seção 5.1), nos quais a não identificação das variáveis correlacionadas pode comprometer o desempenho do algoritmo de otimização.

CAPÍTULO

4

Algoritmos Propostos

As seções a seguir descrevem o funcionamento de todos os algoritmos propostos neste trabalho. A Seção 4.1 explica o CD-BOA, que combina uma adaptação do FA (Seção 3.1.1) com o BOA (Seção 2.6); a Seção 4.2 apresenta o StrOp; a Seção 4.3 propõe uma nova técnica de reamostragem para EDAs; e, por fim, a Seção 4.4 explica as adaptações feitas ao FA, resultando no FA adaptado (aFA, do inglês *adapted Fast Algorithm*) utilizado nas demais técnicas apresentadas neste capítulo.

4.1 Algoritmo de Otimização Bayesiano com Detecção de Comunidades

O novo EDA, denominado CD-BOA, resulta principalmente da modificação substancial do BOA, ao se utilizar um algoritmo de detecção de comunidades (como por exemplo o FA) com o objetivo de construir melhores modelos probabilísticos. Dessa forma, o CD-BOA possui dois novos procedimentos em relação ao BOA original. Isso é ilustrado pela Figura 4.1, que possui duas etapas adicionais (indicadas pelas setas 3 e 4) quando comparada ao fluxograma do algoritmo BOA, apresentado na Figura 2.7. Assim como o BOA, o CD-BOA parte de uma população inicial gerada aleatoriamente. Então, são executadas as etapas representadas pela

setas da Figura 4.1, explicadas a seguir, até que o critério de parada escolhido seja atingido (Seção 2.1).

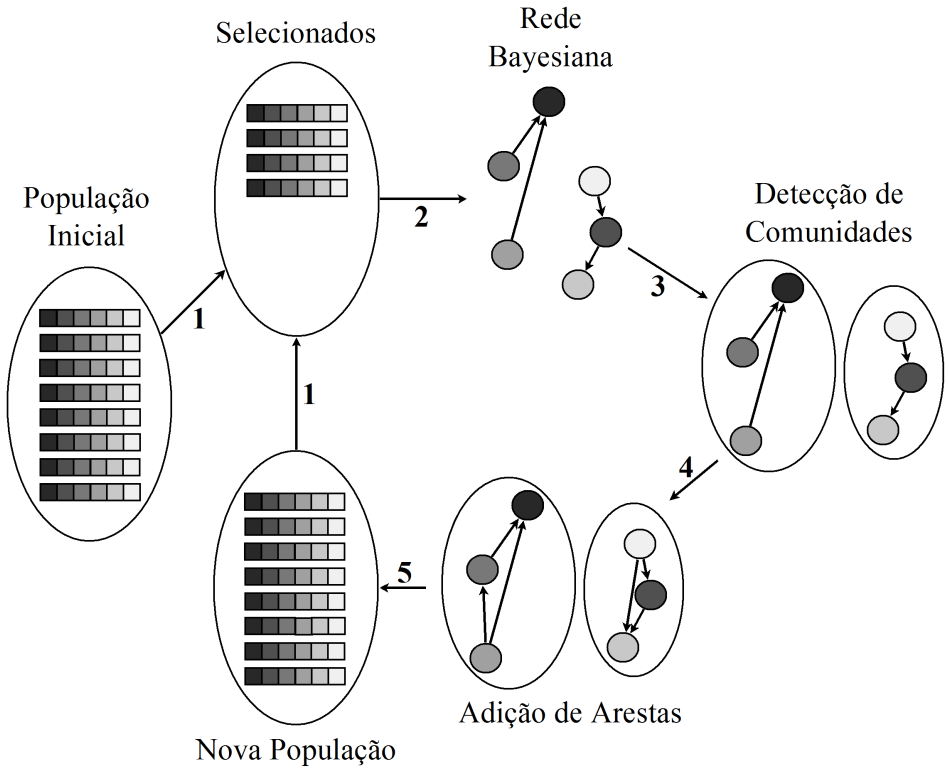


Figura 4.1: Ilustração do funcionamento do CD-BOA.

1. As soluções da população são avaliadas para a determinação de um conjunto de indivíduos promissores (selecionados). No CD-BOA implementado neste trabalho, assim como no BOA, a seleção utilizada foi a de truncamento, sendo selecionada a metade das soluções com melhores valores de *fitness*;

2. A BN é gerada a partir do algoritmo guloso BGS explicado na Seção 2.6.2, passando o parâmetro de entrada v_l no lugar do parâmetro v utilizado no BOA;

3. Um algoritmo de detecção de comunidades é aplicado sobre a BN para identificar as partições do problema (BBv's). O algoritmo utilizado neste trabalho foi o aFA (Seção 4.4), que é uma extensão do FA desenvolvida para ser utilizada no CD-BOA. O tamanho máximo das comunidades encontradas é limitado pelo parâmetro v_u do CD-BOA;

4. Conexões entre arestas de uma mesma comunidade são adicionadas. Com isso, é possível aumentar a força de relação entre as variáveis dentro de um mesmo BBv. Esse é um ponto chave, pois a BN obtida é gerada a partir de uma base de dados amostrada (conjunto de

indivíduos selecionados pelo EDA) e, portanto, algumas relações entre variáveis em geral podem não ser bem amostradas, conforme k e m aumentam (ver Seção 2.4), ou algumas relações podem não ser representadas na BN uma vez que a busca gulosa usada pelos algoritmos de construção da BN não analisam todas as potenciais redes, gerando uma rede subótima, isto é, com imperfeições. Dessa forma, a detecção de comunidades funciona como uma técnica de reparo efetuando algumas mudanças que possibilitam construir BNs mais confiáveis;

5. Novas soluções são construídas a partir da BN obtida e do conjunto de soluções selecionadas, a partir do qual se extraem as instâncias de partições mais prováveis de cada BB. Essa etapa é realizada da mesma forma que no BOA (Etapa 3 da Figura 2.7).

Foram mencionados dois parâmetros de entrada nessas cinco etapas explicadas: v_l e v_u. Os nomes desses parâmetros foram escolhidos para representarem a ideia de limites inferiores e superiores para o parâmetro v utilizado no BOA (do inglês *lower* v e *upper* v). No Passo 2, o parâmetro v_l é passado como parâmetro para a busca gulosa responsável por construir a primeira versão da BN. Ao fazer isso, a BN resultante terá para cada vértice, um máximo de v_l variáveis pais associadas. No entanto, o Passo 4 adiciona novas conexões entre as arestas, aumentando este limite para v_u pais por vértice na BN final encontrada.

Para entender as vantagens em se utilizar esses dois parâmetros, é preciso esclarecer primeiramente algumas características do BOA. Em primeiro lugar, o número esperado de pais por vértice para um problema com partições formadas por k variáveis é de $k-1$. Dessa forma, um parâmetro adequado v para o BOA deve possibilitar que essas ligações sejam encontradas, isto é, $v \geq k-1$. Partindo desse princípio, pode-se pensar que utilizar um grande valor para v é suficiente para encontrar de forma eficiente a solução de qualquer problema em que $k-1 \leq v$. No entanto, esta conclusão não está correta, uma vez que experimentos das Seções 6.2 e 6.4 revelam uma queda no desempenho do BOA à medida que v afasta-se do valor $k-1$, o que é justificado pela complexidade do algoritmo, que cresce exponencialmente com v.

O parâmetro ideal a ser utilizado pelo BOA é $v = k-1$, como feito em (Pelikan *et al.*, 1999), sendo este um valor não conhecido em problemas de caixa preta (He *et al.*, 2007), em que o tamanho dos BBs não é conhecido a priori. Nesse cenário, o CD-BOA apresenta a vantagem de possibilitar o uso de um intervalo estimado de possíveis valores para k ao definir os parâmetros v_l e v_u.

Embora o CD-BOA possua etapas adicionais quando comparado ao BOA, é mostrado que o tempo de execução do mesmo é inferior ao do BOA quando é utilizado $v_l < v < v_u$ (Seção 6.4.1). Para explicação desse comportamento, considere a execução dos dois algoritmos:

BOA utilizando $v = k - 1$ (parâmetro ideal) e CD-BOA utilizando $v_l < v < v_u$. A princípio, é esperado que a BN encontrada pelo BOA após a BGS represente todas as ligações existentes entre variáveis, enquanto a primeira versão da BN encontrada pelo CD-BOA após Etapa 2 (da Figura 4.1) contenha apenas algumas das ligações existentes. No entanto, após a execução das Etapas 3 e 4 da Figura 4.1, as ligações restantes são inferidas a partir das comunidades encontradas e, então, inseridas na BN de uma forma computacionalmente menos custosas. Isso ocorre pois, apesar das etapas adicionais do CD-BOA acrescentarem um termo polinomial em sua complexidade, o tempo de computação do BOA, para costruir a rede utilizada, cresce de forma exponencial em relação a v, enquanto o tempo do CD-BOA, para construir a rede, cresce exponencialmente com v_l, que é menor que v (Seção 6.4.1).

Vale destacar que, embora o tempo de construção da rede no CD-BOA aumente apenas polinomialmente com v_u, é importante utilizar um valor aceitável para este parâmetro. A Tabela de Probabilidades (ver Seção 2.6) requer um tamanho máximo de 2^{v_u} para cada uma das variáveis. Assim, é crucial a restrição de v_u para que não se esgote a memória do computador (*overflow*) durante esse processo. O CD-BOA apresentado nesta Seção é sintetizado no Algoritmo 4.1.

Algoritmo 4.1: Algoritmo de Otimização Bayesiano com Detecção de Comunidades.

Entrada: Uma população P vazia;
Saída: Uma população P contendo soluções promissoras;
1 Inicialize a população P com soluções geradas aleatoriamente;
2 **enquanto** *critério de parada não atingido* **faça**
3 Selecione as 50% melhores soluções da população atual;
4 Construa uma BN usando a métrica K2 com o algoritmo de busca guloso BGS (Seção 2.6.2);
5 Use um algoritmo de detecção de estrutura de comunidades sobre a (BN);
6 Adicione todas as possíveis conexões entre arestas de uma mesma comunidade;
7 Substitua a metade da população original com pior *fitness* por novas soluções criadas a partir da BN e dos indivíduos selecionados (conforme realizado no BOA).
8 **fim**

4.2 Otimização por Decomposição

As definições de problemas aditivamente separáveis e decomponíveis, apresentadas na Seção 4.2.1, partem das ideias apresentadas em (Suh, 1990) e interpretadas em (Goldberg, 2002). Por sua vez, a Seção 4.2.2 apresenta um algoritmo baseado nesses conceitos.

4.2.1 Conceitos Fundamentais

Uma função é dita aditivamente separável se a mesma pode ser representada por uma soma de outras funções que possuem como parâmetros subconjuntos disjuntos das variáveis originais (sem sobreposição de BBs). A Equação 4.1 é um exemplo de uma função separável f.

$$f(x_1, x_2, x_3) = f_1(x_1) + f_2(x_2, x_3). \tag{4.1}$$

Ao tratar problemas de caixa preta (He *et al.*, 2007), o único conhecimento disponível sobre o problema é o número de variáveis e o valor de $f(x)$ para algumas soluções amostradas. O objetivo torna-se então encontrar um particionamento das variáveis do problema de forma que a solução para os subproblemas gere uma aproximação adequada da solução para o problema original. Essa ideia encontra-se representada pela Equação 4.2.

$$f(x) = f'(x) + \epsilon(x), \tag{4.2}$$

em que $f'(x)$ é uma função aditivamente separável, como definida anteriormente (Seção 5.4), e $\epsilon(x)$ representa a diferença entre o melhor valor de $f(x)$ e $f'(x)$. Uma função é dita decomponível se seu argumento (variáveis de entrada) pode ser particionado de alguma forma, tal que $\epsilon(x) \approx 0$. Se $\epsilon(x) = 0$ (a decomposição é perfeita), então $f(x) = f'(x)$ e, portanto, $f(x)$ é aditivamente separável.

Esta seção trata este problema de decomposição sob a perspectiva de EDAs, como o ECGA (Harik *et al.*, 2006) e o BOA (Seção 2.6 e o CD-BOA (Seção 4.1). Esses algoritmos buscam encontrar, a cada iteração, um modelo que identifique fortes relações entre variáveis do problema sendo tratado. Como definido na Seção 2.3, o conjunto de variáveis fortemente relacionadas identificadas por um particionamento das mesmas é chamado de BBv (Seção 2.3). Com base na partição de BBv's e nos valores prováveis para instâncias de cada partição, os EDAs substituem a aplicação de operadores reprodutivos (ver Seção 2.1) pelos seguintes passos:

1. Construção de um modelo probabilístico multivariado, que representa de forma sintética os indivíduos selecionados;

2. Criação de novas soluções a partir do modelo construído.

O Passo 1 consiste em construir um modelo que identifique conjuntos de variáveis correlacionadas. Em seguida, são calculados os valores mais prováveis para cada instância de uma partição. Uma vez construído tal modelo, o mesmo é utilizado para gerar novas soluções similares às selecionadas anteriormente. Note que os Passos 1 e 2 são realizados diversas vezes por um EDA convencional, construindo um modelo a cada geração do EDA.

No entanto, alguns algoritmos (Melo *et al.*, 2011; Vargas e Delbem, 2009) utilizam uma abordagem diferente, que consiste em detectar os BBv's do problema tratado uma única vez. Esse tipo de algoritmo é chamado neste trabalho de Algoritmos de Otimização por Decomposição (DecOA, do inglês *Decomposition Optimization Algorithm*).

A ideia de um algoritmo DecOA, é a de usar um algoritmo para detecção das BBv's do problema uma única vez e, então, aplicar uma busca considerando cada subproblema. A principal diferença é que, enquanto EDAs, como o ECGA ou o BOA, buscam encontrar um modelo que explique as melhores soluções da população atual, um DecOA foca em determinar um modelo que explique a decomposição do problema tratado por meio de um particionamento adequado de variáveis. De forma sintética, um DecOA pode ser descrito por duas etapas, ilustradas pela Figura 4.2 e explicadas a seguir:

- Etapa A: Decomposição de um problema original em subproblemas. Para isso, é utilizado um modelo probabilístico obtido a partir de um conjunto de indivíduos selecionados de uma população;
- Etapa B: Resolução dos subproblemas utilizando um algoritmo de busca como, por exemplo, uma simples busca gulosa (Seção 2.1).

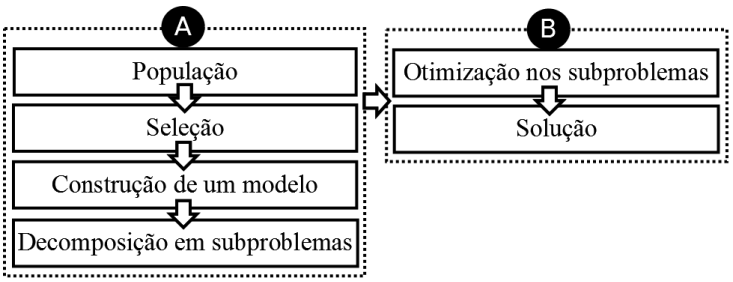


Figura 4.2: Abordagem de dois passos utilizada por um DecOA.

Para que exista uma grande probabilidade de que as partições do problema sejam corretamente encontradas, deve-se utilizar uma amostra inicial de soluções que seja grande o suficiente para que cada instância possível de um BBv esteja representada ao menos uma vez. A Equação 2.2 (ver Seção 2.4) é um limitante inferior do tamanho da população utilizada, que garante a amostragem adequada de cada instância. Como o modelo é encontrado uma única vez em um DecOA (com o foco em compreender o problema sendo tratado), devem ser utilizadas populações maiores (mais representativas do espaço de busca) do que as utilizada em EDAs tradicionais. Observe que não há gerações seguintes em que eventuais erros no modelo possam ser corrigidos.

Durante a Etapa A, um conjunto de soluções selecionadas da população é utilizado na construção de um modelo que identifique as relações entre as variáveis (assim como feito em EDAs). A partir do modelo construído, é identificado um particionamento das variáveis do problema original, isto é, uma decomposição do problema em subproblemas a serem resolvidos na próxima etapa do algoritmo.

A Etapa B utiliza uma busca que, considerando as partições do problema encontradas no Passo 1, encontre soluções para cada subproblema, resultando em uma estimativa de solução para o problema original. Exemplo de algoritmos a serem utilizados nesse passo são: (i) busca exaustiva (Cook *et al.*, 1997), (ii) EAs (Seção 2.1) e (iii) EDAs (Capítulo 2). É importante observar que o modelo proposto não é necessariamente um procedimento iterativo, dependendo da busca utilizada na Etapa B. O DecOA pressupõe que os problemas tratados possam ser decomponíveis e, portanto, pode tratar cada um dos BBv's identificados como problemas independentes. Observe que se o problema não for (perfeitamente) decomponível, as soluções geradas pelo DecOA serão aproximações de soluções para o problema original. Por outro lado, é possível também utilizar uma busca que considere relações entre os subproblemas encontrados, como feito em (Melo *et al.*, 2011), que utiliza o algoritmo DE para otimizar um problema composto por metavariáveis (cada metavariável corresponde a uma partição encontrada).

4.2.2 Otimização Direta

Como exemplo de um DecOA, é proposto neste trabalho o algoritmo de Otimização Direta (StrOp, do inglês *Straight Optimization*) cujo funcionamento é ilustrado na Figura 4.3. A Etapa A deste DecOA (responsável por encontrar as partições do problema) é representada pelas setas numeradas 1, 2 e 3, enquanto a Etapa B (otimização), é representada pela seta 4.

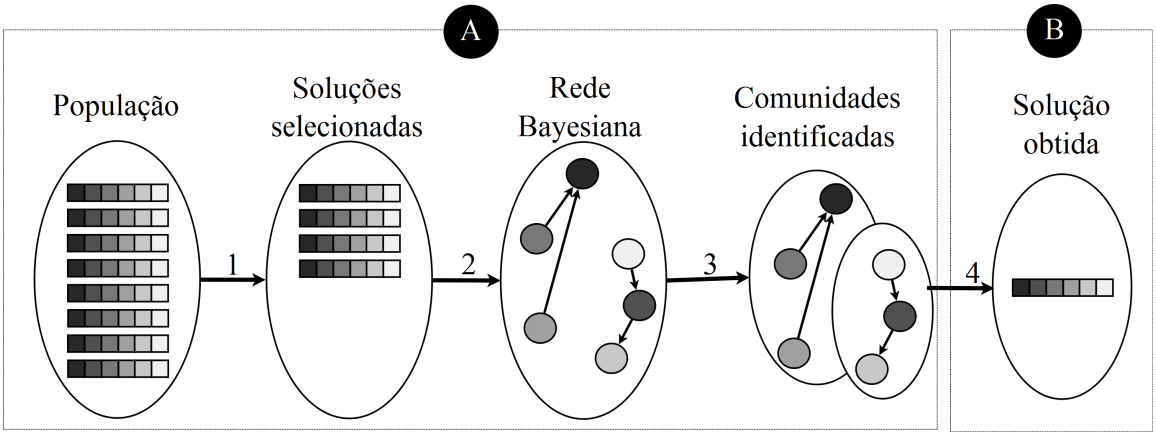


Figura 4.3: Funcionamento do StrOp.

Os Passos 1, 2 e 3 apresentados na Figura 4.3, são equivalentes aos Passos 1, 2 e 3 do CD-BOA (Figura da Seção 4.1). Sucintamente, o Passo 1 representa a seleção dos 50% melhores indivíduos da população. O Passo 2 utiliza a BGS para encontrar uma BN a partir dos indivíduos selecionados. O Passo 3 é a aplicação do aFA para encontrar uma estrutura de comunidades na BN que represente as partições do problema.

A busca utilizada na Etapa B (representada pela seta 4) é uma busca gulosa que, a partir da solução com melhor pontuação na população, testa todas as possíveis instâncias de cada partição, mantendo fixos os valores das demais partições e selecionando a melhor instância encontrada. A busca gulosa é adequada se cada partição possuir tamanho relativamente pequeno, uma vez que o número de avaliações realizado para cada partição cresce exponencialmente com o tamanho do BB. Por esse motivo, é necessário limitar o tamanho das partições encontradas. Isso é garantido pelo algoritmo aFA (Seção (Suh, 1990)), que restringe o tamanho das comunidades identificadas a partir do parâmetro de entrada v_u. O Apêndice H calcula valores limites para v_u, que garantem a eficiência do StrOp para uma faixa relativamente larga de tamanhos de BBs. O Algoritmo 4.2 apresenta o pseudocódigo do StrOp.

Algoritmo 4.2: Pseudocódigo do StrOp.

Entrada: Uma população P vazia;
Saída: Uma solução final (solucaoFinal);
1 InicializaPopulação(P);
2 AvaliaPopulação(P);
3 selecionadas = seleção(P);
`// Encontra a` BN `a partir das soluções selecionadas`
4 BN = encontraBN($selecionadas$);
`// Aplique o aFA pra identificar as BBv's`
5 partições = aFA(BN);
6 solução = melhorSolução(população);
7 **para** *cada* BBv_j *do particionamento* **faça**
`// Aplique busca exaustiva em` BBv_j
8 $instnciaBBv_j$ = buscaExaustiva(BBv_j);
`// Adicione a instância encontrada em uma solução final`
9 adicioneNaSoluçãoFinal(soluçãoFinal,instânciaBBv$_j$);
10 **fim**
11 retorne solucaoFinal.

A Figura 4.4 ilustra a busca gulosa utilizada. No exemplo apresentado, as soluções são compostas de quatro *bits* e as partições encontradas na Etapa B são representadas por BBv_1 e BBv_2, cada uma formada por dois *bits*. Inicialmente, são verificadas as pontuações de cada solução considerando todas as possíveis instâncias da partição representada por BBv_1, enquanto

os valores das varáveis de BBv_2 ficam fixos. A melhor solução é selecionada e o procedimento repete-se, testando todas as possíveis instâncias em BBv_2 com os valores das variáveis de BBv_1 fixos. A união das melhores instâncias encontradas para cada partição é a solução final retornada.

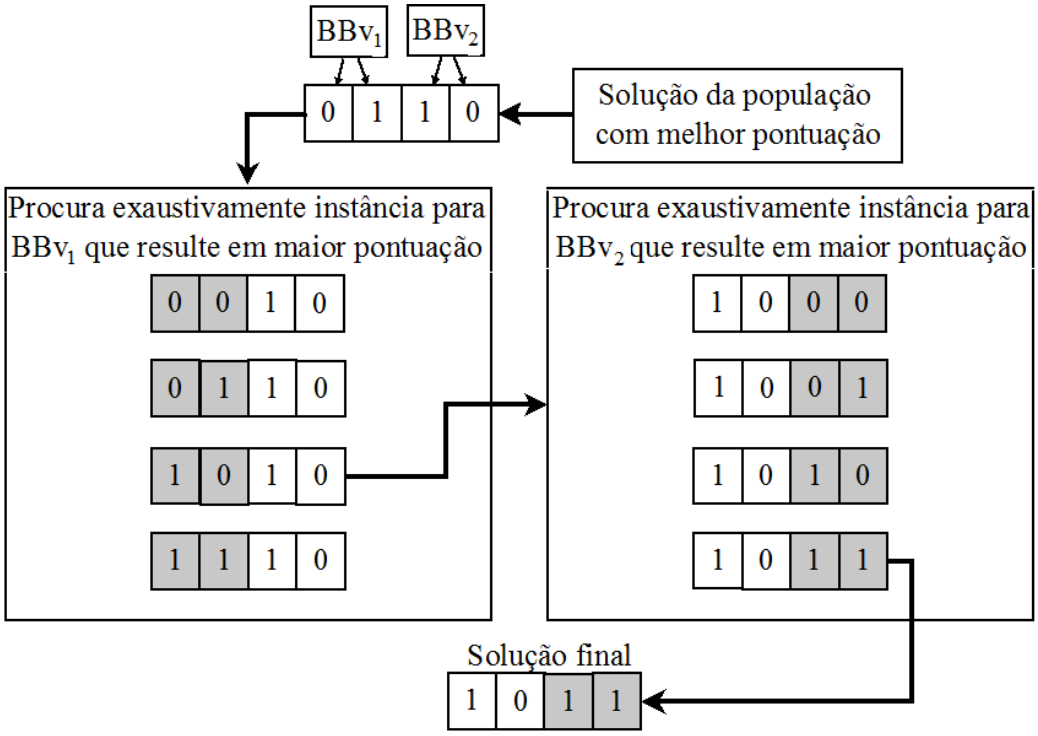


Figura 4.4: Busca gulosa no algoritmo StrOp.

4.3 Reamostragem para EDAs

Baseado na técnica de reamostragem *bootstrap*, explicada na Seção 3.2, foi proposto um método com o objetivo de obter melhores modelos probabilísticos, chamado de Reamostragem para EDAs (REDA, do inglês *Resampling for EDAs*). Ao utilizar este método em conjunto com os algoritmos CD-BOA e StrOp obtém-se, respectivamente, o CD-BOA+REDA e o StrOp+REDA. Esse método substitui a etapa de identificação das BBv's no EDA utilizado (CD-BOA ou StrOp) pelos passos explicados no Algoritmo 4.3.

A Linha 2 do Algoritmo 4.3 realiza reamostragens a partir da população atual, isto é, geram-se várias amostras de populações de selecionados. Uma reamostragem é realizada por meio de um torneio de 8 indivíduos (Seção 2.1), até que se tenha um conjunto de $n/2$ soluções selecionadas, que é o número de indivíduos normalmente selecionado pelo BOA (metade da população). A

Algoritmo 4.3: Pseudocódigo do REDA.

Entrada: Número de reamostragens a serem realizadas: R;
Entrada: População P já avaliada;
1 **para cada** *amostra de 1 até* R **faça**
2 Realize uma reamostragem da população P, selecionando n/2 soluções por torneio;
3 Construa uma BN a partir da reamostragem realizada;
4 Aplique um CDA para identificar BBv's;
5 Guarde BBv's encontradas na Lista A;
6 **enquanto** *houver alguma variável do problema não presente nas BBv's da Lista B* **faça**
7 Lista B recebe BBv com maior número de ocorrências da Lista A.
8 **fim**
9 **fim**

utilização de torneio permite a seleção de cojuntos distintos (cada conjunto corresponde a uma reamostragem)[1] resultando, posteriormente, na construção de diferentes modelos. Os experimentos realizados neste trabalham utilizam torneio de 8 indivíduos, pois o mesmo foi capaz de obter melhores resultados em uma comparação realizada com torneios de 2, 4, 8 e 16 indivíduos (Apêndice E).

Para o caso específico em que o algoritmo é chamado com o parâmetro R igual a 1, considera-se que a técnica REDA não é aplicada, isto é, ao executar o CD-BOA+REDA ou o StrOp+REDA com parâmetro $R = 1$ são executados, respectivamente, o CD-BOA (Seção 4.1) ou o StrOp (Seção 4.2).

A Linha 3 aplica o algoritmo BGS (Seção 2.6), no entanto, utilizando a reamostragem atual. A Linha 4 aplica o aFA (Seção 4.4) para encontrar as comunidades que representam as possíveis BBv's do problema. Todas essas etapas encontram-se ilustradas na Figura 4.5, em um exemplo com quatro reamostragens da população inicial.

As Linhas 5, 6 e 7 sintetizam um processo de seleção das BBv's mais comumente encontrados, até que todas as variáveis do problema estejam representadas em uma lista de BBv's. A aplicação desse método busca obter um consenso dos modelos encontrados, a fim de minimizar a quantidade de variáveis classificadas erroneamente em algumas das comunidades encontradas (representadas por BBv's com baixa ocorrência na lista). Um exemplo da aplicação dessas etapas encontra-se ilustrado pela Figura 4.6. Primeiro, as comunidades encontradas anteriormente são representadas como BBv's candidatas em uma lista e, então, as ocorrências mais comuns de BBv's são selecionadas para compor o modelo a ser utilizado.

[1] Observe que ao aplicar, por exemplo, seleção por truncamento mais de uma vez, o conjunto de selecionados seria o mesmo, o que é inadequado para realizar reamostragens.

É importante ressaltar as duas importantes características do REDA descritas na sequência: A primeira é que as reamostragens são realizadas a partir de uma população de indivíduos já avaliados e, portanto, não aumenta o número de avaliações exigido para a construção dos modelos probabilísticos. A segunda é que, embora nesse trabalho o método proposto tenha sido implementado para execução sequencial, as etapas de reamostragem e aplicação dos CDAs (Figura 4.5) podem ser realizadas paralelamente, melhorando o tempo de execução do algoritmo caso o *hardware* adequado para esse fim esteja disponível.

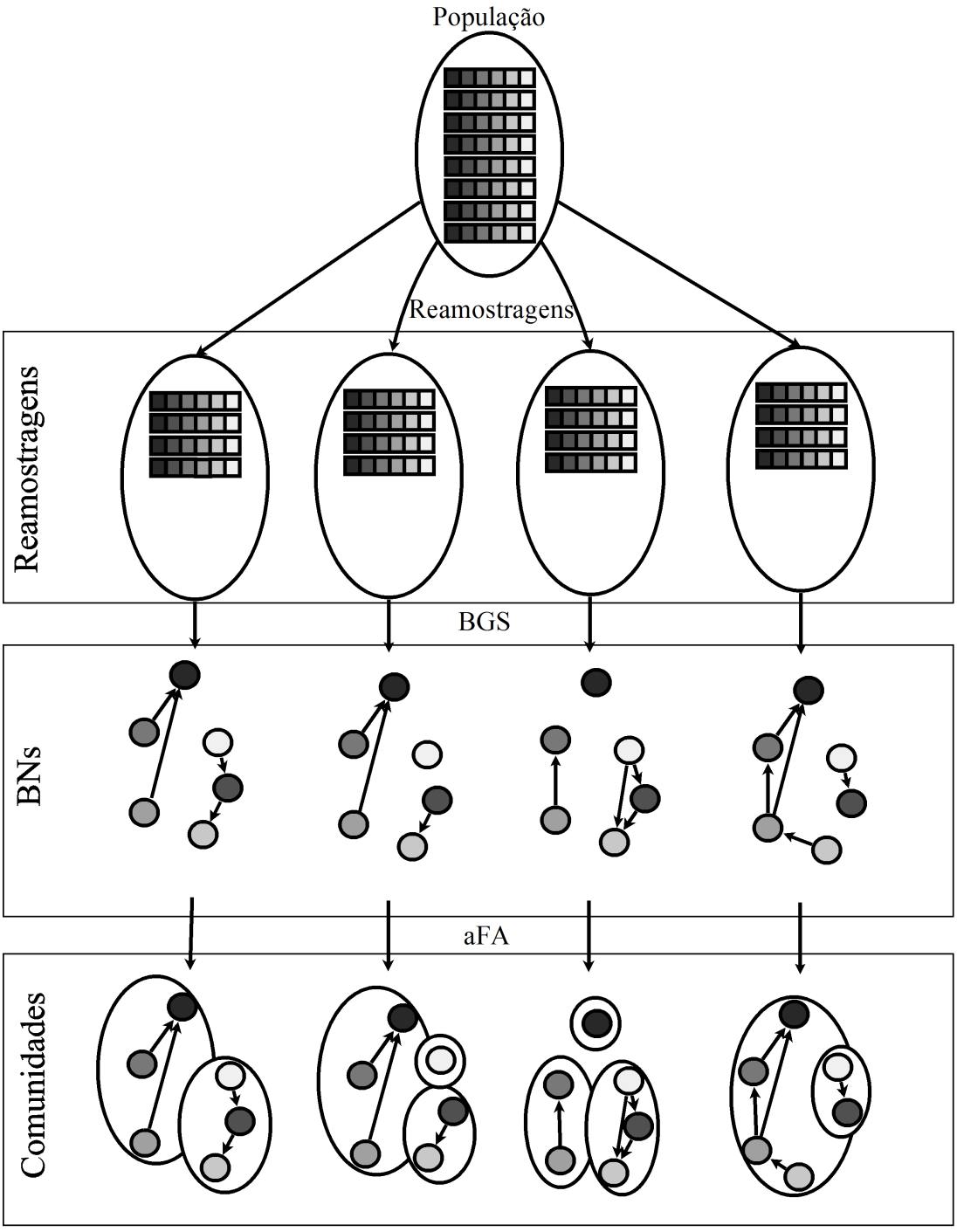


Figura 4.5: Ilustração das Linhas 2, 3 e 4 do Algoritmo 4.3.

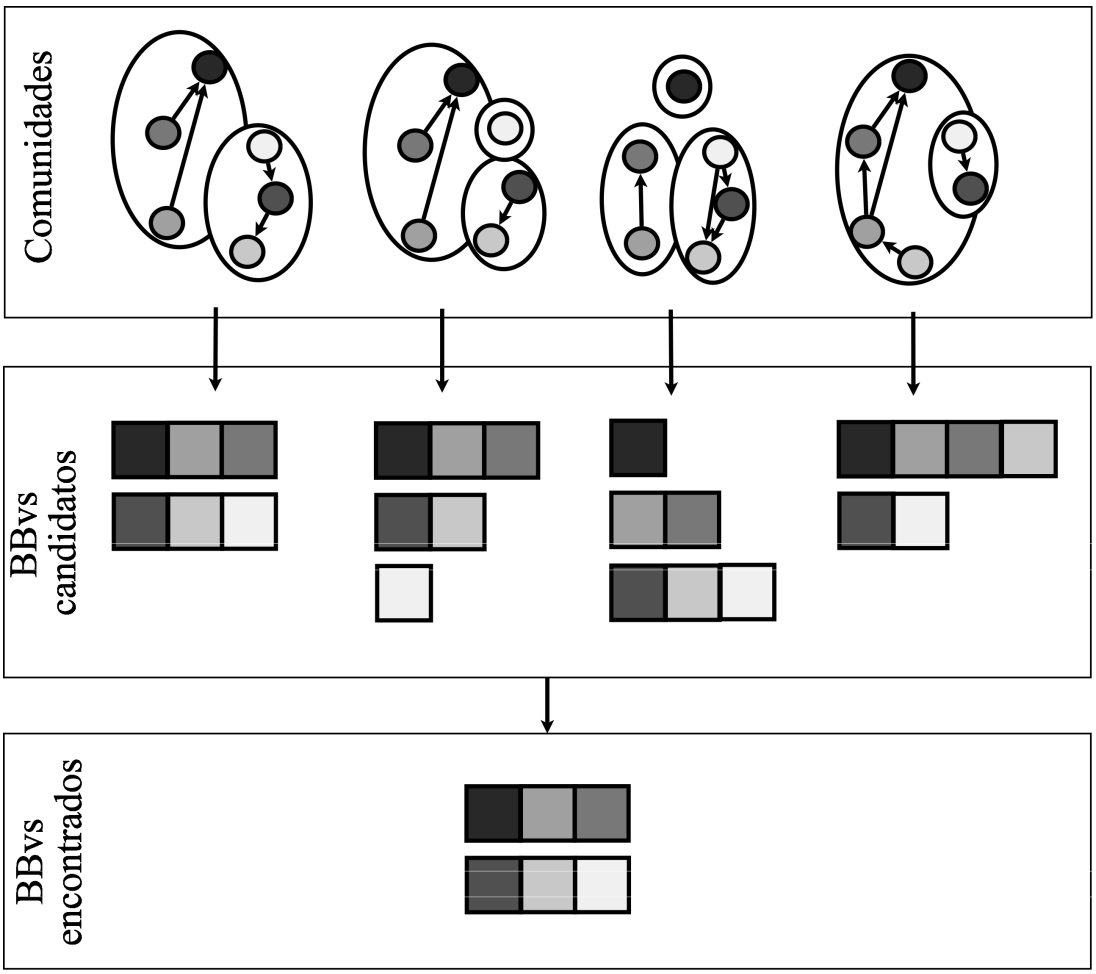


Figura 4.6: Ilustração das Linhas 5, 6 e 7 do Algoritmo 4.3.

4.4 Adaptações ao FA para EDAs

O FA (Seção 3.1.1) foi adaptado para poder ser utilizado pelo CD-BOA e StrOp, resultando no aFA. Para obter o aFA, é necessário aplicar ao FA as três modificações listadas a seguir:

1. A condição de parada "Enquanto número de comunidades > 1" apresentada no Algoritmo 3.1 é modificada para "Enquanto exista uma aresta entre duas comunidades". Essa modificação é necessária porque nem sempre a BN fornecida como entrada para o algoritmo é um grafo conexo (Diestel, 2005);

2. Após a execução normal do FA (Seção 3.1.1), o mesmo é aplicado recursivamente para cada comunidade com tamanho maior que v_u+1, isto é, o FA utiliza um subconjunto da rede original, que corresponde aos vértices pertencentes à respectiva comunidade;

3. A estrutura de comunidades retornada a cada chamada recursiva é a que possui melhor modularidade Q (Seção 3.1.1) e que possui mais do que uma comunidade. Isso garante que o algoritmo não itere indefinidamente (*loop* infinito) ao encontrar um caso em que a melhor modularidade sempre corresponda à estrutura de comunidades com uma única comunidade.

Com o uso do aFA, tem-se a garantia de que os BBv's encontrados não ultrapassam o tamanho v_u. Isso é necessário para manter a eficiência do aFA em seu uso com o algoritmo StrOp, uma vez que o tempo de execução da busca gulosa utilizada cresce exponencialmente com o tamanho das BBv's encontradas. Dessa forma, garante-se um limite superior para o tempo do algoritmo.

Outro benefício do aFA é que, ao encontrar comunidades maiores do que $v_u + 1$ durante a execução do CD-BOA, nem todas as variáveis dentro da comunidade encontrada podem ser diretamente associadas entre si (pois o número máximo de pais por variável é dado por v_u). A busca recursiva por comunidades menores que $v_u + 1$ foca nas variáveis mais fortemente correlacionadas que devem ser associadas.

CAPÍTULO

5

Problemas para validação de EDAs

Além de problemas do mundo real, várias funções são conhecidas e usadas para testar e comparar algoritmos de otimização. Este capítulo reúne problemas que têm sido utilizados para o teste de EAs, em especial, para EDAs. O estudo desses problemas é de interesse para que sejam adequadamente elaborados os experimentos para avaliação das competências e limitações de cada um dos algoritmos propostos neste projeto.

Conforme apresentado na Seção 2.2, vários problemas complexos podem ser decompostos (perfeitamente ou de forma aproximada) em subproblemas. Além da dificuldade de se resolver cada subproblema, é necessário primeiramente encontrar as subestruturas (BBv's) que os delimitam, além de considerar relações entre diferentes BBv's, nos casos em que a decomposição perfeita não é possível. O fato é que as subestruturas de um problema e as formas como essas se relacionam afetam diretamente a complexidade computacional do problema. Dessa forma, entender as propriedades que geram a complexidade dos problemas é fundamental para o desenvolvimento de algoritmos competentes, isto é, que os resolvam de forma eficiente e com robustez (Goldberg, 2002). Nesse sentido, este capítulo apresenta um conjunto de problemas que têm sido utilizados na literatura para validar EDAs, uma vez que tais problemas possuem as principais características envolvendo subestruturas que os tornam complexos.

A Seção 5.1 apresenta os problemas UmMax e BinInt e explica o fenômeno de convergência à deriva; a Seção 5.2 explica funções armadilha; a Seção 5.3 apresenta os problemas multi-

modais de interesse para validação de EDAs; a Seção 5.4 explica os problemas aditivamente separáveis; por fim, a Seção 5.5 apresenta os problemas hierárquicos.

5.1 UmMax, BinInt e o problema de convergência à deriva

O problema UmMax é um ponto comum de partida para se testar EAs. Consiste em maximizar o número de uns em uma *string* binária de tamanho ℓ, conforme representa a Equação 5.1:

$$f_{UM}(X) = \sum_{i=0}^{\ell-1} x_i. \tag{5.1}$$

Assim como o UmMax, o problema BinInt consiste em maximizar o número de uns em uma *string* binária, no entanto, cada *bit* é ponderado conforme mostra a Equação 5.2.

$$f_{BI}(X) = \sum_{i=0}^{\ell-1} x_i 2^i. \tag{5.2}$$

Ambas as funções são utilizadas para se testar a eficiência dos algoritmos ao se aumentar a escala do problema (número de *bits* da *string*) (Goldberg, 2002). Enquanto o UmMax é mais fácil de ser tratado ao se aumentar a escala do problema, o BinInt apresenta uma dificuldade maior, uma vez que *bits* mais significativos no cálculo da função de *fitness* direcionam a seleção das melhores soluções. Por esse motivo, quando os parâmetros de entrada do algoritmo não são adequados, *bits* menos significativos frequentemente convergem prematuramente para valores aleatórios. Para essa convergência prematura, dá-se o nome de convergência à deriva (*drift*, em inglês).

5.2 Funções armadilha

Para verificar o desempenho de diversos EDAs, em geral usam-se funções chamadas de armadilha, cujo comportamento faz com que algoritmos de busca locais geralmente encontrem ótimos locais diferentes do ótimo global (Harik *et al.*, 2006; Pelikan *et al.*, 1999; Vargas e Delbem, 2009). Um exemplo relativamente simples de uma função armadilha é a $ftrap5$, apresentada na Seção 2.4. Funções mais complexas são formadas a partir da composição dessas funções, como descrito pela Equação 5.3:

$$f(X) = \sum_{i=0}^{m} f_i(u_i), \tag{5.3}$$

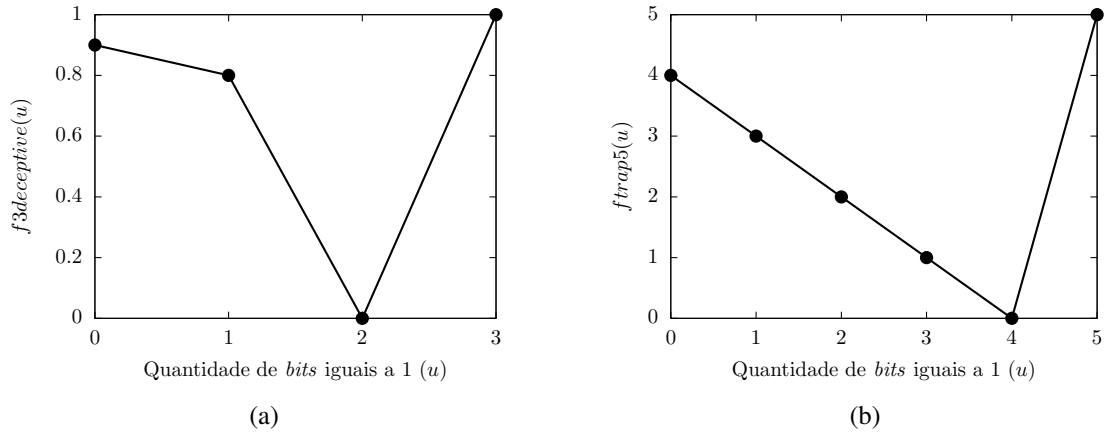
em que:

- X é uma *strings* binária de entrada;
- m é a quantidade de BBs que compõem o problema;
- u_i é a quantidade de *bits* com o valor um que se encontram no BB_i;
- BB_i é um BB do problema;
- f_i é a função a ser aplicada sobre o BB_i;

Problemas complexos podem ser construídos dessa forma, com ou sem sobreposição de BBs. Somente há sobreposição de BBs quando existe pelo menos uma variável em comum em diferentes BBs. Em geral, as funções (representadas por f_i na Equação 5.3) utilizadas são funções armadilha. São exemplos de funções armadilha utilizadas, a $f3deceptive$, representada pela Equação 5.4 e a $ftrap5$, Equação 5.5. As duas funções encontram-se ilustradas pela Figura 5.1.

$$f3deceptive(u) = \begin{cases} 0,9 & se\ u = 0 \\ 0,8 & se\ u = 1 \\ 0 & se\ u = 2 \\ 1 & se\ u = 3. \end{cases} \tag{5.4}$$

$$ftrap5(u) = \begin{cases} 4 - u & se\ u < 5 \\ 5 & caso\ contrário; \end{cases} \tag{5.5}$$



(a) (b)

Figura 5.1: Exemplos de funções armadilhas: (a) $f3deceptive$ à esquerda e (b) $ftrap5$ à direita.

Em funções armadilha, metaheurísticas que utilizam informações locais para orientar a busca tendem a concentrar a população em ótimos locais diferentes do ótimo global. Por esse motivo, GAs convencionais não apresentam desempenho satisfatório ao tratar tais funções (Harik *et al.*, 2006). A Figura 2.5 (ver Seção 2.4) mostra como o tamanho dos BBs aumenta a complexidade desses problemas. Variando o número de BBs (m) da Equação 5.3 e o tamanho k dos mesmos, pode-se gerar problemas com variado grau de complexidade e, com isso, comparar EDAs para problemas de diferentes dificuldades. Generalizando, é possível construir funções armadilha para BBs de tamanho k, como a representada pela Equação 5.6:

$$ftrapk(u) = \begin{cases} (k-1) - u & se\ u < k, \\ k & caso\ contrário. \end{cases} \tag{5.6}$$

Por outro lado, nem todas as funções formadas a partir da Equação 5.6 garantem a decepção para algoritmo que utilizam informações locais. Para se verificar isso, é importante esclarecer a diferença entre funções armadilhas e funções deceptivas (Goldberg, 2002). Uma função armadilha é deceptiva quando ela engana ou, gera informações confusas, ao comparar as pontuações médias dos esquemas menores que o comprimento da *string* (ℓ). Como exemplo, considerando a função $f3deceptive$, a pontuação média do esquema $00*$ é $0,85$, obtida ao calcular a média das pontuações das *strings* 000 e 001, que são respectivamente $0,9$ e $0,8$. Dessa forma, o esquema $00*$ possui uma pontuação média maior que a do esquema $11*$ (dada por $0,5$), o que direciona as soluções para longe do ótimo global 111. Analogamente, a função $ftrap2$ (Figura 5.2, criada por meio da Equação 5.6) não apresenta o mesmo comportamento, uma vez que a pontuação média do esquema $1*$ é maior que a do esquema $0*$[1] e, assim, a informação não é enganosa. Dessa forma, $ftrap2$ não é considerada uma função deceptiva. Para garantir que uma função armadilha seja deceptiva, a inequação 5.7 deve ser satisfeita. Esta inequação foi retirada de (Goldberg, 2002), e utiliza parâmetros que podem ser melhor compreendidos ao observar a Figura 5.3, adaptada do mesmo livro.

$$r \geq \frac{2 - (\ell - z)^{-1}}{2 - z^{-1}}, \tag{5.7}$$

em que:

- r é a taxa a/b (Figura 5.3), dada pela pontuação do ótimo local sobre a pontuação do ótimo global;
- ℓ é o tamanho da *string* de entrada;

[1] A pontuação do esquema $1*$ é 1, dada pela média das pontuações das soluções 11 (2) e 10 (0). Analogamente, a pontuação do esquema $0*$ é $0,5$, dada pela média das pontuações das soluções 01 (0) e 00 (1).

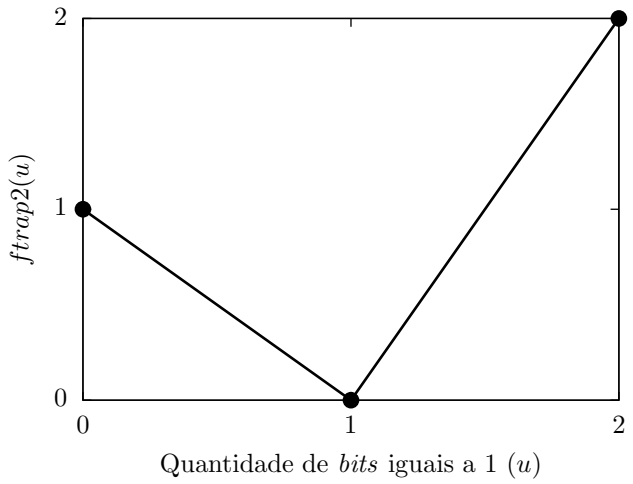


Figura 5.2: Função armadilha $ftrap2$.

- z é o "ponto de quebra" da função armadilha, em que $f(z) = 0$ (Figura 5.3).

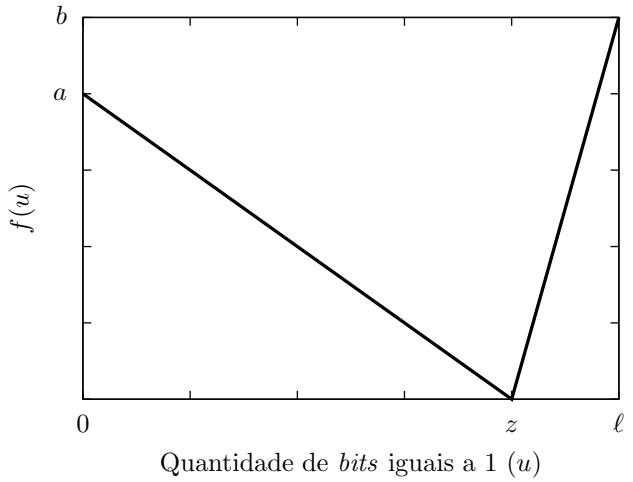


Figura 5.3: Modelo de função armadilha apresentando os parâmetros utilizados para verificar se a função é deceptiva (Equação 5.7).

5.3 Problemas Multimodais

Problemas multimodais apresentam uma grande quantidade de ótimos locais. Um desafio comum envolvendo tais problemas é encontrar não uma solução ótima, mas um conjunto de soluções ótimas, uma vez que esses problemas podem possuir mais de um ótimo global. Estudos de técnicas para resolver esse tipo de problema têm sido realizados (Kronfeld e Zell, 2010;

Li e Deb, 2010; Qu e Suganthan, 2010). Diversos problemas reais de engenharia podem ser classificados como problemas multimodais. Nesses casos é de interesse encontrar diversas soluções ótimas, para depois escolher qual solução será usada, baseando-se em limitações existentes ou outro aspecto prático. Exemplos comuns de problemas reais multimodais são problemas de classificação em aprendizado de máquinas (Mahfoud, 1995).

Algoritmos existentes, voltados a problemas multimodais, normalmente utilizam métodos de *niching* (Mahfoud, 1995) para guardar múltiplas soluções ótimas e evitar que o algoritmo convirja para um ótimo local. O método RTR utilizado no hBOA (Seção 2.7) é um exemplo de uma técnica de *niching*.

Para ilustrar uma função multimodal, considere a função $f6bipolar$ apresentada na Equação 5.8, baseada na Equação 5.4 e ilustrada pela Figura 5.4. Essa função é uma armadilha que possui dois ótimos globais. Ao combinar funções como essas, obtém-se problemas com diversos valores ótimos.

$$f6bipolar = f3deceptive(|3 - u|). \tag{5.8}$$

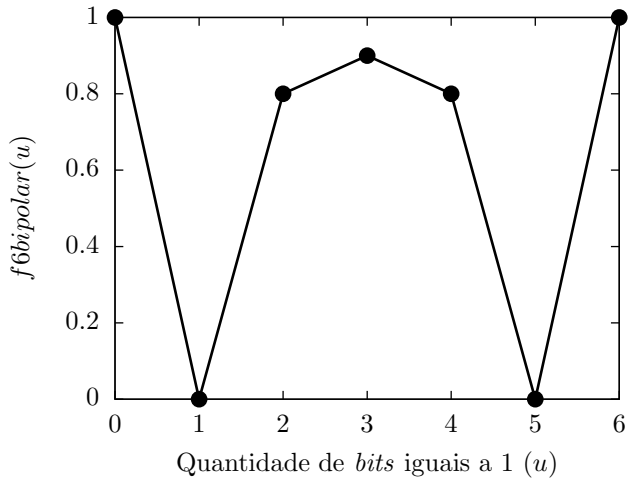


Figura 5.4: Função $f6bipolar$.

São apresentados em (Dick, 2010) outros problemas multimodais, como o Problema da Distância de Hamming Mínima (MINHD, do inglês *Minimum Hamming Distance Problem*), que possui 27 valores ótimos, e 2 170 valores ótimos deceptivos (armadilhas) e o Problema Deceptivo Multimodal Massivo (MMDP, do inglês *Massively Multimodal Deceptive Problem*) que possui 32 ótimos desejados e 5 milhões de ótimos deceptivos.

5.4 Problemas deceptivos aditivamente separáveis

Como visto na Seção 2.2, uma solução para um problema pode ser vista como um conjunto de subestruturas. Considere um problema composto por seis variáveis, cuja função de *fitness* é dada pela soma de duas funções $f3deceptive$ (Equação 5.4), representada pela Equação 5.9.

$$f(x_1, x_2, x_3, x_4, x_5, x6) = f'(x_1, x_2, x_3) + f'(x_4, x_5, x_6). \tag{5.9}$$

em que f é a função de *fitness* e $f'(x_1, x_2, x_3) = f3deceptive(x_1 + x_2 + x_3)$.

Este é um exemplo de um problema composto por funções deceptivas, sem sobreposição de BBs. De forma mais geral, problemas com essas características são definidos por:

$$f(X) = \sum_{i=0}^{m} f_i(V_i), \quad V_a \cap V_b = \{\}, \forall a \neq b. \tag{5.10}$$

em que:

- X é uma *string* binária de entrada;
- m é a quantidade de BBs ou número de partições do problema;
- f_i é a função deceptiva a ser aplicada sobre o conjunto de variáveis V_i;
- V_i é um subconjunto de variáveis de X que compõem o argumento de f_i.

Em (Goldberg, 2002), estes problemas são chamados de deceptivos aditivamente separáveis (ASDP, do inglês *Additively Separable Deceptive Problems*). Tais problemas são casos particulares do modelo *NK-landscapes* (Choi *et al.*, 2008), usados para representar problemas do mundo real. Apesar de ASDPs não representarem os problemas mais complexos de *NK-landscapes*, ASDPs representam problemas computacionalmente difíceis (Goldberg, 2002) e são frequentemente utilizados para se testar a eficiência de EDAs (Harik *et al.*, 2006; Melo *et al.*, 2011; Pelikan *et al.*, 1999). Um problema possui sobreposição de BBs, se não for satisfeita a condição $V_a \cap V_b = \{\}, \forall a \neq b$.

Caso as partições de um ASDP sejam conhecidas, uma busca exaustiva pode encontrar a melhor solução em um número de avaliações dado por $\sum_{i=0}^{m} 2_i^k$, em que k_i é o número de variáveis em V_i. Supondo k não grande, tal busca é computacionalmente viável para se determinar o ótimo global de um ASDP. No entanto, como a relação entre as variáveis geralmente não é conhecida a priori no contexto de EDAs e de otimização de caixa preta (He *et al.*, 2007), é necessário primeiramente encontrar as partições do problema. Isso pode ser realizado por meio da construção de modelos probabilísticos que identifiquem variáveis correlacionadas.

5.5 Sobreposição de BBs e problemas hierárquicos

Os chamados problemas hierárquicos podem ser compreendidos como uma mistura de problemas multimodais e problemas com sobreposição de BBs. Em (Goldberg, 2002), é proposta uma função que auxilia na compreensão de problemas hierárquicos, similar[2] à função dada pela Equação 5.11.

$$f(x_1...x_{72}) = f_1(x_1...x_6)+f_1(x_7...x_{12})+...+f_1(x_67...x_{72})+f_2(x'_1...x'_6)+f_2(x'_7...x'_{12}). \quad (5.11)$$

Na Equação 5.11, tem-se que $f_1(x_i, x_{i+1}, ...x_{i+5}) = f6bipolar(x_i + x_{i+1} + ... + x_{i+5})$ (Equação 5.8) e $f_2(x'_i, x'_{i+1}, ...x'_{i+5}) = ftrap6(x'_i+x'_{i+1}+...+x'_{i+5})$ (Equação 5.6). As variáveis hierárquicas representadas por x'_i são funções das variáveis originais x_i do problema. O valor de uma variável x'_i é dado pelo conjunto de variáveis $\{x_{6i-5}, x_{6i-4}, ..., x_{6i}\}$. Se cada variável do conjunto representado por x'_i possuir o valor 1, então $x'_i = 1$. Caso todas as variáveis do conjunto possuam o valor 0, então $x'_i = 0$. Para qualquer outra combinação de valores dessas variáveis, x'_i é indeterminado e o valor retornado por f_2 é 0. Por exemplo, se $x_1, x_2..., x_6 = 011000$, então x'_1 é indeterminado, e $f_2(x'_1, x'_2, ..., x'_6) = 0$. Se $x_1, x_2..., x_6 = 000000$, então $x'_1 = 0$, e o valor de $f_2(x'_1, x'_2, ..., x'_6)$ depende ainda das outras variáveis hierárquicas $x'_2, ..., x'_6$.

Analisando a Equação 5.11, pode-se perceber que a convergência das soluções tende a ocorrer por etapas. Inicialmente, a função f_1 direciona a convergência das soluções. Por exemplo, o conjunto de variáveis $x_1, x_2, ..., x_6$ pode convergir para 000000 ou 111111. Embora os BBs 000000**...* e 111111**...* contribuam igualmente para o *fitness* nas primeiras gerações, o que ocorre na prática é que apenas um desses BBs domina a população. Note que se 000000**...* dominar completamente a população, não será possível encontrar o ótimo global para o problema, pois em uma etapa posterior do EDA, a função f_2 favoreceria o BB 111111**...*, que não mais se encontra na população. Para evitar isso, técnicas de *niching* (Mahfoud, 1995) podem ser utilizadas para que diferentes soluções sejam guardadas na população, mantendo a diversidade de BBs até o fim da execução do algoritmo. Um exemplo é a técnica de *niching* RTR utilizada no hBOA (Seção 2.7).

Em (Pelikan e Goldberg, 2003), é mostrado que o algoritmo h-BOA é capaz de resolver dois problemas hierárquicos do mundo real, chamados *Ising Spin Glass* e *MAXSAT*, que são explicados a seguir. É importante observar que resolver problemas hierárquicos não é o foco dos algoritmos propostos neste trabalho, no entanto, os resultados relevantes encontrados ao utilizar

[2] As funções f_1 e f_2 utilizadas são variações das utilizadas em (Goldberg, 2002). No entanto, as características hierárquicas do problema apresentado são mantidas.

o CD-BOA (método proposto neste trabalho, Seção 4.1) para problemas com sobreposição de BBs (Seção 6.6) permitem considerar que as adaptações feitas ao BOA (Seção 4.1) podem ser também aplicadas ao hBOA, contribuindo assim também de forma significativa na resolução de problemas como os apresentados nesta seção.

Ising Spin Glass

Um problema de mundo real de interesse na área da física e engenharia de materiais, é encontrar o estado fundamental de um sistema *Ising* para vidros *spin* (do inglês: *Ising Spin Glass*)(Bolthausen e Bovier, 2007). O estado físico desse sistema, é dado por:

1. Um conjunto de valores representado por (σ_0, σ_1, ... σ_{n-1}), em que cada valor σ_i é chamado *spin*, que possui valor -1 ou 1;
2. Um conjunto de valores de acoplamento J_{ij} que relacionam pares de *spins*.

Para encontrar o estado fundamental do sistema, deve-se determinar o conjunto de *spins* que minimizam a energia $H(\sigma)$, calculada por meio da Equação 5.12.

$$H(\sigma) = -\sum_{i,j=0}^{n} J_{ij}\sigma_i\sigma_j. \tag{5.12}$$

A tarefa de encontrar o valor mínimo para $H(\sigma)$ pertence à classe de problemas NP-Completo (Garey e Johnson, 1979; Sipser, 2007). Assim, não existe um algoritmo conhecido que possa resolvê-lo em tempo polinomial. Por outro lado, o hBOA é capaz de resolver exatamente casos deste problema em tempo polinomial (Pelikan e Goldberg, 2003). Tais casos restritos limitam o número de interações entre os *spins*, dispondo-os em uma matriz bidimensional e permitindo apenas interações entre os vizinhos mais próximos. Além disso, são limitados os possíveis valores de acoplamento. Dessa forma, avanços no desenvolvimento de EDAs podem possibilitar avanços significativos na obtenção de soluções para esse problema desafiador.

MAXSAT

Outro problema do mundo real, que pode ser tratado por EDAs, é o de encontrar predicados que satisfaçam o maior número possível de cláusulas em uma expressão lógica na forma normal conjuntiva (CNF, do inglês *Conjunctive Normal Form*) (Sipser, 2007). Tal problema é conhecido como MAXSAT (do inglês: *Maximum Satisfiability Problem*), sendo de especial interesse para as áreas de Teoria da Complexidade e Inteligência Artificial (Pelikan e Goldberg, 2003).

Para ilustrar esse problema, considere a expressão lógica: $(X_1 \vee X_2) \wedge (X_2 \vee \neg X_3) \wedge (\neg X_3 \vee X_1)$, em que X_i's são variáveis binárias e $\vee$, $\wedge$ e $\neg$ são operações de conjunção, disjunção e negação, respectivamente. Encontrar os valores que satisfaçam essa expressão implica em determinar os valores de X_1, X_2 e $X_3 \in \{0, 1\}$ para os quais a expressão resulta em 1. Como exemplo, a *string* binária 101, indica uma solução candidata (X_1=1, X_2=0 e X_3=1). A função de avaliação de uma solução é dada pela quantidade de cláusulas que são satisfeitas. Uma forma de satisfazer todas as cláusulas dessa expressão é atribuindo aos predicados X_i, os valores: X_1=1, X_2=1 e X_3=0, indicando como solução ótima a *string* 110.

Alguns autores apontam MAXSAT como exemplo de casos em que EAs convencionais não são uma solução eficiente e classificam problemas MAXSAT não triviais como sendo problemas armadilha (Rana e Whitley, 1998). No entanto, alguns problemas desse tipo já foram resolvidos de forma eficiente por um EDA, mais especificamente, o hBOA (Pelikan e Goldberg, 2003), que foi testado em dois tipos de problema: (1) expressões na CNF possíveis de serem satisfeitas, geradas aleatoriamente, e (2) problema de coloração de grafos (Galinier e Hao, 1999; Garey e Johnson, 1979), convertidos em MAXSAT. Ambos os problemas foram retirados de (Hoos e Sttzle, 2000) [3]. Esses resultados reforçam a importância do estudo e desenvolvimento de EDAs mais eficientes, buscando resolver MAXSAT para instâncias maiores, por exemplo.

[3]Disponíveis em `http://www.satlib.org/`. Último acesso: 03/02/2011

CAPÍTULO

6

Análise de Desempenho

Os métodos CD-BOA, StrOp e REDA (Capítulo 4) são avaliados por meio das análises descritas neste capítulo. A descrição dos experimentos encontra-se organizada da seguinte maneira: a Seção 6.1 apresenta critérios, códigos e equipamentos utilizados nos experimentos; a Seção 6.2 analisa a eficiência desses algoritmos em termos do número de avaliações; a Seção 6.3 avalia a robustez dos algoritmos com relação à variação de parâmetros de entrada; a Seção 6.4 analisa a eficiência dessas técnicas em termos de tempo de computação; a Seção 6.5, verifica a contribuição da técnica REDA no aumento de desempenho; a Seção 6.6 mostra o desempenho dessas técnicas em cenários envolvendo BBs com sobreposição de *bits* e problemas não deceptivos. Por fim, a Seção 6.7 apresenta uma síntese e discussão dos principais resultados.

6.1 Critérios e recursos utilizados

A seguir, descrevem-se procedimentos experimentais comuns a todos os testes realizados:

- Em geral, os experimentos possuem parâmetros comuns representados pelas letras:

  ℓ: tamanho do problema;

  m: quantidade de BBs que compõem o problema sendo otimizado;

k: tamanho do BB;

n: tamanho da população. Em experimentos específicos, os valores de n utilizados são os propostos em (Pelikan *et al.*, 1999) e, nos demais experimentos o tamanho de n é definido pelo método da bissecção (Sastry, 2001);

v: parâmetro de entrada para o BOA que define o Número Máximo de vértices Pais (NMP) na BN criada (Seção 2.6);

v_l e v_u: são os parâmetro de entrada para o CD-BOA que definem, respectivamente, o número máximo de vértices pais na BN "crua" (isto é, gerada pelo algoritmo BGS Seção 2.6.2), e o número máximo de vértices pais após a estrutura de comunidades ser detectada na BN e ocorrer a adição de arestas (Seção 4.1).

- O critério de parada utilizado pelo BOA e o CD-BOA foi o de convergência, isto é, o algoritmo para quando cada variável possui o mesmo valor em pelo menos 99% das soluções que compõem a população;

- O critério de seleção utilizado por todos os algoritmos foi o de truncamento em 50%, da mesma forma que é realizado no BOA. Ocorre exceção quando a REDA é utilizada. Nesse caso, aplica-se torneio de 8 indivíduos (ver Seção 4.3) para realizar cada nova amostra (reamostragem). A escolha do tamanho do torneio encontra-se justificada no Apêndice E;

- Usam-se problemas de *benchmark* que são descritos por uma combinação de funções núcleo (funções armadilha e deceptivas). Tais funções são explicadas nas Seções 5.2 e 5.3, e encontram-se representadas pelas Equações 6.1, 6.2, 6.3 e 6.4, em que u (argumento) é a quantidade de *bits* 1 na *string* que define uma possível solução para o problema:

$$ftrap5(u) = \begin{cases} 4-u & se\ u < 5 \\ 5 & caso\ contrário; \end{cases} \tag{6.1}$$

$$f3deceptive(u) = \begin{cases} 0,9 & se\ u = 0 \\ 0,8 & se\ u = 1 \\ 0 & se\ u = 2 \\ 1 & se\ u = 3; \end{cases} \tag{6.2}$$

$$f6bipolar = f3deceptive(|3-u|); \tag{6.3}$$

$$ftrapk(u) = \begin{cases} (k-1) - u & se\ u < k, \\ k & caso\ contrário. \end{cases} \tag{6.4}$$

- Cada ponto representado nas figuras deste capítulo é obtido calculando a média de 30 repetições do experimento correspondente (30 execuções do mesmo algoritmo usando sementes do gerador de números aleatórios diferentes em cada execução) realizadas para aumentar a confiança estatística nos resultados. Também são exibidos nos gráficos os intervalos de confiança de 95% para as médias amostrais;
- Os critérios usados para se avaliar a eficiência computacional são os seguintes:
  1. Taxa de Blocos Ótimos (TBO), dada pelo número de BBs para os quais foram encontradas as instâncias ótimas, dividido pelo total de BBs do problema;
  2. Número de Avaliações (NA) até que o critério de convergência tenha sido atingido; esse tem sido o valor mais utilizado para se medir a eficiência de um EA;
  3. Tempo de Execução (TE) do algoritmo. Embora esse critério não tenha sido tão utilizado na literatura quanto o NA para verificar a eficiência de EAs, o mesmo tornou-se mais relevante com o desenvolvimento dos EDAs, dado que a construção de modelos probabilísticos torna cada geração de um EDA significativamente mais demorada.

Os códigos desenvolvidos e utilizados para a execução dos experimentos encontram-se disponibilizados em (Crocomo, 2012) junto com a devida documentação sobre como utilizá-los. São eles:

- *Patch* CD-BOA: a ser aplicado ao algoritmo BOA (Pelikan, 1999) que permite a utilização do CD-BOA proposto neste trabalho;
- *Patch* StrOp: a partir da versão 2.0 do *patch* desenvolvido, é possível utilizar também o algoritmo StrOp;
- CDA aFA: proposto na Seção 4.4. Uma versão eficiente do aFA foi implementada e encontra-se disponibilizada em um arquivo separado, para poder ser utilizada em outras aplicações;
- REDA: é possível utilizar este método em conjunto com outros EDAs existentes, portanto, o REDA é disponibilizado em um arquivo separado do *patch*.

Todos os experimentos foram executados no *cluster* do LCR (ICMC-USP), em máquinas contendo as seguintes especificações: Processador Intel Core i7-2600, Clock 3.4GHz, Cache 8MB, com memória RAM de 8GB, 1333 MHz. É importante destacar que, embora diversos nós do *cluster* tenham sido utilizados para executar os experimentos, a implementação dos algoritmos não é paralela. Portanto, máquinas diferentes foram utilizadas para paralelizar a execução de testes distintos, e não a execução de um mesmo teste.

6.2 Eficiência por Número de Avaliações

Esta Seção está organizada em quatro experimentos que avaliam o desempenho do CD-BOA, StrOp e BOA em relação ao número de avaliações utilizado pelos algoritmos. O **Experimento 1** investiga a importância da técnica aFA na obtenção de uma melhor BN. Os **Experimentos 2 e 3** comparam os resultados do CD-BOA com o BOA utilizando o valor ideal para seu parâmetro v. Por fim, o Experimento 4 repete o experimento 3 para o StrOp, comparando a eficiência da técnica com o CD-BOA.

Experimento 1 - Verificação de que a técnica aFA melhora a representatividade de uma BN com diversas relações entre variáveis não representadas

O conjunto de variáveis que se procura otimizar é representado por uma *string* binária composta por m BBs de tamanho 5 (k=5) a serem aplicados em funções $ftrap5$ (Equação 6.1). Os testes foram realizados para cada tamanho de problema com ℓ variando segundo os valores 30, 60, 90, 120, 150 e 180. O tamanho da população utilizada tanto para o BOA quanto para o CD-BOA variou de $n = 1\,300$ (para $\ell = 30$) até $n = 11\,800$ (para $\ell = 180$), de acordo com os valores sugeridos em (Pelikan *et al.*, 1999). Os algoritmos foram testados utilizando os seguintes parâmetros:

1. $v = 2$ para o BOA;
2. $v_l = 2$ e $v_u = 4$ para o CD-BOA.

O valor ideal de v a ser utilizado para resolver problemas $ftrap5$ é 4, dado que uma variável encontra-se associada a outras 4 variáveis, formando BBs com $k = 5$. O parâmetro $v = 2$ foi escolhido por ser metade do valor ideal e, portanto, não permite a construção de uma BN que identifique todas as relações existentes em uma $ftrap5$. Dessa forma, busca-se verificar a capacidade da técnica de detecção de comunidades (aFA) do CD-BOA em identificar corretamente

os BBs a partir de uma BN que não possui todas as relações existentes entre as variáveis representadas, podendo dessa forma melhorar a BN ao inserir novas relações. A Figura 6.1 mostra a comparação entre o número de avaliações necessárias para convergir utilizando o BOA com $v = 2$ e utilizando o CD-BOA com $v_l = 2$ e $v_u = 4$.

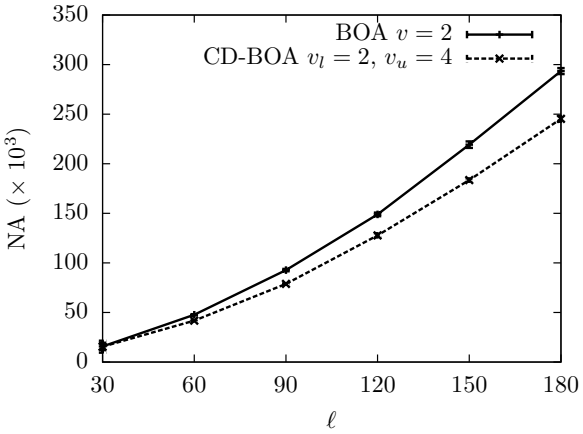


Figura 6.1: Comparação entre BOA com $v = 2$ e CD-BOA com $v_l = 2$ e $v_u = 4$

Os resultados favorecem o uso do CD-BOA, pois o NA foi sempre menor quando utilizando a nova técnica proposta. Observa-se também que a vantagem do CD-BOA aumenta com o tamanho do problema. Todas as TBOs médias foram acima de 97% para ambos os algoritmos. Esses resultados permitem constatar a capacidade da técnica aFA do CD-BOA em melhorar o desempenho do BOA quando constroem-se BNs que não representam todas as relações existentes entre as variáveis.

Experimento 2 - Comparação do CD-BOA com BOA utilizando v ideal

O **Experimento 1** foi estendido incluindo os testes com o BOA utilizando $v = 4$, que é o parâmetro mais adequado para se tratar problemas com $ftrap5$. A Figura 6.2 mostra a comparação dos resultados. É importante notar que, ao utilizar a técnica aFA, o parâmetro v utilizado para o CD-BOA continuou sendo 2 no **Experimento 2**. Embora a técnica de detecção de comunidades tenha sido capaz de melhorar a BN encontrada utilizando v_l=2 e v_u=4 (como mostrado pelo **Experimento 1**) as novas BNs obtidas não foram superiores às encontradas ao utilizar o parâmetro ideal (v=4) para o BOA.

Experimento 3 - Testes com $v_l = 2$ e $v_u = 10$

O efeito de se utilizar v_u maior que o v ideal para o BOA é analisado. Com isso, gera-se maior flexibilidade para o CD-BOA. Basicamente, repete-se o **Experimento 1**, desta vez utilizando

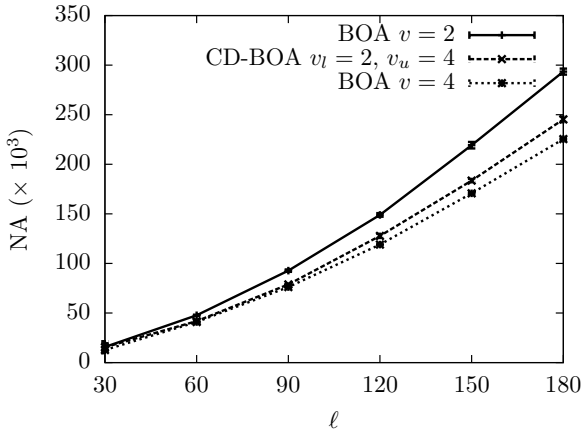


Figura 6.2: Comparação entre BOA com $v = 2$, BOA com $v = 4$, e CD-BOA com $v_l = 2$ e $v_u = 4$.

parâmetros $v_l = 2$ e $v_u = 10$ para o CD-BOA. A Figura 6.3 mostra resultados que indicam que valores de v_u significativamente maiores que o valor de v ideal podem aumentar a eficiência do CD-BOA, aproximando-se da eficiência do BOA com $v = 4$. Observe que o valor ideal do NMP, que é 4, não foi diretamente utilizado nos parâmetros de entrada do CD-BOA. No entanto, sua eficiência foi similar a do BOA usando tal valor.

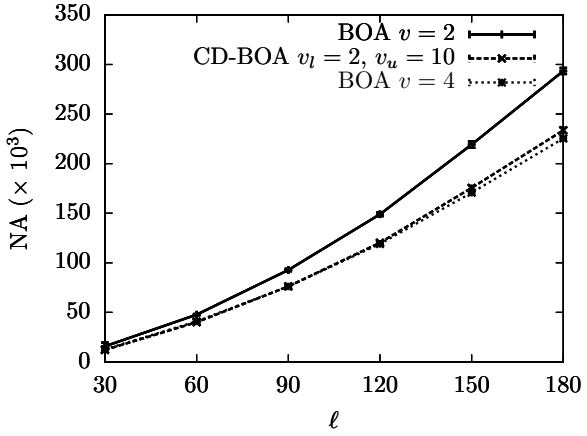


Figura 6.3: BOA ($v = 2$ e $v = 4$) e CD-BOA ($v_l = 2$ e $v_u = 10$).

Experimento 4 - Comparação do CD-BOA e StrOp com $v_l = 2$ e $v_u = 10$

De forma similar ao **Experimento 1**, foi avaliado o algoritmo StrOp (Seção 4.2). Como o tamanho da população utilizada por este algoritmo deve ser maior que o utilizado pelo CD-BOA (Seção 4.2.2), foi utilizada uma população n, para cada caso, 10 vezes maior que a utilizada para o CD-BOA nos **Experimentos 1, 2, 3 e 4**. Os parâmetros utilizados tanto para o CD-BOA

quanto para o StrOp foram $v_l = 2$ e $v_u = 10$. A Figura 6.4 sintetiza os resultados. O StrOp apresenta um crescimento assintótico do NA significativamente menor para resolver o problema de maneira eficaz (TBO superior a 96% em todos os casos). O crescimento do NA aparentemente linear do StrOp, deve-se ao fato do NA esperado do StrOp poder ser dado pelo tamanho da população (que é avaliada uma única vez) mais as avaliações realizadas pela busca gulosa (Seção 4.2). Este cálculo é representado pela equação abaixo 6.5:

$$NA_{ideal} = n + \frac{\ell}{k} 2^k. \tag{6.5}$$

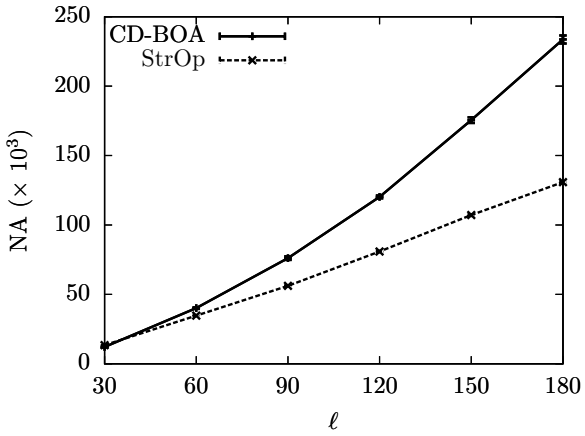


Figura 6.4: Comparação entre CD-BOA e StrOp (ambos com $v_l = 2$ e $v_u = 10$) para o problema composto por funções $ftrap5$.

Com base na Equação 6.5, é possível calcular o número máximo de avaliações que será realizada pelo StrOp em função do parâmetro v_u utilizado. Para isso, considera-se o pior caso: quando as comunidades identificadas pelo aFA forem todas do tamanho máximo permitido: $v_u + 1$ (admitindo erros quando $k < v_u + 1$). Dessa forma, um limite superior para o NA do StrOp é dado pela Equação 6.6:

$$NA_{máximo} = n + \frac{\ell}{v_u + 1} 2^{v_u + 1}. \tag{6.6}$$

A Figura 6.5 mostra os NAs obtidos experimentalmente pelo StrOp, comparado com os valores ideais, calculados seguindo a Equação 6.5 e com o limite superior para o NA de acordo com a Equação 6.6. Os resultados reforçam a validade das equações utilizadas.

A capacidade de calcular o número máximo de avaliações a serem realizadas pelo algoritmo é uma importante característica em problemas do mundo real, em que em geral existe custo significativo a ser considerado para a avaliação de uma solução. A partir da Equação 6.6, é

possível escolher um valor para o parâmetro v_u que não extrapole os recursos disponíveis (como custo financeiro ou tempo disponível).

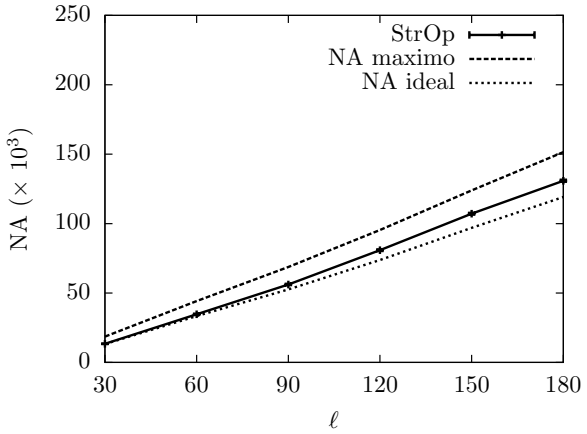


Figura 6.5: Comparação entre os NAs ideais, máximos e obtidos experimentalmente pelo StrOp.

6.3 Robustez relativa aos parâmetros de entrada

Como descrito na Seção 6.2, o BOA necessita de um parâmetro de entrada v que estabelece um número máximo de variáveis pais por variável. No BOA (Pelikan *et al.*, 1999), esse parâmetro foi estabelecido como o tamanho do BB menos um (k-1) em todos os testes realizados. Como indicado pelo **Experimento 2**, este é o valor ideal para o BOA, pois o número de variáveis relacionadas nos problemas de *benchmark* utilizados é sempre o tamanho do BB da função sendo utilizada. No entanto, em problemas do mundo real, essa informação não é normalmente conhecida a priori (são problemas do tipo caixa preta). Além disso,em um mesmo problema podem ocorrer BBs de tamanhos diferentes. Os experimentos desta seção buscam explorar esses aspectos ao variar o parâmetro de entrada v do BOA, verificando o nível de redução de desempenho acarretada pela utilização de valores diferentes do ideal. Nesses experimentos, o NMP indicado nas Figuras refere-se ao parâmetro de entrada v do BOA, e ao parâmetro v_u do CD-BOA.

Experimento 1 - Desempenho do CD-BOA e BOA com imprecisão nos parâmetros de entrada

Tanto o BOA quanto o CD-BOA são testados na resolução de um problema composto por funções $ftrap5$. O parâmetro v é variado de 2 até 10 para o BOA, e fixando v_l em 2 e variando

v_u de 2 a 10 para o CD-BOA. Verificam-se os valores médios do NA e da TBO. O intervalo de valores escolhidos para v neste experimento permite verificar a diminuição de desempenho quando v é maior ou menor que seu valor ideal (4).

Para a realização do **Experimento 1** desta Seção, foi utilizado ℓ=90 e $n = 5\,200$ (como sugerido nos testes exemplos que acompanham o BOA, disponibilizado em (Pelikan, 1999)). O gráfico da Figura 6.6 mostra os resultados, no qual o NA pode ser observado para cada parâmetro usado. Nota-se que os resultados são os esperados para o BOA, uma vez que a eficiência do mesmo atinge seu máximo para o parâmetro ideal (v=4) e piora conforme o parâmetro se distancia do valor ideal. Todos os pontos exibidos na Figura 6.6 possuem TBO médio superior a 97%. No entanto, para o BOA com $v = 9$, tem-se NA$\approx 420\,000$ com TBO$\approx$90%, além disso este valor decai para TBO$\approx$47% com NA$\approx 470\,000$ para o BOA com $v = 10$. O grande intervalo de confiança para o BOA a partir de $v = 8$ mostra uma grande variação entre os NAs requeridos em cada teste realizado, indicando uma instabilidade do algoritmo ao se utilizar valores para v relativamente maiores que o ideal.

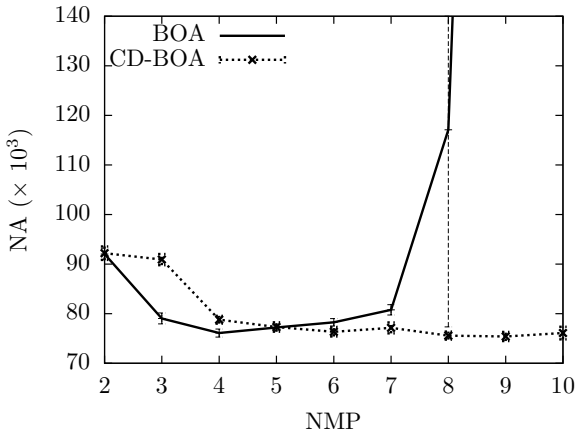


Figura 6.6: Desempenho do BOA e do CD-BOA usando NMP entre 2 e 10 para otimizar uma função composta por $ftrap5$ ($\ell = 90$, $m = 18$).

Por outro lado, os resultados do CD-BOA são em geral melhores, pois o NA apresenta-se mais estável do que no BOA. Conforme os valores de v_u aumentam, o NA diminui, revelando inclusive que o valor ideal para v_u não é 4. Isso deve-se ao fato do tamanho das comunidades (detectadas automaticamente pelo aFA) restringir o número total de arestas que podem ser adicionadas pelo CD-BOA. O aFA estima o tamanho dos BBs (tamanho das comunidades) em tempo de execução do CD-BOA, portanto, um maior valor para v_u concede maior flexibilidade ao CD-BOA (pode encontrar maior variedade de tamanhos de comunidades). Tal característica reflete um importante resultado, uma vez que EDAs são utilizados em geral para problemas do tipo caixa preta.

Experimento 2 - Testes com problemas compostos por funções $f3deceptive$

O **Experimento 1** é estendido verificando se a robustez do CD-BOA e do BOA mantém-se para um problema diferente do $ftrap5$. No caso, utilizou-se o $f3deceptive$ (Equação 6.2). Para tais testes, foi mantido o tamanho $\ell = 90$ e utilizada a população $n = 3\,800$.

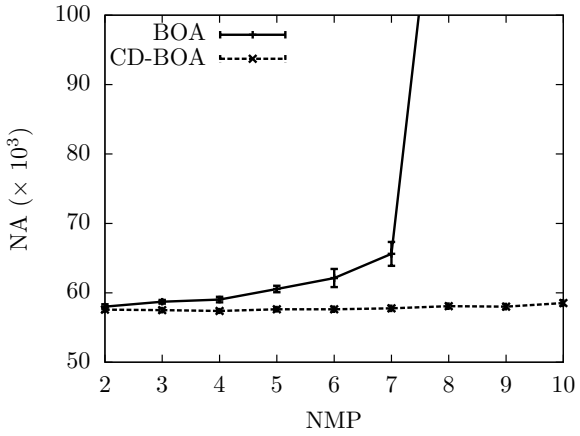


Figura 6.7: Desempenho do BOA e do CD-BOA usando NMP entre 2 e 10 para otimizar uma função composta por $f3deceptive$.

A Figura 6.7 sintetiza os resultados que são similares aos obtidos para o problema com $ftrap5$. A eficiência do BOA decai conforme o NMP aumenta, sem ao menos atingir valores aceitáveis de TBO para $v = 9$ (87%) e $v = 10$ (52%), enquanto o desempenho CD-BOA é mantido com a variação do NMP. Por outro lado, o CD-BOA mantém NAs com valores próximos ao melhor desempenho encontrado para toda a faixa de NMP investigada.

Experimento 3 - Testes com problemas compostos por funções $f3deceptive$ e $ftrap5$

Este experimento tem o objetivo de verificar se o mesmo comportamento dos experimentos anteriores é mantido para problemas compostos por BBs de diferentes tamanhos (em que não existe, à primeira vista, um NMP ideal). Vale destacar que não foram encontrados na literatura testes analisando o desempenho de EDAs para problemas contendo BBs de tamanhos diferentes. Basicamente, o **Experimento 2** foi modificado para um problema composto por 15 funções $f3deceptive$ e 9 $ftrap5$, resultando em um problema de tamanho $\ell = 90$ com $n = 5\,200$.

A Figura 6.8 mostra que os desempenhos do BOA e CD-BOA são similares aos obtidos nos **Experimentos 1 e 2**. Conforme esperado, o CD-BOA não requer a existência de um NMP ideal para resolver de forma eficiente o problema.

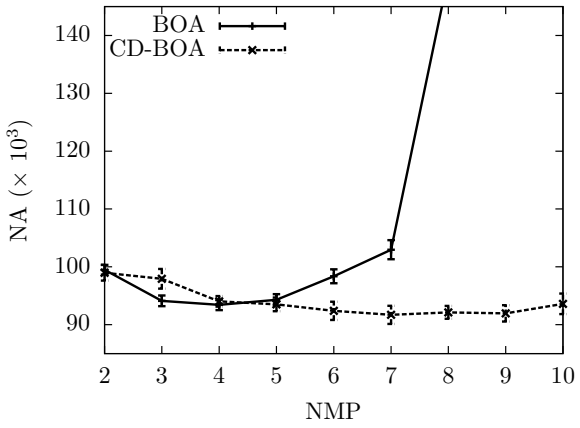


Figura 6.8: Desempenho do BOA e do CD-BOA usando NMP entre 2 e 10 envolvendo uma função composta por 15 $f3deceptive$ e 9 $ftrap5$.

Experimento 4 - Testes com problemas compostos por funções $f6bipolar$

Os Experimentos 2 e 3 foram alterados para considerar mais um cenário diferente: o problema composto por funções $f6bipolar$ (Equação 6.3). Novamente, o tamanho das populações utilizadas foram as sugeridas em (Pelikan *et al.*, 1999) para tais problemas. Os resultados apresentados na Figura 6.9 são diferentes dos apresentados anteriormente, pois a eficiência do BOA não parece diminuir ao aumentar o parâmetro v para os testes realizados. Além disso, mostram que o CD-BOA requer mais NA que o BOA. Isso ocorre, possivelmente, por BNs geradas com $v_l = 2$ não possuírem informação suficiente para que o algoritmo de detecção de comunidades possa identificar os grupos de variáveis relacionadas neste problema de maior complexidade que os testados anteriormente. Para verificar essa hipótese, os testes foram repetidos para todas as combinações de parâmetros v_l e v_u no intervalo de 2 a 10.

Os resultados da Figura 6.10 permitem verificar que a hipótese anterior é correta, pois ao aumentar o parâmetro v_l do CD-BOA, o mesmo resolve o problema de forma mais eficiente. Com $v_l = 4$ ou $v_l = 5$ o algoritmo atinge uma eficiência ainda maior que a obtida pelo BOA utilizando seu parâmetro ideal (5). Tal resultado é importante, pois até o momento, os experimentos constatavam apenas uma maior robustez do algoritmo com relação ao BOA ao utilizar parâmetros de entrada diferentes do ideal. É importante observar também que, neste experimento, o CD-BOA obteve uma eficiência maior, sem usar o NMP ideal (5). Os testes são realizados apenas para os parâmetros com NMP $\geq v_l$, para respeitar a regra de que $v_u \geq v_l$, uma vez que estes parâmetros definem um intervalo de valores.

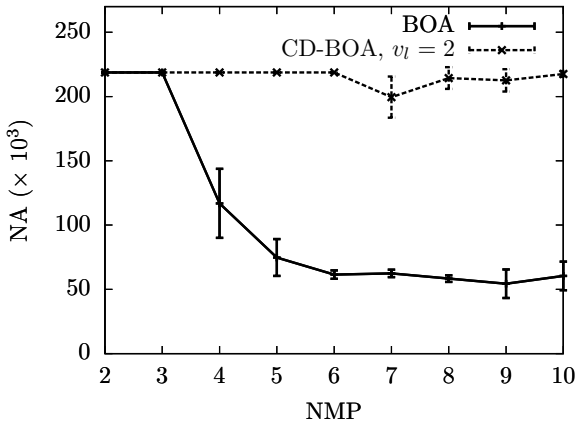


Figura 6.9: Desempenho do BOA e do CD-BOA com $v_l = 2$ usando diferentes valores de NMP para função composta por $fbipolar$ ($\ell = 90$, $m = 15$).

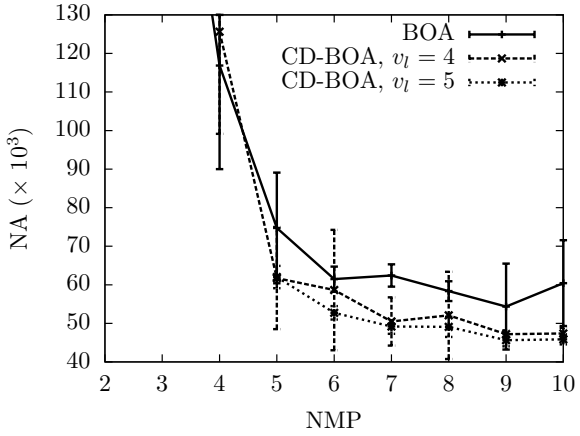


Figura 6.10: Desempenho do BOA e do CD-BOA com $v_l = 4$ e $v_l = 5$, usando diferentes valores de NMP para função composta por $fbipolar$ ($\ell = 90$, $m = 15$).

Experimento 5 - Testes com o StrOp

Os **Experimentos de 1 a 4** desta Seção foram refeitos de forma a incluir o algoritmo StrOp. O comportamento obtido em todos os testes é similar. A Figura 6.11 sintetiza os desempenhos do StrOp em relação ao BOA e CD-BOA. O NA do StrOp é inferior ao do BOA e CD-BOA em todos os casos testados. Observa-se que com o aumento de NMP, o NA obtido pelo StrOp mantém-se quase constante, apresentando pequenos aumentos mesmo diante ao fato de que maiores valores de v_u possibilitam a identificação de comunidades maiores (o NA máximo aumenta com v_u, NMP, devido à busca gulosa do StrOp, como indicado pela Equação 6.6). O fato de tais aumentos serem pequenos demonstra a capacidade do aFA em encontrar as comunidades corretas (do mesmo tamanho dos BBs que compõem o problema) mesmo para $v_u > k - 1$.

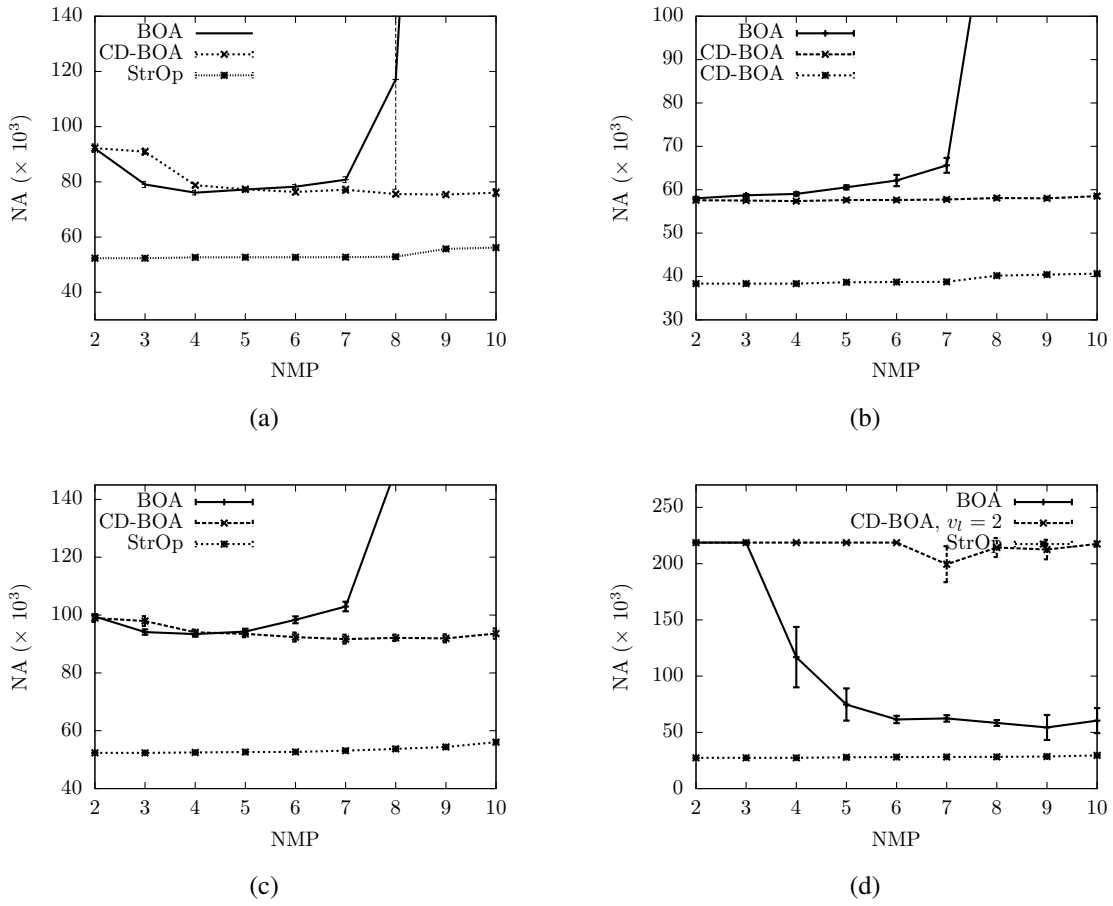


Figura 6.11: Desempenho do BOA, CD-BOA e StrOp usando NMP entre 2 e 10 para otimizar uma função composta por: (a) $ftrap5$ ($\ell = 90$, $m = 18$), (b) $f3deceptive$ ($\ell = 90$, $m = 30$), (c) 15 $f3deceptive$ e 9 $ftrap5$ ($\ell = 90$, $m = 24$) e (c) $fbipolar$ ($\ell = 90$, $m = 15$).

Embora o NA tenha se apresentado similar para todos os NMPs testados, os NMPs 2 e 3 não permitiram que o StrOp encontrasse boas soluções em termos de TBO para o problema, como mostra a Figura 6.12. Isso é devido ao fato de que ao utilizar $v_u < k - 1$, o StrOp não consegue identificar os BBs completamente, e sem a identificação correta dos BBs, a busca gulosa não garante a obtenção de BBs ótimos (normalmente convergindo para ótimos locais em problemas armadilha). No entanto, ao utilizar $v_u \geq k - 1$, o StrOp atinge altas taxas de acerto, como é esperado.

Por outro lado, a qualidade das soluções obtidas pelo CD-BOA não é afetada no caso em que $v_u < k-1$ (parâmetro define um intervalo que não contém o tamanho esperado do BB), obtendo altas TBOs, como mostra a Figura 6.13. As Figuras 6.12, 6.13 e 6.14 mostram que o CD-BOA apresenta-se como o algoritmo mais robusto com relação a variação dos parâmetros de entrada,

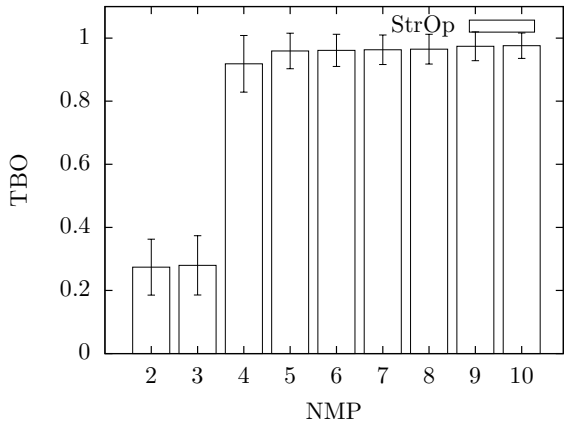


Figura 6.12: TBOs do StrOp usando NMP entre 2 e 10 para função composta por $ftrap5$ ($\ell = 90$, $m = 18$).

uma vez que obtém TBOs relativamente altas para todos os NMPs testados, enquanto o StrOp e o BOA apresentam problemas ao utilizar, respectivamente, baixos e altos valores de NMP.

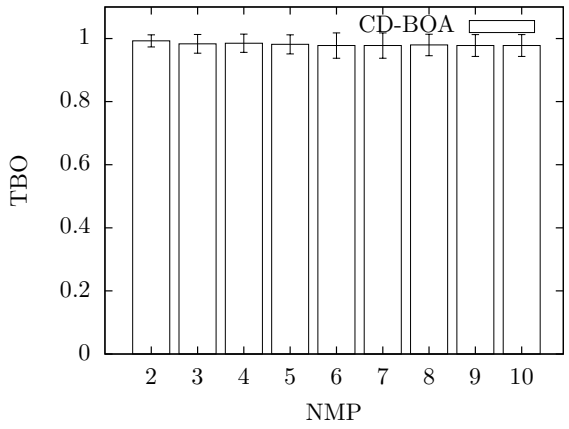


Figura 6.13: TBOs do CD-BOA usando NMP entre 2 e 10 para função composta por $ftrap5$ ($\ell = 90$, $m = 18$).

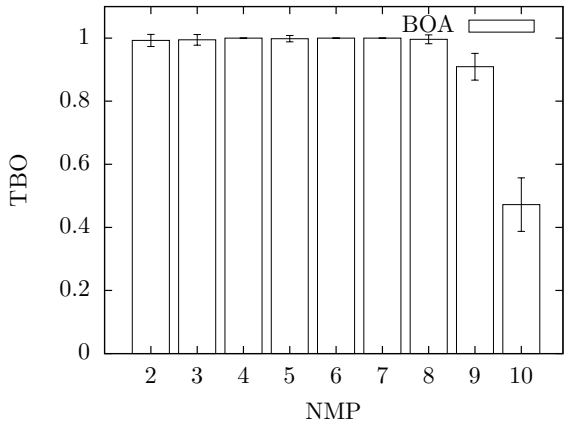


Figura 6.14: TBOs do BOA usando NMP entre 2 e 10 para função composta por $ftrap5$ ($\ell = 90$, $m = 18$).

6.4 Eficiência por Tempo de Execução

Dois experimentos são apresentados para avaliar experimentalmente o comportamento dos algoritmos desenvolvidos. O primeiro avalia o TE do CD-BOA e BOA, enquanto o segundo inclui o algoritmo StrOp.

Experimento 1 - Verificação experimental do comportamento assintótico do CD-BOA e BOA

Durante a realização dos experimentos descritos nas Seções 6.2 e 6.3, verificou-se que o CD-BOA, em geral, requeria menor tempo de execução (TE) do que o BOA. Para confirmar tal observação, os experimentos variando ℓ de 30 a 180 e NMP de 2 a 10 para problemas de $ftrap5$ foram repetidos para estimar adequadamente os TE desse algoritmo e do BOA. As Figuras 6.15 e 6.16 mostram que o tempo de computação do CD-BOA é assintoticamente menor que do BOA. A Figura 6.16 revela que o TE do BOA é superlinear com o aumento do NMP, contrastando com o TE do CD-BOA que se mostra limitado por uma constante.

Experimento 2 - Verificação experimental do comportamento assintótico do StrOp

Para analisar o TE do StrOp, o **Experimento 1** desta Seção foi repetido, utilizando o StrOp com os mesmos parâmetros do CD-BOA ($v_l = 2$ e $v_u = 10$), e população de mesmo tamanho que a utilizada no **Experimento 4** da Seção 6.2. Os resultados das Figuras 6.17 e 6.18 mostram que o TE do StrOp é assintoticamente o menor entre os algoritmos testados, chegando o StrOp a atingir para $\ell = 180$ (Figura 6.17) um tempo de execução aproximadamente 30 vezes menor

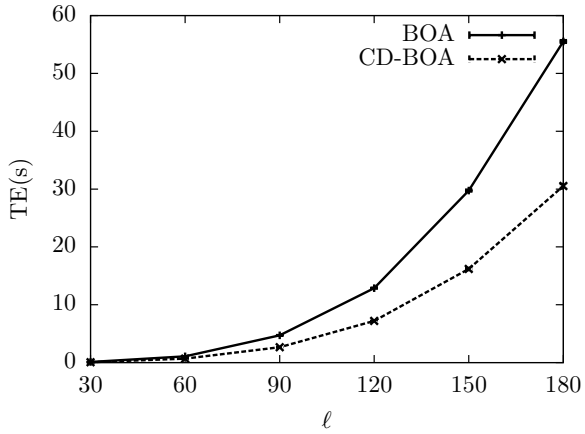


Figura 6.15: TE do BOA ($v = 4$) e do CD-BOA ($v_l = 2$, $v_u = 10$) ao variar ℓ utilizando funções $ftrap5$.

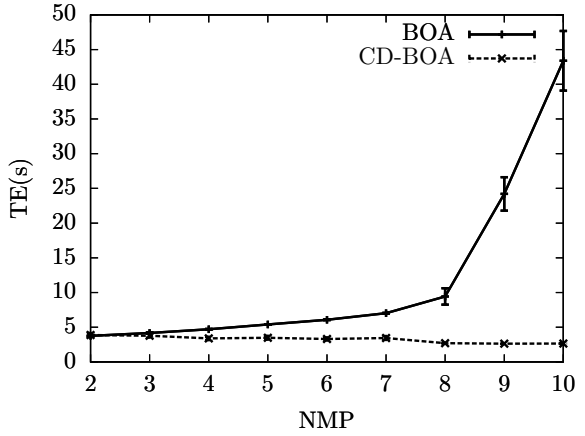


Figura 6.16: TE do BOA ($v = 4$) do CD-BOA ($v_l = 2$) ao variar NMP utilizando funções $ftrap5$.

que o atingido pelo BOA. A Figura 6.18 revela que o TE do StrOp, assim como o CD-BOA, é superlinear com o aumento do NMP.

Buscando melhor compreender os comportamentos sintetizados nas Figuras desta Seção, a complexidade de tempo da construção do modelo utilizado pelo CD-BOA, assim como a complexidade de tempo da construção do modelo e da busca gulosa utilizada no StrOp foram calculadas. As Seções 6.4.1 e 6.4.2 mostram, respectivamente, a determinação destas complexidades.

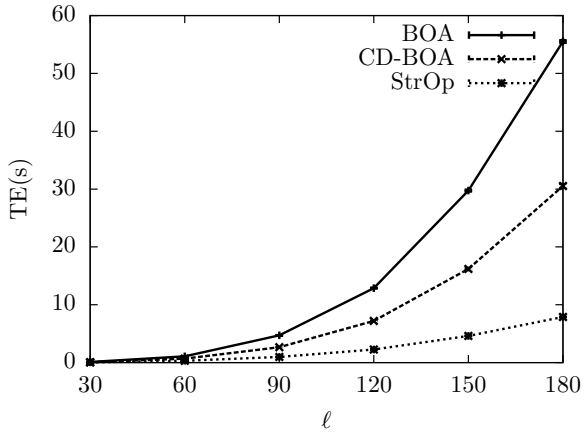


Figura 6.17: TE do BOA ($v = 4$), do CD-BOA ($v_l = 2$ e $v_u = 10$) e do StrOp ($v_l = 2$ e $v_u = 10$) ao variar ℓ utilizando funções $ftrap5$.

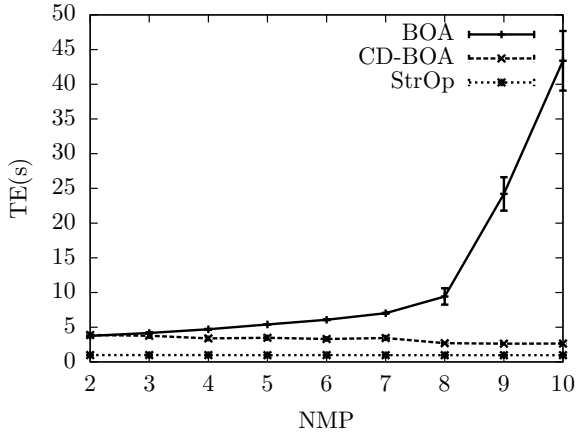


Figura 6.18: TE do BOA ($v = 4$), do CD-BOA ($v_l = 2$ e $v_u = 10$) e do StrOp ($v_l = 2$ e $v_u = 10$) ao variar NMP utilizando funções $ftrap5$.

6.4.1 Complexidade de tempo da construção da rede no CD-BOA

A complexidade de tempo da construção da rede no CD-BOA pode ser descrita pela seguinte expressão: $C_{CD-BOA} = C_{boa} + C_{aFA} + C_{ins}$. em que:

- C_{CD-BOA} é a complexidade de tempo da construção da rede no CD-BOA;
- C_{boa} é a complexidade de tempo da construção da rede no BOA (C_{BOA}) com o valor do parâmetro v_l no lugar de v;
- C_{aFA} é a complexidade de tempo do aFA;

- C_{ins} é a complexidade de tempo para inserção das novas arestas nas comunidades detectadas pelo FA;

C_{boa}:

Em (Pelikan *et al.*, 1999), determinou-se a complexidade da construção da rede pelo BOA (C_{BOA}), que é dada por:

$$C_{BOA} = O(v2^v\ell^2N + v\ell^3). \tag{6.7}$$

em que N é o tamanho da amostra utilizada, que na implementação do BOA proposta em (Pelikan *et al.*, 1999) é igual a metade do tamanho da população (seleção por truncamento). Como definido anteriormente, C_{boa} é C_{BOA} com $v = v_l$. Desta forma, o parâmetro C_{boa} é dado pela Equação 6.8.

$$C_{boa} = O(v_l2^{v_l}\ell^2N + v_l\ell^3). \tag{6.8}$$

C_{aFA}:

A complexidade do algoritmo de detecção de comunidades FA é dada por $C_{FA} = O((a + \ell)\ell)$ (Newman, 2004), em que a é o número de arestas na rede e ℓ, seu número de vértices. A BN de entrada para este algoritmo possui, no máximo, v_l vértices pais por vértice. Sendo assim, em uma BN, $a = [0, v_l\ell]$. Para o cálculo de complexidade considera-se o pior caso ($a = v_l\ell$). Portanto, $C_{aFA} = O((v_l\ell + \ell)\ell R) = O(v_l\ell^2 * R)$, em que R é o número de vezes que o algoritmo é chamado recursivamente (devido à segunda adaptação do algoritmo, descrita no Capítulo 4.4). No pior caso, $R = \ell - (v_u + 1)$, pois a maior comunidade que o aFA poderia retornar possui todos os vértices da rede (ℓ) e, a cada chamada recursiva, somente um vértice seria removido da comunidade até que a mesma atingisse o maior tamanho permitido ($v_u + 1$). Portanto, $C_{aFA} = O(v_l\ell^2(\ell - v_u + 1)) = O(v_l\ell^3 - v_lv_u\ell^2 + v_l\ell^2)$. Sabendo que $v_l < v_u$ e considerando $v_u << \ell$, a expressão anterior pode ser reduzida a $v_l\ell^3$. Dessa forma, a complexidade do aFA aplicado a BNs é dada pela Equação 6.9.

$$C_{aFA} = O(v_l\ell^3). \tag{6.9}$$

C_{ins}:

Para cada par ordenado de vértices dentro da mesma comunidade, verifica-se a possibilidade de inserção de uma aresta conectando-os e, em caso afirmativo, a aresta é inserida. No pior

caso, essa verificação ocorre para todos os pares de arestas da rede (uma única comunidade), resultando em $\ell^2 - \ell$ verificações. Para determinar se uma determinada aresta pode ser inserida, deve-se verificar: i) se o vértice de destino já atingiu o número máximo de vértices pai permitido (v_u) e ii) se a inserção da aresta não criaria um ciclo na rede (BNs são acíclicas). Essas duas verificações são realizadas em $O(1)$, dado que na estrutura da BN, um contador de vértices pais é mantido para cada vértice e uma matriz de caminhos (Apêndice F) é atualizada a cada inserção.

A atualização da matriz de caminhos após a inserção de uma aresta é feita em $O(\ell^2)$ (ver pseudocódigo desta atualização no Apêndice F). Como o número máximo de arestas a serem adicionadas nesta etapa é de v_u por vértice ($v_u\ell$), o tempo total gasto com a atualização da matriz de caminhos é $O(v_u\ell^3)$. Portanto, C_{ins} é dado pela Equação 6.11.

$$C_{ins} = O(\ell^2 - \ell + v_u\ell^3), \tag{6.10}$$

$$C_{ins} = O(v_u\ell^3). \tag{6.11}$$

Conhecendo agora os valores de C_{boa}, C_{FA} e C_{ins}, pode-se reescrever C_{CD-BOA}:

$$C_{CD-BOA} = C_{boa} + C_{aFA} + C_{ins} = O(v_l 2^{v_l}\ell^2 N + v_l\ell^3) + O(v_l\ell^3) + O(v_u\ell^3). \tag{6.12}$$

Analisando a Equação 6.12 acima, verifica-se que o CD-BOA, quando comparado ao BOA (ver Equação 6.7), aumenta o tempo de construção da BN de forma polinomial devido às etapas adicionais, que exigem tempo $O(v_l\ell^3)$ e $O(v_u\ell^3)$. No entanto, ao substituir o termo v que era utilizado no BOA pelo seu correspondente no CD-BOA, v_l, diminui-se exponencialmente o tempo de construção da rede, uma vez que $v_l < v < v_u$ e, portanto, $2^{v_l} < 2^v$. Simplificando a Equação 6.12, C_{CD-BOA} pode ser expressa por:

$$C_{CD-BOA} = O(v_l 2^{v_l}\ell^2 N + (v_l + v_u)\ell^3). \tag{6.13}$$

A expressão do C_{CD-BOA} explica o menor TE experimental do CD-BOA em relação ao BOA com o aumento de ℓ (Figura 6.16) sendo que v_u, que diferencia C_{CD-BOA} de C_{BOA} é, na prática, limitado por uma constante positiva, uma vez que problemas do mundo real possuem certa localidade nas correlações entre variáveis, isto é, as comunidades (v_u) em sua grande maioria são pequenas em relação à rede (ℓ). Adicionalmente, mostra-se no Apêndice G que é possível calcular um limitante superior para v_u, de forma a garantir que $C_{CD-BOA} < C_{BOA}$.

Mostra-se no mesmo apêndice que os limitantes calculados para v_u são muito maiores do que os valores de v_u utilizados nesta seção, e que se mostraram suficientes para resolver o problema de forma eficiente. Por fim, destaca-se que o C_{BOA} cresce exponencialmente com o NMP (v), enquanto o C_{CD-BOA} cresce linearmente com o NMP (v_u). Porém, esse crescimento é em geral desprezível em relação aos demais termos do C_{CD-BOA} explicando os TEs experimentais praticamente constantes em relação a NMP, conforme ilustra a Figura 6.16.

6.4.2 Complexidade de tempo da construção do modelo e busca gulosa do StrOp

A complexidade de tempo da construção do modelo e da busca gulosa do StrOp pode ser descrita pela seguinte expressão: $C_{StrOp} = C_{boa} + C_{aFA} + C_{gs}$.

em que:

- C_{StrOp} é a complexidade calculada;
- C_{boa} é a complexidade de tempo da construção da rede no BOA, utilizando o valor do parâmetro v_l no lugar de v;
- C_{aFA} é a complexidade de tempo do aFA;
- C_{gs} é a complexidade de tempo da busca gulosa utilizada;

Os termos C_{boa} e C_{aFA} foram determinados na Seção 6.4.1. Portanto, é necessário o cálculo de C_{gs}, apresentado na sequência.

$\boldsymbol{C_{gs}}$:

A busca gulosa utilizada, explicada na Seção 4.2.2, avalia uma quantidade de soluções que aumenta exponencialmente com o tamanho dos BBs identificados. No pior caso, os BBs identificados possuem o tamanho máximo permitido, dado por $v_u + 1$. Dessa forma, o número de verificações a ser realizado é de $\frac{\ell}{v_u+1}2^{v_u+1}$. Assim, C_{gs} é dada pela Equação 6.14.

$$C_{gs} = O(\frac{\ell}{v_u+1}2^{v_u+1}). \tag{6.14}$$

Pode-se então calcular C_{StrOp}, conforme mostrado na Equação 6.15.

$C_{StrOp} = C_{boa} + C_{aFA} + C_{gs} = O(v_l 2^{v_l}\ell^2 N + v_l\ell^3) + O(v_l\ell^3) + O(\frac{\ell}{v_u+1}2^{v_u+1})$;

$$C_{StrOp} = O(v_l 2^{v_l}\ell^2 N + v_l\ell^3 + \frac{\ell}{v_u+1}2^{v_u+1}). \tag{6.15}$$

Ao comparar C_{StrOp} (Equação 6.15) com C_{BOA} (Equação 6.7), verifica-se que C_{StrOp} possui um termo adicional que cresce exponencialmente com v_u. Como $v_u > v$, uma análise precipitada pode induzir a conclusão de que tal termo é dominante em C_{StrOp}. No entanto, é possível estabelecer um valor máximo para v_u que garante $\frac{\ell}{v_u+1}2^{v_u+1} < v_l 2^{v_l}\ell^2 N + v_l\ell^3$ (explicado no Apêndice H). Se tal limite para v_u é respeitado, e considerando $v_l < v$, então $C_{StrOp} < C_{BOA}$, uma vez que $v_l 2^{v_l}\ell^2 N + v_l\ell^3 < v2^v\ell^2 N + v\ell^3$, o que explica os resultados experimentais apresentados na Seção 6.4.

Os valores máximos de v_u para se desprezar o termo $\frac{\ell}{v_u+1}2^{v_u+1}$ variam de 19 a 25 para ℓ (N) variando de 30 (1 300) a 180 (11 800), conforme mostrado na Tabela H.1 do Apêndice H referentes aos experimentos envolvendo $trap5$. Esses valores para v_u são significativamente maiores que os tamanhos dos BBs considerados em *benchmarks* de EDAs. Além disso, problemas do mundo real em sistemas de larga escala (com centenas ou milhares de variáveis) correspondem a redes complexas (grafos complexos de larga escala), em que v_u mantém-se significativamente menor que os valores máximos de v_u calculados. Em outras palavras, no Apêndice H, tais v_u's máximos são maiores que as comunidades em geral encontradas em redes complexas, que podem modelar adequadamente as relações entre variáveis de problemas do mundo real de larga escala.

Existe também um fator omitido em C_{BOA}, dado pela Equação 6.7: o número de gerações g que o algoritmo requer para convergir. Ao considerar este parâmetro, obtém-se $C_{BOA} = O(g\ v2^v\ell^2 N + v\ell^3)$. Diferentemente das outras variáveis que são parâmetros definidos pelo usuário, o valor de g varia com os números aleatoriamente gerados na execução do algoritmo (por exemplo, envolvidos na geração da população inicial e de novas soluções).

O fator g não se encontra presente em C_{StrOp}, dado que o mesmo não é um procedimento iterativo (possui apenas uma geração). Alternativamente, pode-se dizer que em $g = 1$ para o StrOp. Ao eliminar um fator que varia a cada execução do algoritmo, é possível afirmar que existe uma menor variação do tempo de execução ao comparar diferentes execuções do StrOp para um problema, do que quando comparado ao BOA. Consequentemente, as variâncias em termos de TE e NA obtidas nos experimentos das Seções 6.2 e 6.4 são menores para o StrOp, o que é refletido nos pequenos intervalos de confiança apresentados nas Figuras de ambas as Seções.

6.5 Método de Reamostragem para EDAs

Nesta Seção, são testadas versões do CD-BOA e StrOp utilizando o REDA, resultando no CD-BOA+REDA e StrOp+REDA. O **Experimento 1** mostra uma limitação do algoritmo StrOp.

No **Experimento 2**, o algoritmo StrOp+REDA mostra-se capaz de corrigir tal limitação. O **Experimento 4** compara o desempenho do StrOp e StrOp+REDA em relação ao NA e TE. Os **Experimentos 4 e 5** buscam compreender a influência do número de amostras utilizado no desempenho do REDA. Por fim, o **Experimento 6** realiza testes comparando a eficiência dos algoritmos CD-BOA e CD-BOA+REDA.

Experimento 1 - Resolver o problema composto por 5 $ftraps$, variando k sem REDA

Para a realização deste experimento, foram executados testes para funções $ftrap3$ ($k = 3$ e $\ell = 15$), $ftrap4$ ($k = 4$ e $\ell = 20$), $ftrap5$ ($k = 5$ e $\ell = 25$) e $ftrap6$ ($k = 6$ e $\ell = 30$) de forma que todos esses subproblemas tenham $m = 5$. Os valores de v_l e v_u utilizados foram respectivamente, $k - 2$ e $k + 2$, de forma a definir um intervalo apropriado para cada função armadilha utilizada. Tais valores são utilizados em todos os experimentos restantes deste capítulo que envolvem funções armadilha. Em adição ao critério de parada de convergência, foi adicionado o critério de parada de número máximo de 50 gerações atingidas para o algoritmo CD-BOA. Este critério foi adotado para obter resultados mais estáveis, dado que alguns casos deste experimento atingiam a TBO desejada (erro máximo de 1 BB) várias gerações antes do critério de convergência ser atingido. Dessa forma, embora o número máximo de gerações tenha sido limitado, a qualidade das soluções encontradas não foi comprometida.

Como não havia tamanhos de população sugeridos pela literatura de (Pelikan *et al.*, 1999) a serem utilizadas nestes experimentos, os valores de n foram obtidos a partir do método da bissecção (Sastry, 2001). No entanto, ao utilizar esse método, foi constatado que o StrOp sempre utilizava a população máxima permitida, por não ser capaz de satisfazer o critério de convergência da bissecção, isto é, no máximo valores de apenas 1 BB errado em cada uma das 30 repetições.

Foi observado que o StrOp, embora atinja TBOs altos na média, é menos estável que o CD-BOA, obtendo erros de mais de um BB em alguns testes. Dessa forma, o tamanho da população utilizada pelo algoritmo foi calculada da mesma forma que no **Experimento 4** da Seção 6.2, isto é, 10 vezes o tamanho da população utilizada no CD-BOA.

As Figuras 6.19 e 6.20 mostram os resultados, que indicam semelhantes NAs em todos os casos, no entanto, uma TBO relativamente baixa do StrOp para $k = 6$. A TBO de 57% ocorre pois, como o StrOp constrói uma única vez um modelo que associa as variáveis, erros únicos no modelo obtido não são corrigidos, resultando em uma menor taxa de acerto. Isso não ocorre com o BOA ou com o CD-BOA, uma vez que esses constroem um novo modelo a cada geração.

Dessa forma, erros obtidos em gerações anteriores podem ser corrigidos durante a execução das próximas gerações.

O REDA possibilita superar essa limitação do StrOp, uma vez que a partir da população inicial, várias populações de indivíduos selecionados para reprodução podem ser reamostradas, construindo em geral BNs com diferenças para cada reamostragem. Como é esperado que os erros ao associar certo conjunto de variáveis estejam presentes na minoria das BNs obtidas, é possível identificar os conjuntos mais comuns (mais verossímeis) de variáveis associadas, gerando um modelo mais confiável para o problema. Além disso, tal modelo melhora a eficácia e escalabilidade do algoritmo sem prejudicar sua eficiência (em termos do NA), uma vez que o número de avaliações necessárias não aumenta.

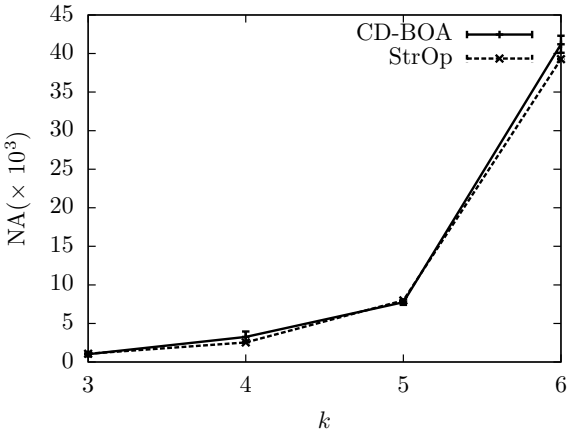


Figura 6.19: Comparação entre NAs requeridos pelo CD-BOA e StrOp (ambos com $v_l = k-2$ e $v_u = k+2$) para problemas compostos por funções armadilha de tamanhos variados.

Experimento 2 - Avaliação do StrOp com REDA

O **Experimento 1** desta Seção foi estendido para avaliar o StrOp modificado com REDA (StrOp+REDA). Dessa vez, foi possível definir o tamanho da população utilizada por meio do método da bissecção. Os resultados mostram que essa abordagem possui uma maior estabilidade, pois foi capaz de satisfazer o critério de realizar 30 repetições para cada problema com erro nos valores das variáveis em no máximo 1 BB em cada teste. Os resultados da Figura 6.21 mostram que a nova abordagem necessitou de um menor NA em todos os casos. A Figura 6.22 mostra que há melhora também no TBO, reforçando a hipótese (**Experimento 1**) de que os erros do StrOp eram devido a problemas na amostragem da população de selecionados usada para construir a BN.

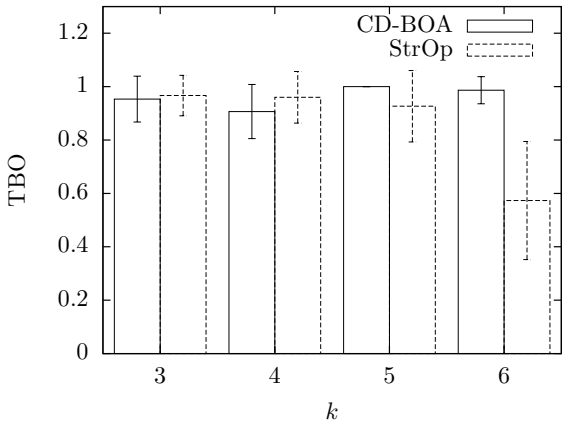


Figura 6.20: Comparação entre TBO dos algoritmos CD-BOA e StrOp (ambos com $v_l = k-2$ e $v_u = k+2$) para problemas compostos por funções trap de tamanhos variados.

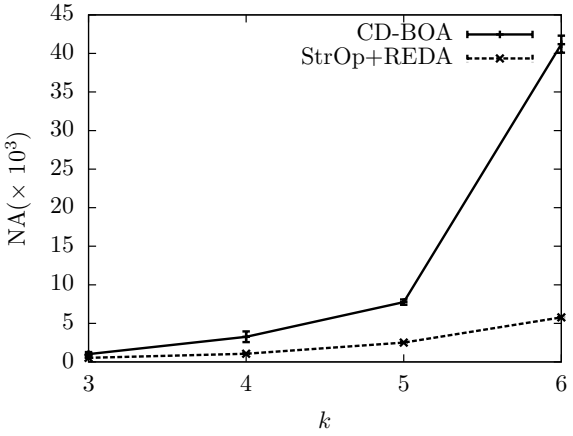


Figura 6.21: Comparação entre NA dos algoritmos CD-BOA e StrOp com REDA (ambos com $v_l = k-2$ e $v_u = k+2$) para problemas compostos por 5 $ftraps$ com k variando de 3 a 6.

Experimento 3 - StrOp+REDA para problemas compostos por trap5

O **Experimento 4** da Seção 6.2 usando o StrOp foi refeito para o StrOp+REDA. Embora resultados satisfatórios tenham sido atingidos pelo StrOp em tal experimento, os testes do **Experimento 2** desta Seção indicam que o StrOp+REDA pode atingir resultados ainda melhores em termos de NA. Para verificar tal afirmação, testes foram feitos utilizando o BOA com $v = 4$ e o StrOp e StrOp+REDA utilizando 10 amostras (ambos com $v_l = 3$ e $v_u = 7$) para resolver o problema composto por funções $ftrap5$. Os valores de n do BOA e do StrOp+REDA foram obtidos pelo método da bissecção, enquanto os valores de n usados no StrOp foram de 10 vezes os usados no BOA. A Figura 6.23 apresenta os resultados que mostram NAs significativamente

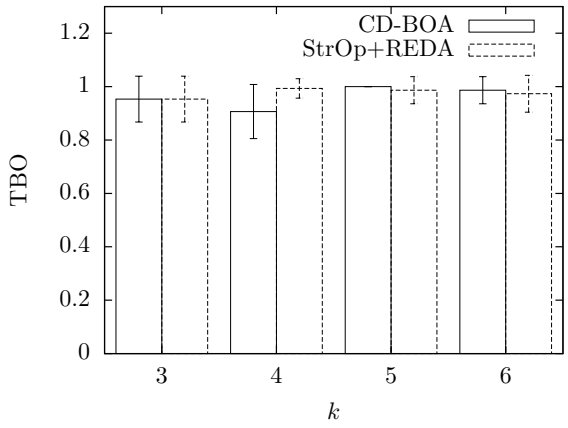


Figura 6.22: Comparação entre TBO do CD-BOA e StrOp com REDA (ambos com $v_l = k - 2$ e $v_u = k + 2$) para problemas compostos por 5 $ftraps$ com k entre 3 e 6.

menores obtidos ao utilizar o REDA, como esperado. Como exemplo, o StrOp+REDA utilizou menos do que 22% do NA utilizado pelo BOA para o caso em que $\ell = 180$.

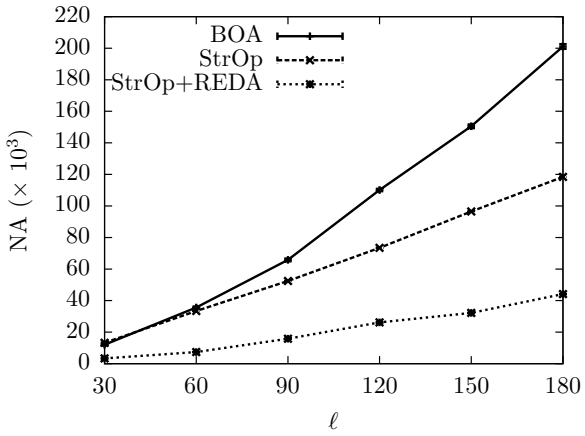


Figura 6.23: Comparação entre NA do BOA ($v = 4$), StrOp e StrOp+REDA (ambos com $v_l = 3$ e $v_u = 7$) para o problema composto por funções $ftrap5$.

A Figura 6.24 mostra o TE dos algoritmos investigados, indicando que o custo por obter um menor NA é um aumento no TE devido as etapas adicionais introduzidas pelo REDA (reamostragens e construção de 10 modelos). No entanto, o TE do StrOp+REDA mostrou-se ligeiramente inferior ao do BOA.

Experimento 4 - Influência da quantidade de amostras

Para verificar a influência da quantidade de amostras utilizada no comportamento do algoritmo, testes foram realizados para o problema composto por $ftrap5$ com $\ell = 90$, va-

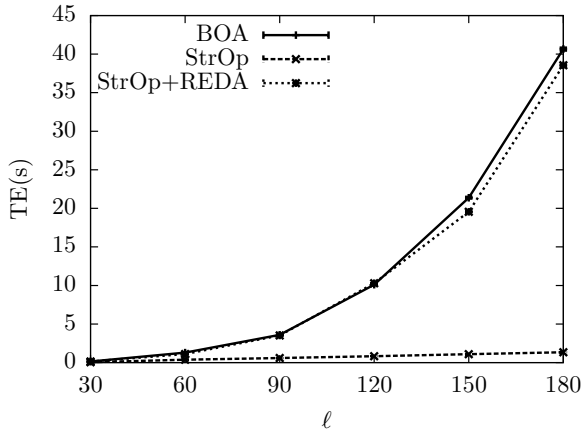


Figura 6.24: Comparação entre TE do BOA ($v = 4$), StrOp e StrOp+REDA (ambos com $v_l = 3$ e $v_u = 7$) para o problema composto por funções $ftrap5$.

riando o número de amostras usadas pelo StrOp+REDA de 1 até 10. Como explicado na Seção 4.3, ao utilizar apenas uma amostra, o StrOp+REDA é equivalente a técnica StrOp (Seção 4.2), já que com apenas uma população de indivíduos selecionados, um único modelo é construído. No entanto, ao utilizar um número de amostras maior do que 1, um modelo é construído para cada amostra e, adicionalmente, um modelo consenso é obtido a partir dos mesmos (Seção 4.3). O tamanho da população utilizado foi obtido pelo método da bissecção. As Figuras 6.25, 6.26, 6.27 e 6.28 mostram, respectivamente, os resultados com relação ao NA, TE, TBO e n. Conforme varia-se a quantidade de amostras entre 1 e 10.

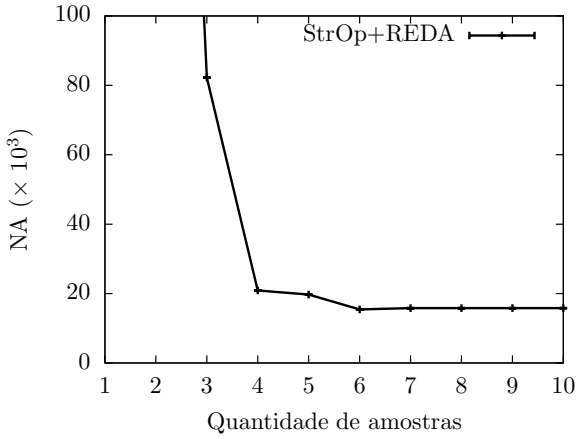


Figura 6.25: NAs do StrOp+REDA variando a quantidade de amostras de 1 a 10 para o problema composto por funções $ftrap5$.

No **Experimento 4**, o critério utilizado na bissecção não foi satisfeito com até 2 amostras. Para esses casos, $n = 400\,000$, que é o tamanho máximo de população permitido pelo método da bissecção utilizado. As Figuras 6.25 e 6.28 evidenciam esse comportamento.

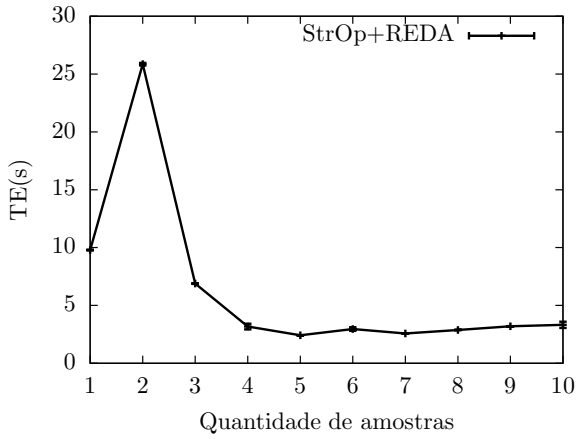


Figura 6.26: TE do StrOp+REDA pela quantidade de amostras de 1 a 10 para o problema composto por funções $ftrap5$.

Os resultados mostram que 2 não é um bom número de amostras a ser utilizado, por não permitir dizer qual dos dois modelos construídos é mais apropriado. Para melhor ilustrar esse comportamento, considere que o modelo 1 afirme a existência de um determinado BBv que, por sua vez, não existe no modelo 2. Com a existência de apenas dois modelos não é possível dizer qual dos dois possui maior probabilidade de estar correto. Este problema é reduzido ao utilizar mais do que duas amostras, uma vez que casos de empate como o ilustrado aqui tornam-se mais improváveis.

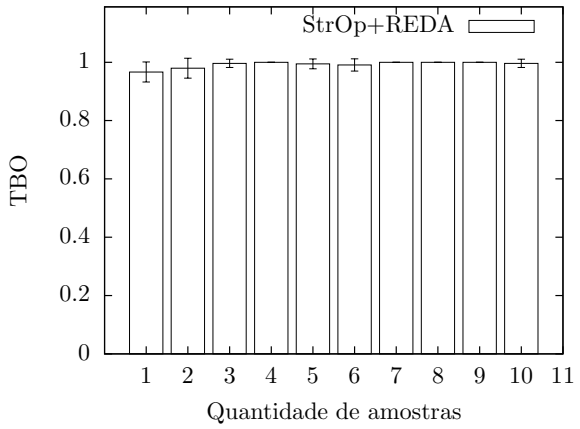


Figura 6.27: TBO do StrOp+REDA pela quantidade de amostras de 1 a 10 para o problema composto por funções $ftrap5$.

Além disso, é possível verificar que usar 2 amostras é ainda pior do que não usar o REDA (número de amostras 1). Isso acontece pois, apesar do segundo modelo não trazer nenhuma melhoria em termos de NA, o uso de 2 amostras aumenta o TE (Figura 6.26) devido ao tempo necessário para a construção de dois modelos, além do tempo para a construção do modelo de consenso, a partir da composição dos dois modelos anteriores (Seção 4.2).

A eficiência do algoritmo melhora quando usando 3 ou mais amostras. O NA diminui conforme o número de amostras aumenta, até que estabiliza ao atingir 6 amostras. O mesmo acontece em relação ao TE, com a diferença de que este apresenta ligeiro aumento a partir de 7 amostras. Este comportamento pode ser compreendido ao verificar o n utilizado (Figura 6.28).

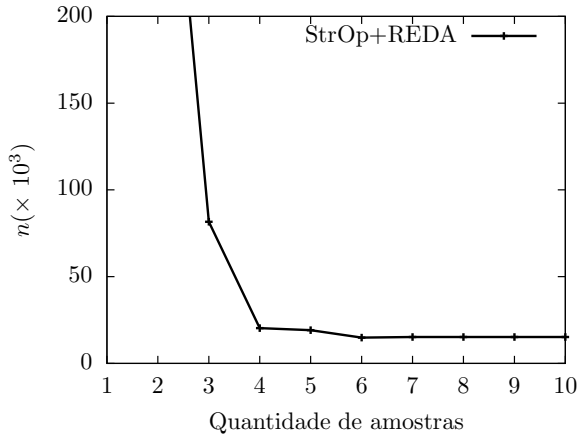


Figura 6.28: n encontrado pelo método da bissecção para o algoritmo StrOp+REDA variando a quantidade de amostras de 1 a 10 para o problema composto por funções $ftrap5$.

Analisando a Figura 6.28, verifica-se que o n encontrado pelo método da bissecção diminui a medida que o número de amostras aumenta, até que 6 amostras são atingidas, quando n se estabiliza. Os valores de n influenciam diretamente no NA e TE, sendo que menores valores de n resultam em menores NA e TE, como indicado pelas Figuras 6.25 e 6.26. No entanto, quando n se estabiliza, o TE aumenta ligeiramente com o número de amostras, devido ao custo computacional em construir um número maior de modelos.

Experimento 5 - Influência da quantidade de amostras para n fixo

O **Experimento 4** permitiu verificar que, mesmo com a adição de mais etapas ao algoritmo com o uso do REDA, o tempo de execução a partir de 3 amostras utilizadas diminui em relação ao StrOp sem REDA (Figura 6.26) por exigir uma população de tamanho menor para se obter modelos bem representativos. Para melhor observar o aumento no TE devido as etapas adicionais inseridas pelo REDA, o **Experimento 4** foi estendido, no entanto, utilizando uma população de tamanho fixo ($n = 52\,000$) para todas as quantidades de amostras testadas.

As Figuras 6.29, 6.30 e 6.31 mostram os resultados referentes ao NA, TE e TBO, respectivamente. A Figura 6.29 mostra que com 3 ou mais amostras, o número médio de NA estabiliza próximo a $52\,576$, que é o NA esperado quando os BBs são corretamente encontrados, calculado a partir da Equação 6.5.

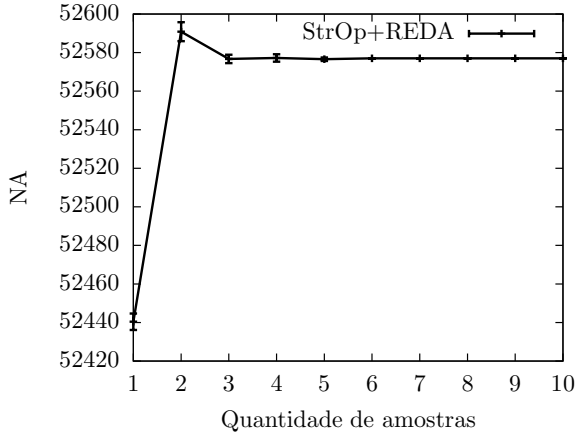


Figura 6.29: NA dos algoritmos StrOp+REDA ($n = 52\,000$) variando a quantidade de amostras de 1 a 10 para o problema composto por funções $ftrap5$.

Como previsto, o TE aumenta com a quantidade de amostras utilizada, devido ao custo adicional em construir múltiplos modelos, como mostra a Figura 6.30. Embora a TBO tenha sido próxima a 1 em todos casos testados, as menores taxas de acerto dizem respeito aos testes com quantidade de amostras 1 e 2, como esperado (justificado no **Experimento 4**).

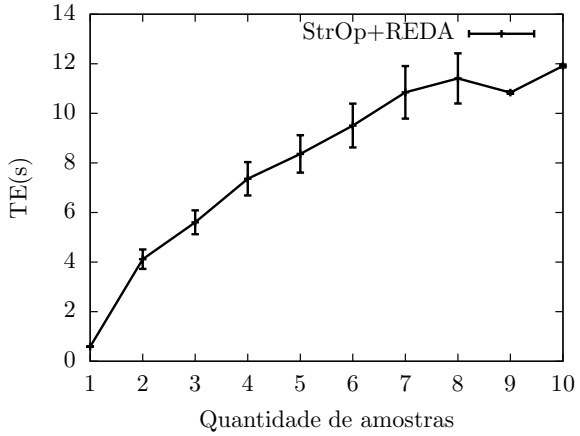


Figura 6.30: TE do StrOp+REDA ($n = 52\,000$) pela quantidade de amostras de 1 a 10 para o problema composto por funções $ftrap5$.

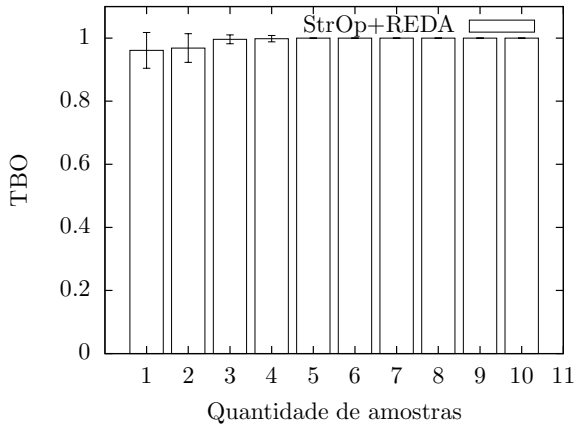


Figura 6.31: TBO do StrOp+REDA ($n = 52\,000$) pela quantidade de amostras de 1 a 10 para o problema composto por funções $ftrap5$.

Experimento 6 - Testes com CD-BOA com REDA

Embora o REDA tenha sido desenvolvido inicialmente para sanar a limitação do StrOp, apresentada no **Experimento 1** desta Seção, a mesma técnica pode ser adaptada a outros EDAs. Com isso, é esperado que a eficiência de tais EDAs seja melhorada devido a capacidade da técnica REDA em permitir a construção de melhores modelos. Para verificar tal possibilidade, foi implementada uma extensão do CD-BOA com reamostragem (CD-BOA+REDA) e o Experimento 3 desta Seção foi repetido para o novo algoritmo.

A Figura 6.32 mostra a capacidade da REDA em auxiliar na obtenção de modelos probabilísticos mais representativos, uma vez que o StrOp+REDA (sem utilizar o valor ideal do NMP diretamente em seus parâmetros de entrada) foi capaz de ultrapassar a eficiência relativa ao NA do BOA utilizando o valor ideal de v.

A eficiência relativa ao NA trazida pelo REDA, em contrapartida, possui um custo adicional na eficiência relativa ao TE, como mostra a Figura 6.33. Como já explorado nos experimentos anteriores desta Seção, tal custo é devido a construção dos modelos adicionais a cada geração. Diferente do experimento feito com o StrOp+REDA, o CD-BOA+REDA obteve um TE maior que o TE do BOA em todos os casos testados. Isso ocorre pelo fato do CD-BOA+REDA ser um algoritmo iterativo, que requer a construção de novos modelos a cada geração do CD-BOA+REDA.

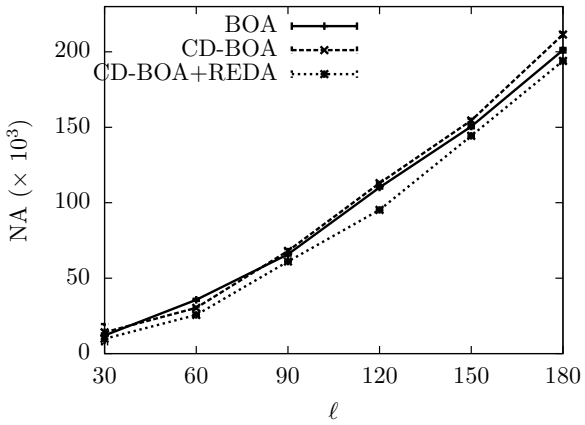


Figura 6.32: Comparação entre NA do BOA ($v = 4$), CD-BOA e CD-BOA+REDA (ambos com $v_l = 3$ e $v_u = 7$) para o problema composto por funções $ftrap5$.

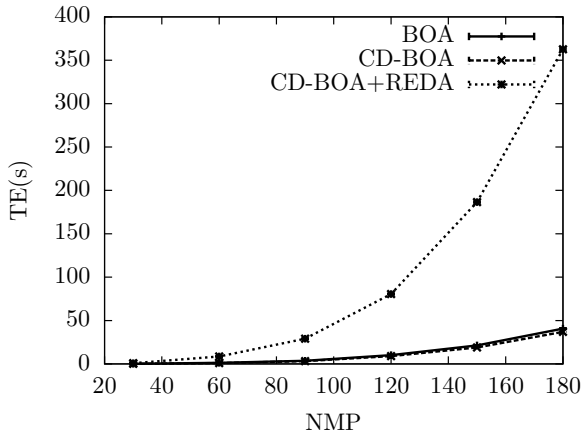


Figura 6.33: Comparação entre TE do BOA ($v = 4$), CD-BOA e CD-BOA+REDA (ambos com $v_l = 3$ e $v_u = 7$) para o problema composto por funções $ftrap5$.

6.6 Avaliação com problemas UmMax, BinInt e com sobreposição de BBs

Os experimentos apresentados a seguir possuem como objetivos testar o desempenho do CD-BOA e StrOp para: i) problemas com sobreposição de funções armadilha e ii) problemas não deceptivos BinInt e UmMax (Seção 5). Os **Experimentos 1 e 2** testam os algoritmos desenvolvidos para problemas envolvendo funções $ftrap5$ com sobreposição de BBs em dois *bits* de cada BB. Esses testes avaliam o desempenho dos algoritmos ao tratar aspectos importantes de problemas hierárquicos (Seção 5.5).

Todos os experimentos anteriores foram realizados com funções armadilha. Tais problemas de difícil resolução modelam aspectos de problemas do mundo real que se colocavam como desafios para algoritmos de busca e otimização. Esses desafios foram importantes para o desenvolvimento de EDAs, uma vez que EAs tradicionais não apresentavam a eficiência desejada para tais problemas. Os **Experimentos 3 e 4** desta Seção testam os algoritmos propostos em dois problemas não deceptivos (UmMax e BinInt, Seção 5.1) que modelam aspectos importantes de problemas do mundo real.

Experimento 1 - Teste do StrOp com problemas contendo sobreposição de funções $ftrap5$

Resolver o problema composto por BBs de tamanho 5 (utilizando $ftrap5$) com sobreposição de BBs em dois *bits* de cada BB. Este tipo de problema é utilizado para representar problemas hierárquicos (Pelikan e Goldberg, 2003). Tal experimento foi elaborado para verificar uma limitação do StrOp, visto que esse foi desenvolvido com foco em resolver problemas decomponíveis deceptivos (Seção 4.2). Foram utilizados os seguintes valores para ℓ: 20, 38, 56, 74, 92, 110. A população foi determinada pelo método da bissecção para o BOA. No entanto, o StrOp+REDA não foi capaz de satisfazer o critério utilizado no método da bissecção (30 testes contendo erro máximo de um BB por teste). Portanto, para todos os testes apresentados a seguir, foi utilizada uma população relativamente grande para o StrOp ($n = 300\,000$).

As Figuras 6.34 e 6.35 mostram os resultados que confirmam a limitação do StrOp para problemas deceptivos não decomponíveis perfeitamente (Seção 7.4). Por outro lado, mostra a capacidade do BOA em solucionar tais problemas. Embora para o caso $\ell = 110$ o StrOp tenha utilizado um número menor de avaliações que o BOA, o mesmo não foi capaz de atingir TBO satisfatória, como indicado pela Figura 6.35. Observa-se ainda que o único caso em que o StrOp obteve uma TBO alta foi para $\ell = 20$.

Experimento 2 - Testes do CD-BOA com problemas contendo sobreposição de funções $ftrap5$

O **Experimento 1** foi repetido, desta vez para o CD-BOA e CD-BOA+REDA. Ambos foram capazes de satisfazer o critério da bissecção e, portanto, possuem TBO próximas a 1,0 em todos os casos testados. Os resultados referentes ao NA são apresentados pela Figura 6.36. Diferentemente do StrOp, é possível observar a capacidade do CD-BOA em resolver problemas com sobreposição de BBs. Embora o BOA com parâmetro $v = 4$ tenha precisado de um menor NA que o CD-BOA, o CD-BOA+REDA obteve uma eficiência maior que o BOA. Tal resultado

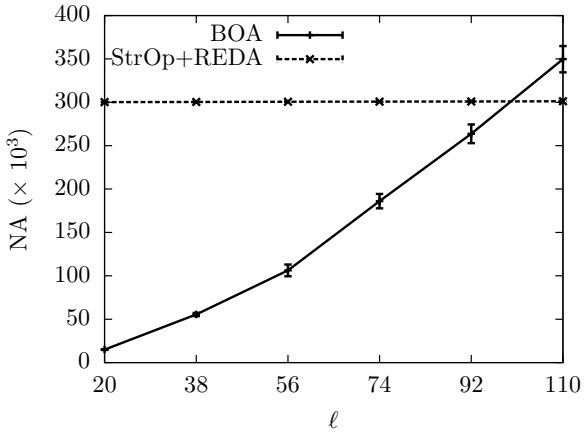


Figura 6.34: Comparação entre NA do BOA ($v = 4$) e StrOp+REDA ($v_l = 3$ e $v_u = 7$) para o problema composto por funções $ftrap5$ com sobreposição de dois *bits*.

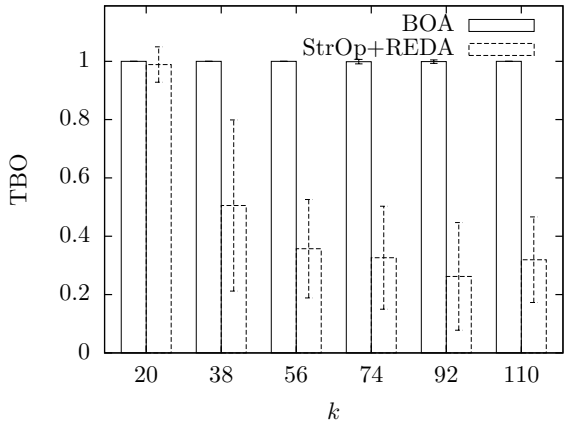


Figura 6.35: Comparação entre TBO do CD-BOA e StrOp+REDA (ambos com $v_l = 2$ e $v_u = 10$) para o problema composto por funções $ftrap5$ com sobreposição de dois *bits*.

é de grande interesse pois, problemas com relação entre BBs são comuns de serem encontrados e de grande importância, como o MAXSAT e o ISIS SPIN Glasses (Seção 5.5).

O parâmetro ideal para o BOA resolver subproblemas $ftrap5$ é $v = 4$, no entanto, ocorrência de sobreposição entre todos os BBs, acarreta em todas as variáveis do problema estarem relacionadas, mesmo que indiretamente. Dessa forma, é possível que um NMP maior possibilite resolver o problema de forma mais eficiente. Experimentos adicionais com o CD-BOA utilizando $v_l = 4$ e $v_u = 10$ confirmam tal suposição, uma vez que o mesmo apresentou uma eficiência maior que a do BOA com $v = 4$ sem utilizar a REDA (Figura 6.37). Isso significa que uma BN gerada com NMP igual a 4 pode ser melhorada ao adicionar um maior número de relações entre variáveis.

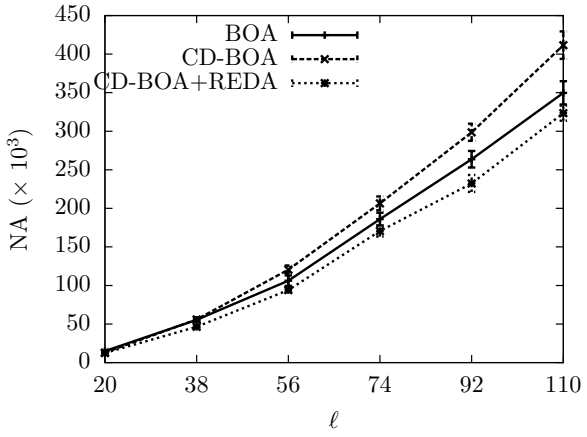


Figura 6.36: NA do CD-BOA, CD-BOA+REDA (ambos com $v_l = 3$ e $v_u = 7$) e BOA ($v = 4$) para o problema composto por funções $ftrap5$ com sobreposição de dois *bits*.

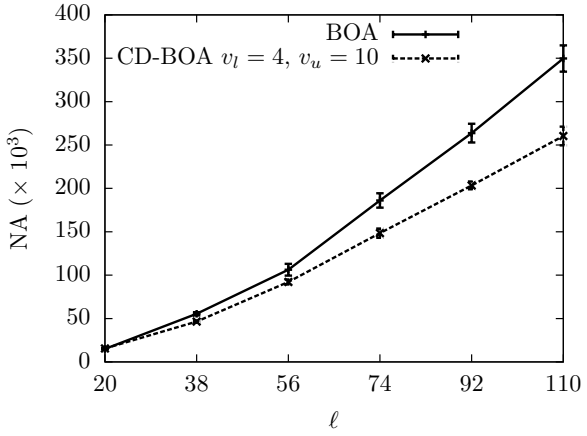


Figura 6.37: NA do CD-BOA ($v_l = 4$ e $v_u = 10$) e BOA ($v = 4$) para o problema composto por funções $ftrap5$ com sobreposição de dois *bits* em cada BB.

Uma alternativa ao utilizar o CD-BOA com maiores valores de v_l e v_u, é utilizar maiores valores para v no BOA. No entanto, a complexidade do BOA cresce exponencialmente com v, como mostrado na Seção 6.4, o que resultaria em maiores TE.

Experimento 3 - Teste com o UmMax

O problema UmMax (Seção 5.1) foi testado para ℓ igual a 5, 10, 15, 20 e 25. A população foi determinada por meio do método da bissecção, como nos experimentos anteriores, admitindo erro máximo de 1 *bit* para os 30 testes realizados. Os algoritmos testados foram: CD-BOA com $v_l = 1$ e $v_u = 1$ (equivalente ao BOA com $v = 1$) e o algoritmo StrOp+REDA, usando os mesmos parâmetros.

Os resultados da Figura 6.38 revelam que ambos os algoritmos foram capazes de resolver o problema de forma eficaz (erro máximo de 1 *bit* por solução obtida). No entanto, o algoritmo StrOp necessitou de um NA significativamente menor, como esperado. Isso deve-se ao fato da natureza da busca gulosa utilizada pelo StrOp, que garante a obtenção da solução ótima em um problema em que não existe relação entre as variáveis (BBs de tamanho 1).

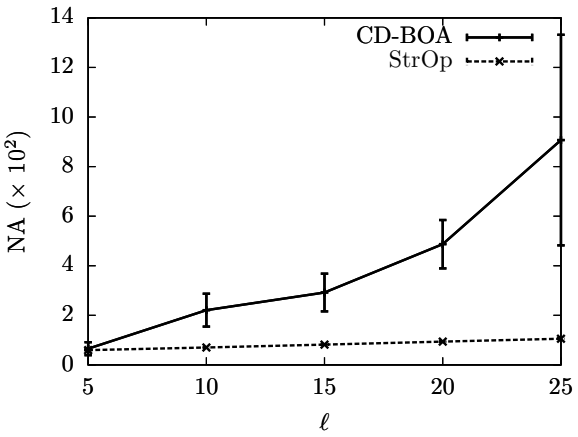


Figura 6.38: Comparação entre o NA dos algoritmos CD-BOA e StrOp+REDA (ambos com $v_l = 1$ e $v_u = 1$) para o problema UmMax.

Experimento 4 - Teste com o BinInt

O **Experimento 3** foi repetido utilizando desta vez o problema BinInt. Embora similar ao UmMax, tal problema estressa a possibilidade de algoritmos probabilísticos convergirem por deriva, como explicado na Seção 5.1. Os resultados na Figura 6.39 mostram que mais uma vez, o StrOp precisou de um NA bem menor em todos os casos testados.

A Figura 6.40 apresenta resultados que enfatizam o aumento na dificuldade ao passar do problema UmMax para o BinInt. Isso ocorre devido a convergência por deriva (ver Seção 5.1) que pode ocorrer para o CD-BOA e outros algoritmos aplicados ao problema BinInt. No entanto, o StrOp não utiliza um procedimento iterativo (possui apenas uma geração, com a construção de somente um modelo probabilístico) como o BOA ou o CD-BOA, e assim, não ocorre deriva com ele. Por isso, o StrOp mantém desempenho similar (NAs) para os problemas UmMax e BinInt como mostra a Figura 6.41. Com isso, identifica-se outra classe de problemas para a qual existem vantagens na utilização do StrOp: a classe dos problemas que apresentam deriva.

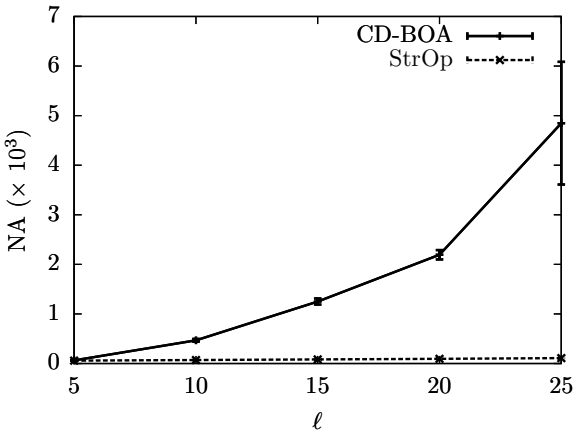


Figura 6.39: Comparação entre o NA dos algoritmos CD-BOA e StrOp+REDA (ambos com $v_l = 1$ e $v_u = 1$) para o problema BinInt.

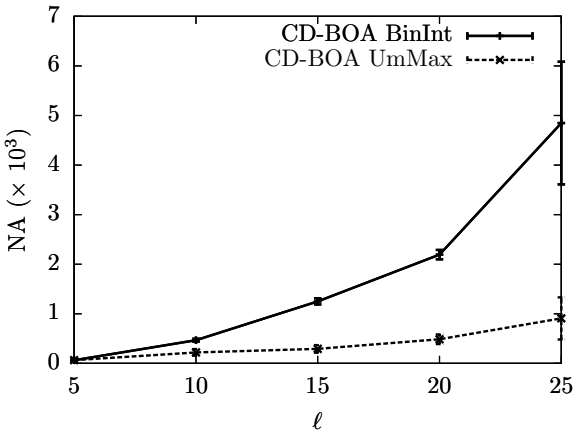


Figura 6.40: Comparação entre os NAs do CD-BOA para os problemas UmMax e BinInt.

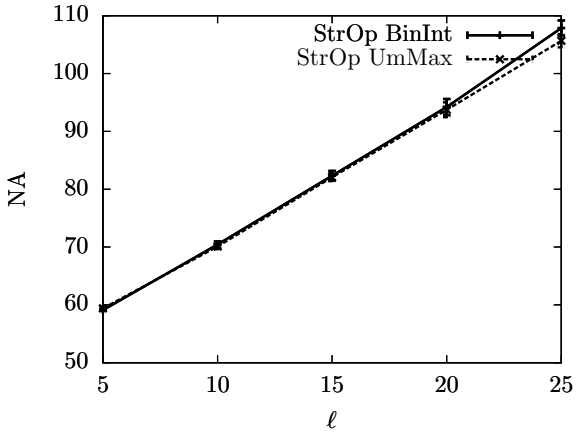


Figura 6.41: Comparação entre os NAs do algoritmo StrOp para os problemas UmMax e BinInt.

6.7 Discussão dos resultados

Os resultados de maior interesse, apresentados neste capítulo, são sintetizados a seguir:

1. Primeiramente, verificou-se a capacidade da técnica de detecção de estruturas de comunidades, aFA, em aprimorar os modelos obtidos na primeira iteração do BOA, gerando algoritmos eficientes como o CD-BOA e o StrOp, tanto em relação ao NA, quanto em relação ao TE;

2. Maior robustez do CD-BOA em relação ao BOA ao utilizar parâmetros de entrada diferentes dos ideais, como mostram os experimentos da Seção 6.3. A robustez é mantida também em problemas compostos por BBs de tamanhos diferentes. O BOA é um EDA que apresenta muitos resultados relevantes na literatura, sendo que muitas extensões do mesmo já foram desenvolvidas, como o BOA hierárquico (hBOA) (Pelikan e Goldberg, 2003), BOA para problemas multiobjetivos (Pelikan *et al.*, 2005) ou o BOA para números reais (rBOA) (Ahn *et al.*, 2004). No entanto, este importante algoritmo utiliza um parâmetro relativo ao tamanho dos BBs, porém, essa informação não está disponível em problemas do tipo caixa preta, nem mesmo pode-se saber se um problema possui todos os BBs de tamanho igual. Apesar dessa limitação, nenhuma implementação deste algoritmo havia sido criada com a finalidade de abordar problemas com BBs de tamanhos desconhecidos, que é comum de ser encontrado ao se tratar problemas do mundo real. Como solução a esta limitação, o CD-BOA demonstra ter a capacidade de estimar o tamanho dos BBs do problema em tempo de execução;

3. Constatação de que o algoritmo StrOp pode ser significativamente mais eficiente que o CD-BOA e BOA para problemas deceptivos decomponíveis (Seções 6.2 e 6.4): O surgimento dos EDAs foi fortemente motivado pelos problemas deceptivos, em que os AEs tradicionais (como os GAs) possuem dificuldade em solucionar de maneira eficiente. Para esses problemas, os EDAs surgiram como uma alternativa para esses EAs. No entanto, uma subcategoria desses problemas (representantes de uma grande quantidade de problemas do mundo real), os problemas deceptivos decomponíveis, podem ser solucionados de forma mais eficiente do que com a utilização dos atuais EDAs. Com este propósito, foi proposto o StrOp, capaz de resolver tal classe de problemas complexos com uma eficiência significativamente maior do que os EDAs atuais;

4. Melhoria introduzida pela técnica REDA, que tornou o StrOp muito mais confiável do que sua primeira versão (Experimentos 1 e 2 da Seção 6.5). Tal técnica de reamostragem

pode ser utilizada também em outros EDAs, conforme foi avaliado no CD-BOA+REDA, melhorando os modelos utilizados sem aumentar o NA necessário. Este resultado mostra que técnicas de reamostragem podem ser utilizadas para melhorar a qualidade dos modelos probabilísticos utilizados em EDAs. Isso é feito sem a necessidade de aumentar o NA para a construção desses modelos, uma vez que as soluções utilizadas para compor uma maior quantidade de modelos são reamostradas de um conjunto de soluções já avaliadas;

5. Prova da complexidade de tempo de etapas do CD-BOA e do StrOp comprovando que são eficientes para instâncias de larga escala dos problemas investigados;

6. Identificação de classes de problemas que cada um dos algoritmos propostos possui maior probabilidade de sucesso, dado pela Tabela 6.1.

Tabela 6.1: Problemas e algoritmos sugeridos.

Tipo de problema	Algoritmo sugerido
Tamanho de BBs desconhecido	CD-BOA ou StrOp
Tamanho de BBs conhecido	BOA, CD-BOA ou StrOp
Com convergência à deriva	StrOp
Sem convergência à deriva	BOA, CD-BOA ou StrOp
Com sobreposição de BBs	CD-BOA
Sem sobreposição de BBs	StrOp

CAPÍTULO

7

Conclusões

O objetivo deste trabalho foi o de mostrar como algoritmos de detecção de comunidades (CDAs, Seção 3.1) podem contribuir na construção de EDAs baseados em Redes Bayesianas (BN, Seção 2.6). Para isso, foram propostos e implementados o CD-BOA (Seção 4.1), o StrOp (Seção 4.2), um método de reamostragem que usa CDAs, chamado REDA (Seção 4.3), além de experimentos com os mesmos em problemas de *benchmark* considerados desafios na literatura da área.

O BOA é um dos EDAs mais bem sucedidos atualmente, dados os tipos de diferentes problemas complexos que resolveu (Ahn *et al.*, 2004; Pelikan e Goldberg, 2003; Pelikan *et al.*, 1999, 2005). Esta tese colaborou com o estado da arte ao propor avanços significativos em relação a esse algoritmo, em termos de eficiência e robustez. Tais avanços podem ser também estendidos às demais variações do BOA (Seção 2.7), contribuindo assim na resolução de uma maior diversidade de problemas.

As principais contribuições deste trabalho são destacadas nas seções seguintes, que encontram-se organizadas da seguinte maneira: Seção 7.1 discute a generalização do BOA para problemas de caixa preta; Seção 7.2 apresenta considerações sobre a eficiência computacional dos algoritmos desenvolvidos; Seção 7.3 avalia as contribuições do REDA para EDAs; Seção 7.4 discute a importância dos DecOAs; Seção 7.5 contextualiza o trabalho sob a perspectiva do *No free lunch theorem* (Wolpert e Macready, 1997). Por fim, a Seção 7.6 propõe extensões consideradas relevantes neste trabalho.

7.1 Generalização do BOA

EDAs têm se destacado entre os EAs por resolverem uma quantidade significativa de problemas deceptivos considerados computacionalmente complexos. Em particular, o BOA é um EDA que apresenta resultados relevantes, com extensões deste para resolver problemas no domínio dos números reais (Ahn *et al.*, 2004), problemas hierárquicos (Pelikan, 2005) e multiobjetivos (Pelikan *et al.*, 2005). No entanto, uma limitação do BOA é a necessidade do conhecimento sobre o tamanho dos BBs do problema sendo tratado, para garantir sua eficiência. Neste trabalho, foi possível constatar que o uso de CDAs possibilitou a generalização do desempenho do BOA em problemas para os quais tal conhecimento não esteja disponível, uma vez que o CDA desenvolvido (aFA) identifica, em tempo de execução, os tamanhos dos BBs do problema. Dessa forma, o CD-BOA é mais robusto ao utilizar parâmetros de entrada referentes ao tamanho dos BBs, relativamente distantes dos tamanhos dos BBs do problema. Com isso, a eficiência do CD-BOA é preservada também para problemas com BBs de tamanhos diferentes.

Tal robustez é importante para problemas do mundo real, em que o tamanho dos BBs não é conhecido a priori. Este benefício do CD-BOA pode também atingir problemas considerados pelas outras extensões do BOA, por exemplo, aumentando a robustez dos algoritmos para tratamento de problemas hierárquicos (Pelikan, 2005) e multiobjetivos (Pelikan *et al.*, 2005).

7.2 Eficiência Computacional

O critério mais utilizado para se medir a eficiência de EAs é o Número de Avaliações (NA, Seção 6.2). No entanto, é importante o Tempo de Execução (TE, Seção 6.4) de EDAs, dado que a construção de modelos probabilísticos com adequada representatividade é normalmente realizada por algoritmos computacionalmente custosos. Dessa forma, um grande desafio é encontrar modelos probabilísticos representativos em um tempo computacional relativamente baixo (Pelikan *et al.*, 2002). Uma limitação do BOA tem sido o TE alto para a geração do modelo, como apontado em (Ahn *et al.*, 2004).

Foi mostrado neste trabalho que, ao utilizar técnicas de CDA, o CD-BOA possui TE menor que o BOA e NA similar, possuindo dessa forma um melhor compromisso entre esses dois critérios de avaliação (Seções 6.2 e 6.4). Este benefício foi constatado tanto experimentalmente, quanto ao calcular e comparar a complexidade de tempo para a criação da rede nos dois algoritmos.

A partir da análise da complexidade de tempo de etapas dos algoritmos, foi possível estabelecer condições para os parâmetros de entrada v_l e v_u de forma a garantir um crescimento

moderado do tempo de execução, tanto para o CD-BOA quanto para o StrOp. Verificou-se que, ao aumentar o parâmetro responsável pelo número máximo (v_u para o CD-BOA e o StrOp, e v para o BOA) de variáveis correlacionadas, identificadas no modelo utilizado, tanto o StrOp quanto o CD-BOA apresentam um crescimento do TE assintoticamente menor que o do BOA. Isso indica que uma maior quantidade de problemas (com diferentes quantidades de variáveis correlacionadas) pode ser resolvido com TE aceitável. O StrOp apresentou uma redução significativa do TE, chegando a ser até 30 vezes mais rápido que o BOA para os problemas testados. Esses resultados são importantes para problemas que necessitem de TE relativamente baixo para encontrar uma solução. Tais tipos de problemas, em geral, ocorrem em sistemas que requerem respostas em tempo real, como o aprendizado *on-line* em jogos eletrônicos (Crocomo, 2008), o restabelecimento de energia em sistemas elétricos (Santos *et al.*, 2008) e a estimação de frequência e fase (Coury *et al.*, 2012).

A utilização de CDAs na obtenção dos modelos probabilísticos também tornou possível diminuir o NA necessário para encontrar a solução de diversos problemas, principalmente nos experimentos relativos aos algoritmos StrOp e StrOp+REDA. Para este último, a redução do NA atingiu até 78% quando comparado ao BOA.

O REDA também beneficiou positivamente o desempenho do CD-BOA. A reamostragem de modelos do REDA possibilitou a construção de um melhor modelo para o algoritmo, reduzindo o NA. Por outro lado, a utilização de tal método acarreta o aumento do TE, uma vez que a construção do modelo utilizado exige a execução de rotinas adicionais. É importante lembrar que a execução dessas rotinas adicionais pode ser paralelizada, dada a natureza do método proposto, o que reduziria significativamente o aumento no TE caso o *hardware* necessário se encontre disponível.

7.3 Reamostragem

O REDA aplica CDAs múltiplas vezes, sobre diferentes BNs que representam conjuntos de soluções reamostradas. Dessa forma, é possível eliminar erros nas construções dos modelos que ocorrem para uma outra amostragem mas não surgem na maior parte delas. A partir desse processo, encontra-se um modelo mais confiável para representar as variáveis correlacionadas no problema.

Como discutido na Seção 7.2, o REDA afeta direta e positivamente a eficiência dos algoritmos, a qual pode ainda ser melhorada a partir da paralelização dos processamentos relativos a cada reamostragem. Outro benefício é a redução da população necessária para se executar os algoritmos que utilizam o método, constatado experimentalmente neste trabalho. Isso deve-se

ao fato de que, por se tratar de um método que possui certa robustez a erros nos modelos, é necessária uma menor quantidade de informações sobre o problema para a construção deles.

Embora o REDA não esteja restrito à identificação de comunidades, a utilização de CDAs permitiu o desenvolvimento deste método em específico. O mesmo pode ser diretamente adaptado em qualquer EDA que busque a identificação de partições do problema e, portanto, torna-se uma contribuição relevante deste trabalho.

Além disso, o método proposto possibilitou a correção de uma limitação do algoritmo StrOp, que é a incapacidade deste em corrigir erros no modelo encontrado ao longo do processo de otimização, uma vez que um único modelo probabilístico é construído no StrOp. Este aspecto torna o REDA ainda de maior importância para DecOAs, que nem sempre são procedimentos iterativos, assim como ocorre com o algoritmo StrOp.

7.4 Otimização por Decomposição

EDAs tradicionais buscam por modelos probabilísticos durante o processo de otimização, tendo sempre como foco, a otimização de um problema. No entanto, alguns EDAs (em pesquisas centralizadas no Laboratório de Computação Reconfigurável da USP) (Crocomo e Delbem, 2011; Melo *et al.*, 2011; Vargas e Delbem, 2009) têm sido desenvolvidos com duas etapas principais: a primeira, voltada exclusivamente a compreensão/modelagem do problema a ser otimizado e, a segunda, responsável pelo processo de otimização em si.

Neste trabalho, foi proposta uma classe de algoritmos com a finalidade de englobar tais métodos, chamada de DecOA. Também foi proposto e implementado um algoritmo pertencente a tal classe, o StrOp. Os resultados com relação ao StrOp identificaram uma maior eficiência do mesmo com relação ao BOA para vários dos problemas testados, como discutido na Seção 7.2. Foi também identificada a capacidade do StrOp em otimizar problemas que normalmente convergem à deriva em EAs convencionais.

Uma limitação encontrada do algoritmo StrOp é a de resolver problemas que possuem sobreposição de BBs. Isso deve-se ao fato dele utilizar uma busca gulosa na etapa de otimização, que ignora relações que possam existir entre BBs do problema. É importante ressaltar que essa é uma limitação do algoritmo StrOp como proposto e não de todos os algoritmos pertencentes à classe DecOA. Uma alternativa, para corrigir essa limitação, seria substituir a busca gulosa por outra que utilize as BBv's como metavariáveis, usando uma heurística para tratar relações entre as metavariáveis. Por exemplo, o método PhyDE (Melo *et al.*, 2011) utiliza como metaheurística o DE (Storn e Price, 1997) para encontrar os melhores valores das metavariáveis (identificadas por uma etapa preliminar de decomposição do problema original).

Além disso, em (Pelikan e Goldberg, 2001) é enfatizada a importância de desenvolver algoritmos que possam resolver de forma eficiente problemas não-hierárquicos. A partir de pequenas alterações nesses algoritmos, problemas hierárquicos passam a ser resolvidos, conforme ocorre do BOA para o h-BOA (Seção 2.7). Nesse contexto, o StrOp foi capaz de resolver os mesmos problemas com uma eficiência significativamente maior que o BOA, gerando a perspectiva de que tal vantagem possa ser explorada para problemas hierárquicos.

7.5 Almoço grátis e escolha de algoritmos

Segundo o teorema da inexistência de almoço grátis (*No free lunch theorem*) (Wolpert e Macready, 1997), não existe um algoritmo que seja o melhor a ser utilizado em todos os problemas de otimização existentes. Por esse motivo, não é objetivo deste trabalho encontrar qual é o melhor EDA (ou variação) existente.

No entanto, os resultados deste trabalho possibilitam identificar tipos de problemas para os quais é mais interessante a utilização de um ou outro algoritmo. A partir desses resultados, a Tabela 6.1 (Seção 6.7) foi gerada buscando ser um "guia" relevante na escolha do algoritmo a ser utilizado para a otimização de um determinado problema. Dessa forma, esse "guia" é uma contribuição relevante alcançada.

7.6 Perspectivas de Novos Avanços

A partir dos resultados obtidos neste trabalho, novas ideias surgem para o aprimoramento dos métodos apresentados e criação de novos algoritmos. Dentre essas, podem ser destacadas:

- **Versão paralela do REDA**: como discutido anteriormente, o REDA apresentou resultados relevantes, que podem ser melhorados a partir da implementação de sua paralelização, dado que a obtenção de cada reamostragem e geração de cada modelo pode ser executado independentemente;

- **Extensões do algoritmo StrOp**: uma limitação constatada do algoritmo StrOp é na resolução de problemas que possuem fortes correlações entre BBs. Uma possível estratégia para superar essa limitação é a substituição do método de busca gulosa utilizado. Para isso, poder-se-ia utilizar, por exemplo, o algoritmo DE, de forma similar a que ocorre no PhyDE (Melo *et al.*, 2011). Outra possibilidade é o desenvolvimento de uma extensão multiobjetivo para o StrOp.

- **Desenvolvimento de DecOAs**: como constatado, o algoritmo StrOp implementado possui vantagens ao tratar problemas para os quais EAs e mesmo outros EDAs normalmente convergem à deriva. Dado que DecOAs possuem uma primeira etapa responsável apenas por compreender/modelar o problema, é possível identificar os aspectos de cada problema que o torna difícil e, então, selecionar um algoritmo de busca adequado a ser utilizado na segunda etapa. Tal estratégia segue como uma continuação do trabalho proposto em (Wolpert e Macready, 1997), que apresenta o *No Free Lunch Theorem*. Como tal teorema afirma que não existe um algoritmo de busca que seja, em média, superior a outro, propõe-se a criação de uma ferramenta de dois passos: o primeiro é responsável por compreender particularidades da função objetivo (desconhecida em otimização de caixa-preta) relativa à qual será realizada a busca; o segundo monta ou escolhe o algoritmo de busca a ser utilizado.

- **Expansão das vantagens do CD-BOA para derivações do BOA**: o algoritmo BOA possui derivações para a resolução de diferentes problemas, como problemas de números reais (Ahn *et al.*, 2004), multiobjetivos (Pelikan *et al.*, 2005) e hierárquicos (Pelikan, 2005). Da mesma forma que este projeto gerou o CD-BOA a partir do BOA, as derivações do BOA também podem ser estendidas com base em CDAs e REDA. Na prática, isso requer a adaptação do *patch* implementado para o BOA (Capítulo 6) às suas derivações (Seção 2.7), aumentando assim a quantidade de problemas beneficiados pelos avanços de desempenho para EDAs obtidos neste trabalho.

- **Uso de conhecimento sobre o problema**: um DecOA pode ser visto como um algoritmo que acrescenta uma primeira etapa para compreender/modelar o problema tratado. No entanto, os métodos apresentados partem do princípio de que nenhum conhecimento ainda existe (problemas de caixa-preta). Nem sempre, na prática, a situação é tão extrema. Há casos em que existem certas informações sobre o problema, que podem ser utilizadas, auxiliando assim na eficiência e eficácia do algoritmo. Tais informações podem ser apresentadas ao CD-BOA ou StrOp por meio de uma lista de variáveis correlacionadas previamente, por exemplo. Com isso, o desempenho desses algoritmos poderia ser significativamente aumentado uma vez que tais relações não precisariam ser encontradas em tempo de execução.

Bibliografia

ABRAHAM, A.; JAIN, L.; GOLDBERG, R., eds. *Evolutionary Multiobjective Optimization: Theoretical Advances and Applications*. 1 ed. New York: Springer, 302 p., 2005.

AHN, C. W.; RAMAKRISHNA, R. S.; GOLDBERG, D. E. Real-Coded Bayesian Optimization Algorithm: Bringing the Strength of BOA into the Continuous World. In: DEB, K., ed. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2004)*, Springer-Verlag, Berlin, 2004, p. 840–851.

APORNTEWAN, C.; CHONGSTITVATANA, P. Building-block identification by simultaneity matrix. *Soft Comput.*, v. 11, n. 6, p. 541–548, 2007.
Disponível em `http://dx.doi.org/10.1007/s00500-006-0097-z`

BABA, N.; HANDA, H. Commons Game Made More Exciting by an Intelligent Utilization of the Two Evolutionary Algorithms. In: BABA, N.; JAIN, L. C.; HANDA, H., eds. *Advanced Intelligent Paradigms in Computer Games*, New York: Springer. (Studies in Computational Intelligence, 71), p. 1–16, 2007.

BÄCK, T. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford: Oxford University Press, 328 p., 1996.

BÄCK, T.; FOGEL, D.; MICHALEWICZ, Z., eds. *Evolutionary computation 1: Basic algorithms and operators*. Bristol: Institute of Physics Publishing, 339 p., 2000.

BARKAT ULLAH, A. S.; SARKER, R.; CORNFORTH, D. Search space reduction technique for constrained optimization with tiny feasible space. In: *Annual conference on Genetic and evolutionary computation*, New York: ACM, 2008, p. 881–888.

BAROLLI, L.; IKEDA, M.; MARCO, G.; DURRESI, A.; KOYAMA, A.; IWASHIGE, J. A search space reduction algorithm for improving the performance of a GA-based qos routing method in ad-hoc networks. *Int. J. Distrib. Sen. Netw.*, v. 3, n. 1, p. 41–57, 2007.

BEYER, H. G.; ARNOLD, D. V. *Theory of evolution strategies: A tutorial* London: Springer-Verlag, p. 109–133, 2001.

BOETTCHER, S.; PERCUS, A. G. Optimization with extremal dynamics. *Complex.*, v. 8, n. 2, p. 57–62, 2002.

BOLTHAUSEN, E.; BOVIER, A. *Spin glasses*, v. 1900 de *Lecture Notes in Mathematics*. New York: Springer, 368 p., 2007.

BONETTI, D. R.; DELBEM, A. C.; TRAVIESO, G.; SOUZA, P. S. L. Enhanced van der waals calculations in genetic algorithms for protein structure prediction. *Concurrency and Computation: Practice and Experience*, p. n/a–n/a, 2012.
Disponível em `http://dx.doi.org/10.1002/cpe.2913`

BRAGA, A. P.; CARVALHO, A. P. L. F.; LUDEMIR, T. B. *Redes neurais artificiais : Teoria e aplicações*. Rio de Janeiro: LTC Editora, 238 p., 2007.

BRASIL, C. R. S. *Algoritmo evolutivo de muitos objetivos para predição ab initio de estrutura de proteínas*. Tese de Doutorado, Universidade de São Paulo, São Carlos, 2012.

CASTRO, L. *Computação natural - uma jornada ilustrada*. LIVRARIA DA FISICA, 266 p., 2010.

CHOI, S.-S.; JUNG, K.; KIM, J. H. Phase transition in a random nk landscape model. *Artificial Intelligence*, v. 172, n. 2Ű3, p. 179 – 203, 2008.
Disponível em `http://www.sciencedirect.com/science/article/pii/S0004370207001099`

COLE, N.; LOUIS, S.; MILES, C. Using a genetic algorithm to tune first-person shooter bots. In: *Evolutionary Computation, 2004. CEC2004. Congress on*, 2004, p. 139 – 145 Vol.1.

COOK, W. J.; CUNNINGHAM, W. H.; PULLEYBLANK, W. R.; SCHRIJVER, A. *Combinatorial optimization*. New York: John Wiley, 1997.

COOPER, G. F.; HERSKOVITS, E. A bayesian method for the induction of probabilistic networks from data. *Machine Learning*, v. 9, p. 309–347, 1992.

COTTA, C. Protein structure prediction using evolutionary algorithms hybridized with backtracking. In: *International Work-Conference on Artificial and Natural Neural Networks: Part II: Artificial Neural Nets Problem Solving Methods*, Berlin: Springer-Verlag, 2003, p. 321–328.

COURY, D. V.; DELBEM, A. C. B.; DE CARVALHO, J. R.; OLESKOVICZ, M.; DO VALLE SIMÕES, E.; BARBOSA, D.; DA SILVA, T. V. Frequency estimation using a genetic algorithm with regularization implemented in fpgas. *IEEE Trans. Smart Grid*, v. 3, n. 3, p. 1353–1361, 2012.

CROCOMO, M. K. *Um algoritmo evolutivo para aprendizado on-line em jogos eletrônicos*. Dissertação de Mestrado, Universidade de São Paulo, São Carlos, 2008.

CROCOMO, M. K. Repositório dos códigos implementados - algoritmo fast newman, reda, patch cd-boa e strop. 2012.
Disponível em `http://lcrserver.icmc.usp.br/colab/cd-boa` (Acessado em 02 oct. 2012)

CROCOMO, M. K.; DELBEM, A. C. B. *Otimização por decomposição*. Relatório Técnico 3131, Universidade de São Paulo, 2011.
Disponível em `http://www.icmc.usp.br/~biblio/relatorios_tecnicos.php` (Acessado em 10 set. 2012)

DARWIN, C. R. *On the origin of species or the preservation of favoured races in the struggle for life*. Kent, UK: John Murray, 502 p., 1859.

DE JONG, K. A. *Evolutionary computation: A unified approach*. Cambridge: MIT Press, 256 p., 2006.

DEB, K.; PRATAP, A.; AGARWAL, S.; MEYARIVAN, T. A fast and elitist multiobjective genetic algorithm: NSGA-II. *Evolutionary Computation, IEEE Transactions on*, v. 6, n. 2, p. 182 –197, 2002.

DELBEM, A. C. B.; LIMA, T.; TELLES, G. P. Efficient forest data structure for evolutionary algorithms applied to network design. *Evolutionary Computation, IEEE Transactions on*, v. PP, n. 99, p. 1, 2012.

DICK, G. Automatic identification of the niche radius using spatially-structured clearing methods. In: *IEEE Congress on Evolutionary Computation*, IEEE, 2010, p. 1–8.

DIESTEL, R. *Graph Theory*, v. 173 de *Graduate Texts in Mathematics*. 3 ed. Springer-Verlag, Heidelberg, 2005.
Disponível em <`http://www.math.uni-hamburg.de/home/diestel/books/graph.theory/GraphTheoryIII.pdf`> (Acessado em 08 fev. 2011)

DONETTI, L.; MUÑOZ, M. A. Detecting network communities: a new systematic and efficient algorithm. *Journal of Statistical Mechanics: Theory and Experiment*, v. 2004, n. 10, p. P10012, 2004.

DUCH, J.; ARENAS, A. Community detection in complex networks using extremal optimization. *Physical Review E*, v. 72, n. 2, p. 027104+, 2005.

EFRON, B.; GONG, G. A leisurely look at the bootstrap, the jackknife, and cross-validation. *The Amer. Stat.*, v. 37, p. 36–48, 1983.

EMMENDORFER, L. R. *Aprendizado da ligação entre genes em computação evolutiva: Uma nova abordagem baseada em estatísticas de baixa ordem*. Tese de Doutorado, Universidade Federal do Paraná, 2007.

ENGELBRECHT, A. P. *Fundamentals of computational swarm intelligence*. New York: John Wiley & Sons, 672 p., 2007.

FELSENSTEIN, J. *Inferring phylogenies*. Connecticut: Sinauer Associates, 664 p., 2003.

FLOREANO, D.; MATTIUSSI, C. *Bio-inspired artificial intelligence: Theories, methods, and technologies*. Intelligent Robotics and Autonomous Agents. Cambridge, MA, USA: The MIT Press, 674 p., 2008.

FOGEL, D. B. *Evolutionary computation : Toward a new philosophy of machine intelligence*. IEEE Press, 296 p., 2005.

GALINIER, P.; HAO, J.-K. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, v. 3, p. 379–397, 10.1023/A:1009823419804, 1999. Disponível em `http://dx.doi.org/10.1023/A:1009823419804` (Acessado em 10 set. 2012)

GAREY, M. R.; JOHNSON, D. S. *Computers and intractability: A guide to the theory of np-completeness*. W. H. Freeman, 338 p., 1979.

GASPAR-CUNHA, A.; TAKAHASHI, R.; ANTUNES, C. H. *Manual de computação evolutiva e metaheurística*. Coimbra: Imprensa da Universidade de Coimbra, 2012.

GOLDBERG, D. E. *Genetic algorithms in search, optimization, and machine learning*. Boston: Addison-Wesley Professional, 432 p., 1989.

GOLDBERG, D. E. *The design of innovation: Lessons from and for competent genetic algorithms*. Norwell: Kluwer Academic Publishers, 272 p., 2002.

HADI, A.; RASHIDI, F. Design of optimal power distribution networks using multiobjective genetic algorithm. In: *Advances in Artificial Intelligence*, New York: Springer, 2005, p. 203–215.

HARIK, G. R.; LOBO, F. G.; GOLDBERG, D. E. The compact genetic algorithm. *IEEE Trans. Evolutionary Computation*, v. 3, n. 4, p. 287–297, 1999.

HARIK, G. R.; LOBO, F. G.; SASTRY, K. Linkage learning via probabilistic modeling in the extended compact genetic algorithm (ECGA). In: *Scalable Optimization via Probabilistic Modeling*, p. 39–61, 2006.

HE, J.; REEVES, C.; WITT, C.; YAO, X. A note on problem difficulty measures in black-box optimization: Classification, realizations and predictability. *Evol. Comput.*, v. 15, n. 4, p. 435–443, 2007.

HOLLAND, J. H. *Adaptation in natural and artificial systems*. Michigan: The University of Michigan Press, 1975.

HOOS, H. H.; STTZLE, T. SATLIB: An online resource for research on SAT. 2000. Disponível em `<http://www.cs.ubc.ca/~hoos/Publ/sat2000-satlib.pdf>` (Acessado em 09 fev. 2011)

HOOS, H. H.; STÜTZLE, T. *Stochastic local search: Foundations & applications*. Burlington: Elsevier: Morgan Kaufmann, 672 p., 2004.

JAEGER, M. Probabilistic decision graphs - combining verification and ai techniques for probabilistic inference. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, v. 12, p. 2004, 2004.

JAIN, A. K.; MURTY, M. N.; FLYNN, P. J. Data clustering: a review. *ACM Comput. Surv.*, v. 31, n. 3, p. 264–323, 1999.
Disponível em `http://doi.acm.org/10.1145/331499.331504`

KRONFELD, M.; ZELL, A. Towards scalability in niching methods. In: *IEEE Congress on Evolutionary Computation*, IEEE, 2010, p. 1–8.
Disponível em <`http://dx.doi.org/10.1109/CEC.2010.5585916`> (Acessado em 09 fev. 2011)

LARRAÑAGA, P.; LOZANO, J. A. *Estimation of distribution algorithms: A new tool for evolutionary computation (genetic algorithms and evolutionary computation)*. New York: Springer, 382 p., 2001.

LI, X.; DEB, K. Comparing lbest PSO niching algorithms using different position update rules. In: *IEEE Congress on Evolutionary Computation*, Barcelona: IEEE, 2010, p. 1–8.

MACQUEEN, J. B. Some methods for classification and analysis of multivariate observations. In: CAM, L. M. L.; NEYMAN, J., eds. *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, University of California Press, 1967, p. 281–297.

MAHFOUD, S. W. *Niching methods for genetic algorithms*. Relatório Técnico 1894, Department of Computer Science, University of Illinois, Urbana, 1995.

MELO, V. V.; DELBEM, A. C. Using smart sampling to discover promising regions and increase the efficiency of differential evolution. *International Conference in Intelligent Systems Design and Applications*, v. 0, p. 1394–1399, 2009.

MELO, V. V.; DELBEM, A. C. B.; PINTO, JUNIOR, D. L.; FEDERSON, F. M. Improving global numerical optimization using a search-space reduction algorithm. In: *Annual conference on Genetic and evolutionary computation*, New York: ACM, 2007, p. 1195–1202.

MELO, V. V.; DUQUE, T. S. P. C.; DELBEM, A. C. B. Efficiency enhancement of ECGA through population size management. In: *Intelligent Systems Design and Applications*, Pisa: IEEE Computer Society, 2009, p. 19–24.

MELO, V. V.; VARGAS, D. V.; CROCOMO, M. K.; DELBEM, A. C. B. Phylogenetic differential evolution. *International Journal of Natural Computing Research*, v. 2, n. 1, p. 21–38, 2011.

MOURA, A.; RIJO, R.; SILVA, P.; CRESPO, S. A multi-objective genetic algorithm applied to autonomous underwater vehicles for sewage outfall plume dispersion observations. *Applied Software Computing*, v. 10, n. 4, p. 1119–1126, 2010.

MÜHLENBEIN, H. The equation for response to selection and its use for prediction. *Evolutionary Computation*, v. 5, n. 3, p. 303–346, 1997.

NEWMAN, M. E. J. The structure of scientific collaboration networks. 2001. Disponível em <http://www.pubmedcentral.gov/articlerender.fcgi?artid=14598> (Acessado em 09 fev. 2011)

NEWMAN, M. E. J. Fast algorithm for detecting community structure in networks. *Physical Review E*, v. 69, n. 6, p. 066133+, 2004. Disponível em http://dx.doi.org/10.1103/PhysRevE.69.066133

NEWMAN, M. E. J.; GIRVAN, M. Finding and evaluating community structure in networks. *Physical Review E*, v. 69, n. 2, p. 026113+, 2004. Disponível em <http://dx.doi.org/10.1103/PhysRevE.69.026113> (Acessado em 09 fev. 2011)

NIEDERMAYER, D. An introduction to bayesian networks. 2009. Disponível em <http://www.niedermayer.ca/papers/bayesian/index.html> (Acessado em 09 fev. 2011)

OGATA, A. K. O. *Multialinhamento de seqüências biológicas utilizando algoritmos genéticos*. Dissertação de Mestrado, Universidade de São Paulo, São Carlos, 2006.

PEARL, J. *Probabilistic reasoning in intelligent systems:networks of plausible inference*. Burlington: Morgan Kaufmann, 552 p., 1988.

PELIKAN, M. Bayesian optimization algorithm (BOA) code in C++. 1999. Disponível em http://illigal.org/1999/03/24/bayesian-optimization-algorithm-boa-code-in-c/ (Acessado em 02 oct. 2012)

PELIKAN, M. *Hierarchical bayesian optimization algorithm: Toward a new generation of evolutionary algorithms*. Studies in Fuzziness and Soft Computing, 1 ed. New York: Springer, 2005.

PELIKAN, M.; GOLDBERG, D. E. Escaping hierarchical traps with competent genetic algorithms. In: *Genetic and Evolutionary Computation Conference (GECCO2001)*, San Francisco: Morgan Kaufmann, 2001, p. 511–518.

PELIKAN, M.; GOLDBERG, D. E. Hierarchical boa solves ising spin glasses and MAXSAT. In: *Proceedings of the 2003 international conference on Genetic and evolutionary computation: PartII*, Berlin: Springer-Verlag, 2003, p. 1271–1282. Disponível em <http://portal.acm.org/citation.cfm?id=1756582.1756586> (Acessado em 09 fev. 2011)

PELIKAN, M.; GOLDBERG, D. E.; CANTÚ-PAZ, E. Boa: The bayesian optimization algorithm. In: *Conference on Genetic and Evolutionary Computation*, Orlando: Morgan Kaufmann, 1999, p. 525–532.

PELIKAN, M.; GOLDBERG, D. E.; LOBO, F. G. A survey of optimization by building and using probabilistic models. *Comput. Optim. Appl.*, v. 21, n. 1, p. 5–20, 2002.

PELIKAN, M.; SASTRY, K.; GOLDBERG, D. E. Multiobjective hboa, clustering, and scalability. In: *Conference on Genetic and evolutionary computation*, New York: ACM, 2005, p. 663–670.

PESSIN, G.; OSÓRIO, F. S.; WOLF, D. F.; BRASIL, C. R. S. Improving efficiency of a genetic algorithm applied to multi-robot tactic operation. In: MORALES, Á. F. K.; SIMARI, G. R., eds. *Advances in Artificial Intelligence, 12th Ibero-American Conference on AI, Bahía Blanca, Argentina, November 1-5, 2010. Proceedings*, Springer, 2010, p. 50–59 (*Lecture Notes in Computer Science*, v.6433).

PUJOL, J. M.; BÉJAR, J.; DELGADO, J. Clustering algorithm for determining community structure in large networks. *Physical Review*, v. 74, p. 016107, 2006.
Disponível em `http://www.biomedsearch.com/nih/Clustering-algorithm-determining-community-structure/16907151.html`

QU, B.-Y.; SUGANTHAN, P. N. Novel multimodal problems and differential evolution with ensemble of restricted tournament selection. In: *IEEE Congress on Evolutionary Computation*, Los Alamitos: IEEE, 2010, p. 1–7.

RAJAGOPALAN, R.; MOHAN, C. K.; MEHROTRA, K. G.; VARSHNEY, P. K. Multi-Objective Evolutionary Algorithms for Sensor Network Design. In: BUI, L. T.; ALAM, S., eds. *Multi-Objective Optimization in Computational Intelligence: Theory and Practice*, Hershey: Information Science Reference, p. 208–238, 2008.

RANA, S.; WHITLEY, D. Genetic algorithm behavior in the MAXSAT domain. Berlin: Springer, 1998, p. 785–794.

REICHARDT, J.; BORNHOLDT, S. Detecting Fuzzy Community Structures in Complex Networks with a Potts Model. *Physical Review Letters*, v. 93, n. 21, p. 218701+, 2004.
Disponível em `http://dx.doi.org/10.1103/PhysRevLett.93.218701`

SAITOU, N.; NEI, M. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Mol Biol Evol*, v. 4, n. 4, p. 406–425, 1987.
Disponível em `<http://mbe.oxfordjournals.org/content/4/4/406.abstract>` (Acessado em 09 fev. 2011)

SANTOS, A. C.; DELBEM, A. C. B.; BRETAS, N. G. Energy restoration for large-scale distribution system using ea and a new data structure. In: *Power and Energy Society General Meeting - Conversion and Delivery of Electrical Energy in the 21st Century, 2008 IEEE*, 2008, p. 1 –8.

SANTOS, A. C.; DELBEM, A. C. B.; LONDON, J. B. A.; BRETAS, N. G. Node-Depth Encoding and Multiobjective Evolutionary Algorithm Applied to Large-Scale Distribution System Reconfiguration. *IEEE Transactions on Power Systems*, 2010.
Disponível em `http://dx.doi.org/10.1109/TPWRS.2010.2041475`

SANTOS, E. B. *A ordenação das variáveis no processo de otimização de classificadores bayesianos: Uma abordagem evolutiva*. Dissertação de Mestrado, Universidade Federal de São Carlos, São Carlos, 2007.

SASTRY, K. *Evaluation-relaxation schemes for genetic and evolutionary algorithms*. Dissertação de Mestrado, University of Illinois at Urbana-Champaign, Department of General Engineering, Urbana, IL, 2001.

SIMÕES, E. V.; BARONE, D. A. C. Predation: An approach to improving the evolution of real robots with a distributed evolutionary controller. In: *International Conference on Robotics and Automation*, IEEE, 2002, p. 664–669.

SIPSER, M. *Introdução à teoria da computação*. São Paulo: Cengage Learning, 459 p., 2007.

STORN, R.; PRICE, K. Differential evolution, a simple and efficient heuristic for global optimization over continuous spaces. *J. of Global Optimization*, v. 11, n. 4, p. 341–359, 1997.

SUH, N. *The principles of design*, v. 7. Oxford University Press New York, 1990.

TALBI, E.-G. *Metaheuristics : from design to implementation*, v. 10 de *The Sciences Po series in international relations and political economy*. San Francisco: John Wiley & Sons, 1–6 p., 2009.

TEO, J.; NERI, L. D.; NGUYEN, M. H.; ABBASS, H. A. Walking with EMO: Multi-Objective Robotics for Evolving Two, Four, and Six–Legged Locomotion. In: BUI, L. T.; ALAM, S., eds. *Multi-Objective Optimization in Computational Intelligence: Theory and Practice*, Hershey: Information Science Reference, p. 300–332, 2008.

VAN VELDHUIZEN, D. A. *Multiobjective evolutionary algorithms: classifications, analyses, and new innovations*. Tese de Doutorado, Air Force Institute of Technology, Wright Patterson AFB, OH, USA, aAI9928483, 1999.

VARGAS, D. V.; DELBEM, A. C. B. *Algoritmo filogenético*. Relatório Técnico 350, Universidade de São Paulo, 2009.
Disponível em `http://www.icmc.usp.br/~biblio/relatorios_tecnicos.php` (Acessado em 10 set. 2012)

WOLPERT, D. H.; MACREADY, W. G. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, v. 1, n. 1, p. 67–82, 1997.

YE, Z.; HU, S.; YU, J. Adaptive clustering algorithm for community detection in complex networks. *Physical Review E (Statistical, Nonlinear, and Soft Matter Physics)*, v. 78, n. 4, p. 046115+, 2008.

YONG, Z.; SANNOMIYA, N. A method for solving large-scale flowshop problems by reducing search space of genetic algorithms. In: *Conference on Systems, Man, and Cybernetics*, Los Alamitos: IEEE Computer Society, 2000, p. 1776 – 1781.

ZACHARY, W. W. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, v. 33, p. 452–473, 1977.

ZYDALLIS, J. B. *Explicit building-block multiobjective genetic algorithms: theory, analysis, and development*. Tese de Doutorado, Air Force Institute of Technology, aAI3077559, 2003.

APÊNDICE

A

Cálculo do Tamanho da População

Neste apêndice é apresentada a dedução da Equação 2.2, baseada no trabalho apresentado em (Goldberg, 2002), que calcula o tamanho da população necessário para que cada instância de um BB seja representada pelo menos uma vez na população. Como exemplo, considere uma solução de 4 *bits* em que os dois primeiros *bits* constituem um BB. É preciso então determinar o tamanho n da população inicial de forma que esta possua pelo menos uma de cada possível instância desta partição. Mais formalmente, a população deve conter os seguintes esquemas (Seção 2.3): 00**, 01**, 10** e 11**. Sendo * a representação de um valor qualquer para o *bit*.

A probabilidade p_k de que uma instância dessa partição ocorra uma ou mais vezes, é dada pela Equação A.1

$$p_k = 1 - \left[1 - \frac{1}{\chi^k}\right]^n, \tag{A.1}$$

em que χ é a cardinalidade do alfabeto (no exemplo anterior, $\chi = \{0, 1\}$), k é o tamanho do BB (no mesmo exemplo, $k = 2$), e n é o tamanho da população. Tal probabilidade pode ser compreendida como 1 menos a chance de que a instância não seja representada nenhuma vez na população, dada por $\left[1 - \frac{1}{\chi^k}\right]^n$. Substituindo $\frac{n}{\chi^k}$ por r, tem-se:

$$p_k = 1 - \left[1 - \frac{n}{n\chi^k}\right]^n = 1 - \left[1 - \frac{r}{n}\right]^n, \tag{A.2}$$

Utilizando a aproximação (1 - $\frac{p}{q}$)$^q \approx e^{-p}$, válida para $p << q$. A Equação A.2 é reescrita como:

$$p_k = 1 - e^{-\frac{n}{\chi^k}}.$$

Fazendo $n_k = \chi^k$, obtém-se a Equação A.3:

$$p_k = 1 - e^{-\frac{n}{n_k}}. \tag{A.3}$$

Ao considerar uma inicialização aleatória das soluções em que as ocorrências de esquemas, representando BBs de mesmo tamanho, possuam a mesma taxa de sucesso, a probabilidade de pelo menos uma ocorrência de cada um dos possíveis esquemas na população inicial é dada por $p_s = p_k^{n_k}$ (Goldberg, 2002), em que n_k é o número de possíveis instâncias do BB. Substituindo p_k pela expressão da Equação (A.3), é obtida a Equação A.4.

$$p_s = (1 - e^{-\frac{n}{n_k}})^{n_k}. \tag{A.4}$$

Utilizando novamente a aproximação (1 - $\frac{p}{q}$)$^q \approx e^{-p}$, tem-se que:

$$p_s = \left(1 - \frac{n_k e^{-\frac{n}{n_k}}}{n_k}\right)^{n_k} = \left(1 - \frac{s}{n_k}\right)^{n_k},$$

em que $s = n_k e^{-\frac{n}{n_k}}$. Com isso, a expressão para p_s resulta em:

$$p_s = e^{-n_k e^{-\frac{n}{n_k}}}. \tag{A.5}$$

Para estimar então o tamanho de uma população que amostre adequadamente todas as instâncias de um BB com uma dada probabilidade p_s, é necessário isolar n na Equação A.5.

Seja α a chance de que um BB não tenha todas as suas instâncias representadas. Então, $p_s = 1 - \alpha$ e, portanto,

$$1 - \alpha = e^{-n_k e^{\frac{n}{n_k}}},$$

assim,

$$\ln(1 - \alpha) = -n_k e^{-\frac{n}{n_k}}.$$

Sendo $\ln(1 - x) \approx -x$, para pequenos valores de x,

$$-\alpha = -n_k e^{-\frac{n}{n_k}},$$

então,

$$\ln\left(\frac{\alpha}{n_k}\right) = -\frac{n}{n_k},$$

$$\ln \alpha - \ln n_k = -\frac{n}{n_k},$$

portanto,

$$n = n_k(\ln n_k - \ln \alpha),$$

Substituindo n_k por χ^k,

$$n = \chi^k(k \ln \chi - \ln \alpha). \tag{A.6}$$

Admitindo como tolerável o erro de um único BB, isto é, no máximo um BB não estar representado pelas soluções da população inicial, adota-se $\alpha = \frac{1}{m}$. Reescrevendo a Equação A.6, obtém-se:

$$n = \chi^k(k \ln \chi - \ln m^{-1}). \tag{A.7}$$

Como $\ln m^{-1} = -\ln m$, a Equação A.7 resulta na Equação 2.2, isto é:

$$n = \chi^k(k \ln\ \chi + \ln\ m).$$

APÊNDICE

B

Cálculo Empírico do Tamanho da População

A Figura 2.5, replicada na Figura B.1 deste apêndice, apresenta linhas contínuas que correspondem a valores de tamanhos de população calculados pela Equação 2.2 utilizando os valores de m e k indicados na própria Figura B.1. Para cada par de valores de k e m é determinado um tamanho da população n. O Algoritmo B.1 é executado 30 vezes. Os valores de n mostrados na Figura B.1 correspondem a média dos valores de n obtidos nas 30 execuções.

Algoritmo B.1: Otimização Direta Multicritério.

1 Crie uma população inicial com 0 indivíduo;
2 Enquanto houverem instancias não representadas de algum BB na população, inclua um novo individuo aleatório;
3 Retorne o tamanho da população final (que contém todas as instâncias de cada BB).

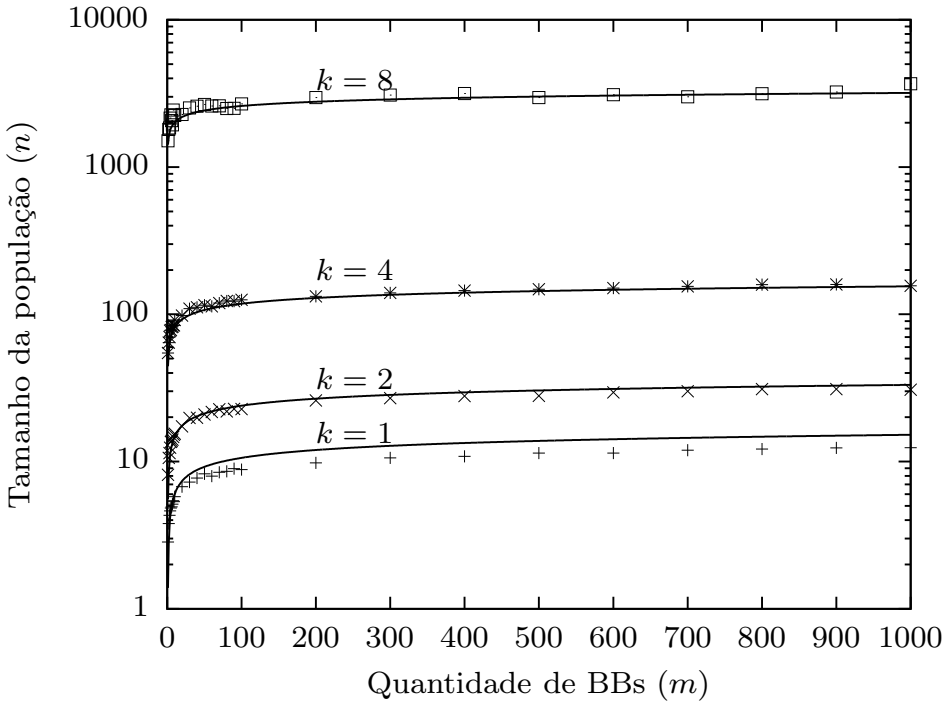


Figura B.1: Tamanho da população inicial, n, necessária para que exista pelo uma instância de cada possível BB: curvas teóricas (Apêndice A) e resultados experimentais

.

APÊNDICE

C

Problemas Multiobjetivo

Na resolução de problemas multiobjetivo, as soluções são avaliadas de acordo com múltiplas funções objetivo, as quais podem envolver objetivos conflitantes e difíceis de serem ponderados por meio de um conhecimento a priori. Por exemplo:

- Minimização do custo de um produto e maximização do lucro;
- Maximização do desempenho de um motor e minimização do consumo de combustível.

Nesses problemas, considera-se um vetor de funções objetivo $f(X) = (f_1(X), f_2(X), ..., f_{N_f}(X))$, em que N_f é o número total de funções. Por meio de $f(X)$, cada possível solução X é mapeada em um espaço multidimensional de objetivos S_{obj}, no qual as soluções podem ser comparadas apropriadamente. Esse fato mostra a principal dificuldade de solucionar tais problemas quando comparados a problemas mono-objetivo, que são mapeados em espaço de objetivos unidimensional. Devido a essa diferença, EAs multiobjetivos têm sido propostos nas últimas décadas, possibilitando a obtenção de melhores soluções para uma diversidade de problemas do mundo real (Abraham *et al.*, 2005; Pelikan *et al.*, 2005).

Em problemas com um único objetivo, encontram-se normalmente um ou poucos ótimos globais bem definidos. No entanto, em problemas de otimização multiobjetivo, existem conjuntos de soluções de qualidade equivalente, que oferecem diferente *trade-off* (compromisso) em relação a cada um dos objetivos sendo otimizados. O conjunto de soluções com compromisso ótimo é chamado de conjunto Pareto-ótimo (Abraham *et al.*, 2005) e uma solução desse conjunto é chamada Pareto-ótima. Normalmente, o objetivo de um algoritmo de otimização multiobjetivo é simplesmente encontrar soluções que se aproximem do conjunto Pareto-ótimo, dada a dificuldade computacional de tais problemas.

Para melhor esclarecer o que é o conjunto Pareto-ótimo, considere duas soluções: X_1 e X_2. É dito que X_1 domina X_2 se duas condições são satisfeitas: i) $f_i(X_1)$ é pelo menos igual a

$f_i(X_2)$ para cada função i sendo otimizada, e $ii)$ $f_i(X_1)$ é superior a $f_i(X_2)$ em pelo menos uma dessas funções. É chamado conjunto Pareto-ótimo, o conjunto de soluções não dominadas do problema. A Figura C.1 mostra a Fronteira de Pareto em um problema de minimização de custo de um produto e maximização do lucro [1]. Tal fronteira é formada pelos valores das funções objetivo das soluções contidas no conjunto Pareto-ótimo. Cada ponto no gráfico representa um vetor $f(X) = (f_1(X), f_2(X))$ obtido para uma dada solução X.

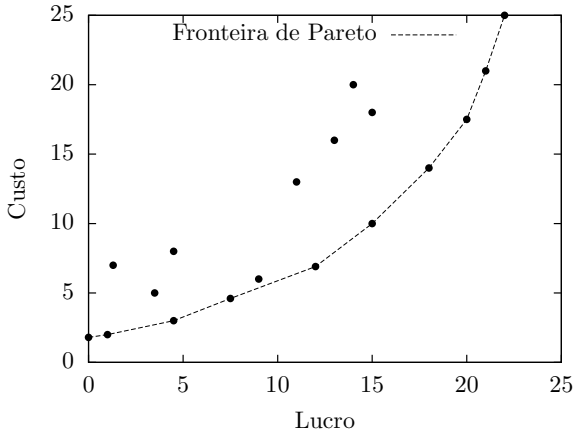


Figura C.1: Exemplo que ilustra várias opções de custo e lucro obtidas pela venda de um determinado produto.

Em (Pelikan *et al.*, 2005), dois ASDP (Seção 5.4) multiobjetivos foram propostos para avaliar EDAs multiobjetivos. O primeiro, composto pela função UmMax (ver Seção 2.4) e pela função ZeroMax, que é similar à função anterior, mas que retorna a soma do número de zeros ao invés de uns. O gráfico que mostra essas duas funções encontra-se na Figura C.2. Qualquer modificação no solução (*string* binária) resulta em um incremento na pontuação de uma função e igual decremento na outra função. Devido a esse fato, todas as soluções desse problema encontram-se no conjunto Pareto-ótimo. O segundo teste proposto, resulta da combinação das funções $ftrap5$ (Equação 5.5) e da $ftrap5$ inversa ($invftrap5$), dada pela Equação C.1. Essas funções são ilustradas na Figura C.2.

$$invftrap5(u) = \begin{cases} u-1 & se\ u > 0 \\ 5 & caso\ contrário, \end{cases} \tag{C.1}$$

em que u é o número de uns.

A Figura C.3 ilustra os possíveis valores de *fitness* para uma solução de tamanho 100 sendo avaliado pela função $ftrap5$ e $invftrap5$, os pontos circulados são os que formam o conjunto Pareto-ótimo.

[1] Dados meramente ilustrativos.

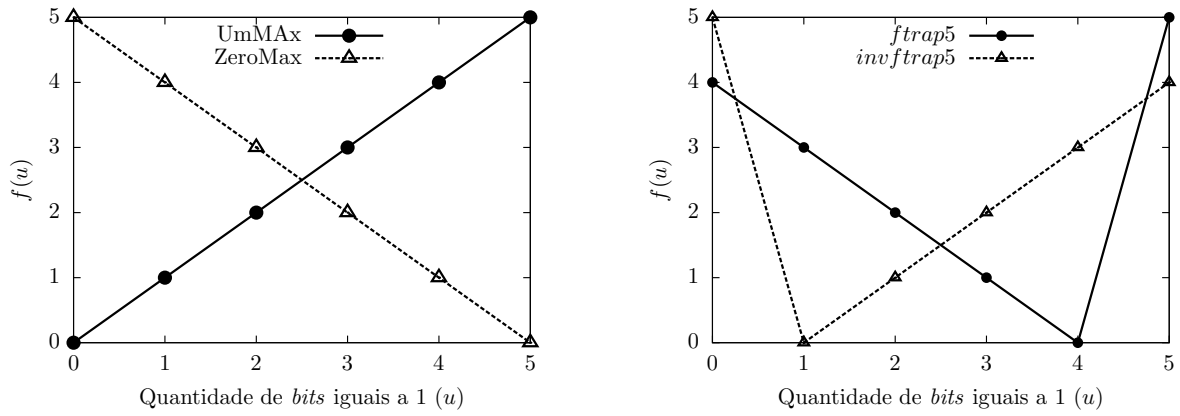


Figura C.2: Problemas multiobjetivos linearmente separáveis: (a) funções UmMax e ZeroMax, (b) funções $ftrap5$ e $invftrap5$.

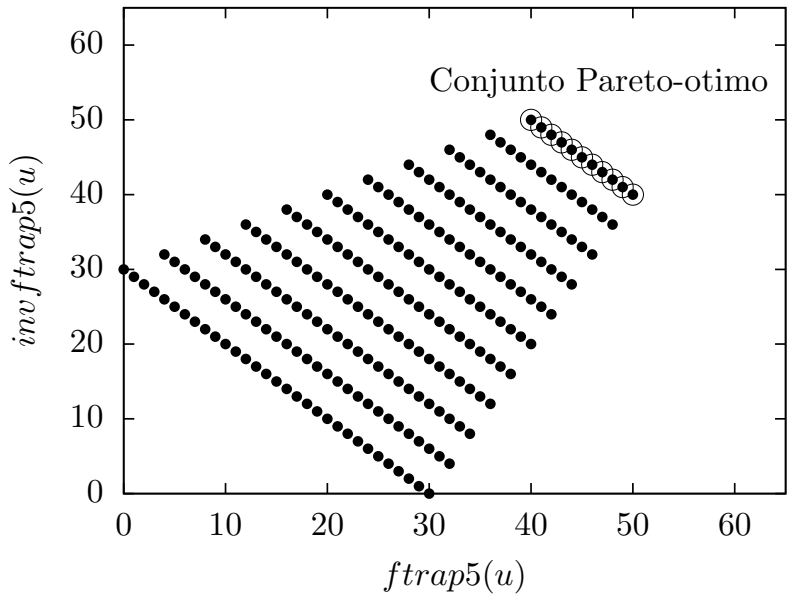


Figura C.3: Valores para soluções existentes de 50 *bits*, utilizando função $ftrap5$ e $invftrap5$. Conjunto Pareto-ótimo representado pelos pontos circulados.

APÊNDICE

D

Experimentos com a versão do CD-BOA utilizando adClust

Este apêndice relata um dos experimentos preliminares feitos com a versão do algoritmo CD-BOA utilizando o algoritmo de detecção de comunidades AdClust, e com uma versão utilizando o algoritmo. Foi repetido o **Experimento 1** relatado na Seção 6.2. As Figuras D.1 e Figuras D.2 mostram os resultados, que indicam um desempenho próximo, porém inferior da versão do algoritmo que utiliza a técnica adClust. Devido ao fato do FA possuir menor complexidade computacional, o mesmo foi escolhido como algoritmo padrão a ser utilizado no restante dos experimentos.

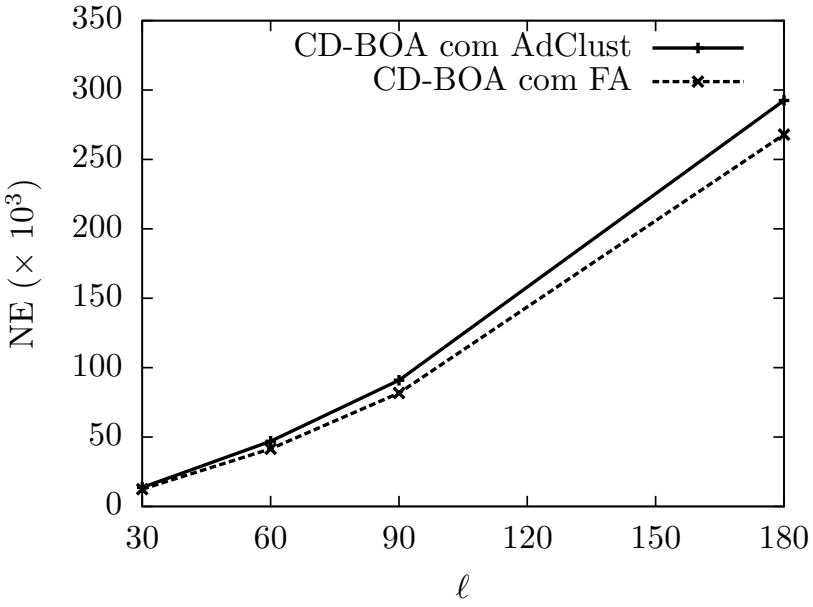


Figura D.1: Comparação entre NA do CD-BOA com FA e CD-BOA com adClust (ambos com $v_l = 2$ e $v_u = 4$).

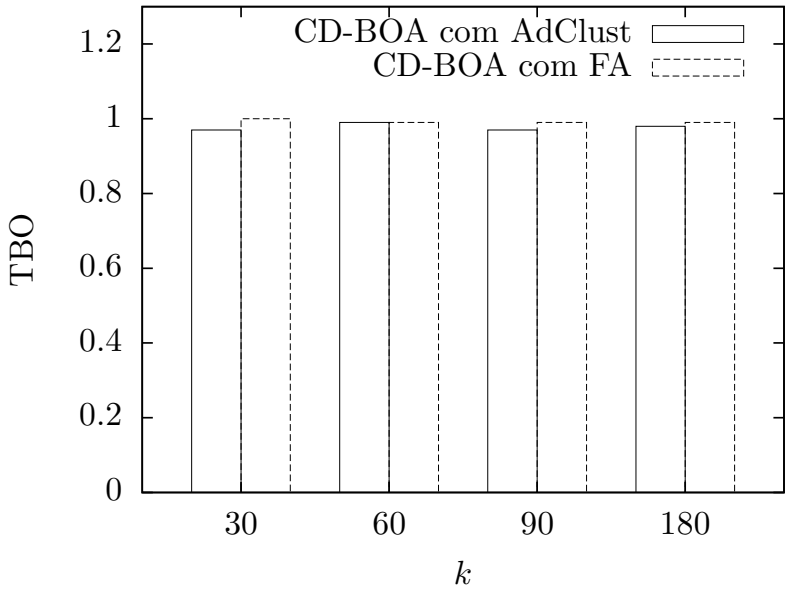


Figura D.2: Comparação entre TBO do CD-BOA com FA e CD-BOA com adClust (ambos com $v_l = 2$ e $v_u = 4$).

APÊNDICE

E

Tamanho do torneio utilizado

O tamanho do torneio utilizado pelo REDA 4.3 nos experimentos deste trabalho, foi definido através de uma comparação entre o REDA utilizando torneio de 2, 4, 8 e 16 indivíduos, aplicado ao CD-BOA em um problema com $\ell = 90$, $n = 5200$, composto por funções $ftrap5$. A Figura E.1 mostra os resultados, em que cada ponto foi obtido pela média de 30 experimentos. Tais resultados mostram que o menor NA médio foi obtido pelo algoritmo utilizando torneio de 8 indivíduos.

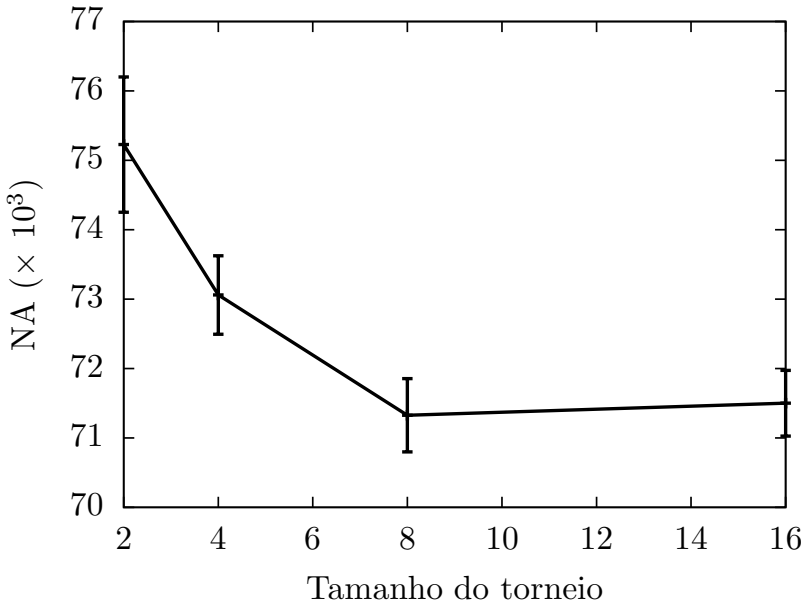


Figura E.1: NA do CD-BOA+REDA, utilizando torneio de 2, 4, 6 e 8 indivíduos para o problema com $\ell = 90$ composto por funções $ftrap5$.

APÊNDICE

F

Atualização da matriz de caminhos

O Algoritmo F.1 apresenta o pseudocódigo da atualização da matriz de caminhos do BOA (Seção 4.1). Esse código foi adaptado do código disponível em (Pelikan, 1999). A atualização dessa matriz é importante para se detectar em tempo $O(1)$ a presença de ciclos em um grafo, no caso, uma rede Bayesiana.

Algoritmo F.1: Pseudocódigo do procedimento de atualização da matriz de caminhos, utilizada para detectar ciclos em grafos.

Entrada: Número de vértices (ℓ) na rede;
Entrada: O par ordenado de vértices (i, j) que representa a nova aresta adicionada;
Entrada: A matriz de caminhos $path$. Se $path[a][b] == 1$ existe um caminho conectando o vértice a ao vértice b;
Saída: Uma matriz de caminhos atualizada.

```
1 para (a = 0; a < ℓ; a++) faça
2     se (path[a][i] == 1) então
3         para (b = 0; b < ℓ; b++) faça
4             se ((b != a) and (path[j][b] == 1)) então
5                 path[a][b] = 1;
6             fim
7         fim
8     fim
9 fim
```

APÊNDICE

G

Considerações relativas à complexidade do BOA

A complexidade de tempo da construção da rede CD-BOA possui dois termos, um quadrático e outro cúbico com relação a ℓ, como mostra a Equação G.1 (determinada na Seção 6.4.1):

$$C_{CD-BOA} = O(v_l 2^{v_l} \ell^2 N + (v_l + v_u)\ell^3). \tag{G.1}$$

Na sequência, verificam-se as situações em que cada um destes termos é o dominante, buscando inicialmente a situação em que os termos se equivalem.

$$\begin{aligned} v_l 2^{v_l} \ell^2 N &= (v_l + v_u)\ell^3 \\ v_l 2^{v_l} \ell^2 N &= v_l \ell^3 + v_u \ell^3 \\ v_l 2^{v_l} N &= v_l \ell + v_u \ell \\ v_u &= \frac{v_l 2^{v_l} N - v_l \ell}{\ell} \\ v_u &= \frac{v_l (2^{v_l} N - \ell)}{\ell}. \end{aligned} \tag{G.2}$$

Com base na Equação G.2, conclui-se que somente com v_u exponencial em relação a v_l, a contribuição do termo cúbico $(v_l + v_u)\ell^3)$ poderá ser assintoticamente maior que a do termo quadrático $(v_l 2^{v_l} \ell^2 N)$ para a complexidade do CD-BOA (C_{CD-BOA}). Em outras palavras, admitindo um limitante para v_u, dado por:

$$v_u < \frac{v_l (2^{v_l} N - \ell)}{\ell}, \tag{G.3}$$

então, C_{CD_BOA} (Equação G.1) pode ser reescrita como:

$$C_{CD-BOA} = O(v_l 2^{v_l} \ell^2 N). \tag{G.4}$$

Considerando a Equação G.3 e a C_{BOA} (Equação G.4), pode-se afirmar que se forem satisfeitas as condições $v_l < v$, e $v_u < \frac{v_l(2^{v_l}N-\ell)}{\ell}$, então $C_{CD-BOA} < C_{BOA}$.

$$C_{CD-BOA} = O(v_l 2^{v_l} \ell^2 N). \tag{G.5}$$

Com a finalidade de verificar se a Inequação G.3 define limitantes aceitáveis para os valores de v_u, os mesmos foram calculados a partir dos parâmetros utilizados nos testes realizados no **Experimento 1** da Seção 6.4.1, variando ℓ de 30 a 180, com $v_l = 2$, e N variando de 650 a 5900. Os valores calculados encontram-se na Tabela G.1.

Tabela G.1: Limitantes calculados para v_u utilizado no CD-BOA.

ℓ	N	v_u máximo
30	650	171
60	1 650	218
90	2 600	229
120	3 650	241
150	4 800	254
180	5 900	260

Observa-se que os limitantes calculados para v_u são muito grandes, uma vez que o número máximo de pais por variável possível em uma rede de tamanho ℓ é dado por $\ell - 1$, e todos os limitantes encontrados (Tabela G.1) encontram-se acima deste valor. Adicionalmente, o maior valor de v_u utilizado na Seção 6.4.1 foi $v_u = 10$ e, embora muito inferior aos limitantes apresentados, se mostrou capaz de resolver todos os casos apresentados de maneira eficaz. Assim, os resultados experimentais da Seção 6.4.1 são reforçados pelas análises realizadas neste Apêndice.

APÊNDICE

H

Considerações relativas à complexidade do StrOp

A complexidade da construção do modelo e da busca gulosa do StrOp é dada pela Equação H.1, calculada na Seção 6.4.2.

$$C_{StrOp} = O(v_l 2^{v_l} \ell^2 N + v_l \ell^3 + \frac{\ell}{v_u + 1} 2^{v_u+1}). \tag{H.1}$$

Deseja-se determinar uma condição que garanta que o termo $\frac{\ell}{v_u+1}2^{v_u+1}$ não seja dominante na complexidade do StrOp, ou seja:

$$\begin{aligned}
\frac{\ell}{v_u + 1} 2^{v_u+1} &< v_l 2^{v_l} \ell^2 N + v_l \ell^3 \\
\frac{1}{v_u + 1} 2^{v_u+1} &< v_l 2^{v_l} \ell N + v_l \ell^2 \\
\log_2(\frac{2^{v_u+1}}{v_u + 1}) &< \log_2(v_l 2^{v_l} \ell N + v_l \ell^2) \\
\log_2(2^{v_u+1}) - \log_2(v_u + 1) &< \log_2(v_l 2^{v_l} \ell N + v_l \ell^2) \\
v_u + 1 - \log_2(v_u + 1) &< \log_2(v_l 2^{v_l} \ell N + v_l \ell^2).
\end{aligned}$$

Assim, para que o termo $\frac{\ell}{v_u+1}2^{v_u+1}$ não seja dominante em C_{StrOp}, a condição apresentada em H.2 deve ser satisfeita:

$$v_u - \log_2(v_u + 1) < \log_2(v_l 2^{v_l} \ell N + v_l \ell^2) - 1. \tag{H.2}$$

Caso a condição H.3 seja satisfeita, garante-se que H.2 também é satisfeita, uma vez que $v_u - \log_2(v_u + 1) < v_u$. Assim, H.3 apresenta um limitante superior para v_u que garante que o termo que cresce exponencialmente com v_u na complexidade do StrOp não seja o termo dominante.

$$v_u < \log_2(v_l 2^{v_l} \ell N + v_l \ell^2) - 1. \tag{H.3}$$

O limitante para v_u foi calculado para todos os casos testados no **Experimento 2** da Seção 6.4, referente ao tempo de execução do algoritmo StrOp para os problemas compostos por armadilhas do tipo $ftrap5$, variando ℓ de 30 a 180, com $v_l = 2$, e N variando de $6\,500$ a $59\,000$. Os valores calculados encontram-se na Tabela H.1.

Tabela H.1: Limitantes calculados para v_u utilizado no StrOp.

ℓ	N	v_u máximo
30	6 500	19
60	16 500	21
90	26 000	23
120	36 500	24
150	48 000	24
180	59 000	25

É possível notar que todos os limitantes encontrados estão acima do valor de $v_u = 10$ utilizado nos experimentos. O menor limitante calculado (19, para $\ell = 30$) está significativamente acima do tamanho dos BBs de casos resolvidos na literatura (Harik *et al.*, 2006; Melo *et al.*, 2011; Pelikan *et al.*, 1999). Como exemplo, uma das características do EDA PhyDE (Seção 2.5) é a capacidade de resolver problemas com blocos de tamanhos considerados grandes, para mostrar tal característica em (Melo *et al.*, 2011), são utilizados problemas compostos por blocos de tamanho 8.

Dessa forma, para os problemas enfrentados na literatura de EDAs (e possivelmente de otimização em geral em que haja BBs evidenciados), os valores necessários de v_u são significativamente menores que os da Tabela H.1. Portanto, o termo $\frac{\ell}{v_u+1}2^{v_u+1}$ não é o dominante na complexidade do StrOp para tais problemas. Com isso, é possível reescrever C_{StrOp} como segue:

$$C_{StrOp} = O(v_l 2^{v_l} \ell^2 N + v_l \ell^3). \tag{H.4}$$

Considerando a Equação H.4 e que $v_l < v$, obtém-se que $C_{StrOp} < C_{BOA}$. Esse resultado é corroborado pelos resultados do **Experimento 2** da Seção 6.4.