UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO
DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO E AUTOMAÇÃO INDUSTRIAL

ALGORITMOS BIO-INSPIRADOS APLICADOS À OTIMIZAÇÃO DINÂMICA

Autor: Fabrício Olivetti de França
Orientador: Prof. Dr. Fernando José Von Zuben
Co-orientador: Prof. Dr. Leandro Nunes de Castro

Dissertação de Mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos exigidos para a obtenção do título de Mestre em Engenharia Elétrica.
Área de Concentração: Engenharia de Computação

Comissão Examinadora

Fernando José Von Zuben, Dr. .....DCA/FEEC/Unicamp
Wagner Caradori do Amaral, Dr. .....DCA/FEEC/Unicamp
Romis Ribeiro de Faissol Attux, Dr. ..... DECOM /FEEC/Unicamp
Eduardo Raul Hruschka, Dr. .....UNISANTOS/SP

Campinas – SP – Brasil

Dezembro de 2005

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA E ARQUITETURA - BAE - UNICAMP

França, Fabrício Olivetti de
F844a Algoritmos bio-inspirados aplicados à otimização dinâmica / Fabrício Olivetti de França. --Campinas, SP: [s.n.], 2005.

Orientadores: Fernando José Von Zuben, Leandro Nunes de Castro
Dissertação (Mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Pesquisa operacional. 2. Otimização matemática. 3. Programação não-linear. 4. Otimização combinatória. 3. Computação bio-inspirada. I. Von Zuben, Fernando José. II. Castro, Leandro Nunes de. III. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. IV. Título.

Título em Inglês: Bio-inspired algorithms applied to dynamic optimization
Palavras-chave em Inglês: Dynamic optimization, Nonlinear optimization, Combinatorial optimization, Artificial immune systems, Ant colony optimization, Biologically-inspired computing
Área de concentração: Engenharia de Computação
Titulação: Mestre em Engenharia Elétrica
Banca examinadora: Wagner Caradori do Amaral, Romis Ribeiro de Faissol Attux e Eduardo Raul Hruschka
Data da defesa: 01/12/2005
Programa de Pós-Graduação: Engenharia Elétrica

COMISSÃO JULGADORA – DISSERTAÇÃO DE MESTRADO

Candidato: Fabrício Olivetti de França

Data da Defesa: 01 de dezembro de 2005

Título da Tese: "Algoritmos bio-inspirados aplicados à otimização dinâmica"

Prof. Dr. Fernando José Von Zuben (Presidente): Fernando José Von Zuben
Prof. Dr. Eduardo Raul Hruschka: Eduardo R. Hruschka 310198
Prof. Dr. Romis Ribeiro de Faissol Attux: 291777
Prof. Dr. Wagner Caradori do Amaral: 46663

Agradecimentos

Ao Prof. Dr. Fernando José Von Zuben pelas aulas, ensinamentos, desafios e orientação durante esses anos de pós-graduação.
Ao Prof. Dr. Leandro Nunes de Castro pelo grande auxílio em minhas pesquisas.
Aos membros da banca pelas contribuições e comentários que levaram ao amadurecimento da tese.
Ao Prof. Santo Scuderi pelo estímulo durante a graduação para ingressar na área de pesquisa acadêmica.
Ao Prof. André Luiz Vizine Pereira por me apresentar ao programa de mestrado da Unicamp e me orientar durante o período de graduação.
Aos meus pais por todo o apoio à escolha do caminho que resolvi trilhar.
À Faculdade de Engenharia Elétrica e de Computação e ao Departamento de
Engenharia de Computação e Automação Industrial pelo fornecimento de instalações e condições de trabalho apropriadas.
À CAPES pelo apoio financeiro através da concessão de bolsa de mestrado.

Resumo

de França, F. O., **Algoritmos Bio-Inspirados aplicados à Otimização Dinâmica.** Campinas: FEEC/Unicamp, Dezembro de 2005. Dissertação de Mestrado – Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas.

Esta dissertação propõe algoritmos bio-inspirados para a solução de problemas de otimização dinâmica, ou seja, problemas em que a superfície de otimização no espaço de busca sofre variações diversas ao longo do tempo. Com a variação, no tempo, de número, posição e qualidade dos ótimos locais, as técnicas de programação matemática tendem a apresentar uma acentuada degradação de desempenho, pois geralmente foram concebidas para tratar do caso estático. Algoritmos populacionais, controle dinâmico do número de indivíduos na população, estratégias de busca local e uso eficaz de memória são requisitos desejados para o sucesso da otimização dinâmica, sendo contemplados nas propostas de solução implementadas nesta dissertação. Os algoritmos a serem apresentados e comparados com alternativas competitivas presentes na literatura são baseados em funcionalidades e estruturas de processamento de sistemas imunológicos e de colônias de formigas. Pelo fato de considerarem todos os requisitos para uma busca eficaz em ambientes dinâmicos, o desempenho dos algoritmos imuno-inspirados se mostrou superior em todos os critérios considerados para comparação dos resultados dos experimentos.

Palavras-chave: otimização dinâmica, otimização não-linear, otimização combinatória, algoritmos bio-inspirados, sistemas imunológicos artificiais, otimização por colônia de formigas.

Abstract

de França, F. O., **Bio-Inspired Algorithms applied to Dynamic Optimization.** Campinas: FEEC/Unicamp, December 2005. Master Thesis – School of Electrical and Computer Engineering, University of Campinas.

This dissertation proposes bio-inspired algorithms to solve dynamic optimization problems, i.e., problems for which the optimization surface on the search space suffers several changes over time. With such variation of number, position and quality of local optima, mathematical programming techniques may present degradation of performance, because they were usually conceived to deal with static problems. Population-based algorithms, dynamic control of the population size, local search strategies and an efficient memory usage are desirable requirements to a proper treatment of dynamic optimization problems, thus being incorporated into the solution strategies implemented here. The algorithms to be presented, and compared with competitive alternatives available in the literature, are based on functionalities and processing structures of immune systems and ant colonies. Due to the capability of incorporating all the requirements for an efficient search on dynamic environments, the immune-inspired approaches overcome the others in all the performance criteria adopted to evaluate the experimental results.

Keywords: Dynamic optimization, nonlinear optimization, combinatorial optimization, bio-inspired algorithms, artificial immune systems, ant colony optimization.

Sumário

Índice de Tabelas

Índice de Figuras

Índice de Algoritmos

Tabela de Siglas

ACA – Ant Clustering Algorithm
ACO – Ant Colony Optimization
aiNet – Artificial Immune Network
AIS – Artificial Immune System
AS – Ant System
BCA – B-Cell Algorithm
CLONALG – Clonal Selection Algorithm
copt-aiNet – Artificial Immune Network for Combinatorial Optimization
dopt-aiNet – Artificial Immune Network for Dynamic Optimization
HCF-ACO – Hyper-Cube Framework for Ant Colony Optimization
IE - Inteligência de Enxame
IMP.MMAS – Improved Max-Min Ant System
MMAS – Max-Min Ant System
opt-aiNet –Artificial Immune Network for Multimodal Optimization
PCV – Problema do Caixeiro Viajante
PI – Programação Inteira
PL - Programação Linear
PMC – P-medianas Capacitadas
PNL - Programação Não-linear
PO - Pesquisa Operacional
PSO – Particle Swarm Optimization
SI – Sistemas Imunológicos
VLSI – Very Large System Integration

Capítulo 1

Introdução

A Pesquisa Operacional (PO) estuda métodos para a obtenção de soluções em processos de otimização do mundo real, como processos industriais, econômicos, sociais e biológicos. Estes problemas são modelados e formulados como problemas matemáticos de otimização, tanto de minimização como de maximização, por exemplo, maximizar o lucro e minimizar prejuízos (Bazaraa et al., 1990; Bazaraa et al., 1993; Bazaraa & Shetty, 1976; Churchman et al., 1957).

Esses problemas podem ser divididos em dois domínios distintos: *contínuos* e *discretos*. Os problemas contínuos são aqueles em que os valores a serem otimizados pertencem ao conjunto dos números reais, enquanto nos problemas discretos, ou problemas *combinatórios*, os valores pertencem a um conjunto finito, sendo usual o emprego de números inteiros ou binários.

Até recentemente, as ferramentas desenvolvidas para resolver esses problemas lidavam com ambientes estáticos, isto é, que não têm suas características alteradas com o passar do tempo. Entretanto, em muitos problemas de mundo real esse ambiente é afetado por diversos fatores temporais. Essa classe de problemas é denominada de *problemas de otimização em ambientes dinâmicos*, ou simplesmente *otimização dinâmica*, e sua dificuldade adicional está no fato de que nem sempre é viável reiniciar uma busca toda vez que o ambiente muda, além do fato de nem sempre existir uma forma de detectar esta mudança. Nesse caso, surge a questão: "Como adaptar as ferramentas existentes que operam em ambientes estáticos de forma que possam lidar com ambientes variantes no tempo, aproveitando as informações já adquiridas ao longo do processo de otimização?".

Muitos dos problemas a serem resolvidos pertencem à classe de problemas NP-Difíceis (Garey & Johnson, 1979), caracterizada pela ausência de uma solução exata que possa ser obtida em tempo polinomial. Para encontrar uma solução aproximada para essa classe de problemas, muitas vezes são utilizadas as chamadas *heurísticas* ou *meta-heurísticas* (Glover & Kochenberger, 2002; Michalewicz & Fogel, 2000), que envolvem algoritmos de busca que recorrem a estratégias empíricas de tratamento do problema, visando buscar uma solução que seja, no mínimo, satisfatória. Esses algoritmos não garantem a obtenção do ótimo global como resultado final, e alguns deles nem sequer garantem convergência. Apesar disso, são inúmeros os exemplos de sucesso da aplicação de meta-heurísticas relatados na literatura, principalmente quando combinados com métodos de busca tradicionais, como métodos de programação não-linear, inteira e dinâmica (Bazaraa et al., 1990; Bazaraa et al., 1993; Bazaraa & Shetty, 1976; Churchman et al., 1957). Em geral, a ausência de propostas de solução tratáveis computacionalmente que possuam convergência ao ótimo global é uma motivação necessária para o emprego de meta-heurísticas, seja em otimização estática ou dinâmica, onde geralmente tem-se um

custo computacional viável e grandes chances de conseguir encontrar uma boa solução, possivelmente até o ótimo global.

Uma classe de meta-heurísticas que vem ganhando destaque nos últimos anos é a dos algoritmos bio-inspirados, que são algoritmos baseados no comportamento de certos animais, e também em processos e modelos de sistemas e fenômenos biológicos (de Castro & Von Zuben, 2005).

São considerados neste trabalho os algoritmos baseados em Sistemas Imunológicos (Dasgupta, 1998; de Castro, 2001; de Castro & Timmis, 2002b) e os baseados no comportamento das Colônias de Formigas (Dorigo, 1992). Os primeiros se inspiram em diversas propriedades do sistema imunológico humano, como a capacidade de identificar, eliminar e extrair características de microorganismos causadores de doenças. Características como identificação de múltiplos padrões por células diferentes, expansão e contração adaptativa do número dessas células, entre outros, fornecem peculiaridades necessárias para processos de otimização, principalmente no caso dinâmico. O segundo grupo de algoritmos bio-inspirados refere-se, principalmente, àqueles baseados no comportamento de colônias de formigas ao executarem certas tarefas, como busca por alimento e limpeza de ninhos. Para tanto, as formigas têm a capacidade de se auto-organizarem de maneira a realizar a operação de forma coletiva e cooperativa, utilizando apenas comunicação indireta entre elas.

A escolha desses algoritmos deve-se ao fato de que eles possuem características desejáveis para o processo de otimização em ambiente dinâmico, como: serem algoritmos populacionais, tornando possível realizar uma exploração por diversas soluções ao mesmo tempo; promoverem a manutenção de diversidade, enfatizando a característica populacional e mantendo cada proposta de solução em uma região diferente do espaço de busca; possuírem um controle dinâmico do número de indivíduos na população, ajudando na manutenção de diversidade e mantendo apenas os indivíduos que permanecem em pontos estratégicos; trabalharem com estratégias de busca local, de forma a realizar uma busca acelerada ao ótimo local mais próximo de cada proposta de solução; e realizarem um uso eficaz de memória, mantendo indivíduos que já foram ótimos e que podem voltar a ser, por exemplo.

Principais Contribuições da Pesquisa:

- Detecção de limitações e restrições de desempenho do algoritmo opt-aiNet (do inglês *artificial immune network for multimodal optimization*) e criação subseqüente do algoritmo dopt-aiNet (do inglês *artificial immune network for dynamic optimization*), contendo:
    - busca unidimensional para definir o tamanho do passo ótimo em cada direção definida durante a mutação;
    - novo algoritmo de supressão que identifica quando duas soluções representam o mesmo ótimo local;
    - alteração na estrutura populacional, dividindo o conjunto de soluções em duas populações diferentes: população atual e população de memória, sendo a última utilizada para classificar os ótimos locais já encontrados;

    - dois novos algoritmos de mutação que definem um maior conjunto de direções possíveis para a busca.
- Melhorias no algoritmo MMAS (do inglês *Max Min Ant System*), dando origem ao IMP.MMAS (do inglês *Improved Max Min Ant System*), onde os procedimentos de atualização do feromônio e detecção de convergência foram otimizados.
- Introdução de uma busca local, no algoritmo copt-aiNet (do inglês *artificial immune network for combinatorial optimization*), visando melhora e rapidez na busca por soluções ótimas em ambientes dinâmicos.

Organização do Texto

Esta dissertação está organizada em três partes, de uma forma que facilite o entendimento e conduza a uma melhor organização dos assuntos abordados. Na primeira parte, composta pelo Capítulo 2, são explicados os processos de otimização, a modelagem matemática do problema e é apresentado um breve histórico dos métodos clássicos de solução.

A segunda parte da dissertação é dedicada aos sistemas contínuos, e constará do Capítulo 3, que mostra os algoritmos baseados em Sistemas Imunológicos Artificiais e suas aplicações em sistemas contínuos, e do Capítulo 4, onde será abordada a aplicação de um desses algoritmos para o caso de ambientes dinâmicos.

A terceira parte abordará os problemas combinatórios. No Capítulo 5 são detalhados os conceitos de algoritmos inspirados em Colônia de Formigas, com análise de desempenho em ambientes estáticos, sendo seguido pelo Capítulo 6, onde um algoritmo inspirado em Sistemas Imunológicos para otimização em ambiente discreto tem suas características analisadas. Finalmente, o Capítulo 7 apresenta o desempenho desses algoritmos em ambientes dinâmicos, além de uma análise detalhada de suas características.

Por último, o Capítulo 8 conclui o trabalho com uma análise criteriosa de desempenho dos algoritmos propostos e aponta perspectivas para trabalhos futuros.

Parte I

Considerações Iniciais

Capítulo 2

Conceitos Gerais de Otimização

O conceito de otimização refere-se à tomada de decisões que melhorem certo processo partindo de uma formulação matemática do problema (Bazaraa et al., 1990; Bazaraa et al., 1993; Bazaraa & Shetty, 1976; Churchman et al., 1957).

Em muitas áreas, existe a necessidade freqüente de ajuste dos procedimentos utilizados para obter o melhor desempenho de uma dada tarefa. Por exemplo, quando estamos a caminho do trabalho e percebemos um congestionamento à frente, tomamos a decisão de alterar nossa rota de modo que não percamos tempo no trânsito. Na indústria, existem políticas de produção em que se busca fabricar apenas a quantidade que atende a demanda, reduzindo ao máximo os custos de armazenagem. Todos esses problemas fazem parte do que é chamado de *programação matemática* (Bazaraa et al., 1990; Bazaraa et al., 1993; Bazaraa & Shetty, 1976; Churchman et al., 1957), em que através de uma formulação matemática do problema e da definição do procedimento de solução, otimiza-se um processo de forma a melhorar o desempenho ou diminuir custos.

Um problema de programação matemática tem a seguinte forma:

$$\begin{aligned} &\text{otimizar } f(x) \\ &\text{Sujeito a :} \\ &\begin{cases} g(x) \le 0 \\ h(x) = 0 \end{cases} \end{aligned} \qquad \textbf{(2.1)}$$

onde otimizar refere-se a maximizar ou minimizar, dependendo do problema x é um vetor representando as variáveis de otimização, $f(x)$ é chamada de *função-objetivo*, que relaciona desempenho do sistema (p. ex., lucro, tempo na execução de uma tarefa) às variáveis otimização em x (p. ex., quantidade de material, alocação de máquinas, rota a ser percorrida), e $g(x)$ e $h(x)$ são as restrições de desigualdade e igualdade, respectivamente, aplicadas sobre x.

Os tipos de programação ou otimização mais comuns são:

- *otimização contínua*, que compreende a *otimização linear*, onde a função-objetivo é linear em x, com x sendo um vetor de variáveis contínuas do problema; e a *otimização não-linear*, onde pelo menos um dos componentes da função-objetivo é não-linear em x;
- *otimização combinatória*, geralmente representados por grafos, e na qual as variáveis que compõe o vetor x pertencem a conjuntos discretos.

Existem vários algoritmos capazes de resolver problemas de otimização linear de forma exata e em tempo polinomial, como o método Simplex e suas variantes (Bazaraa et al., 1990; Bazaraa & Shetty, 1976). Entretanto, a maioria dos problemas referentes aos outros dois tipos de programação pertence a uma classe de problemas denominada NP-difícil (do inglês *NP-hard*), na qual só é possível determinar a melhor solução para o problema a partir de métodos exatos em tempo exponencial ou fatorial.

Desse ponto em diante, todos os problemas de otimização serão considerados de minimização.

2.1 Vizinhança e Ótimos Locais

Antes de explicarmos os algoritmos desenvolvidos para encontrar a solução dos problemas, devemos estabelecer alguns conceitos necessários para o entendimento dos mecanismos de busca por solução.

Tradicionalmente, os algoritmos de busca trabalham com uma região fechada do problema, chamada de *espaço de busca*, que geralmente é definida pela representação escolhida para as soluções candidatas e pelas restrições do problema. A partir de um ponto x qualquer pertencente ao espaço de busca B, uma função $V(x)$ que retorna um sub-espaço de B, formado por uma região em torno de x (veja Figura 2.1), é chamada de função de vizinhança de x.

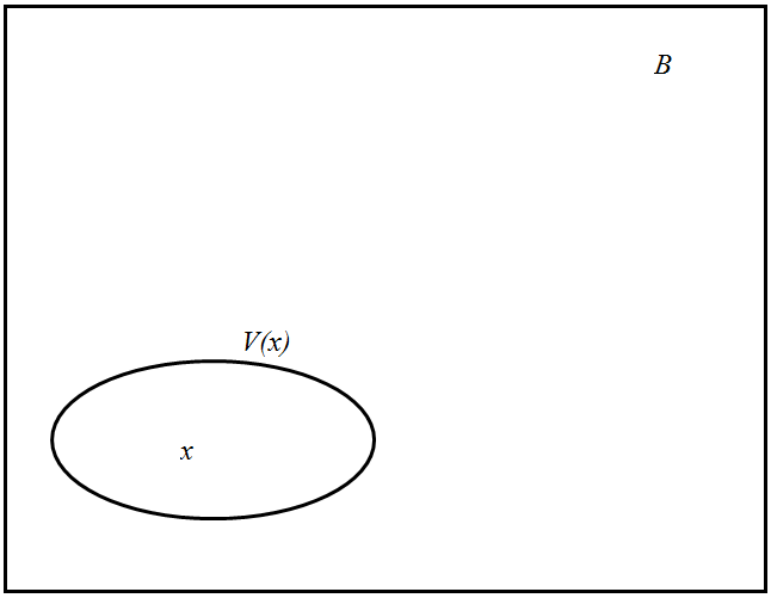


Figura 2.1: Espaço de busca e vizinhança de *x*.

Essa função de vizinhança geralmente é definida através de uma medida de distância, como, por exemplo, a distância euclidiana entre os pontos, e limitada por um valor $\varepsilon > 0$. Neste caso, a definição de vizinhança de x fica:

$$V(x) = \{y \in B \mid dist(x, y) \leq \varepsilon\} \tag{2.2}$$

onde

$$dist(x, y) = \sqrt{\sum_{i=1}^{N} (x_i - y_i)^2} \tag{2.3}$$

com N sendo a dimensão do espaço euclidiano utilizado.

Se, para um certo ponto x, tem-se que:

$$f(x) \leq f(y), \forall y \in V(x), \tag{2.4}$$

diz-se, então, que x é um ótimo local na vizinhança V(x) do espaço de busca B. Entretanto, se x for tal que:

$$f(x) \leq f(y), \forall y \in B, \tag{2.5}$$

diz-se que x é um ótimo global.

Uma das características presentes em muitos problemas e que dificulta o processo de otimização é a existência de múltiplos ótimos locais, pois muitas vezes os algoritmos de busca restringem o tamanho da região de vizinhança onde irão procurar por uma solução e podem, equivocadamente, supor que um determinado ótimo local é a melhor solução possível.

2.2 Programação Não-Linear

A *programação não-linear* (PNL) teve origem com o trabalho de Davidon (1959) e estuda métodos de solução de problemas de otimização nos quais pelo menos um dos fatores da função-objetivo é não-linear, podendo haver restrições tanto lineares como não-lineares. Esses métodos de solução são geralmente caracterizados por recorrer a processos iterativos de busca.

Uma classe de funções não-lineares muito freqüente e, por isso, muito estudada, é composta pelas chamadas funções multimodais. As funções multimodais são aquelas que possuem múltiplos ótimos locais, tornando a maioria dos métodos de busca tradicionais ineficazes, pois estes convergem para o ótimo mais próximo do ponto inicial de busca, e este ótimo local pode ser de má qualidade.

2.3 Problemas Combinatórios

Outra classe muito comum de problemas de otimização são os *problemas combinatórios* (Lawer et al., 1985). Neste caso, é necessário definir, por exemplo, quais itens devem pertencer à solução, qual a ordem de execução de uma série de processos, ou então qual estado, dentre um conjunto finito de valores, será assumido pelas variáveis do problema. Exemplos de tais problemas podem ser vistos em diversas áreas como: roteamento de veículos, circuitos integrados (VLSI), alocação de tarefas e planejamento de produção.

Esses problemas podem ser representados tanto na forma de *grafos* como na forma de um problema de *programação inteira* (PI). A seguir, será brevemente discutido um dos problemas mais estudados dessa classe: o *Problema do Caixeiro Viajante* (PCV).

2.3.1 Problema do Caixeiro Viajante

O Problema do Caixeiro Viajante (PCV) é descrito como sendo o problema de encontrar, dentro de um conjunto de *n* cidades, o *ciclo hamiltoniano* de menor custo. Um ciclo hamiltoniano é um ciclo de um grafo contendo todos os vértices (cada vértice está associado a uma cidade), sendo que cada vértice aparece apenas uma vez neste ciclo. Este problema pertence à classe de problemas NP-difíceis e é amplamente estudado por ser um caso geral de diversos outros problemas.

Uma possível formulação do PCV é sob a forma de um problema de *programação inteira*, como proposto em Orman & Williams (2004):

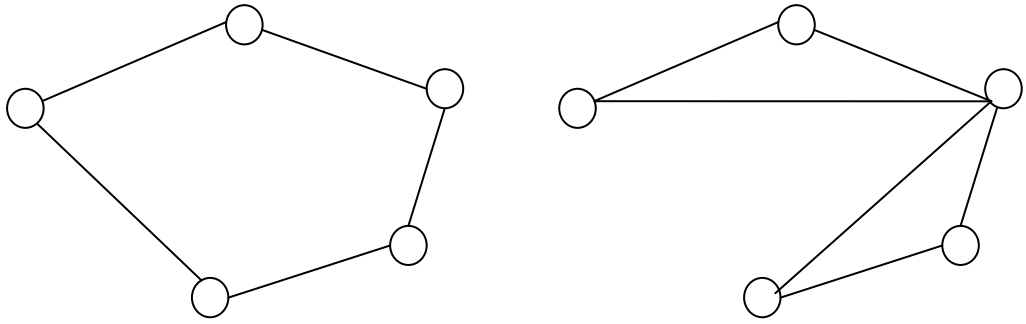
$$\text{minimizar } f(x) = \sum_{\substack{i,j \\ i \neq j}} c_{ij} x_{ij}$$

Sujeito a :

$$\begin{cases} \text{R1} & \sum_{\substack{j \\ i \neq j}} x_{ij} = 2, & \forall i \in N \\ \text{R2} & \sum_{\substack{i \\ i \neq j}} x_{ij} = 2, & \forall j \in N \\ \text{R3} & \sum_{\substack{i,j \in M \\ i \neq j}} x_{ij} \leq |M| - 1, & \forall M \subset N \\ \text{R4} & x_{ij} \in \{0,1\} \end{cases} \tag{2.6}$$

onde c_{ij} é o custo para percorrer o caminho entre as cidades i e j e x_{ij} pertence ao conjunto binário {0,1} (de acordo com R4) e indica se o caminho (i,j) pertence à solução, sendo x_{ij}=1 quando o caminho (i,j) pertence à solução e x_{ij}=0 caso contrário. As duas primeiras restrições do problema servem para garantir que uma mesma cidade não seja visitada mais de uma vez na solução. A terceira restrição indica que o percurso deve ser completo e deve conter todas as cidades. Já a última restrição é referente ao domínio de x. O parâmetro N refere-se ao conjunto de cidades e M a um sub-conjunto de N. Um exemplo do uso das restrições pode ser visto na Figura 2.2.

Na forma de um problema de grafos, tem-se que, dentro de um grafo completo $G = \{E,V\}$, onde E é o conjunto de arestas e V é o conjunto de vértices, encontre o sub-grafo $G' = \{E',V\}$, sendo $G' \subset G$ tal que cada vértice apareça em exatamente duas arestas do conjunto e exista apenas um único ciclo. Para um número N de vértices, existem $(N-1)!/2$ soluções possíveis para o problema, já que não importa qual será a cidade de início do percurso fechado, que pode ser determinada arbitrariamente, e este percurso pode ser realizado em sentido horário ou anti-horário.



(a) (b)

Figura 2.2: O problema do caixeiro viajante. (a) Solução factível para o PCV com 5 cidades. (b) Solução que viola as restrições R1, R2 e R3 para o mesmo problema.

2.4 Métodos de Solução

Nesta seção, serão apresentadas as técnicas mais comuns para solucionar tanto problemas contínuos como discretos. A seção 2.4.1 aborda os métodos exatos, que garantem a solução ótima global. Uma adaptação dos métodos de enumeração descritos na seção 2.4.1 é realizada na seção 2.4.2 para espaços de busca contínuos. A seção 2.4.3 descreve os métodos de busca local, que exploram apenas uma região em torno de um ponto. Heurísticas que adicionam uma espécie de *guia* aos métodos de busca são apresentadas na seção 2.4.4, enquanto a Seção 2.4.5 discute as meta-heurísticas, que implementam definições para melhorar o desempenho das heurísticas. Já a seção 2.4.6 trata de uma classe especial de meta-heurística, vinculadas à computação bio-inspirada (de Castro & Von Zuben, 2005), e que será abordada nesta dissertação.

2.4.1 Método de Enumeração e Algoritmos Exatos

Para problemas da classe NP-difícil, uma das formas conhecidas até o momento de se obter com segurança um ótimo global é através de uma busca exaustiva em cada elemento ou combinação de elementos possíveis. Entretanto, esse é um processo custoso e o tempo e/ou memória necessários para se obter a solução cresce exponencial ou fatorialmente quando a dimensão do problema cresce linearmente. Além disso, também existe a questão de como gerar todas as possíveis soluções sem repetição.

Como os métodos exaustivos requerem a avaliação de todas as soluções candidatas do problema, eles são chamados de *métodos de enumeração* ou *métodos enumerativos*. Esses métodos operam da seguinte forma: a partir de uma solução, gere uma outra sem que esta já tenha sido testada anteriormente, ou seja, sendo S o conjunto de soluções candidatas, gere uma seqüência de soluções com a cardinalidade de S, e onde $S_i \neq S_j$, $\forall S_i, S_j \in S$ e $i \neq j$.

Enumerando Problemas Discretos

Como a enumeração em otimização discreta é mais intuitiva, ilustraremos inicialmente a solução para o PCV. Esse problema é enumerado gerando todas as possíveis permutações possíveis, de um problema com N vértices, de forma recursiva a partir de um vértice inicial, como pode ser visto no Algoritmo 2.1.

```
[opt_caminho] = Função enumere(i,vertice,k),
      vertice = vertice + 1;
      caminho[i] = vertice;
      Se k == N,
            Se dist(caminho) > dist(opt_caminho),
                  opt_caminho = caminho;
            Fim
      Fim
      Para p = 1..N,
            Se caminho[p] == 0,
                  enumere(p,vertice,k+1);
            Fim
      Fim
      k = k - 1;
      caminho[i] = 0;
Fim
```

Algoritmo 2.1: Algoritmo de enumeração para o PCV.

Este algoritmo é inicialmente executado pela chamada da função `enumere(1,0,0)`, que indica que a enumeração parte do vértice 1 e que as outras variáveis estão em estados iniciais. Em seguida, esse vértice é designado como primeiro elemento do caminho e, para cada vértice que ainda não faz parte da solução, o procedimento é chamado recursivamente com os novos estados (*vertice* = 1, $k = k+1$), até o momento em que $k = N$, quando o caminho se torna completo e a solução é avaliada. Caso a solução seja melhor do que a melhor solução encontrada até o momento, esta é atribuída à variável `opt_caminho`. Após um caminho se completar, a função retorna para o passo anterior e remove este vértice da solução, iniciando o processo com o próximo.

2.4.2 Método de Enumeração em Espaços de Busca Contínuos

Como em domínios contínuos existem infinitos pontos, mesmo em um intervalo fechado, torna-se impossível enumerar todas as soluções possíveis do problema. Mesmo assim, utilizando esse conceito pode-se aproximar uma enumeração subdividindo o intervalo (domínio) em várias partes menores e, escolhido o intervalo de menor valor (ou maior se for caso de maximização), subdivide-se este em intervalos ainda menores, até que se atinja uma determinada precisão.

Embora não seja possível considerar esta enumeração como sendo um método exato, se os intervalos forem subdivididos em espaços muito pequenos, no limite é possível se atingir o ótimo global. A partir disso, surgem duas questões fundamentais: i) Se o tamanho da divisão for muito pequeno, então estaremos aumentando o espaço de busca excessivamente, o que torna a convergência muito lenta; e ii) Se o tamanho da divisão for

muito grande, então as chances de atingir um ótimo global se tornam muito pequenas, dependendo da complexidade do problema.

Um algoritmo de enumeração para problemas contínuos pode ser descrito como apresentado no Algoritmo 2.2.

```
[x*] = Função enumere(n,xa,xb)
       S = [xa; (xa + (xb-xa)/n); (xa + 2*(xb-xa)/n); ... xb];
       Se |S(1) - S(2)| > ε,
             Para i = 1..n-1 faça,
                   x[i] = enumere(n, S(i), S(i+1));
             Fim
       Senao
             x = S;
       Fim
       x* = min_x(f(x));
Fim
```

Algoritmo 2.2: Algoritmo de enumeração para um problema contínuo do tipo não-linear.

O algoritmo de enumeração recebe três parâmetros de entrada: n, referente ao número de intervalos em que a função será dividida; x_a e x_b, que representam o início e o fim desse intervalo, respectivamente. No início do algoritmo, é gerado o vetor S de intervalos de soluções e a distância entre estes intervalos é verificada. Caso seja menor que um dado ε, o algoritmo devolve $x \in S$ para o qual o valor da função-objetivo é mínimo. Caso contrário, o algoritmo sub-divide cada um dos intervalos em n partes até que a condição de distância dos intervalos seja satisfeita.

2.4.3 Métodos de Busca Local

Os métodos de busca local são métodos que partem de uma solução qualquer, percorrendo sua vizinhança à procura de soluções melhores. Caso uma solução melhor nesta vizinhança seja encontrada, o mesmo procedimento é repetido iterativamente na nova solução até que um ótimo local seja identificado. O que difere uma busca local de outras e determina sua eficácia é a escolha da vizinhança. Outro fator que afeta o desempenho da busca, tanto na qualidade dos resultados quanto no custo computacional, independentemente da determinação de vizinhança, é se a busca é do tipo *first improvement* (primeira melhoria) ou *best improvement* (melhor melhoria). No *first improvement*, a primeira solução que apresentar melhora na vizinhança será a substituta da solução atual. Já no *best improvement*, é feita uma busca dentro de toda a vizinhança para só então determinar a próxima solução a ser avaliada.

Para problemas contínuos não-lineares, serão apresentados dois algoritmos que necessitam de informação de derivadas da função para determinação da direção de busca, mas que não garantem a convergência para o ótimo global, e são eles: o Método do Gradiente e o Método de Newton. No caso de problemas discretos, serão apresentados três algoritmos clássicos de busca local: 2-opt, *k*-opt e *Lin-Kernighan*, utilizando-se do problema do caixeiro viajante para ilustrar as explicações.

Método Contínuo de 1a. Ordem: Método do Gradiente

O *método do gradiente* consiste em, dado um ponto inicial x_0, determinar a direção associada à maior taxa de decrescimento da função-objetivo, indicada pela informação contida em seu vetor gradiente. A estrutura básica do algoritmo é descrita no Algoritmo 2.3.

```
[x*] = Função gradiente(x0,ε)
       k = 0;
       Enquanto ||∇f(x^k)|| ≥ ε
              d^k = -∇f(x^k);
              α_k = busca(x^k,d^k);
              x^{k+1} = x^k + α_k.d^k;
              k = k + 1;
       Fim
Fim
```

Algoritmo 2.3: Algoritmo do método do Gradiente.

A função gradiente recebe dois parâmetros, x_0 que se refere ao ponto inicial, e ε, que é o valor de limiar utilizado no critério de parada do algoritmo. A cada iteração, o gradiente da função no ponto atual é calculado e a direção de busca é determinada como sendo oposta ao do vetor gradiente. Em seguida, é feita uma busca unidimensional (Bazaraa et al., 1993; Bazaraa & Shetty, 1976) para determinar qual o tamanho do passo nessa direção, e então é calculado o novo ponto. O passo pode ser otimizado ou simplesmente busca-se um passo que conduza a uma melhora do critério de otimização. Esses passos se repetem até que a norma do vetor gradiente seja menor que um dado limiar ε, correspondente ao critério de parada. O ponto x atual é retornado como sendo o ótimo local encontrado (Figura 2.3).

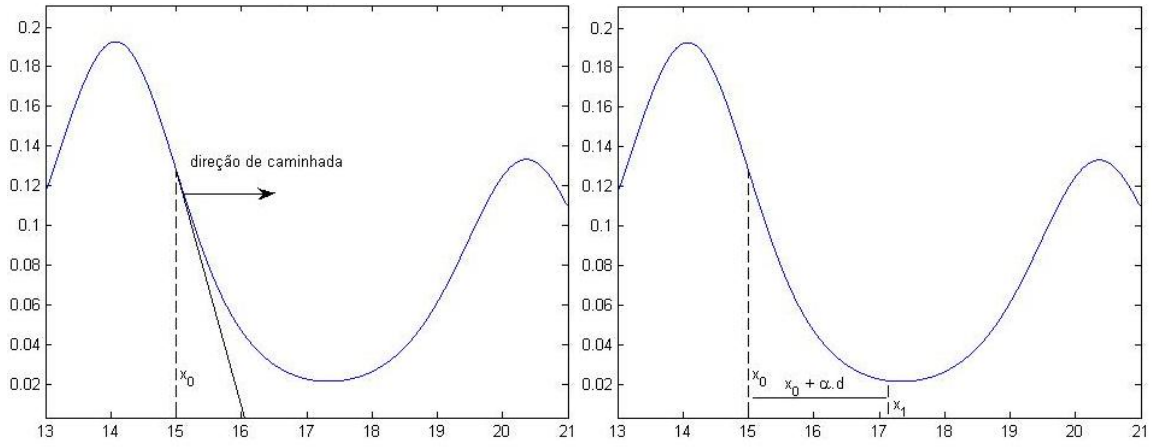


Figura 2.3: Uma iteração do Método do Gradiente considerando a otimização unidimensional. (a) A informação de primeira ordem define a direção de caminhada e (b) o ponto x_1 é encontrado através de $x_0 + \alpha.d$.

Método Contínuo de 2a. Ordem: Método de Newton

Um método mais preciso e de convergência mais rápida é o *método de Newton* (Bazaraa et al., 1993; Luenberger, 1984). Sua estrutura é muito parecida com a do método do gradiente, exceto pelo fato de que a direção de busca é determinada por uma aproximação quadrática da função custo, obtida através de sua matriz hessiana. Utilizando a expansão truncada em série de Taylor, uma função pode ser aproximada por:

$$f(x) \cong f(x_0) + \nabla f(x_0)^T .(x - x_0) + \frac{1}{2}(x - x_0)^T F(x_0)(x - x_0), \tag{2.7}$$

onde $F(x_0)$ é a hessiana da função $f(x)$ no ponto x_0, e $(\cdot)^T$ corresponde à operação de transposição vetorial.

O ponto ótimo da função encontra-se em $\nabla f(x) = 0$ e, portanto:

$$\nabla f(x) \cong \nabla f(x_0) + F(x_0).(x - x_0) = 0, \tag{2.8}$$

$$x = x_0 - F(x_0)^{-1}.\nabla f(x_0). \tag{2.9}$$

O método de Newton é resumido no Algoritmo 2.4 e ilustrado na Figura 2.4.

```
[x*] = Função Newton(x0,ε)
       k = 0;
       Enquanto ||∇f(x^k)|| ≥ ε
              d^k = -F(x_0)^-1.∇f(x^k);
              x^k+1 = x^k + d^k;
       Fim
Fim
```

Algoritmo 2.4: Algoritmo do método de Newton.

Note que, para funções quadráticas, este algoritmo chega ao ótimo em apenas uma iteração. Assim como o método do gradiente, o método de Newton também retorna, geralmente, um mínimo local da função.

Um aspecto interessante é que nem sempre $f(x^{k+1}) < f(x^k)$ e, portanto, o algoritmo tende a oscilar em torno do ótimo local, em alguns casos necessitando até mesmo de mais iterações que o método do gradiente, podendo até divergir. Para resolver este problema, é possível utilizar um passo $\alpha \in (0,1]$ da mesma forma que no método do gradiente, transformando a Eq. (2.9) na Eq. (2.10).

$$x = x_0 - \alpha \cdot F(x_0)^{-1} \cdot \nabla f(x_0). \tag{2.10}$$

Além do fato de necessitarmos da informação da segunda derivada da função, existe também a limitação no cálculo da inversa da matriz hessiana, que normalmente possui um custo computacional muito grande. Muitos métodos baseados neste, denominados *quase-Newton*, utilizam-se de uma aproximação dessa matriz inversa para reduzir o custo computacional, ao mesmo tempo em que usam uma informação de segunda ordem aproximada.

Como uma última limitação do método de Newton, existe a necessidade de a matriz hessiana ser definida positiva para problemas de minimização. Caso não seja, essa matriz deverá sofrer modificações para se tornar definida positiva.

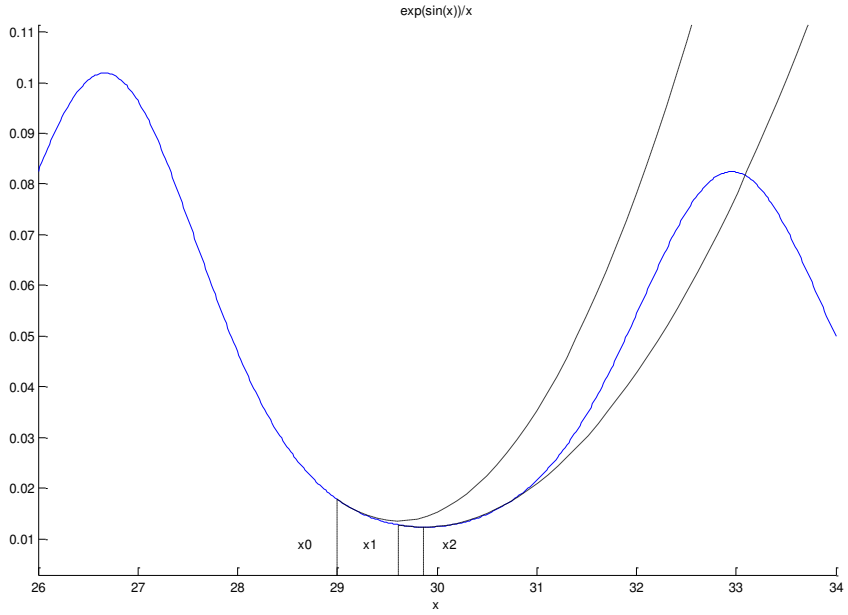


Figura 2.4: Duas iterações do método de Newton para $x_0 = 29$ e f(x) = $\frac{e^{sen(x)}}{x}$.

Embora esses métodos funcionem bem, retornando um mínimo local e com convergência quadrática (a raiz quadrada do erro em uma data iteração é proporcional ao erro da próxima), eles requerem que a função seja contínua e diferenciável no espaço de busca percorrido.

Método Discreto 1: Algoritmo 2-opt

Este é o método de busca mais simples para o problema do caixeiro viajante, sendo a vizinhança $N(S)$ da solução S definida pelo conjunto de soluções que podem ser alcançadas através da permutação de duas arestas não-adjacentes em S. Este movimento é chamado de 2-*interchange* e é descrito na Figura 2.5.

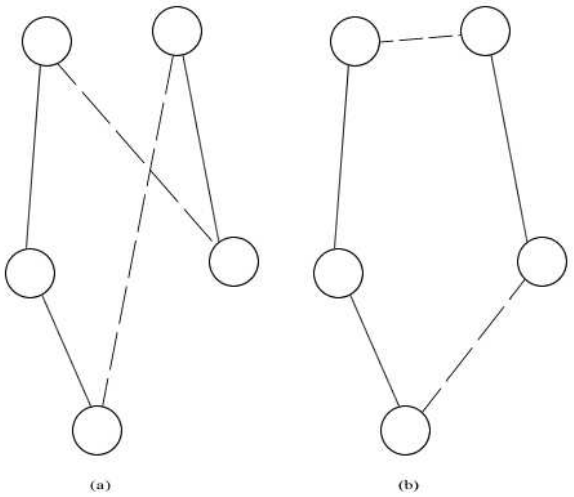


Figura 2.5: Exemplo de uma operação 2-*interchange*. (a) Antes da troca. (b) Depois da troca.

Note que, como dito anteriormente, dependendo do tipo de implementação de busca local todas as trocas possíveis são efetuadas antes de ser tomada a decisão, ou pelo menos até que se encontre a primeira troca que melhore a solução, se essa troca existir.

Método Discreto 2: Algoritmo *k*-opt

Este método nada mais é do que a generalização do anterior, ou seja, a vizinhança $N(S)$ da solução S é definida pelo conjunto de soluções obtidas através da permutação de até k arestas. O número escolhido para k definirá o tamanho da vizinhança e conseqüentemente o desempenho do algoritmo. Um valor baixo de k definirá uma vizinhança pequena, mas também uma menor chance de encontrar uma solução melhor. Por outro lado, um valor muito alto de k resulta em grandes chances de encontrar a solução ótima global, mas também resulta em uma vizinhança muito grande, tornando o algoritmo computacionalmente intratável. Geralmente um valor de $k \leq 3$ é utilizado.

Método Discreto 3: Algoritmo *Lin-Kernighan*

Este é o algoritmo de busca local desenvolvido por Lin & Kernighan (1973), atualmente a mais conhecida e utilizada busca local para o problema do caixeiro viajante. Este método se baseia no *k*-opt, mas com uma diferença: o valor de k passa a ser variável. Ou seja, além de determinar qual a melhor troca, este algoritmo também determina qual o melhor k para aquela iteração.

O algoritmo busca a seqüência de arestas $\{x_1, x_2, ..., x_k\}$ que, quando trocadas pelas arestas $\{y_1, y_2, ..., y_k\}$, retornam um caminho factível e de menor custo. Uma descrição mais detalhada é apresentada no Algoritmo 2.5 e na discussão a seguir.

```
[caminho] = Função lin-kernighan(),
      tour = caminho_aleatório();
      G* = 0;
      i = 1;
      n2i-1 = sorteie_nó();
      n2i = escolha_nó(n2i-1);
      x[i] = (n2i-1,n2i);
      n2i+1 = escolha_nó_troca(n2i);
      y[i] = (n2i,n2i+1);
      g[i] = gain(x_aresta[i], y_aresta[i]);
      G = g[i];
      Repita
            i = i + 1;
            n2i = escolha_nó(n2i-1);
            x[i] = (n2i-1,n2i);
            n2i+1 = escolha_nó_troca(n2i);
            y[i] = (n2i,n2i+1);
            g[i] = gain(x_aresta[i], y_aresta[i]);
            G = G + g[i];
            Se não existem mais trocas de arestas que melhorem,
                  Se G > G*,
                        G* = G;
                        caminho = troca(caminho,x_aresta,y_aresta);
                  Fim
            Fim
      Até que G ≤ G*;
Fim
```

Algoritmo 2.5: Algoritmo de Lin-Kernighan para o PCV.

O algoritmo inicia construindo um caminho aleatório e sorteando um nó n_1 por onde irão começar as trocas. Em seguida, é escolhida uma aresta contendo o nó inicial $x_1 = (n_1,n_2)$ e uma aresta de troca que parta de n_2 e gere um ganho positivo $y_1 = (n_2,n_3)$. Escolhida a primeira aresta de troca, começa então o processo iterativo com $i = 2$, onde a cada passo uma aresta é escolhida contendo o último nó escolhido na iteração anterior $x_i = (n_{2i-1},n_{2i})$, e escolhe-se uma aresta partindo de n_{2i} tal que algumas condições sejam satisfeitas:

i) Se a aresta (n_1,n_{2i}) é criada, então um caminho completo é formado;
ii) A aresta y_i é uma aresta não utilizada contendo o nó n_{2i};
iii) Para garantir a disjunção entre x_i e y_i, x_i não pode ser uma aresta já escolhida para o conjunto y, e y_i não pode ser uma aresta já escolhida pelo conjunto x;
iv) O ganho deve ser positivo;
v) A aresta y_i deve permitir a quebra de x_{i+1} para possibilitar que na iteração seguinte existam trocas factíveis;
vi) Antes da decisão final da escolha de y_i é verificado se, ao fechar n_{2i} com n_1, gera-se um caminho com custo menor do que o original.

Terminada a construção do conjunto de arestas originais (x) e o conjunto de arestas de troca (y), o novo caminho é construído e os passos são executados novamente até que não ocorra mais nenhum ganho. Este processo é ilustrado passo a passo nas Figuras 2.6 à 2.10, considerando um caso de estudo contendo seis vértices.

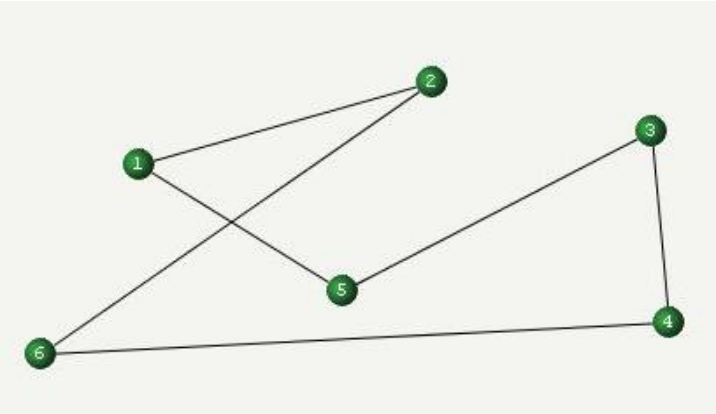


Figura 2.6: Rota inicial não ótima. O primeiro vértice escolhido aleatoriamente é o "5".

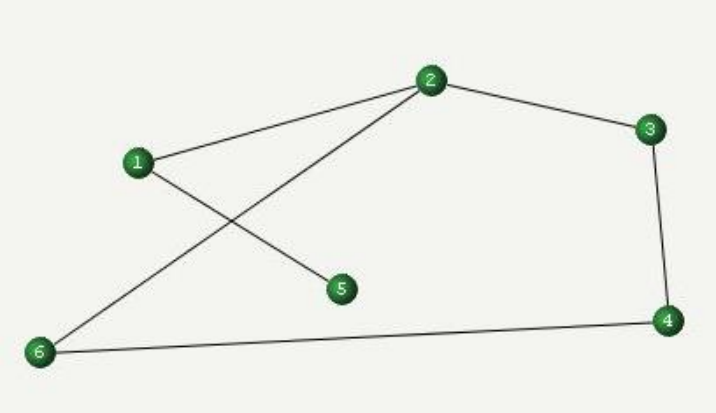


Figura 2.7: Após a escolha do vértice inicial a aresta (5,3) é incluída na lista *x* de vértices a serem removidos e a aresta (3,2) é incluída na lista *y* de vértices a serem incluídos.

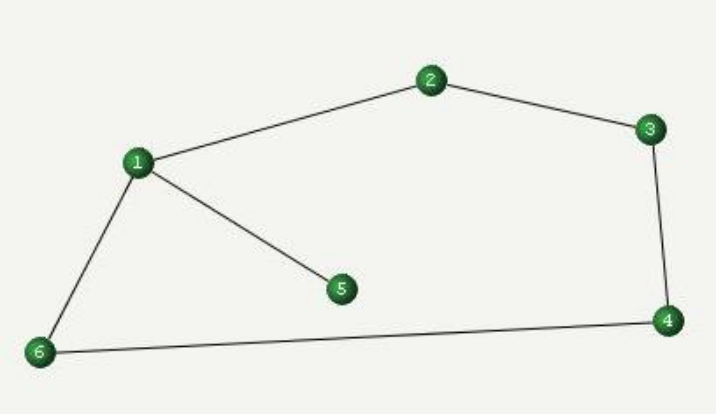


Figura 2.8: Em seguida, partindo do vértice "2" é incluído em *x* a aresta (2,6) e em *y* a aresta (6,1).

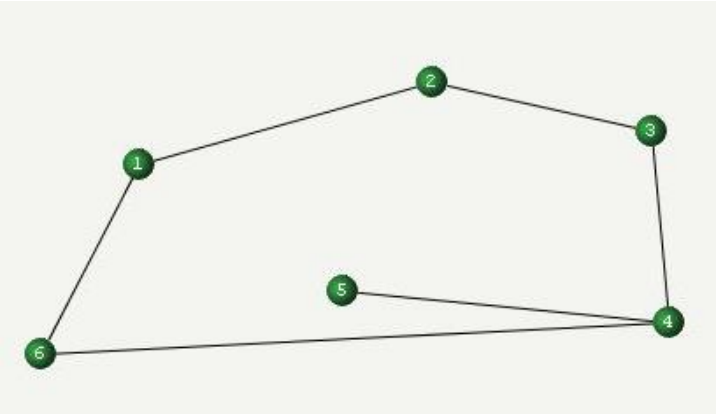


Figura 2.9: A lista *x* recebe a aresta (1,5) e a lista *y* a aresta (5,4).

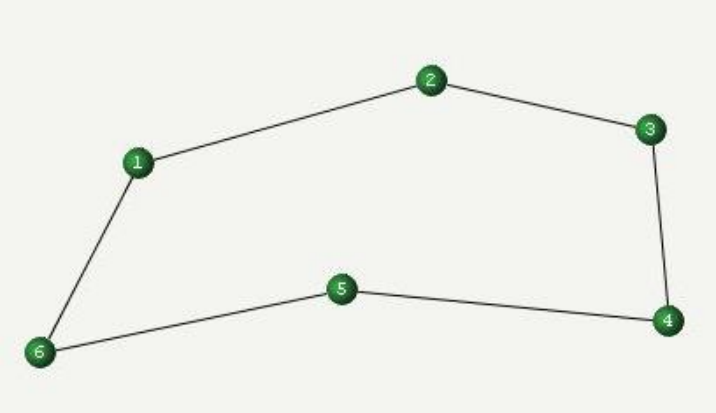


Figura 2.10: Por último, a aresta escolhida para remoção é a (4,6) e para inserção a (6,5), terminando assim as escolhas possíveis de arestas e chegando na solução ótima. As listas finais de remoção e inserção de arestas ficam: *x* = {(5,3), (2,6), (1,5), (4,6)} e *y* = {(3,2), (6,1), (5,4), (6,5)}, gerando um movimento 4-opt.

Este é o melhor algoritmo de busca local conhecido para o problema do caixeiro viajante e gera soluções cujas qualidades diferem de aproximadamente 2% do ótimo, além de ter um custo computacional relativamente baixo (Michalewicz & Fogel, 2000).

2.4.4 Métodos Heurísticos

A palavra *heurística* vem da palavra grega *heuriskein*, que significa *descobrir*. A idéia de heurística na área de Inteligência Artificial varia de acordo com suas sub-áreas, mas no conceito de busca e otimização é empregada como o método que utiliza uma ou mais informações referentes ao problema para *guiar* o processo de busca para uma solução ótima (Michalewicz & Fogel, 2000). Por exemplo, no PCV, para construir uma solução, poderíamos partir de um vértice inicial e ir acrescentando o vértice mais próximo seqüencialmente, até formar um caminho completo. A *informação heurística*, neste caso, seria a distância entre os vértices. Este algoritmo é denominado *heurística gulosa de construção*.

Método *Branch-and-Bound*

Os métodos de enumeração, como visto na seção 2.4.1, são muito custosos, uma vez que eles verificam todas as rotas possíveis, dado um conjunto de vértices. Outra forma de se resolver o problema é utilizando uma heurística para excluir uma área do espaço de busca que seguramente não contém a solução ótima.

Uma maneira de agilizar a enumeração é utilizando-se de informações ou heurísticas para o problema. Uma técnica que utiliza tal artifício é chamada de *branch-and-bound*. Basicamente, este algoritmo constrói uma árvore de soluções, onde é calculada uma estimativa ótima da solução de cada ramificação e apenas a menor destas irá gerar novas ramificações. Para ilustrar, suponha um problema de caixeiro viajante com quatro cidades, $n = 4$, e com a seguinte matriz de distâncias, onde cada elemento c_{ij} da matriz representa a distância entre os vértices i e j:

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 7 & 6 & 3 \\ 7 & 0 & 2 & 4 \\ 6 & 2 & 0 & 5 \\ 3 & 4 & 5 & 0 \end{bmatrix} \end{array}.$$

Uma possível estimativa, talvez a mais intuitiva, seria a somatória dos dois menores valores de cada linha, pois cada vértice aparecerá duas vezes em cada solução e, idealmente, em uma solução ótima teríamos as duas menores distâncias partindo de cada vértice. A vantagem de se estudar uma melhor estimativa se dá na velocidade com que se consegue encontrar a solução para o problema. Neste exemplo, teríamos:

$$(3+6)+(2+4)+(2+5)+(3+4)=29.$$

Tendo a estimativa e expandindo a árvore para cada folha (Figura 2.11), faz-se um novo cálculo levando em conta a aresta que estará presente em cada ramificação.

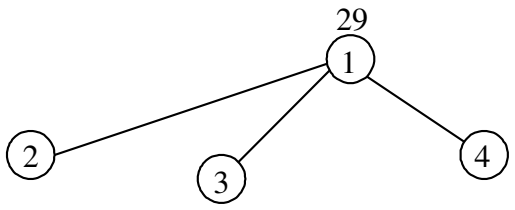


Figura 2.11: Ramificação inicial para um PCV de tamanho $n = 4$. Os valores dentro dos círculos identificam o vértice e os números acima são os valores das estimativas.

Por exemplo, vamos calcular o valor da estimativa para a ramificação 1-2 levando em conta que as arestas (1,2) e (2,1) deverão estar presentes no cálculo:

$$(3+7)+(2+7)+(2+5)+(3+4)=33$$

Fazendo o cálculo para as ramificações (1,3) e (1,4) tem-se na Figura 2.12

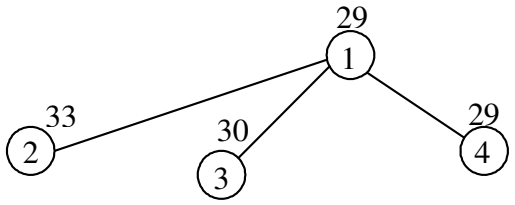


Figura 2.12: Valores limites para cada nó na ramificação inicial.

Esses valores limites significam que, a partir desses pontos esta será a menor solução possível. Intuitivamente, devemos seguir pelo vértice de menor valor que, neste exemplo, é o vértice de número 4. A árvore final é ilustrada na Figura 2.13.

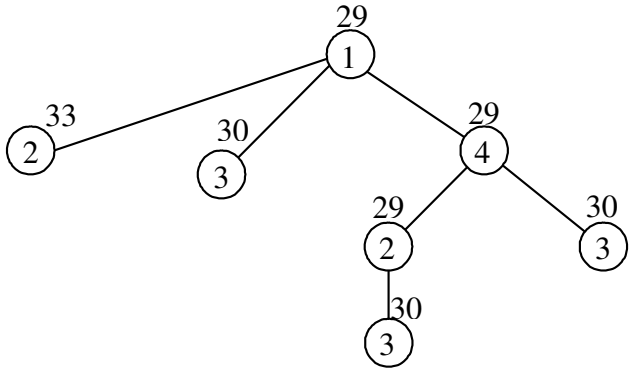


Figura 2.13: Árvore completa para um PCV de tamanho *n=4* com os valores limites para cada nó. Os valores dentro dos círculos identificam o vértice e os números acima são os valores das estimativas.

Note que, neste caso, apenas uma solução completa foi gerada, enquanto em uma busca exaustiva seriam necessárias três soluções. Quanto melhor a estimativa, mais rápida se torna a busca, pois a redução do espaço de busca tende a ser ainda maior.

2.4.5 Meta-Heurísticas

O prefixo *meta* vem do grego e significa *mudança*, *além*, *após*. No presente contexto, é possível denominar *meta-heurística* como *além da heurística*, ou seja, uma heurística que define heurísticas. A idéia principal desse conceito é a criação de heurísticas generalizadas que possam resolver diversos tipos de problemas sem a necessidade de grandes alterações na estrutura do algoritmo. Um exemplo disso é a *busca tabu* (Glover, 1986), onde simplesmente é feita a busca local no problema guardando os passos da busca em uma memória, denominada, geralmente, de *lista tabu*. Em determinado ponto, quando não há melhorias, o algoritmo parte para uma solução diferente das que já constam na memória,

possivelmente direcionando a busca para outros ótimos locais. Neste caso, a meta-heurística seria a utilização da memória, enquanto a heurística seria o uso da busca local, que deve ser diferente para cada problema.

2.4.6 Computação Bio-Inspirada

Uma classe de meta-heurísticas que tem recebido bastante atenção nos últimos tempos é constituída por ferramentas de *computação bio-inspirada* ou *computação inspirada na biologia*. A computação bio-inspirada é a linha de pesquisa que emprega metáforas e modelos de sistemas biológicos no projeto de ferramentas computacionais de solução de problemas complexos (de Castro & Von Zuben, 2005; Paton, 1994).

É sabido que muitos desses processos biológicos têm um comportamento otimizado e, muitas vezes, têm uma capacidade de processamento de informação muito maior que a do melhor computador digital já feito até hoje. Prova disso é todo o processo de evolução que as espécies tiveram ao longo de milênios, assim como a capacidade do cérebro humano para guardar informações em forma de imagens, cheiros, sabores ou sons, e também inferir raciocínios lógicos e conferir um processo rápido de aprendizado.

Um dos primeiros algoritmos bio-inspirados originou-se da tentativa de entender o funcionamento do sistema nervoso com o trabalho de McCulloch & Pitts (1943), que resultou no primeiro modelo matemático do funcionamento de um neurônio biológico. Apenas mais tarde, Rosenblatt (1957) desenvolveu a idéia e criou a primeira rede neural capaz de aprender a classificar padrões linearmente separáveis.

Outro algoritmo bio-inspirado que deve ser mencionado foi desenvolvido por J. H. Holland (1992), publicado inicialmente em 1975. Neste trabalho, Holland utilizou-se do conceito de evolução das espécies segundo Darwin, onde cada espécie evolui através da competição entre indivíduos pela sobrevivência e reprodução. Os indivíduos mais aptos a sobreviverem terão mais chances (maiores probabilidades) de propagar seu material genético para as gerações seguintes. Durante a reprodução sexuada, ocorre um procedimento denominado *recombinação*, ou *crossover*, através do qual parte do material genético do pai combina-se com parte do material genético da mãe. Como resultado, um material genético que dará origem a um novo indivíduo com características de ambos os pais será gerado, combinando o material genético de indivíduos mais aptos à sobrevivência. Seu material genético tem chance de ser melhor do que o de seus ancestrais, ou pelo menos melhor que a média da população, por causa da polarização da seleção, conhecida como *pressão seletiva*. Esta idéia deu origem aos *algoritmos genéticos* (Holland, 1992).

Além do conceito de algoritmos genéticos, outros conceitos parecidos aplicáveis a diferentes áreas formam o conjunto de técnicas denominadas *algoritmos evolutivos* (Glover & Kochenberger, 2002). Em resumo, pode-se dizer que os algoritmos evolutivos são compostos pelos algoritmos genéticos, pela *programação genética* (Koza, 1992), pelas *estratégias evolutivas* (Rechenberg, 1965; Schwefel, 1965), pela *programação evolutiva* (Fogel et al., 1966) e pelos *sistemas classificadores* (Beasley, 2001).

Os dois tipos de algoritmos bio-inspirados que serão tratados em detalhes nos próximos capítulos são os *sistemas imunológicos artificiais* (SIAs) (de Castro, 2001; de Castro & Timmis, 2002b) e a *inteligência de enxame* (IE) (Bonabeau et al., 1999). Enquanto os SIAs são motivados pela capacidade dos sistemas imunes de aprender e

reconhecer invasores dentro do organismo, a inteligência de enxame é baseada no comportamento de insetos e outros organismos sociais, como colônias de formigas.

2.5 Ambientes Dinâmicos

Os problemas discutidos até agora têm um comportamento estático, ou seja, os ótimos locais e o ótimo global são invariantes, i.e., permanecem na mesma posição durante todo o processo de busca. Entretanto, existem problemas que se comportam de maneira dinâmica, ou seja, sofrem variação em suas propriedades ou até em suas funções-objetivo, de forma que os ótimos sempre estão em movimento para outras posições enquanto o algoritmo busca por uma solução.

Por exemplo, se pensarmos em um problema de caixeiro viajante onde é preciso encontrar o melhor caminho para entregar certo produto em determinados locais, o trânsito entre um ponto e outro está sujeito a diversas variações temporais, como tráfego intenso, obras na pista, acidentes de trânsito e muitos outros obstáculos que podem alterar o que antes podia ser o caminho ótimo. No caso de ambientes contínuos, pode-se citar mudança no custo de fabricação de um determinado componente, a introdução de uma nova metodologia que altera o comportamento do sistema, desgaste em equipamentos, perturbações externas, dentre outros.

Intuitivamente, a solução mais fácil para estes problemas seria simplesmente reiniciar o algoritmo cada vez que o ambiente se alterasse. Porém, essa abordagem necessitaria que o ambiente se mantivesse estático durante todo o processo de otimização e, como é necessário reiniciar a cada mudança do ambiente, seria preciso detectar a não estacionariedade do ambiente, e nem sempre isso é possível.

A maneira mais eficiente de resolver esta dificuldade seria aproveitar as informações obtidas até o momento para procurar pelo novo ótimo assim que o ambiente fosse alterado. Isso funcionaria muito bem nos casos em que a mudança é suave e o novo ótimo estivesse próximo do antigo. Entretanto, novamente isso não é regra e, portanto, teríamos uma perda de desempenho considerável caso o novo ótimo estivesse muito distante do anterior.

Outro problema a considerar é que a maioria dos algoritmos converge para uma única solução após um determinado número de iterações e, com isso, perdem sua capacidade de acompanhar a dinâmica do problema. Em um estudo feito por Jin & Branke (2005), foram apontadas quatro características fundamentais que um algoritmo deve possuir para ser capaz de tratar problemas de otimização dinâmica. São elas:

- **Capacidade de gerar diversidade:** durante as iterações de algoritmos populacionais, gerar e incluir novas soluções em sua população de forma a espalhar os indivíduos estrategicamente pelo espaço de busca.

- **Capacidade de manter essa diversidade:** manter um controle da população de forma a evitar uma convergência prematura de suas soluções.

- **Memória de soluções:** guardar em uma memória algumas boas soluções encontradas no passado, o que é especialmente útil quando o ótimo oscila entre diversas soluções e mesmo quando ele se aproxima destas soluções passadas.

- **Multipopulação:** subdividir uma população em várias sub-populações pode ser útil para manter a diversidade, pois isso permite procurar por vários ótimos locais ao invés de apenas um, mantendo assim uma população diversa e espalhada.

Jin e Branke (2005) também apontam alguns algoritmos com essas características, sendo que nenhum deles possui as quatro características simultaneamente.

As alterações que um ambiente pode sofrer com o tempo são divididas em quatro tipos, como definido por Farina et al. (2004) e ilustrado nas Figuras 2.14 e 2.15, utilizando-se da função Rastrigin:

$$f(\mathrm{x}) = \sum_{i=1}^{N} (x_i^2 - 10.\cos(2.\pi.x_i) + 10)\,. \tag{2.11}$$

No primeiro tipo, denominado Tipo I, a mudança ocorre na posição dos ótimos no espaço de busca, que se deslocam em função do tempo, mas os seus valores correspondentes não mudam. Em ambientes contínuos, um método para simular este comportamento seria alterando a função a ser otimizada para:

$$f(x) => f(x + g(t)), \tag{2.12}$$

onde $g(t)$ define em quanto a função será deslocada, em relação à posição original, no instante t.

O segundo tipo, chamado Tipo II, refere-se ao tipo de função que não só altera a posição de seus ótimos, mas também os seus valores. Para ambientes contínuos, um exemplo seria alterar a função para:

$$f(x) => f(x + g(t)) + h(t), \tag{2.13}$$

onde $h(t)$ é a função que desloca a altura dos picos ou profundidade dos vales.

O terceiro tipo, Tipo III, é o oposto do Tipo I, onde a posição dos ótimos permanece constante, mas seus valores mudam com o passar do tempo. Em ambientes contínuos poderíamos fazer:

$$f(x) => f(x)^{g(t)}, \tag{2.14}$$

com $g(t) > 0$.

O quarto e último tipo, Tipo IV, refere-se à adição de uma outra função-objetivo ou simples substituição da função-objetivo anterior, podendo causar uma alteração completa do ambiente de busca.

$$f(x) => f(x) + g(x)\,. \tag{2.15}$$

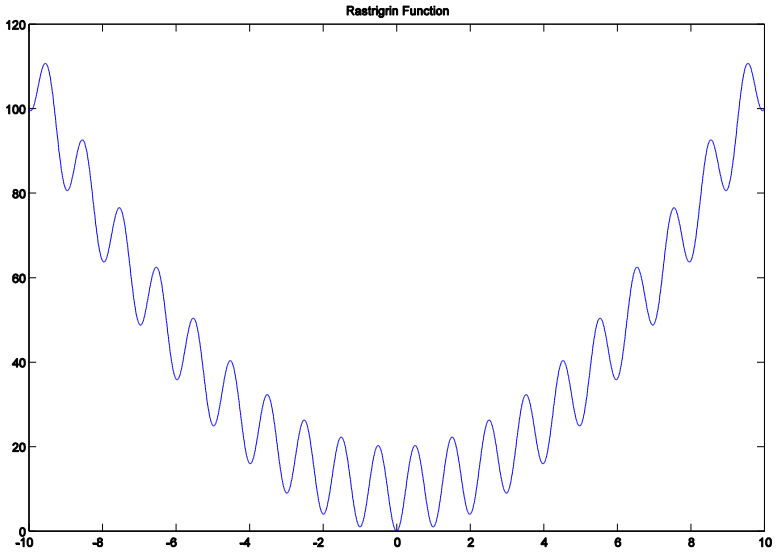


Figura 2.14: Função Rastrigin estática para $x \in [-10; 10]$.

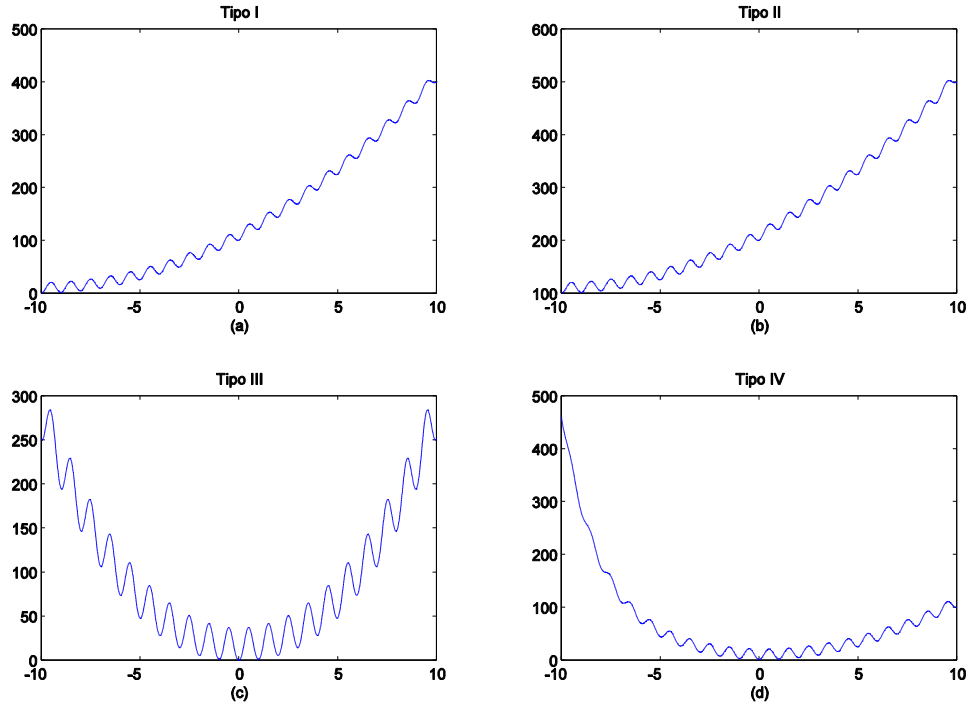


Figura 2.15: Exemplos da aplicação de dinamismo na função Rastrigin. (a) Tipo I, onde os ótimos foram deslocados em 10 unidades ($g(t)$ = 10); (b) Tipo II, onde além de os ótimos serem deslocados em 10 unidades, a altura dos picos aumentou em 100 ($g(t)$=10, $h(t)$=100); (c) Tipo III, onde apenas os picos foram elevados ($g(t)$ = 1,2); e (d) Tipo IV, onde uma função ($g(x)=e^{-x/1,7}$) foi acrescentada ao objetivo.

Para a maioria dos algoritmos de busca, todos os tipos descritos acima são equivalentes, pois eles simplesmente procuram por novas soluções ao detectar mudanças no ambiente. Portanto, um algoritmo que obtém sucesso na otimização em um dos ambientes, possivelmente terá sucesso nos outros tipos também. Caso o tipo de mudança que ocorre

seja conhecido a priori, pode-se criar mecanismos que facilitam a busca dos novos ótimos, utilizando-se das características do ambiente. Neste trabalho, exceto quando especificado de outra forma, serão testados problemas do Tipo I.

2.6 Síntese do Capítulo

Neste capítulo foram apresentados os conceitos gerais de otimização que serão utilizados daqui em diante nessa dissertação, assim como uma introdução aos métodos clássicos de soluções para os problemas estudados, divididos em: métodos exatos, métodos de busca local, métodos heurísticos, métodos meta-heurísticos e computação bio-inspirada. Ao final do capítulo foi dedicada uma seção à explicação dos ambientes dinâmicos em otimização aqui estudados.

Parte II

Domínio Contínuo

Capítulo 3

Sistemas Imunológicos Artificiais

Sistemas imunológicos artificiais podem ser definidos como sistemas adaptativos inspirados em imunologia teórica e prática e desenvolvidos com o objetivo principal de resolver problemas complexos (de Castro & Timmis, 2002). Uma das aplicações propostas utilizando SIAs foi uma ferramenta de proteção de redes de computadores contra invasão, bem como ferramenta contra ataques por vírus de computador, recriando processos análogos à função do sistema imune em organismos vivos. Posteriormente, essa inspiração foi utilizada para resolver problemas diversos (de Castro, 2001), como reconhecimento de padrões, agrupamento de dados, aprendizado de máquina e otimização, que é o alvo desta dissertação.

Este capítulo apresenta um breve histórico da teoria e desenvolvimento desta meta-heurística, partindo da inspiração biológica e detalhando seus princípios e algoritmos. Por fim, será apresentado o algoritmo a ser analisado, melhorado e, então, adaptado para utilização junto a problemas de otimização em ambientes dinâmicos.

3.1 Sistemas Imunológicos e Imunologia

Uma das áreas atuais da biologia amplamente estudada e que gera grande controvérsia e teorias concorrentes é a imunologia, que é a área de pesquisa responsável pelo estudo dos sistemas imunológicos. Seu estudou foi motivado pelas tentativas em conter uma epidemia que assolava a Europa no século XVIII e a observação de que os bovinos da mesma região adquiriram a mesma doença mas não foram afetados por ela.

3.1.1 As Primeiras Descobertas em Imunologia

Os primeiros passos para a formalização desta teoria foram dados no final do século XVIII com os estudos e observações do médico experimental Eduard Jenner. Naquela época, a varíola tinha se tornado uma epidemia, matando uma em cada dez crianças, na Suécia e na França, e uma em cada sete, na Rússia e Inglaterra. Mas Jenner observou, durante algum tempo, que os bovinos também tinham desenvolvido uma espécie de varíola, só que muito mais branda, não causando suas mortes. Também foi observado que as pessoas que cuidavam desses animais e adquiriam essa doença não adquiriam a varíola humana. Diante desses fatos, Jenner propôs inocular pus da varíola bovina em crianças sadias e mais tarde inocular pus de um adulto gravemente doente nessas mesmas crianças. Da repetição desse mesmo experimento em adultos e da percepção de que eles não desenvolviam a doença, foi criada a primeira *vacina* (termo originário da palavra latina *vacca* que significa vaca).

Alguns anos mais tarde, o médico francês Louis Pasteur apresentou um estudo conhecido como "teoria germinal das enfermidades infecciosas", onde dizia que toda doença infecciosa era causada por microorganismos que tinham a capacidade de se multiplicar e se propagar entre as pessoas. Seu interesse por esses microorganismos causadores de enfermidades, então denominados *patógenos* (do grego *patho gen*, que significa "que causa sofrimento"), levaram a um estudo importante na área de imunologia, onde estes patógenos eram geralmente atenuados para serem inoculados no organismo humano e gerarem uma defesa contra os mesmos. Este estudo abriu caminho para a criação de vacinas preventivas contra doenças que afligiam os seres humanos na época, como a cólera.

3.1.2 Algumas Teorias de Funcionamento do Sistema Imunológico

Para um melhor entendimento do funcionamento do *sistema imunológico* (SI), vamos começar a análise no instante em que um patógeno é identificado. Cabe ressaltar que, na descrição que se segue, nem todos os agentes do sistema imunológico serão considerados e os processos envolvidos serão descritos de forma simplificada e, muitas vezes, aproximada. Assim que um conjunto de células brancas, chamadas *macrófagos,* encontram esse patógeno, inicia-se o processo de *fagocitose*, onde o patógeno é fragmentado. Várias partes que identificam o patógeno, denominadas *antígenos*, são apresentadas na superfície externa do macrófago. Os macrófagos que apresentam antígenos são enviados até um *linfonodo*, que são nódulos espalhados pelo corpo e servem como um local onde as células brancas (que são as células de defesa) de dois tipos diferentes, denominadas "células T" e "células B", obtêm informações sobre patógenos. As células B e T realizam funções específicas na defesa do organismo.

As células T têm várias funções, dependendo do seu tipo. As principais são as *células T destrutivas* (também chamadas de *células T citotóxicas* ou *células T matadoras*) e as *células T de defesa* (conhecidas também como *células T ajudantes*). As células T destrutivas, uma vez ativadas pelos antígenos, passam a procurar por células do próprio organismo que já foram infectadas pelo patógeno e as destroem injetando uma toxina. As células T ajudantes fornecem sinais químicos de alerta para induzir a produção de mais células B e células T destrutivas.

As células B, dentre outros papéis, têm a função de produzir os chamados *anticorpos*. Um dos principais papéis dos anticorpos é localizar os patógenos que ainda não infectaram uma célula do organismo, se ligar a estes patógenos e sinalizar para que outras células do sistema imune, como as células T e os macrófagos, os destruam. A produção dos anticorpos é determinada através dos antígenos apresentados pelos macrófagos. Para ajudar no entendimento, assuma que os antígenos possuem formas geométricas bem definidas. Os anticorpos são então produzidos como o complemento dessa geometria, formando um encaixe que tenha a capacidade de se ligar (grudar) a estes antígenos. Quanto melhor o encaixe, maior a afinidade.

Quando um anticorpo apresenta alta afinidade com um patógeno, as células B responsáveis pela sua produção passam a gerar *clones* (conjunto de células descendentes da mesma célula) produzindo outros anticorpos praticamente idênticos. Durante o processo de clonagem, os anticorpos dos clones sofrem pequenas variações genéticas (mutação) em sua

formação e, ao se ligarem aos antígenos, sinalizam para que algum macrófago ou célula T destrutiva elimine o complexo formado pelo antígeno e pelo anticorpo. Uma vez criada uma célula B com receptor (anticorpo) capaz de identificar um determinado patógeno, esta célula passa a persistir no organismo, resultando na criação de uma espécie de *memória imunológica* capaz de promover uma rápida reação do sistema imune caso volte a ocorrer essa mesma infecção ou alguma infecção suficientemente similar a esta. Estes processos de ligação ao antígeno, *expansão clonal*, mutação e seleção de células com receptores mais bem adaptados ao reconhecimento dos antígenos são denominados de *seleção clonal* (Burnet, 1955). O objetivo da seleção clonal é, portanto, permitir uma maturação das células B, no sentido de que os anticorpos passem a ser mais aptos a reconhecerem antígenos e formarem a memória imunológica.

Apesar desses mecanismos básicos de ligação entre antígenos e anticorpos serem bem conhecidos atualmente, ainda existe uma discussão acerca da dinâmica do sistema imunológico e dos mecanismos de regulação das respostas imunes. É possível identificar pelo menos quatro principais vertentes (de Castro, 2003):

- A Teoria do Auto-Reconhecimento (*Self-Nonself Theory*): também conhecida como *teoria do próprio/não-próprio*, criada por Bretscher & Cohn (1970), foi a primeira formalização sobre como o sistema imunológico combate infecções. Esta teoria defende que existe uma separação entre os elementos (p. ex. células e moléculas) que pertencem ao organismo e aqueles que não pertencem, sendo estes últimos considerados patógenos e promotores de uma resposta imunológica.
- A Teoria da Auto-Afirmação (*Self-Assertion View*): De acordo com esta perspectiva, não há dicotomia no comportamento do sistema imunológico, ou seja, o SI não se comporta distintamente para elementos próprios e não próprios (Bersini, 2002). O SI é visto como um sistema dinâmico que não requer estímulos externos para apresentar atividade; não há diferença fundamental entre próprio/não-próprio. Boa parte das propostas do SI como um sistema auto-afirmativo provém da *teoria da rede imunológica*, preconizada por Jerne (1974).
- A Teoria dos Múltiplos Sistemas (*Multi-Systemic View*): Esta teoria propõe que o sistema imunológico é um sistema vital integrado com outros sistemas orgânicos e, assim, compartilha mecanismos de reconhecimento, ativação e adaptação. Há fortes indícios da interdependência entre o sistema imune e os sistemas nervoso e endócrino através de moléculas mensageiras, neurotransmissores e hormônios, além de diversas analogias funcionais (Besendovsky & del Rey, 1996).
- Além dessas três vertentes, outra proposta que vem ganhando aceitação ao longo dos últimos anos é a chamada *teoria do perigo* (*Danger Theory*). Ela foi proposta por Matzinger (1994), e defende que o sistema imunológico é ativado apenas quando um microorganismo começa a causar danos ao organismo, como a destruição de células saudáveis que passam a liberar sinais de perigo ou dano.

A validade de todas essas teorias foi comprovada apenas parcialmente e, portanto, elas são alvo de muita discussão no meio científico, cada uma com suas linhas de argumentação próprias.

3.2 Sistemas Imunológicos Artificiais

O uso de teorias de funcionamento do sistema imunológico para fins computacionais começou a ter representatividade dentro da inteligência computacional há cerca de uma década e passou a ser reconhecida como uma área de pesquisa denominada Sistemas Imunológicos Artificiais (SIAs). Os sistemas imunológicos artificiais foram motivados, inicialmente, pela eficiência do SI na defesa do organismo e pela forma com que seu ambiente e funcionalidade se assemelha a alguns problemas em ambientes computacionais, como detecção de intrusão em redes de computadores e combate a vírus computacionais.

Como conseqüência das habilidades do SI e do sucesso dos SIAs na proteção de sistemas computacionais contra vírus e intrusos de rede, outras áreas passaram a aplicar diversos conceitos e modelos para tratar problemas nos mais variados domínios, como reconhecimento de padrões, análise de dados e, como será visto mais adiante, otimização de funções.

Recentemente, vários algoritmos baseados nos SIs foram formalizados de maneira a abranger diversas áreas da engenharia (de Castro, 2001). A seguir, serão apresentados alguns desses algoritmos.

3.2.1 CLONALG: Classificação e Otimização via Seleção Clonal

A inspiração de como os SIs maturam os anticorpos levou à criação de um algoritmo, conhecido como algoritmo de seleção clonal, CLONALG (*CLONal Selection ALGorithm*) (de Castro & Von Zuben, 2002), que pode ser aplicado tanto para aprendizagem de máquina e reconhecimento de padrões como para problemas de otimização. Esse processo de maturação, conforme explicado resumidamente na Seção 3.1.2, refere-se à adaptação de um anticorpo para a identificação de um antígeno. Toda vez que um anticorpo tem um certo sucesso em identificar e se ligar ao antígeno, as células que carregam anticorpos deste tipo iniciam um processo de proliferação, na qual uma grande quantidade de clones da célula sofre mutações na região de anticorpo, com o propósito de produzir anticorpos ainda mais adaptados para reconhecer o antígeno. A pressão seletiva durante este processo é muito grande e permite a eliminação de todos os anticorpos que não conseguem se adaptar ao antígeno.

CLONALG: Versão para Reconhecimento de Padrões

O processo de funcionamento do algoritmo consiste em gerar diversos anticorpos (reconhecedores, representados no algoritmo como **Ab**) aleatoriamente e apresentá-los a antígenos (padrões a serem reconhecidos, representados no algoritmo como **Ag**) selecionando aqueles que têm maior afinidade, iniciando então o processo de clonagem (conjunto de indivíduos **C** no algoritmo) e mutação (ou maturação) tentando aumentar a capacidade de reconhecimento desse anticorpo. O pseudocódigo apresentado no Algoritmo 3.1 resume a versão do CLONALG para reconhecimento de padrões. Repare que o

algoritmo não faz distinção entre as células que carregam anticorpos e os anticorpos propriamente ditos.

```
[Ab{m},F] = Function clonalg(Ab,Ag,M,L,max_it,n,β,d);
      t = 0;
      F = 0;
      Enquanto t < max_it E F < (M*L),
            Para j = 1 .. M,
                  f(j,:) = afinidade(Ab,Ag(j,:));
                  Ab{n}(j,:)= seleciona(Ab,f,n);
                  C(j,:) = clona(Ab{n},β,f);
                  C*(j,:) = hipermutação(C,f);
                  f(j,:) = afinidade(C*,Ag(j,:));
                  Ab* = seleciona(C*,f,1);
                  Ab{m}(j,:)= insere(Ab{m}(j,:),Ab*);
                  Ab{d} = gera(d,L);
                  Ab{r} = substitui(Ab{r},Ab{d},f);
            Fim
            f = afinidade(Ab{m},Ag);
            F = soma(f);
            t = t + 1;
      Fim
Fim
```

Algoritmo 3.1: Pseudo-código do algoritmo CLONALG para reconhecimento de padrões.

Como parâmetros de entrada para o algoritmo CLONALG, tem-se a população de anticorpos iniciais **Ab**, a população de antígenos a serem reconhecidos **Ag** de tamanho M e comprimento L (antígenos e anticorpos correspondem a um vetor de L elementos), o número máximo de iterações, o número de clones n, um fator multiplicativo β para o processo de clonagem (descrito abaixo), e o número d de anticorpos a serem substituídos por novos indivíduos gerados aleatoriamente.

No início do laço principal, é gerada a matriz **F,** cujos elementos f_{ij} representam a afinidade do anticorpo Ab_i com o antígeno Ag_j. A afinidade é uma medida da capacidade ou aptidão para o reconhecimento de antígenos. Em seguida, são selecionados os n anticorpos que melhor reconhecem o antígeno Ag_j. Estes passam por um processo de clonagem proporcional à afinidade f e pelo fator multiplicativo β, onde os indivíduos mais aptos geram mais clones. A população de clones é alterada através de um processo de *hipermutação*, conforme será explicado em detalhes a seguir. Esses clones mutados são comparados com o antígeno Ag_j e é calculada sua afinidade, seguindo os passos que serão mostrados na seqüência. O clone com maior afinidade é selecionado e inserido na população de memória (índice $\{m\}$). Novos indivíduos são gerados e substituem alguns indivíduos da população restante (índice $\{r\}$, $\mathbf{Ab \backslash Ab}_{\{m\}}$) proporcionalmente à sua afinidade. Após repetir esse processo para todos os antígenos, a matriz de afinidade é novamente gerada e a soma dos elementos dessa matriz é então calculada. O laço é quebrado quando o número de iterações atinge o máximo ou quando a soma das afinidades for igual a $M*L$, que significa que todos os anticorpos foram reconhecidos.

Cálculo da Afinidade – Função de aptidão

A função de aptidão deve refletir, de maneira concisa, o quanto uma determinada população está apta para reconhecer antígenos. Ela pode ser calculada utilizando uma métrica de distância, da seguinte forma:

$$E = \sum_{j=1}^{M} \min_{i} dist(Ab_i, Ag_j), \tag{3.1}$$

onde $dist(\mathrm{Ab}_i, \mathrm{Ag}_j)$ por exemplo, quando $Ab_i, Ag_j \in \Re^L$, refere-se à distância euclidiana entre os indivíduos *i* e *j*.

Já para um espaço discreto essa medida não é adequada e, para isso, pode-se utilizar uma medida de distância equivalente à distância de Hamming (*H*):

$$s(i) = \begin{cases} 1, & Ab_i \neq Ab_j, \forall j \neq i \\ 0, & \text{caso contrário} \end{cases}, \tag{3.2}$$

$$H = \sum_{i=1}^{N} s(i). \tag{3.3}$$

Caminho para a Solução Vizinha – Função hipermutação

A função de hipermutação também deve ser definida de forma diferente, dependendo da representação adotada (por exemplo, no espaço de Hamming ou espaço euclidiano). As mutações definidas são do tipo *mutação de múltiplos pontos*, onde cada elemento do vetor tem probabilidade p_m de mutação definida pelo usuário. Assim, os elementos escolhidos são substituídos aleatoriamente por outros elementos pertencentes ao alfabeto, no caso de espaço discreto, ou sofrem uma pequena perturbação, no caso de espaço contínuo, definida por:

$$\alpha = e^{-\rho E^*}, \tag{3.4}$$

onde ρ refere-se à taxa de decaimento da função exponencial e E^* refere-se ao valor da função de aptidão normalizada no intervalo [0, 1].

CLONALG: Versão para Otimização

Na adaptação do algoritmo CLONALG, para resolver problemas de otimização, algumas poucas alterações precisam ser feitas. Neste contexto, não há uma população de antígenos a ser reconhecida e, portanto, a afinidade será associada ao valor da função-objetivo para um dado anticorpo *i* da população **Ab**. Também não haverá um sub-conjunto de anticorpos específicos que irão compor a memória imunológica. Adicionalmente, após a clonagem dos anticorpos, *d* novos elementos serão inseridos no conjunto **Ab**.

```
[Ab,f] = Function clonalg(Ab,L,max_it,n,β,d)
      Para t = 1 .. max_it,
            f = avalia(Ab);
            Ab{n} = seleciona(Ab,f,n);
            C = clona(Ab{n},β,f);
            C* = hipermutação(C,f);
            f = avalia(C*);
            Ab{n} = seleciona(C*,f,n);
            Ab = insere(Ab,Ab{n});
            Ab{d} = gera(d,L);
            Ab = substitui(Ab,Ab{d},f);
      Fim
      f = avalia(Ab);
Fim
```

Algoritmo 3.2: Pseudo-código do algoritmo CLONALG para otimização.

Nessa versão do algoritmo, a função `avalia(·)` retorna um vetor coluna onde cada elemento f_i representa o valor da função-objetivo para o anticorpo $\mathbf{Ab}_i$. Da mesma forma que na outra versão do CLONALG, serão selecionados n anticorpos para sofrerem clonagem e mutação. Desses clones, os n melhores serão selecionados e inseridos na população **Ab**, e o processo se repete iterativamente, até que a condição de parada (número máximo de iterações) seja satisfeita.

3.2.2 aiNet: Agrupamento de Dados Usando Redes Imunológicas

O problema de agrupamento (*clusterização*) consiste em: a partir de um conjunto de dados ou amostras não rotuladas (sem classe definida), agrupá-los de maneira que amostras que contêm atributos similares pertençam ao mesmo grupo, ao mesmo tempo em que amostras que contêm atributos dissimilares pertençam a grupos distintos. Para isso, os algoritmos de agrupamento empregam medidas de similaridade ou dissimilaridade entre os dados de forma a agrupá-los baseado nessas relações (Everitt et al., 2001; Rui & Wunsch, 2005). Existem diversas áreas de aplicação para os algoritmos de agrupamento de dados, como mineração de dados (*data mining*), reconhecimento de padrões e compactação de informação.

Inspirado na teoria da rede imunológica proposta por Jerne (1974) e nos conceitos do CLONALG, de Castro e Von Zuben (2000) propuseram um modelo simplificado de rede imunológica artificial, denominada aiNet (Artificial Immune NETwork), para fazer agrupamento de dados e compactação de informação. Esse algoritmo consiste em gerar anticorpos (*protótipos*) que identificam grupos de antígenos (dados apresentados). O pseudocódigo da aiNet é apresentado no Algoritmo 3.3.

```
[Ab{m},S] = Função aiNet(Ag,L,max_it,n,ζ,σd,σs,d);
      Ab := gera(N0,L);
      Para t = 1 .. max_it,
            Para j = 1 .. M,
                  f(j,:) = afinidade(Ab,Ag(j,:));
                  Ab{n}(j,:) = seleciona(Ab,f(j,:),n);
                  C = clona(Ab{n},1,f(j,:));
                  C* = mutação(C,Ag(j,:),f(j,:));
                  f(j,:) = afinidade (C*,Ag(j,:));
                  M = seleciona(C*,f(j,:),ζ);
                  [M,f(j,:)] = suprime(M,1/f(j,:),σd);
                  S = afinidade (M,M);
                  [M,S] = suprime(M,S,σs);
                  Ab{m} = insere(Ab{m},M);
            Fim
            S = afinidade(Ab{m},Ab{m});
            [Ab{m},S] = suprime(Ab{m},S,σs);
            Ab{d} = gera(d,L);
            Ab = insere(Ab{m},Ab{d});
      Fim
Fim
```

Algoritmo 3.3: Pseudo-código do algoritmo aiNet.

De forma similar ao CLONALG, a aiNet inicia gerando uma população de anticorpos **Ab** de forma aleatória. Para cada antígeno apresentado à rede, o algoritmo calcula a afinidade do antígeno a todos os anticorpos, selecionando os n melhores e executando a clonagem seguida do processo de mutação:

$$\mathbf{C}_{\mathbf{k}}^{*} = \mathbf{C}_{\mathbf{k}} + \alpha(\mathbf{Ag}_{\mathbf{j}} - \mathbf{C}_{\mathbf{k}}), \tag{3.5}$$

onde o clone k do anticorpo i é movido na direção do antígeno j com um passo α ($0 < \alpha < 1$) proporcional à distância entre eles (quanto mais próximo menor o passo).

Após esse processo, são selecionados os ζ clones com menor similaridade em relação à população de anticorpos de acordo com a função de afinidade e estes são colocados em uma população de memória **M**. Em seguida, os elementos dessa população são suprimidos de duas maneiras: i) quando a afinidade deles em relação ao antígeno for menor ou igual a um limiar σ_d; ou ii) quando existirem dois elementos da população que tenham uma distância menor ou igual a um limiar σ_s, conhecido como limiar ou fator de supressão. Finalmente, os elementos que restaram entram para a população de anticorpos **Ab**. Em seguida, o segundo critério de supressão é aplicado a toda a população **Ab** e, então, d novos elementos são inseridos.

Vale notar, nesse ponto, que a aiNet contém uma característica incomum à maioria dos algoritmos populacionais, que é o controle dinâmico do tamanho da população. Através do processo de supressão, que reduz o tamanho da população, seguido de um passo de inserção que aumenta esse tamanho, o algoritmo gera um efeito de contração e expansão da população, onde após cada ciclo a quantidade de anticorpos na rede não será necessariamente o mesmo do início do processo adaptativo. Em um contexto de agrupamento de dados, essa funcionalidade é útil especialmente para identificar automaticamente quantos grupos existem nos dados analisados, algo que geralmente requer uma pré-análise do espaço de dados.

Para ilustrar o resultado desse algoritmo, suponha um conjunto de dados espalhados em um espaço bi-dimensional, como ilustrado na Figura 3.1. Neste exemplo, é possível visualizar facilmente a existência de 4 grupos distintos de pontos no espaço. O papel da aiNet para esse problema é posicionar um conjunto de protótipos (anticorpos) neste espaço, tais que eles representem cada um desses grupos.

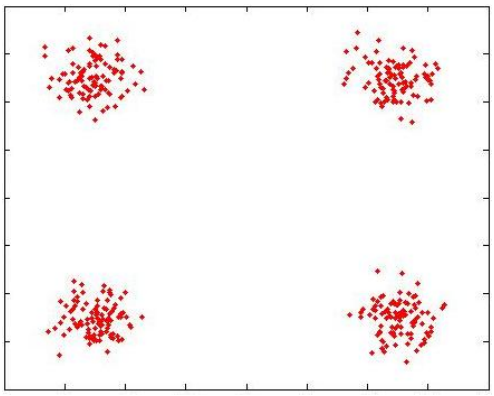


Figura 3.1: Conjunto de dados que devem ser representados e agrupados. Embora nesse caso seja fácil separá-los visualmente, a maioria dos problemas de agrupamento requer a análise de um conjunto de dados em um espaço de elevada dimensão, R^n, com n > 2, tornando impossível uma inspeção visual.

Executando o algoritmo da aiNet para este conjunto de dados, é possível obter uma saída como a apresentada na Figura 3.2. Neste exemplo a aiNet gerou trinta e dois anticorpos contendo mais de um representante por grupo, sendo que em média cada representante conta com 8 protótipos. Conforme já apresentado na proposta original da aiNet (de Castro & Von Zuben, 2001a), é possível aplicar procedimentos para determinação automática do número de grupos após a convergência da rede. Esta etapa não será apresentada aqui, mas a escolha mais usual é produzir uma árvore geradora mínima (Zahn, 1971), tomando os protótipos obtidos como entrada, e eliminar arestas inconsistentes, ou seja, aquelas que têm um comprimento bem maior que suas arestas adjacentes, indicando assim grande separação entre dados.

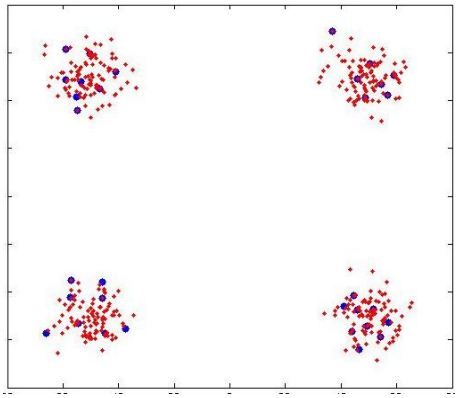


Figura 3.2: Saída do algoritmo aiNet para o conjunto de dados apresentado. Os pontos azuis representam os anticorpos obtidos ao final da execução. Note que o algoritmo retorna um pequeno conjunto de pontos em volta de cada cluster.

3.2.3 opt-aiNet: aiNet Estendida para Otimização Multimodal

Por causa da característica de população dinâmica no algoritmo aiNet e de sua possível utilidade na solução de problemas de otimização, em de Castro & Timmis (2002a), os autores propuseram uma versão da aiNet, denominada opt-aiNet (Artificial Immune NETwork for OPTimization), para resolver problemas de otimização multimodal.

O processo adaptativo é similar àquele feito na versão de otimização do algoritmo CLONALG (Algoritmo 3.2), onde no lugar dos antígenos a serem reconhecidos é utilizada uma função-objetivo para definir a adaptabilidade do anticorpo ao ambiente. O Algoritmo 3.4 resume a opt-aiNet.

```
[Ab,S] = Função opt-aiNet(N0,L,Nc,max_it,β,σs,d,ε);
      Ab := gera(N0,L);
      fit = f(Ab);
      Para t = 1 .. max_it,
            C = clona(Ab,1,Nc);
            C* = mutação(C,β,norm(fit));
            fit* = f(C*);
            M = seleciona(C*,fit*);
            [Ab,fit] = insere(Ab,fit,M,fit*);
            Se erro_médio < ε,
                  [Ab,fit] = suprime(Ab,σs);
                  D = gera(d,L);
                  fit_d = f(D);
                  [Ab,fit] = insere(Ab,fit,D,fit_d);
            Fim
      Fim
Fim
```

Algoritmo 3.4: Pseudo-código do algoritmo opt-aiNet.

O algoritmo começa gerando uma população inicial e avaliando o seu desempenho através de uma função-objetivo que tem o papel de medir a afinidade ou aptidão de cada anticorpo. Em seguida, é iniciado o laço principal do algoritmo, em que é feito um processo de clonagem. Cada anticorpo irá gerar exatamente *Nc* clones, sendo este número definido pelo usuário. Após esse passo, inicia-se o processo de mutação, no qual é gerado um vetor direção através de um gerador de números aleatórios gaussiano de média 0 e desvio padrão 1, $N(0,1)$. O anticorpo caminha nessa direção com um passo de tamanho α proporcional ao inverso do valor da função-objetivo normalizada (Eqs. (3.6) e (3.7)). Portanto, quanto melhor o anticorpo relativo aos demais, menor o tamanho do passo.

$$C_k^* = C_k + \alpha N(0,1), \tag{3.6}$$

$$\alpha = \frac{1}{\beta} e^{-norm(f)}, \tag{3.7}$$

onde β é um parâmetro definido pelo usuário que regula o tamanho do passo.

Em seguida, cada clone é avaliado e o melhor clone de cada anticorpo é selecionado para substituir o seu anticorpo-pai, caso tenha um desempenho melhor que o dele. Neste ponto, se o erro médio, ou seja, a diferença da média do valor da função-objetivo da

iteração passada com a iteração atual for menor que um determinado limiar ε, então é executado um processo de supressão. A supressão permite que anticorpos muito similares entre si, parâmetro definido pela distância euclidiana entre os anticorpos $\{i,j|dist(i,j) < \sigma_s\}$, compitam entre si e o pior seja eliminado. Nessa fase, também são gerados novos anticorpos e introduzidos na população. O processo então se repete até que o número máximo de iterações definido pelo usuário seja atingido.

Esse algoritmo herda a característica da aiNet de ter um controle dinâmico do tamanho da população e procedimentos explícitos para manutenção de diversidade, sendo útil no caso de otimização para identificar uma variedade de ótimos locais ao invés de apenas um, como é o caso da maioria dos algoritmos de busca e otimização. Essa é uma característica importante para a otimização em ambientes dinâmicos, conforme será verificado no próximo capítulo.

A opt-aiNet se difere do CLONALG principalmente no modo como o anticorpo reage ao ambiente, tanto através de seu valor objetivo quanto da sua similaridade com outros anticorpos. Uma grande vantagem da opt-aiNet em relação não apenas ao CLONALG, mas também a outros da mesma classe, é o controle dinâmico do tamanho da população. Essa característica o torna capaz de manter apenas um número de anticorpos reduzido e necessário para o processo de busca e identificar e manter protótipos em diversos ótimos locais simultaneamente. Estas são algumas das características necessárias para tratar um ambiente de otimização dinâmico e, portanto, a opt-aiNet será o alvo de estudo para o próximo capítulo, com a proposta de extensões relevantes.

3.3 Síntese do Capítulo

Neste capítulo, foram apresentados o histórico e conceitos de Sistemas Imunológicos Artificiais, iniciando pelo estudo do sistema imune humano até a criação de ferramentas de engenharia inspiradas em conceitos e teorias imunológicas já propostas. Algumas dessas ferramentas foram descritas em detalhes em uma ordem que mostrasse a sua evolução, terminando no algoritmo aplicado ao conceito de otimização multimodal denominado opt-aiNet, que será estudado em mais detalhes no próximo capítulo e estendido para a criação de um novo algoritmo capaz de lidar com ambientes dinâmicos.

Capítulo 4

Sistemas Imunológicos Artificiais para Ambientes Dinâmicos: dopt-aiNet

Este capítulo fará uma análise do algoritmo opt-aiNet apresentado anteriormente, identificando suas limitações e propondo melhorias para aumentar o seu desempenho em situações críticas. Em seguida, essas alterações serão unificadas e será proposto um novo algoritmo, denominado dopt-aiNet, para ser aplicado em ambientes dinâmicos. Este algoritmo será testado inicialmente para casos estáticos, visando avaliar seu potencial para melhoria de desempenho em relação à opt-aiNet e também a outros algoritmos da literatura. Depois de verificado o seu desempenho na otimização de funções estáticas, será especificado o ambiente para testes dinâmicos, onde seu desempenho será comparado a outros dois algoritmos a serem apresentados ainda neste capítulo.

4.1 Funções para Análise de Desempenho

Para avaliar o desempenho dos algoritmos apresentados nesse capítulo, serão investigadas quatro funções com características distintas, conforme descrito na Tabela 4.1.

Tabela 4.1: Funções contínuas que serão utilizadas durante os testes em ambientes estáticos. A primeira coluna refere-se ao nome com que a função é conhecida, a segunda coluna apresenta a função propriamente dita, enquanto a terceira coluna aponta a dimensão do espaço de busca. Já a quarta refere-se ao domínio que será utilizado nos testes, e as duas últimas ao vetor de *x* ótimo e seu valor de função correspondente.

Nome da Função	***f(x)***	***N***	**Domínio de *x***	x^*	$f(x^*)$
Esfera	$f_1(\mathbf{x}) = \sum_{i=1}^{N} x_i^2$	30	[–100; 100]	**0**	0
Rosenbrock	$f_2(\mathbf{x}) = \sum_{i=1}^{N-1} (100.(x_{i+1} - x_i)^2 + (x_i - 1)^2)$	30	[–30; 30]	**1**	0
Rastrigin	$f_3(\mathbf{x}) = \sum_{i=1}^{N} (x_i^2 - 10.\cos(2.\pi.x_i) + 10)$	30	[–5,12; 5,12]	**0**	0
Griewank	$f_4(\mathbf{x}) = \frac{1}{4000} \sum_{i=1}^{N} x_i^2 - \prod_{i=1}^{N} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	30	[–600; 600]	**0**	0

Essas funções são amplamente utilizadas na literatura por apresentarem, individualmente, características que as tornam desafiadoras para ferramentas de otimização, com exceção da função Esfera que tem o único propósito de testar a convergência do algoritmo. A função *Esfera* (Figura 4.1a) é uma função simples que contém um único ótimo (unimodal) e é utilizada como primeiro teste para verificar se o algoritmo tem capacidade de convergir em situações em que algoritmos de busca convencionais conseguem a solução de forma rápida. A função *Rosenbrock* (Figura 4.1b) tem uma característica muito interessante, pois apresenta uma inclinação muito pequena de sua curva criando uma superfície de decrescimento lento, e retardando a convergência da maioria dos algoritmos. As duas últimas funções, *Rastrigin* (Figura 4.1c) e *Griewank* (Figura 4.1d), apresentam uma superfície com muitos ótimos locais e com alturas parecidas, dificultando o processo de busca pelo ótimo global. Note que a Figura 4.1 contém apenas uma representação dessas funções no espaço $\Re^2$ mas os experimentos efetuados nesse trabalho serão no espaço $\Re^{30}$.

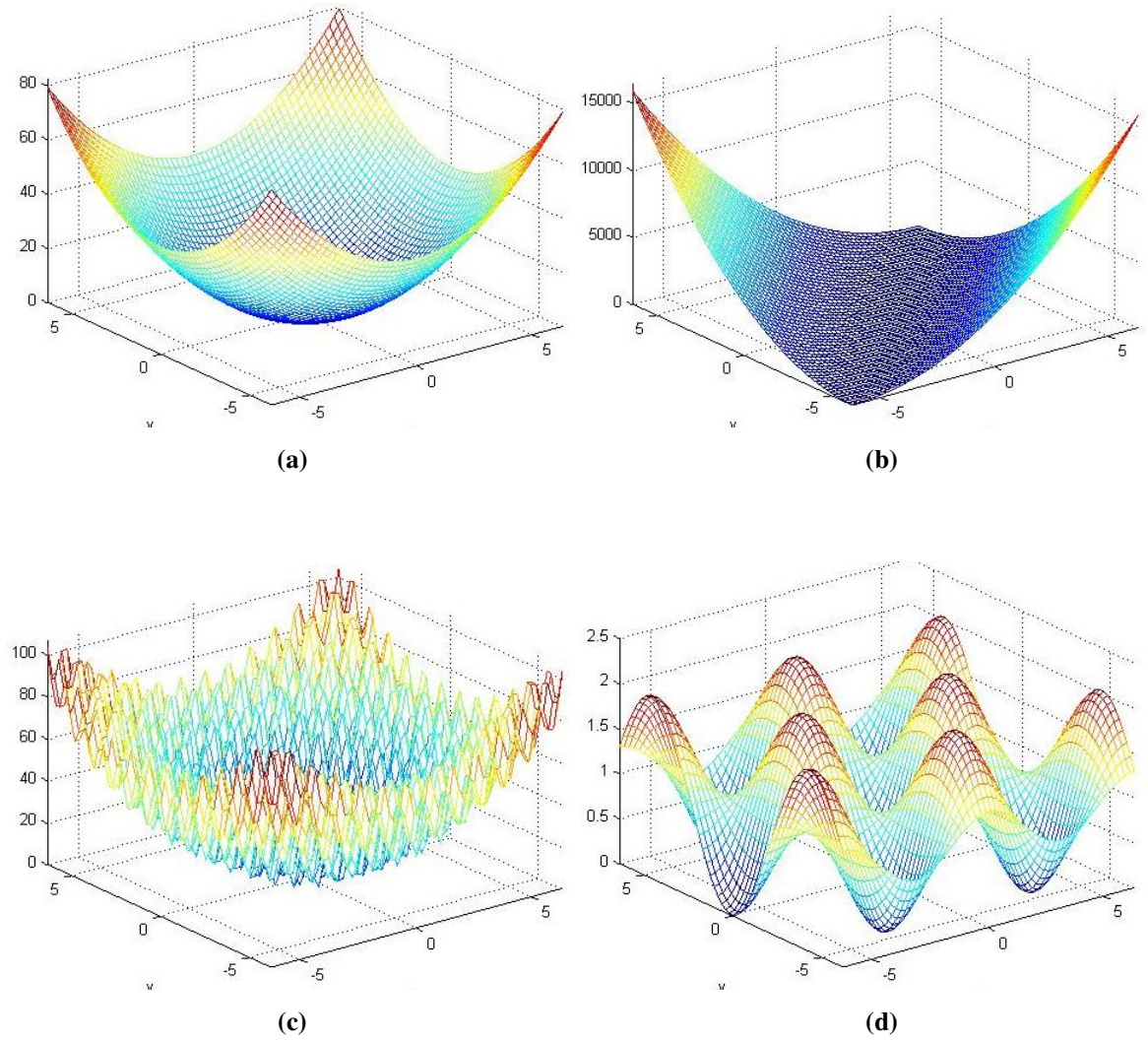


Figura 4.1: Funções utilizadas na análise de desempenho do algoritmo opt-aiNet. (a) Função *Esfera*, (b) Função *Rosenbrock*, (c) Função *Rastrigin* e (d) Função *Griewank*. Para efeito de visualização, o espaço de busca foi reduzido de 30 para 2 dimensões.

4.2 Análise de Desempenho da opt-aiNet

Para verificar o desempenho da opt-aiNet diante de funções com características diferentes e em grandes dimensões, foram utilizadas as 4 funções descritas acima rodando 30 testes com no máximo 5.500.000 avaliações de função (aproximadamente 50.000 iterações da opt-aiNet quando gerados 10 clones por indivíduo e quando a população permanece fixa com tamanho 10), conforme sugerido em de Castro & Timmis (2002a). O erro tolerado para os ótimos foi da ordem de 10^{-3}. Os resultados são descritos na Tabela 4.2. Esse limite de avaliações de funções foi utilizado, pois foi percebido durante experimentos preliminares que o algoritmo opt-aiNet nem sempre convergia para a solução de ótimo global nas condições do experimento.

Como é possível ver na Tabela 4.2, os resultados para as funções f_3 e f_4 estão longe do erro de tolerância. O comportamento observado nessas funções foi de muitos ótimos locais encontrados no espaço de busca em f_3, fazendo com que os anticorpos perdessem tempo em ótimos locais de baixa qualidade. No caso da função f_4, foi observada uma convergência muito lenta, além da presença de múltiplos ótimos.

Tabela 4.2: Valores obtidos pela opt-aiNet nas funções estudadas: mínimo refere-se ao menor valor da função-objetivo obtido nas 30 rodadas, máximo refere-se ao maior, e médio é o valor médio obtido acompanhado do desvio padrão, quando houver.

		opt-aiNet		
Função	**Ótimo Global**	**Mínimo**	**Máximo**	**Médio**
f_1	0	0,00	0,00	0,00
f_2	0	0,08	0,43	0,21 ± 0,08
f_3	0	121,45	185,12	153,54 ± 13,58
f_4	0	220,78	433,97	340,70 ± 61,94

Tabela 4.3: Número de avaliações de funções para se chegar aos resultados da Tabela 4.2 pelo opt-aiNet nas funções estudadas: mínimo refere-se ao menor valor obtido nas 30 rodadas, máximo refere-se ao maior, e médio é o valor médio obtido acompanhado do desvio padrão, quando houver.

	opt-aiNet		
Função	**Mínimo**	**Máximo**	**Médio**
f_1	2.519.630	3.715.230	$3.109.986 \pm 3{,}6\times10^5$
f_2	5.500.000	5.500.000	5.500.000
f_3	5.500.000	5.500.000	5.500.000
f_4	5.500.000	5.500.000	5.500.000

Tabela 4.4: Número final de anticorpos obtidos pelo opt-aiNet nas funções estudadas: mínimo refere-se ao menor valor obtido nas 30 rodadas, máximo refere-se ao maior, e médio é o valor médio obtido acompanhado do desvio padrão, quando houver.

	opt-aiNet		
Função	**Mínimo**	**Máximo**	**Médio**
f_1	9	10	9,90 ± 0,30
f_2	2	8	5,20 ± 1,13
f_3	200	210	205,00 ± 5,08
f_4	10	10	10,00

4.3 Pontos Críticos da opt-aiNet

Como visto na seção anterior, a opt-aiNet tem seu desempenho degradado em problemas de grandes dimensões, necessitando de um tempo maior para atingir um valor aceitável da função-objetivo e nem sempre encontrando um valor adequado nesse tempo. Nesta seção, iremos analisar cada um dos pontos críticos do algoritmo e propor soluções visando aumentar o desempenho.

4.3.1 Mutação

A mutação deve ser modelada de tal forma que através dela seja possível chegar a um ponto de ótimo local na vizinhança de um determinado anticorpo. A forma como a mutação está definida na opt-aiNet requer um parâmetro que determina o tamanho do passo que o clone irá efetuar na direção dada por um vetor gaussiano gerado aleatoriamente. A dificuldade de escolher o tamanho certo reside no fato de que o passo não pode ser muito grande e nem muito pequeno: i) se o passo for muito grande, o algoritmo tenderá a passar direto pelo ótimo e não mais alcançá-lo; ii) caso o passo seja pequeno demais, o algoritmo irá percorrer o espaço de busca muito lentamente, podendo perder direções que o levariam mais próximo do ótimo, caso elas fossem melhor aproveitadas.

Outro aspecto da mutação refere-se ao vetor direção que é gerado. Como este é definido por um vetor gaussiano aleatório, ele tem igual probabilidade de ter valor negativo e positivo entre seus elementos. Quanto maior a dimensão do espaço de busca, mais distribuída ficará a proporção de elementos negativos e positivos. Essa característica elimina uma grande parte dos possíveis vetores que tenham todos ou maioria de seus valores positivos ou negativos, tornando assim certos pontos do espaço de busca inatingíveis em tempo razoável.

Ainda na mutação, poderiam ser aproveitadas informações presentes nos melhores indivíduos para que estes gerassem indivíduos diferentes (fora de sua vizinhança) de forma que herdassem características dos anticorpos de origem.

4.3.2 Supressão

O outro ponto crítico do algoritmo opt-aiNet é referente ao controle de diversidade, ou função de supressão, que remove todos os anticorpos que residem na mesma vizinhança que outro anticorpo de melhor valor.

O problema principal é determinar se dois pontos fazem parte da mesma vizinhança de forma precisa e sem conhecer a superfície de adaptação gerada pela função-objetivo. No algoritmo original, esta tarefa é feita simplesmente utilizando a distância euclidiana entre dois pontos, necessitando de um parâmetro σ_s que define a distância mínima para o corte. A dificuldade reside justamente em determinar qual o melhor valor para este parâmetro, uma vez que um valor muito pequeno faz com que o algoritmo demore a detectar pontos parecidos, e um valor muito grande pode fazer com que a vizinhança de dois ótimos seja detectada como apenas uma só. Além disso, ainda há casos de funções em que, em certas regiões do espaço de busca, um valor pequeno seria o ideal, enquanto em outros um valor grande é necessário. A Figura 4.2 ilustra as dificuldades na definição do limiar de supressão σ_s.

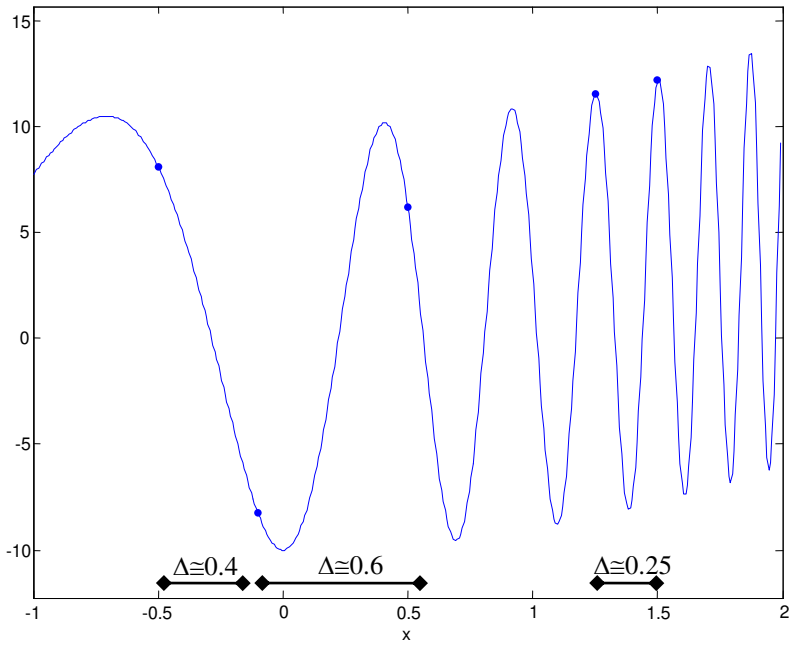


Figura 4.2: Pontos na função associados a ótimos locais diferentes têm uma distância menor do que pontos associados ao mesmo ótimo local. Uma escolha errada de parâmetros poderia causar uma eliminação equivocada, ou então fazer com que várias células que deveriam ser eliminadas permanecessem na população.

4.3.3 Custo Computacional Excessivo

Como visto nos experimentos reportados acima, em algumas situações o número de anticorpos da população cresce de uma forma expressiva e nem sempre produtiva. Isso torna o processo de otimização lento, pois muitos destes anticorpos poderão convergir para um ótimo local de má qualidade. Além disso, uma vez que um anticorpo converge para um ótimo, não existem critérios para parar o processo de clonagem e mutação deste anticorpo, gerando um custo computacional adicional e desnecessário, já que o anticorpo encontra-se em um ótimo local e, portanto, não é possível melhorar localmente a qualidade de sua solução.

4.4 dopt-aiNet: Extensões da opt-aiNet

Nesta seção, serão apresentadas melhorias no algoritmo opt-aiNet de forma a atenuar suas limitações explicitadas anteriormente, melhorar seu desempenho e possibilitar sua aplicação a ambientes dinâmicos de forma competitiva.

Com o objetivo de melhorar a qualidade de soluções e o tempo necessário para obtê-las, foram propostas melhorias visando englobar todas as características desejáveis para otimização em ambientes dinâmicos, criando assim novas funcionalidades do algoritmo ou melhorando as já existentes. Essas melhorias compreendem a criação de uma memória de boas soluções, busca no tamanho do passo da mutação, duas novas mutações que

introduzem novos tipos de vizinhança, um novo algoritmo de supressão mais preciso e elaborado, e um limite do tamanho da população, de forma a evitar gastos computacionais excessivos.

4.4.1 Divisão da População: População Ativa × População de Memória

Como apontado anteriormente, foi observado experimentalmente que, em determinado momento do processo de busca, um anticorpo atinge um ponto do espaço de busca a partir do qual ele não consegue gerar um clone melhor, ou por estar em um ótimo local ou pela natureza da mutação. Neste momento, o custo computacional aplicado a este anticorpo durante o processo de clonagem e mutação é desperdiçado. Para economizar recursos computacionais, os anticorpos incapazes de melhorar seu desempenho são colocados em uma população separada, denominada *população de memória*.

Para determinar quando um anticorpo deve fazer parte da população de memória, um processo de *recompensa* e *punição* é executado da seguinte forma. Inicialmente, para cada anticorpo um valor *rank* é designado e, toda vez que esse anticorpo apresenta melhoras em uma mutação, este valor é incrementado. Caso contrário, *rank* é decrementado. Quando este valor chega a zero, o indivíduo é transferido para outra população onde ele permanece apenas para identificar, ao final do processo adaptativo, um certo ótimo local ou global. O resultado desta simples heurística é que o algoritmo não desperdiçará mais tempo de processamento computacional com anticorpos possivelmente localizados em ótimos locais, bastando armazená-los em uma população de memória.

4.4.2 Busca Unidimensional: Seção Áurea

O primeiro aspecto levantado sobre a opt-aiNet foi em relação ao tamanho do passo que o clone deve dar em uma dada direção. Em algoritmos clássicos de otimização, como os métodos de gradiente e de Newton, dada a direção de maior decrescimento o tamanho desse passo é calculado através de uma busca unidimensional. Tendo o ponto inicial e a direção de caminhada, o problema de encontrar o parâmetro α assume a forma:

$$\mathbf{min}\ f(\mathbf{x} + \alpha\mathbf{d}), \tag{4.1}$$

onde **x** é o ponto atual, **d** a direção de busca e α o tamanho do passo.

Note que este problema resulta numa busca em uma única dimensão, ou seja, em função de α. Existem diversos métodos que encontram o ótimo global para esse problema, mas a maioria necessita da informação de primeira derivada da função. Um método que possui características de convergência global para funções unimodais e convexas, sem a necessidade de informações sobre a função, é o método de *seção áurea* (Bazaraa et al., 1993). Esse método é baseado nos algoritmos de enumeração para programação não-linear, nos quais são definidos intervalos fechados do espaço de busca cada vez menores em torno do ótimo global.

O ponto chave desse algoritmo é a forma como ele subdivide o intervalo de busca utilizando um número conhecido como *razão áurea*. A razão áurea é um número encontrado em proporções de figuras geométricas diversas na natureza, como em espirais nas conchas de moluscos, o crescimento de uma concha sem alteração de seu formato, a razão entre a medida de nosso braço e o antebraço, e muitos outros. Essa medida tem sido

utilizada, de longa data, em projetos arquitetônicos da Grécia antiga e até em pinturas feitas por Leonardo da Vinci. A razão áurea é representada pela letra grega φ e seu valor é o número irracional $\varphi = \frac{1+\sqrt{5}}{2}$ que vale aproximadamente 1,618.

O método da seção áurea funciona da seguinte forma: dado um intervalo fechado pré-estabelecido [a,b], dois pontos dentro desse intervalo são gerados utilizando a razão áurea φ de forma que:

$$\alpha = a + (1 - [\varphi - 1]).(b - a) \text{ e} \tag{4.2}$$

$$\beta = a + (\varphi - 1).(b - a). \tag{4.3}$$

O método da seção áurea está resumido no Algoritmo 4.1.

```
[alpha] = Função aurea(x0,d,a,b,f,ε);
      r = (√5-1)/2;
      α = a + (1-r)*(b-a);
      β = a + r*(b-a);
      y1 = f(x0 + α *d);
      y2 = f(x0 + β *d);
      Enquanto |b-a|> ε,
            Se y1>y2,
                  a = α;
                  α = β;
                  y1=y2;
                  beta = a + r*(b-a);
                  y2 = f (x0 + β *d);
            Senão,
                  b = β;
                  β = α;
                  y2=y1;
                  α = a + (1-r)*(b-a);
                  y1 = f(x0 + α *d);
            Fim;
      Fim;
      α = (b+a)/2;
Fim
```

Algoritmo 4.1: Método da seção áurea para busca unidimensional (versão para minimização).

O algoritmo começa escolhendo dois pontos, α e β, especificados por φ dentro do intervalo dado. Em seguida, esses pontos são avaliados e o laço principal iniciado. Caso α seja pior que β, o novo intervalo fica sendo [α, *b*] e um novo β é calculado para este intervalo. Caso contrário, o novo intervalo será [a, β] e um novo α é calculado. O laço é repetido até que o tamanho do intervalo seja menor que um limiar pré-especificado.

Para garantir a convergência global deste procedimento, a função deve necessariamente ser convexa, unimodal e não apresentar descontinuidade na direção estabelecida. Em uma função convexa no intervalo *I* definido por [*a*, *b*], tem-se que para todo $x_1, x_2 \in I$, sendo que $f(x_1) \neq f(x_2)$ e $x_1 < x_2$:

$$f(\lambda x_1 + (1-\lambda)x_2) < \mathbf{max}\{f(x_1), f(x_2)\}, \tag{4.4}$$

e também que:

$$\begin{array}{l} Se\ f(x_1) > f(x_2), então\ f(x) \geq f(x_2)\ para\ todo\ x \in [a, x_1) \\ Se\ f(x_1) \leq f(x_2), então\ f(x) \geq f(x_2)\ para\ todo\ x \in (x_2, b] \end{array}. \quad (4.5)$$

Em outras palavras, é possível reduzir o intervalo verificando dois pontos distintos pertencentes ao mesmo.

Para que o algoritmo funcione, deve-se escolher um intervalo em que a função seja convexa. Uma forma de aumentar as chances de isso ocorrer é percorrer espaços reduzidos, de maneira que a função tenha maiores probabilidades de ser unimodal dentro do intervalo. Visando aumentar a chance de convergência global para funções não convexas, foi definido que a seção áurea será executada em 4 intervalos idênticos, definidos a partir do intervalo original de busca, que é feito pelo usuário. Embora ainda assim possam ocorrer falhas no processo de busca, elas diminuem substancialmente se levarmos em conta que o processo é executado para vários clones de vários anticorpos e em várias iterações.

4.4.3 Novos Procedimentos de Mutação

Para complementar a mutação gaussiana e tornar possível uma exploração mais ampla do espaço de busca, dois novos procedimentos de mutação foram criados. O primeiro irá agir em conjunto com a mutação gaussiana e irá englobar as direções que esta geralmente não percorre. Já o segundo irá permitir que um anticorpo se movimente de uma região para outra. Esta última, na verdade, gera um novo anticorpo na população.

Mutação Unidimensional

O procedimento de *mutação unidimensional* gera um conjunto de direções que são praticamente impossíveis de serem obtidas utilizando uma distribuição gaussiana. Ele gera uma matriz **D** com elementos $d_{ii} = 1$ e $d_{ij} = 0$ para $i \neq j$, ou seja, faz com que seja possível percorrer uma dimensão de cada vez, dando o nome à mutação.

Adicionalmente, também são utilizados os vetores direcionais **1** e **−1** (compostos por todos os elementos iguais a 1 e −1, respectivamente) e a matriz gaussiana da mutação tradicional contendo *Nc* direções. Ou seja, a partir de agora serão gerados $Nc + n + 2$ clones para mutação, sendo *n* a dimensão do espaço de busca.

Determinadas as direções, as demais etapas ocorrem de forma similar à mutação tradicional. O pseudocódigo para a mutação unidimensional é apresentado no Algoritmo 4.2.

```
[C] = Função mut_unidimensional(C,f)
      G = gaussiana();
      D = [identidade(n); 1; -1; G]
      Para cada vetor "c" da matriz C de clones faça,
            Para cada linha "d" da matriz D faça,
                  α = aurea(c,d,f)
                  c' = c + α*d
                  Se f(c') é melhor do que f(c),
                        c = c'
                  Fim
            Fim
      Fim
Fim
```

Algoritmo 4.2: Algoritmo de Mutação Unidimensional.

No Algoritmo 4.2, os parâmetros de entrada são a matriz de clones **C** e a função a ser otimizada *f*. A função `identidade`(*n*) gera uma matriz identidade *n*×*n*, e a função `aurea`(**c**,**d**,*f*) retorna o tamanho do passo na direção **d** partindo do ponto **c**, como explicado anteriormente. Note que essa mutação já inclui a mutação gaussiana, tornando desnecessária a execução desta parte em separado.

Mutação por Duplicação Gênica

A segunda mutação proposta é inspirada em um processo de mutação biológica denominado *duplicação gênica* (*gene duplication*), na qual no momento em que a cadeia de DNA está sendo transcrita, partes de seus elementos são duplicadas e transcritas em outras posições. Em Holland et al. (1994), Ohno (1970), foi observado que essa mutação teve grande participação na evolução das espécies.

Utilizando dessa idéia para otimização de funções, foi criado o algoritmo de *mutação por duplicação gênica*, que atua em um vetor **x** de dimensão $n > 2$.

```
[x] = Função mut_dupgenica(x,f)
      Dup = x[rand(1,n)]
      x' = x;
      Para cada elemento de x faça,
            x'[j] = Dup;
            Se f(x') for melhor do que f(x),
                  x[j] = x'[j]
            Senão
                  x'[j] = x[j]
            Fim
      Fim
Fim
```

Algoritmo 4.3: Algoritmo de Mutação por Duplicação Gênica.

A função recebe como parâmetro de entrada um vetor candidato à solução e a função a ser otimizada. Primeiramente, é sorteado um elemento desse vetor, colocando-o na variável `Dup`, e é gerada uma cópia desse vetor para ser modificada. Em seguida, para cada elemento do vetor copia-se o valor de `Dup` e verifica-se se houve melhora. Em caso

negativo, o valor original é restaurado. Ao final do laço, o vetor modificado é retornado, mas não substitui o vetor original na população e sim é inserido como um novo elemento.

4.4.4 Supressão dos Anticorpos por Distância de Segmento de Reta

O objetivo principal do controle de similaridade ou supressão de anticorpos é que não existam dois ou mais anticorpos identificando um mesmo ótimo. Para isso, é preciso definir uma função de vizinhança $N(\mathbf{x})$ que compreenda todo o espaço em volta do ponto $\mathbf{x}$ e que envolva apenas um ponto de ótimo. Essa vizinhança deve representar um espaço convexo.

O problema em usar a distância euclidiana como medida de vizinhança é a dificuldade em escolher um limiar de supressão para a distância entre anticorpos, uma vez que ele não é de fácil determinação e não poderia ser único para uma função como aquela ilustrada na Figura 4.2. Nestes casos, pontos que estão associados ao mesmo ótimo local e deveriam ser eliminados têm uma distância maior entre si do que pontos que pertencem a ótimos locais diferentes. Portanto, ao definirmos um limiar de supressão muito grande, vários anticorpos que identificam ótimos diferentes serão cortados; caso contrário, se o limiar for muito pequeno, o algoritmo irá demorar a perceber a similaridade entre os anticorpos e acarretará em um gasto excessivo de tempo computacional.

Para minimizar este problema, foi idealizado um novo algoritmo de supressão no qual, para definir se dois pontos $P_1 = [\mathbf{x}_1, f(\mathbf{x}_1)]$ e $P_2 = [\mathbf{x}_2, f(\mathbf{x}_2)]$ estão na mesma região de vizinhança, é medida a distância entre o ponto médio $P = \left[\frac{\mathbf{x}_1 + \mathbf{x}_2}{2}, f\left(\frac{\mathbf{x}_1 + \mathbf{x}_2}{2} \right) \right]$ e o segmento de reta definido por esses dois pontos. Se P_1 e P_2 estiverem associados ao mesmo ótimo local, essa distância tende a ser muito menor do que no caso em que os pontos estão associados a ótimos locais distintos.

Este procedimento foi denominado de *algoritmo de supressão por distância de segmento de reta*, está ilustrado nas Figuras 4.3 e 4.4 (para o caso de maximização) e é apresentado sob a forma de pseudocódigo no Algoritmo 4.4.

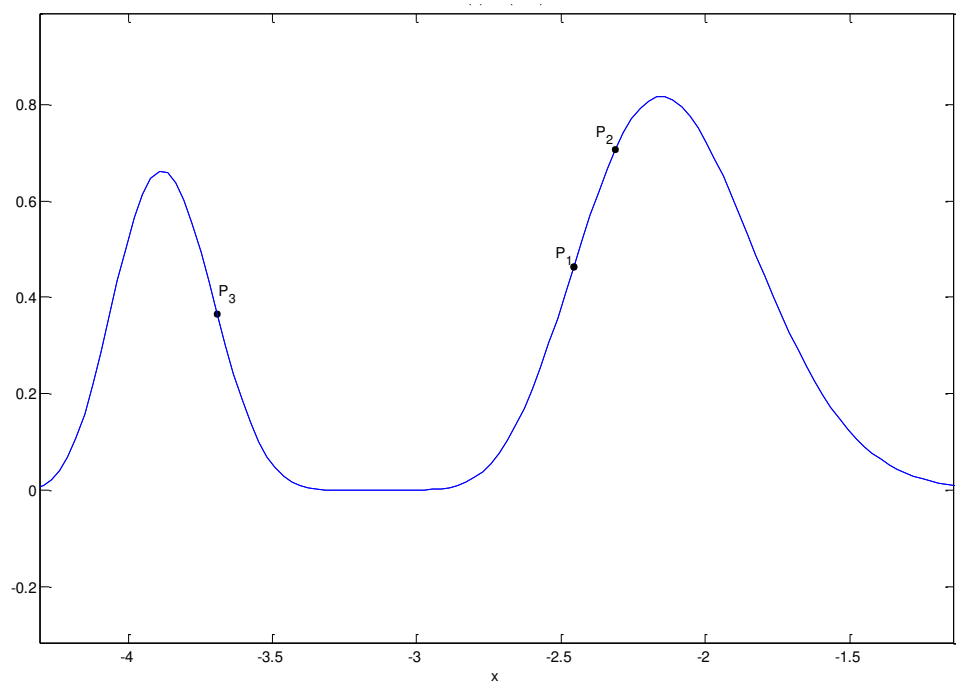
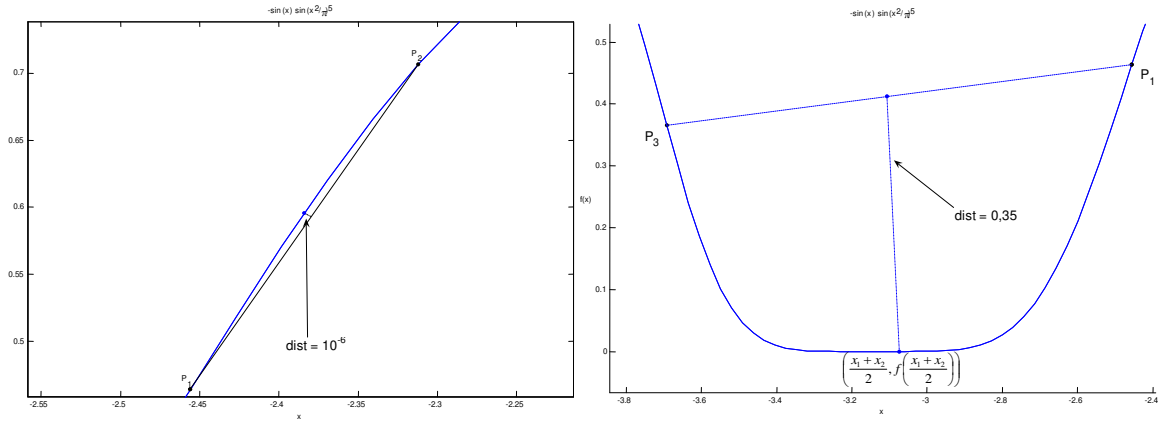


Figura 4.3: Exemplo do algoritmo de Supressão por Distância de Segmento de Reta. Desses três pontos, P_1 e P_2 estão associados a um mesmo ótimo local e um deles deve ser eliminado, e P_3 está associado a um outro ótimo local e deverá permanecer.



(a) (b)

Figura 4.4: Analisando de perto cada segmento formado, em (a) temos uma distância muito pequena em relação ao ponto médio na curva e a reta, mas em (b) essa distância se mostra muito maior.

```
[X] = Função supressão_reta(X, σs)

  Para cada par de anticorpos x1,x2 faça,
      P1 = [x1, f(x1)];
      P2 = [x2, f(x2)];
      P = [x1+0.5(x2-x1), f(x1+0.5(x2-x1))];
      v = P2 – P1;
      w = P – P1;
      c1 = w.v;    % produto escalar dos vetores w e v
      c2 = |v|;    % norma do vetor v
      b = c1/c2;   % projeção de P no segmento de reta P1P2
      Se c1 <= 0, % ponto mais próximo está em P1
            d = dist(P,P1);
      Senão Se c2 <= c1, % ponto mais próximo está em P2
            d = dist(P,P2);
      Senão, % ponto mais próximo está no ponto Pb onde PPb ⊥ P1P2
            Pb = P1 + b.v;
            d = dist(P,Pb);
      Fim
      Se d < σs,
            elimina o anticorpo com o pior valor de função-objetivo.
      Fim
  Fim
Fim
```

Algoritmo 4.4: Algoritmo de Supressão por Distância de Segmento de Reta.

Inicialmente, os pontos P_1, P_2 e P (ponto central de P_1 e P_2) são criados e os dois vetores $\mathbf{v} = \overline{P_1P_2}$ e $\mathbf{w} = \overline{PP_1}$ são calculados. O ponto P é então projetado no segmento de reta definido por **v** (Figura 4.5). Essa projeção nos leva ao ponto P_b pertencente à reta $\overline{P_1P_2}$, onde o ponto P forma 90º, e é calculado por $\frac{\mathbf{w}.\mathbf{v}}{|\mathbf{v}|}$. Caso o ponto P_b pertença ao segmento, é calculada então a distância entre P e P_b; caso contrário, é utilizada a distância menor entre *dist*(P,P_1) e *dist*(P,P_2) para formar uma medida mais justa (Figura 4.6).

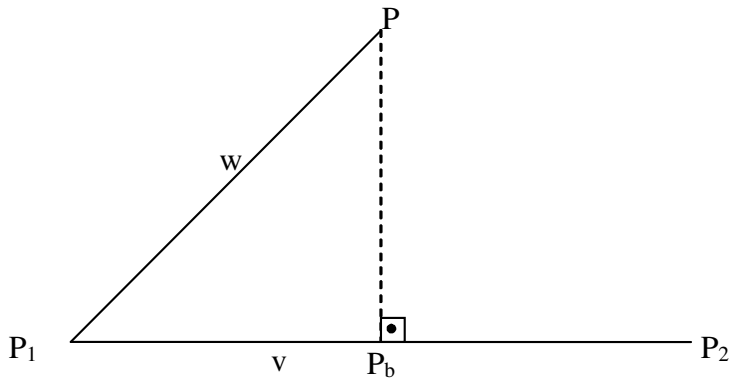


Figura 4.5: Projeção do ponto médio P na reta $\overline{P_1P_2}$.

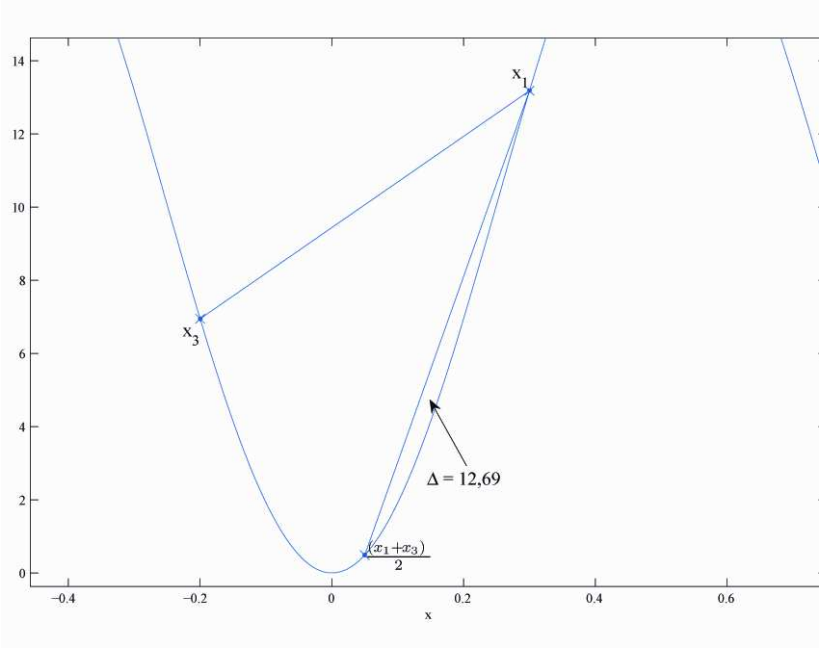


Figura 4.6: Caso em que o ponto P_b se encontra fora do segmento $\overline{P_1P_2}$.

Se esta distância for menor que o limiar de supressão, então o anticorpo com menor valor de função-objetivo é eliminado da população. Uma das vantagens desse algoritmo reside justamente no fato de que, uma vez que re-escalonamos a função-objetivo ($g(\mathbf{x}) = K.f(\mathbf{x})$), a distância entre o segmento de reta e a função aumenta de forma insignificante nos casos em que os dois pontos do segmento estão associados a um mesmo ótimo local. Nos outros casos, essa distância aumenta proporcionalmente a K. Essa propriedade será demonstrada a seguir.

Suponha dois pontos $P_1 = [\mathbf{x}_1, \mathbf{y}_1]$ e $P_2 = [\mathbf{x}_2, \mathbf{y}_2]$ onde $\mathbf{y}$ é definido pela função-objetivo $f(\mathbf{x})$. Foi definido que $g(\mathbf{x})$ é a função $f(\mathbf{x})$ escalonada por uma constante K suficientemente grande que define os pontos $P_1' = [\mathbf{x}_1, K.\mathbf{y}_1]$ e $P_2' = [\mathbf{x}_2, K.\mathbf{y}_2]$.

Calculando $P_b = \left[\frac{x_1 + x_2}{2}, Y_m\right]$ e $P_b^{'} = \left[\frac{x_1 + x_2}{2}, K \cdot Y_m\right]$, onde $Y_m = f\left(\frac{x_1 + x_2}{2}\right)$.

A reta $\overline{P_1P_2}$ é definida por:

$$P_1P_2 : \begin{vmatrix} x & y & 1 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{vmatrix} = \quad , \tag{4.6}$$
$$(y_1 - y_2)\cdot x + (x_2 - x_1)\cdot y + (x_1 \cdot y_2 - x_2 \cdot y_1) = 0$$

e a distância entre um ponto qualquer e esta reta é calculada como:

$$d[(x_0, y_0), ax + by + c] = \frac{|ax_0 + by_0 + c|}{\sqrt{a^2 + b^2}}. \tag{4.7}$$

Então, a distância de P_b até este segmento assume a forma:

$$D_1 = d[P_0, P_1P_2] = d\left[\left(\frac{x_1 + x_2}{2}, Y_m\right), P_1P_2\right] =$$

$$= \frac{\left|(y_1 - y_2)\cdot \frac{x_1 + x_2}{2} + (x_2 - x_1)\cdot Y_m + (x_1y_2 - x_2y_1)\right|}{\sqrt{(y_1 - y_2)^2 + (x_2 - x_1)^2}} \tag{4.8}$$

$$D_1 = \frac{\left|\frac{(x_1y_1 + x_2y_1 - x_1y_2 - x_2y_2)}{2} + (x_1y_2 - x_2y_1) + (x_2 - x_1)Y_m\right|}{\sqrt{(y_1 - y_2)^2 + (x_2 - x_1)^2}} \tag{4.9}$$

$$D_1 = \frac{\left|\frac{(x_1y_1 - x_2y_1 + x_1y_2 - x_2y_2)}{2} + (x_2 - x_1)Y_m\right|}{\sqrt{(y_1 - y_2)^2 + (x_2 - x_1)^2}} \tag{4.10}$$

$$D_1 = \frac{\left|\frac{(x_1 - x_2)(y_1 + y_2)}{2} + (x_2 - x_1)Y_m\right|}{\sqrt{(y_1 - y_2)^2 + (x_2 - x_1)^2}} \tag{4.11}$$

$$D_1 = \frac{\left|\frac{-(x_2 - x_1)(y_1 + y_2)}{2} + (x_2 - x_1)Y_m\right|}{\sqrt{(y_1 - y_2)^2 + (x_2 - x_1)^2}} \tag{4.12}$$

$$D_1 = \frac{\left|(x_2 - x_1)\left(Y_m - \frac{(y_1 + y_2)}{2}\right)\right|}{\sqrt{(y_1 - y_2)^2 + (x_2 - x_1)^2}}. \tag{4.13}$$

Logo, quando Y_m se aproxima de $\frac{y_1 + y_2}{2}$, a distância se aproxima de zero, ou seja, quando dois pontos estão em um mesmo segmento de curva, quanto mais próximo um do outro mais próxima a curva será da reta formada pelos dois pontos.
Fazendo os mesmos cálculos com P_1', P_2' e P_3' temos:

$$P_1'P_2' : \begin{vmatrix} x & y & 1 \\ x_1 & Ky_1 & 1 \\ x_2 & Ky_2 & 1 \end{vmatrix} = \tag{4.14}$$

$$K(y_1 - y_2)\cdot x + (x_2 - x_1)\cdot y + K\cdot(x_1 \cdot y_2 - x_2 \cdot y_1) = 0$$

$$D_2 = \frac{\left|(x_2 - x_1)\left(K \cdot Y_m - \frac{(K \cdot y_1 + K \cdot y_2)}{2}\right)\right|}{\sqrt{(K \cdot y_1 - K \cdot y_2)^2 + (x_2 - x_1)^2}} \tag{4.15}$$

$$D_2 = \frac{\left|(x_2 - x_1)\left(K \cdot Y_m - \frac{K \cdot (y_1 + y_2)}{2}\right)\right|}{\sqrt{K^2 \cdot (y_1 - y_2)^2 + (x_2 - x_1)^2}} \tag{4.16}$$

$$D_2 = \frac{\left|K \cdot (x_2 - x_1)\left(Y_m - \frac{(y_1 + y_2)}{2}\right)\right|}{\sqrt{K^2 \cdot (y_1 - y_2)^2 + \frac{K^2}{K^2} \cdot (x_2 - x_1)^2}} \tag{4.17}$$

$$D_2 = \frac{K \cdot \left|(x_2 - x_1)\left(Y_m - \frac{(y_1 + y_2)}{2}\right)\right|}{K \cdot \sqrt{(y_1 - y_2)^2 + \left(\frac{x_2 - x_1}{K}\right)^2}} \tag{4.18}$$

$$D_2 = \frac{\left|(x_2 - x_1)\left(Y_m - \frac{(y_1 + y_2)}{2}\right)\right|}{\sqrt{(y_1 - y_2)^2 + \left(\frac{x_2 - x_1}{K}\right)^2}}. \tag{4.19}$$

Então, como podemos ver na Eq. 4.19, a influência de K é limitada, particularmente quando x_1 e x_2 são mais próximos, o que tende a ocorrer quando ambos pertencem ao mesmo ótimo local. E quando ambos pertencem a ótimos locais distintos, y_1 e y_2 podem ser bem distintos, o que novamente reduz a influência de K.

Entretanto, quando Y_m é maior do que y_1 e y_2, o ponto mais próximo se afastará do segmento definido por $\overline{P_1P_2}$ quando K aumenta. Portanto, se $\mathbf{v}\cdot\mathbf{w} \le 0$, então o ponto mais próximo está antes de P_1. Por outro lado, se $\mathbf{v}\cdot\mathbf{w} \ge 0$ e $\mathbf{v} \le \mathbf{v}\cdot\mathbf{w}$, então o ponto mais próximo estará depois de P_2 ($\left|\frac{\mathbf{v}\cdot\mathbf{w}}{|\mathbf{v}|}\right| \ge 1$).

No primeiro caso, e tomando $y_1 \prec y_2$, temos:

$$\mathbf{v} \cdot \mathbf{w} \leq 0 \tag{4.20}$$

$$(x_2 - x_1) \cdot \left(\frac{x_1 + x_2}{2} - x_1 \right) + (y_2 - y_1) \cdot (Y_m - y_1) \leq 0 \tag{4.21}$$

$$(x_2 - x_1) \cdot \left(\frac{x_2 - x_1}{2} \right) + (y_2 - y_1) \cdot (Y_m - y_1) \leq 0 \tag{4.22}$$

$$\left(\frac{(x_2 - x_1)^2}{2} \right) + (y_2 - y_1) \cdot (Y_m - y_1) \leq 0 \cdot \tag{4.23}$$

Mas:

$$\left(\frac{(x_2 - x_1)^2}{2} \right) \geq 0, (y_2 - y_1) \geq 0 \tag{4.24}$$

$$\therefore Y_m \leq y_1 . \tag{4.25}$$

E no segundo caso temos:

$$\mathbf{v} \cdot \mathbf{w} \geq 0 \text{ e } \mathbf{v} \leq \mathbf{v} \cdot \mathbf{w} \tag{4.26}$$

$$(x_2 - x_1)^2 + (y_2 - y_1)^2 \leq \left(\frac{(x_2 - x_1)^2}{2} \right) + (y_2 - y_1) \cdot (Ym - y_1), \tag{4.27}$$

como

$$(x_2 - x_1)^2 \geq \left(\frac{(x_2 - x_1)^2}{2} \right), \tag{4.28}$$

então

$$(y_2 - y_1)^2 \leq (y_2 - y_1) \cdot (Y_m - y_1) \tag{4.29}$$

$$(y_2 - y_1) \leq (Y_m - y_1) \tag{4.30}$$

$$y_2 \leq Y_m . \tag{4.31}$$

Portanto, apenas quando os pontos P_1 e P_2 não pertencerem ao mesmo segmento de curva, a distância será influenciada por K, como demonstrado abaixo:

$$D_2^{'} = \sqrt{\left(\frac{x_2 - x_1}{2} \right)^2 + K^2 \cdot (Y_m - y_1)^2} \tag{4.32}$$

$$D_2^{'} = K \cdot \sqrt{\left(\frac{x_2 - x_1}{2K}\right)^2 + \left(Y_m - y_1\right)^2} \,. \tag{4.33}$$

Então, enquanto o valor de D_2 é limitado por $\frac{\left|\left(x_2 - x_1\right)\left(Y_m - \frac{\left(y_1 + y_2\right)}{2}\right)\right|}{\sqrt{\left(y_1 - y_2\right)^2}}$ quando o valor de K caminha em direção ao infinito, o valor de D_2' não é limitado por qualquer valor e tende ao infinito juntamente com K.

Logo, quando uma função é aumentada em K a distância entre dois pontos pertencentes à mesma curva estará sujeita a um aumento insignificante, enquanto as distâncias dos outros pontos aumentarão proporcionalmente.

Empiricamente, foi determinado o valor de 0,5 para o limiar de supressão em todas as funções utilizadas. Note que é possível manter esse valor para qualquer função ajustando apenas o parâmetro K de acordo com o valor de $f(\mathbf{x})$, por exemplo.

Cabe alertar que esta metodologia não garante a decisão correta em qualquer circunstância. Mas como muitos procedimentos de supressão devem ser realizados, a eficiência da proposta fica sustentada pela freqüência muito maior de decisões corretas.

4.4.5 População Limitada

Conforme visto na Seção 4.2, quando se deseja otimizar uma função em que o número de ótimos é muito grande como, por exemplo, no caso da Figura 4.7, a população auto-ajustável da opt-aiNet tende a crescer de forma acelerada e improdutiva, pois nem todos os ótimos são interessantes e existe um tempo até que dois anticorpos residentes no mesmo ótimo sejam detectados. Isso causa um aumento desnecessário no custo computacional do algoritmo. Para solucionar esse problema foi simplesmente imposto um limite no número de anticorpos que a população ativa pode alcançar. Os anticorpos excedentes são eliminados de acordo com seus respectivos valores de aptidão: anticorpos com valores baixos da função-objetivo são eliminados, respeitando o tamanho mínimo da população.

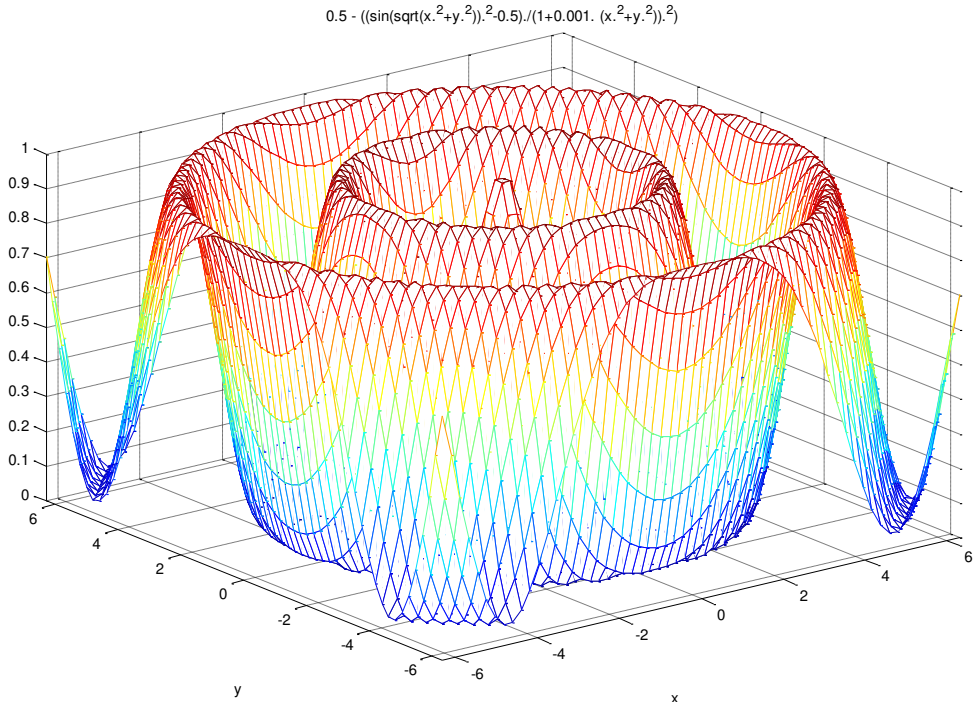


Figura 4.7: Um exemplo de função com infinitos ótimos locais. Se não for imposto um limite, a população tende a crescer demasiadamente.

4.5 Algoritmos para Comparação

Nesta seção, dois outros algoritmos bio-inspirados serão apresentados para confrontar os resultados da dopt-aiNet na seqüência. O primeiro algoritmo apresentado será o *método de otimização por enxame de partículas* (*Particle Swarm Optimization* - PSO) proposto por Eberhart & Kennedy (1995). O PSO é baseado no comportamento de partículas e sistemas sociais humanos (de Castro & Von Zuben, 2005; Kennedy & Eberhart, 1995). O PSO apresenta um bom desempenho na literatura, além de já ter sido empregado em testes similares aos feitos neste trabalho em ambientes dinâmicos (Carlisle, 2002). O segundo algoritmo estudado é chamado de *algoritmo da célula B* (*B-Cell Algorithm* - BCA), que é um algoritmo de otimização também baseado em sistemas imunológicos (Kelsey & Timmis, 2003). Em Kelsey & Timmis (2003), aponta-se um melhor desempenho do BCA em relação à opt-aiNet para alguns casos de estudo.

4.5.1 PSO: Otimização por Enxame de Partículas

O algoritmo PSO (Particle Swarm Optimization) foi criado em 1995 (Eberhart & Kennedy, 1995; Kennedy & Eberhart, 1995) e é baseado no comportamento dinâmico e social de grupos de animais, sendo um dos precursores da área conhecida como *inteligência de enxame* ou *inteligência coletiva* (*swarm intelligence*). Na inteligência de enxame, vários agentes adquirem informações locais do ambiente e, através de comunicação direta ou indireta, trocam informações para tentar atingir algum objetivo.

Um dos precursores dessa idéia foi o trabalho de Reynolds (1987), que investigou e simulou o comportamento de vôo de pássaros usando regras comportamentais simples para cada agente (pássaro) da revoada.

Esse foi um dos comportamentos que motivaram Eberhart & Kennedy (1995) a imaginarem a seguinte situação: em um ambiente contínuo, diversas partículas percorrem o espaço de busca em uma trajetória que, a cada instante, tem sua direção de deslocamento composta pela combinação linear de um vetor direção que aponta para a melhor solução da partícula até o momento e de um vetor direção da melhor solução encontrada por um conjunto de partículas vizinhas, sendo que a vizinhança é definida *a priori*.

Esse comportamento incorpora as idéias de competição e colaboração simultaneamente: i) *competição*, pois cada partícula procura uma solução melhor que seus vizinhos, utilizando tanto a região que seu melhor vizinho encontrou (*informação global*) como a região da melhor solução encontrada por ela própria (*informação local*); e ii) *colaboração*, pois cada partícula compartilha suas soluções ou experiências umas com as outras.

O espaço de busca de cada partícula é determinado por dois vetores: o primeiro vetor aponta para a melhor solução obtida até o momento por ela própria e o segundo vetor aponta para a melhor solução obtida considerando todas as partículas vizinhas (a melhor entre n partículas mais próximas, por exemplo). Para atualizar a posição corrente da partícula, é utilizada a Eq. (4.34):

$$\mathbf{p} = \mathbf{p} + \mathbf{v} \tag{4.34}$$

onde $\mathbf{v}$ é chamado de vetor velocidade e é calculado segundo a Eq.(4.35),

$$\mathbf{v} = K.(\mathbf{v} + \rho_1.rand1.(\mathbf{p_{best}} - \mathbf{p}) + \rho_2.rand2.(\mathbf{g_{best}} - \mathbf{p})) \tag{4.35}$$

onde K é uma constante de amortização, ρ_1 e ρ_2 correspondem à importância dada para a informação local e global, respectivamente, *rand1* e *rand2* são variáveis aleatórias que assumem valores no intervalo [0, 1] e são utilizadas para diversificar os graus de importância dadas às informações locais e globais, $\mathbf{p_{best}}$ representa o melhor vetor solução obtido por essa partícula até o momento e $\mathbf{g_{best}}$ representa o melhor vetor solução obtido por seus vizinhos. O procedimento é explicado em maiores detalhes no Algoritmo 4.5.

```
[x] = Function PSO(max_it, N, ρ1, ρ2, K)
      P = inicia_particulas(N);
      v = inicia_velocidade(N);
      fP = avalia(P);
      Pbest = P;
      fPbest = fP;
      fGbest = min(fPbest);
      Gbest = P[min_i(fPbest)];
      It = 0;
      Enquanto it < max_it faça,
            v = K.(v + ρ1.rand1.(Pbest - P) + ρ2.rand2.(Gbest - P));
            P = P + v;
            fP = avalia(P);
            [Pbest, fPbest] = atualiza_melhor(P,Pbest);
            [Gbest, fGbest] = atualiza_melhor(Pbest,Gbest);
            it = it + 1;
      Fim
Fim
```

Algoritmo 4.5: Algoritmo Particle Swarm Optimization (PSO).

O algoritmo inicia gerando aleatoriamente as partículas (soluções) e seus correspondentes vetores de velocidade **v** (direção de caminhada + passo). Em seguida essas soluções são avaliadas e esse valor é atribuído ao vetor de melhor solução de cada partícula (*Pbest*), assim como o melhor valor entre todas as partículas é atribuída a *Gbest*.

Após a inicialização, o algoritmo entra em seu laço principal, no qual é calculado o novo vetor de velocidade baseado na melhor solução da partícula e na melhor solução entre as vizinhas. Os parâmetros utilizados são: *K*, constante de amortização para limitar o valor da velocidade; ρ_1, referente à importância da informação local; ρ_2, referente à importância da informação global; *rand1* e *rand2* são variáveis aleatórias no intervalo [0, 1], empregada para que o processo não se torne determinístico.

4.5.2 BCA: Algoritmo da Célula B

O algoritmo BCA (Kelsey & Timmis, 2003) utiliza um processo similar à opt-aiNet, no qual cada elemento (aqui chamado de célula B, ou B-Cell) sofre um processo de clonagem e mutação. As principais diferenças são:

- A representação não é por vetor de números reais, e sim um vetor binário de 64 bits de dupla precisão por variável;
- A mutação utilizada é denominada *mutação contígua* e será descrita abaixo; e
- O tamanho da população é fixo.

O procedimento para implementação do BCA é apresentado no Algoritmo 4.6.

```
[x] = Function BCA(max_it,N,Nc)
      Bcells = inicializa(N);
      xBcells = decodifica(Bcells);
      fBcells = f(xBcells);
      it = 0;
      Enquanto it < max_it faça,
            Para i = 1..N,
                  C = clona(Bcells(i), Nc);
                  C(Nc) = inicializa();
                  C = mut_contigua(C);
                  xC = decodifica(C);
                  fC = avalia(xC);
                  j = melhor(C);
                  Se C(j) for melhor que Bcells(i),
                        Bcell(i) = C(j);
                  Fim
            Fim
            it = it + 1;
      Fim
Fim
```

Algoritmo 4.6: Algoritmo BCA.

O algoritmo inicia atribuindo valores aleatórios para cada célula B, que corresponde aos anticorpos da opt-aiNet e dopt-aiNet. Em seguida, cada célula B é decodificada para valores reais e, então, avaliada. O laço principal do algoritmo executa, para cada célula B, um processo de clonagem, no qual *Nc* clones são gerados, o último clone é substituído por uma nova célula B aleatória, resultando então em $Nc-1$ clones idênticos e um novo indivíduo. Cada um dos clones passa por um processo de mutação, denominado pelos autores *mutação contígua* (Figura 4.8). Nesse processo, diferente da mutação multi-pontos, para cada segmento de 64 bits (representando uma variável) dois números são sorteados: um que será o ponto inicial e o outro que será a extensão da mutação. Então, todos os bits a partir do ponto inicial até o tamanho sorteado serão complementados. Após esse processo, caso algum dos clones tenha um desempenho melhor que a célula B originária, este a substitui na população.

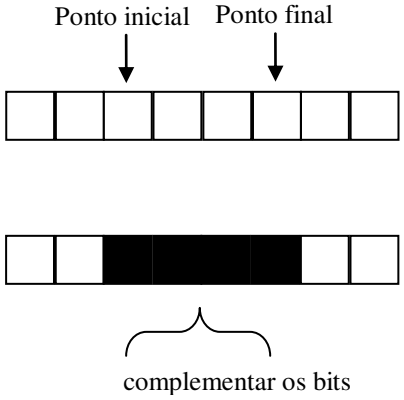


Figura 4.8: Mutação Contígua do algoritmo BCA.

4.6 Resultados Experimentais para Casos Estáticos

Nesta seção, o desempenho da dopt-aiNet será confrontado com o desempenho do PSO e do BCA em ambientes estáticos, utilizando-se para isso das mesmas funções introduzidas no início do capítulo.

Para obter resultados comparativos nos ambientes estáticos, cada algoritmo foi executado por 30 rodadas, e como critérios de parada, a função-objetivo atingir o valor de 10^{-3} ou quando 100.000 avaliações da função-objetivo fossem executadas; o que ocorresse primeiro. Após esse processo, a média, o desvio padrão, o menor e o maior valor da função-objetivo, obtido pelo melhor indivíduo da população, e o número de avaliações da função-objetivo foram calculados.

Para o PSO, os parâmetros foram escolhidos de acordo com Carlisle (2002), que realizou um trabalho completo de análise de sensibilidade do algoritmo aos parâmetros. Com isso, espera-se que o PSO apresente bom desempenho nas mesmas funções utilizadas aqui. No caso do BCA, os únicos parâmetros necessários são o número de células B e o número de clones, ambos tendo 4 como valor de referência, conforme sugerido em Kelsey & Timmis (2003). Na Tabela 4.5, são apresentados os parâmetros utilizados pelos três algoritmos.

Tabela 4.5: Parâmetros utilizados nos experimentos com os algoritmos PSO, BCA e dopt-aiNet.

PSO	
N	30
K	0,729
ρ_1	2,8
ρ_2	1,3
BCA	
N	4
Nc	4
dopt-aiNet	
Número de células	10
Número de clones	4
Limite máximo de células	200
Rank	15
σ_s	0,5

Inicialmente, os resultados da dopt-aiNet foram comparados com os obtidos anteriormente pela opt-aiNet, os quais são apresentados na Tabela 4.6.

Tabela 4.6: Comparação entre os resultados obtidos pelos algoritmos dopt-aiNet e opt-aiNet apresentados anteriormente. A primeira parte contém o número de avaliações até o ótimo e, na segunda parte, o valor ótimo encontrado.

	dopt-aiNet			opt-aiNet		
No. de Avaliações						
	Min.	**Max.**	**Média (± desvio)**	**Min.**	**Max.**	**Média (± desvio)**
f_1	375	3278	1420,63±742,21	2.519.630	3.715.230	$3.109.986 \pm 3{,}6\text{x}10^5$
f_2	1199	50128	5831,13±9873,70	5.500.000	5.500.000	5.500.000
f_3	370	4473	1155,00±839,68	5.500.000	5.500.000	5.500.000
f_4	1450	17968	8728,00±5452,00	5.500.000	5.500.000	5.500.000
Valor da Função-objetivo						
	Min.	**Max.**	**Média (± desvio)**	**Min.**	**Max.**	**Média (± desvio)**
f_1	0	0	0	0	0	0
f_2	0	0	0	0,08	0,43	0,21 ± 0,08
f_3	0	0	0	121,45	185,12	153,54 ± 13,58
f_4	0	0	0	220,78	433,97	340,70 ± 61,94

Como pode ser observado, a diferença de desempenho da dopt-aiNet em relação à opt-aiNet é muito grande, confirmando a melhora obtida através dos procedimentos propostos, que resultaram na criação da dopt-aiNet. Na Tabela 4.7, os tempos médios gastos por esses dois algoritmos são comparados para se ter uma melhor idéia do custo computacional utilizado por cada um deles. Para obter estes tempos médios, foi utilizada uma máquina com as seguintes especificações: Athlon XP+2000 @ 1,67GHz com 512MB de memória RAM no Sistema Operacional Linux Slackware 9.1 e compilados com gcc 3.2.

Tabela 4.7: Comparação entre os tempos médios em segundos para obtenção dos resultados descritos na Tabela 4.7 entre os algoritmos dopt-aiNet e opt-aiNet.

Tempo médio de execução (seg.)		
	dopt-aiNet	**opt-aiNet**
f_1	0,00	68,53
f_2	0,01	146,80
f_3	0,02	171,80
f_4	0,12	204,60

Os resultados descritos na Tabela 4.8 mostram um desempenho superior da dopt-aiNet em comparação aos outros dois algoritmos, destacando as funções Rastrigin (f_3) e Griewank (f_4), nas quais o PSO e o BCA nem sempre conseguiram encontrar o ótimo global e, quando encontraram, realizaram um número de avaliações da função muito maior que a dopt-aiNet. No caso da função Griewank (f_4), a dopt-aiNet apresentou uma leve desvantagem em relação ao BCA provavelmente devido à natureza da mutação utilizada pela dopt-aiNet, na qual é dada uma direção e calculado um tamanho do passo através da Seção Áurea. Uma vez que esse procedimento tem como objetivo chegar ao ótimo local mais próximo e essa função possui muitos ótimos locais com valores de função-objetivo

iguais localizados próximos entre si, o algoritmo tende a explorar mais o espaço de busca em torno dessas regiões antes de seguir adiante na região de busca.

Tabela 4.8: Comparação entre os resultados obtidos pelos algoritmos dopt-aiNet, PSO e BCA. A primeira parte contém o número de avaliações até o ótimo e na segunda parte o valor ótimo encontrado.

	dopt-aiNet			PSO			BCA		
No. de Avaliações									
	Min.	**Max.**	**Média (± desvio)**	**Min.**	**Max.**	**Média (± desvio)**	**Min.**	**Max.**	**Média (± desvio)**
f_1	375	3278	1420,63±742,21	6930	10260	8482,00±748,23	2948	6564	5236,00±909,99
f_2	1199	50128	5831,13±9873,70	100000	100000	100000,00	100000	100000	100000,00
f_3	370	4473	1155,00±839,68	100000	100000	100000,00	2116	4564	2913,33±822,87
f_4	1450	17968	8728,00±5452,00	7560	100000	72644,00±42545,83	4004	6676	5068,33±628,80
Valor da Função-objetivo									
	Min.	**Max.**	**Média (± desvio)**	**Min.**	**Max.**	**Média (± desvio)**	**Min.**	**Max.**	**Média (± desvio)**
f_1	0	0	0	0,00	0,00	0,00	0,00	0,00	0,00
f_2	0	0	0	0,47	842,54	82,82±155,72	27,18	28,9	28,20±0,47
f_3	0	0	0	21,88	79,59	51,60±14,53	0,00	0,00	0,00
f_4	0	0	0	0,00	0,24	0,06±0,07	0,00	0,00	0,00

Outro ponto que merece atenção é o controle do tamanho da população, principalmente na função Rastrigin, onde a opt-aiNet (ver Tabela 4.4) aumentava excessivamente o seu tamanho e não realizava a supressão de elementos que estavam no mesmo pico. A Tabela 4.9 evidencia que essa dificuldade foi suprimida com o uso do novo algoritmo de supressão.

Tabela 4.9: Número de células ao final da execução do algoritmo dopt-aiNet para cada função.

	Tamanho da população ao final da execução do algoritmo		
	Min.	**Max.**	**Média (± desvio)**
f_1	7	10	9,73±0,87
f_2	7	10	9,60±0,72
f_3	1	11	5,63±3,79
f_4	3	15	6,70±3,36

A seguir, serão apresentados os modelos de ambiente dinâmico que serão utilizados para os testes finais dessa ferramenta.

4.7 Ambientes Dinâmicos

Nesta seção, serão abordadas as metodologias para criação de ambientes dinâmicos a partir de funções estáticas. O modelo utilizado neste trabalho será o mesmo proposto por Angeline (1997), com algumas alterações de modo a torná-lo ainda mais desafiador para os algoritmos.

4.7.1 Modelo proposto por Angeline (1997)

Para ter um controle da dinâmica das funções, Angeline (1997) propôs a introdução de dois parâmetros extras para a função a ser otimizada: a amplitude do deslocamento dos ótimos da função (*amp*) e a freqüência de mudança dos ótimos (φ), ou seja, de quantas em quantas iterações a função sofrerá uma mudança. As funções, então, sofrem alterações diretamente em suas variáveis, como descrito na Eq. 4.36:

$$f'(x) = f(x + \Delta k), \tag{4.36}$$

onde Δk é um vetor que define a nova posição do ótimo em relação à posição original.

Para este trabalho, foram criados três tipos de movimentos:

- *Degrau*: o incremento Δk é calculado através de uma função degrau com deslocamento aleatório τ entre [–*amp*, *amp*].

$$\Delta k = \Delta k + \tau; \tag{4.37}$$

- *Rampa*: o incremento Δk aumenta linearmente, de Δk_0 até $\Delta k_0 + \tau$, por um certo período de tempo *T*, e com τ entre [–*amp*, *amp*]:

$$\Delta k = \Delta k_0 + \left(\frac{\tau - \Delta k_0}{T} \right) . t; \tag{4.38}$$

- *Quadrático*: o incremento Δk aumenta de forma quadrática, partindo de Δk_0 até $\Delta k_0 + \tau$, a cada passo de iteração *t* por um certo período de tempo *T*, e com τ entre [–*amp*, *amp*]:

$$\Delta k(t) = \tau \cdot \left(\frac{-t^2}{T^2} + \frac{2t}{T} \right) + \Delta k_0 . \tag{4.39}$$

4.8 Resultados Comparativos: dopt-aiNet × PSO × BCA

Como um primeiro teste de desempenho, foram executadas 5.000 iterações de cada algoritmo para cada uma das funções em cada um dos três tipos de ambientes dinâmicos definidos anteriormente. As posições dos ótimos de cada função foram alteradas a cada 400

iterações. No caso dos ambientes dinâmicos tipo *rampa* e *quadrático*, o período de atualização foi de 100 iterações e o tamanho do deslocamento foi definido como $amp = 100$.

Para analisar o desempenho de cada algoritmo junto a cada experimento foram calculados o erro médio, a média dos menores erros obtidos dentro dos intervalos onde o ambiente é estático, a média do número de iterações para se chegar ao menor erro nesse intervalo, e o maior erro encontrado. Para todos os experimentos, esses valores estarão resumidos em tabelas.

O erro médio, juntamente com seu desvio, mostra a oscilação dos resultados de cada algoritmo quando submetido à mudança no ambiente. A média dos menores erros indica a capacidade do algoritmo em alcançar novos ótimos. O número de iterações necessárias até esse valor de erro dá origem a diversas situações:

a) Se o erro tem um valor igual a zero, então o número de iterações indica a velocidade de reação;
b) Se o erro não é zero e o número de iterações é próximo da freqüência de atualização, então, embora o algoritmo não tenha alcançado o ótimo, dado um maior tempo ele continuaria melhorando suas soluções;
c) Se o erro não é zero e o número de iterações está distante da freqüência de atualização, então o algoritmo estagnou em um ótimo local e possivelmente não conseguiria reduzir o erro mesmo que um maior tempo fosse permitido.

Tabela 4.10: Resultados dos experimentos com o algoritmo dopt-aiNet.

		dopt-aiNet				
		Erro Médio	**Média do Menor Erro**	**Menor Erro**	**Maior Erro**	**Média de Iterações**
f_1	**Pulso**	75,81±74,30	**0,18±0,46**	**0,00**	345,74	215,67±124,35
	Rampa	**25,02±38,54**	**0,00**	**0,00**	218,03	298,22±48,04
	Quadrático	**28,31±48,17**	**0,00**	**0,00**	251,99	274,00±51,59
f_2	**Pulso**	**154,64±88,06**	**67,02±52,58**	**3,76**	493,24	293,56±102,62
	Rampa	**93,27±65,71**	**34,82±31,49**	**3,78**	269,27	366,44±45,08
	Quadrático	**37,32±43,99**	**2,80±2,02**	**0,02**	206,01	317,33±125,71
f_3	**Pulso**	94,05±66,69	41,00±23,73	23,32	366,03	174,33±122,32
	Rampa	**42,99±41,34**	**13,71±10,19**	**4,67**	256,30	367,56±46,36
	Quadrático	**52,13±45,26**	**21,04±17,90**	**6,21**	243,66	390,00±22,00
f_4	**Pulso**	**66,65±75,29**	**5,66±7,04**	**0,00**	324,23	174,00±137,98
	Rampa	**24,29±44,53**	**0,60±1,81**	**0,00**	221,24	347,33±32,78
	Quadrático	**35,73±62,86**	**3,28±3,15**	**0,00**	238,04	236,22±110,6

Tabela 4.11: Resultados dos experimentos com o algoritmo PSO.

		PSO				
		Erro Médio	**Média do Menor Erro**	**Menor Erro**	**Maior Erro**	**Média de Iterações**
f_1	**Pulso**	**75,49±95,39**	0,34±0,60	**0,00**	331,15	206,89±139,89
	Rampa	53,51±70,13	0,09±0,16	**0,00**	320,48	400,78±0,67
	Quadrático	72,30±92,1	0,04±0,05	**0,00**	333,40	400,44±0,73
f_2	**Pulso**	571,27±1120,68	82,26±201,76	4,49	11361,88	227,11±153,67
	Rampa	2747,36±3157,80	2478,50±3249,18	5,15	10065,06	263,22±194,28
	Quadrático	1518,37±2379,21	1270,66±2417,17	3,73	8528,18	137,67±174,01
f_3	**Pulso**	**88,57±79,33**	**24,58±24,72**	**10,78**	325,75	269,11±128,68
	Rampa	61,58±54,30	21,04±17,73	8,66	256,51	365,11±30,74
	Quadrático	83,31±75,45	30,76±47,01	9,40	258,39	344,22±54,26
f_4	**Pulso**	94,51±103,05	17,39±16,15	0,03	349,80	249,89±150,66
	Rampa	75,67±88,02	36,58±27,50	**0,00**	319,73	231,00±65,46
	Quadrático	91,27±108,69	30,24±32,29	**0,00**	326,66	221,44±93,32

Tabela 4.12: Resultados dos experimentos com o algoritmo BCA.

		BCA				
		Erro Médio	**Média do Menor Erro**	**Menor Erro**	**Maior Erro**	**Média de Iterações**
f_1	**Pulso**	649,15±267,67	654,59±227,08	276,46	979,63	178,00±139,49
	Rampa	257,97±72,37	277,86±34,29	229,22	338,84	395,33±6,16
	Quadrático	272,81±66,70	289,27±23,54	250,60	346,11	394,11±6,94
f_2	**Pulso**	537,81±219,27	543,23±177,43	273,62	867,07	146,44±149,11
	Rampa	256,46±95,04	274,86±25,07	233,33	1113,74	367,11±50,53
	Quadrático	270,12±80,59	293,83±15,87	276,79	332,86	387,33±17,16
f_3	**Pulso**	540,70±212,38	552,54±156,37	321,19	858,87	208,11±152,55
	Rampa	251,35±82,72	281,46±23,73	239,96	321,65	387,56±17,44
	Quadrático	257,93±82,56	284,97±23,70	239,37	327,27	384,22±25,37
f_4	**Pulso**	621,98±245,43	587,36±161,20	308,94	912,62	175,89±135,87
	Rampa	285,34±156,3	284,44±28,98	234,02	351,24	396,22±4,55
	Quadrático	279,42±148,43	286,01±30,42	224,53	336,13	398,78±1,72

Como pode ser visto nas Tabelas 4.10 a 4.12Tabela 4.12, a dopt-aiNet apresentou os menores erros médios e média dos menores erros (colunas 3 e 4 das tabelas, respectivamente) em comparação com os outros algoritmos, exceto nos ambientes dinâmicos do tipo *pulso* nas funções f_1, onde apesar do PSO ter um erro médio menor, a média dos menores erros é maior e f_3, para a qual o PSO teve o melhor desempenho. Também é importante notar que o maior erro obtido durante as iterações foram menores para a dopt-aiNet, indicando uma reação mais rápida quando o ambiente é alterado.

As figuras a seguir servem para ilustrar a velocidade de reação de cada um dos algoritmos. Como apresentado na Figura 4.9, o eixo x representa a iteração e o eixo y o valor do erro em relação ao ótimo global da iteração atual. A linha serrilhada representa a mudança causada no ambiente, calculada pela distância euclidiana entre o ótimo global

original e o novo ponto ótimo. A linha cheia representa o erro do algoritmo em relação ao novo ótimo e é calculado pela distância entre o melhor indivíduo da iteração e o ótimo global atual. Quanto mais próximo o valor de *y* da linha sólida estiver de zero, mais próximo o algoritmo está do ótimo global em uma dada iteração. E quanto mais próxima a linha sólida estiver da linha tracejada, menor a capacidade de resposta ao deslocamento do ótimo.

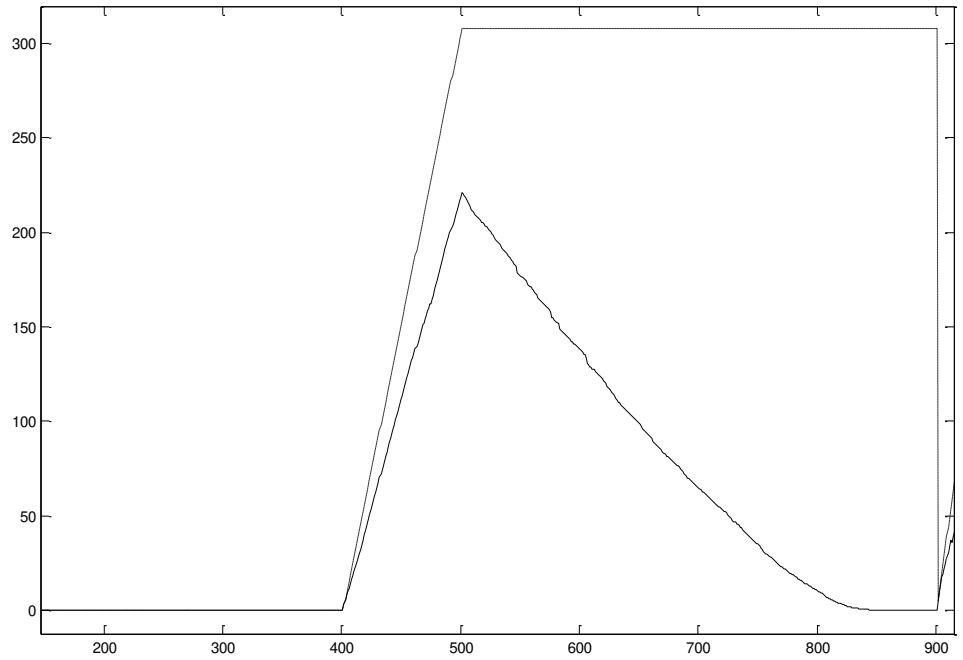
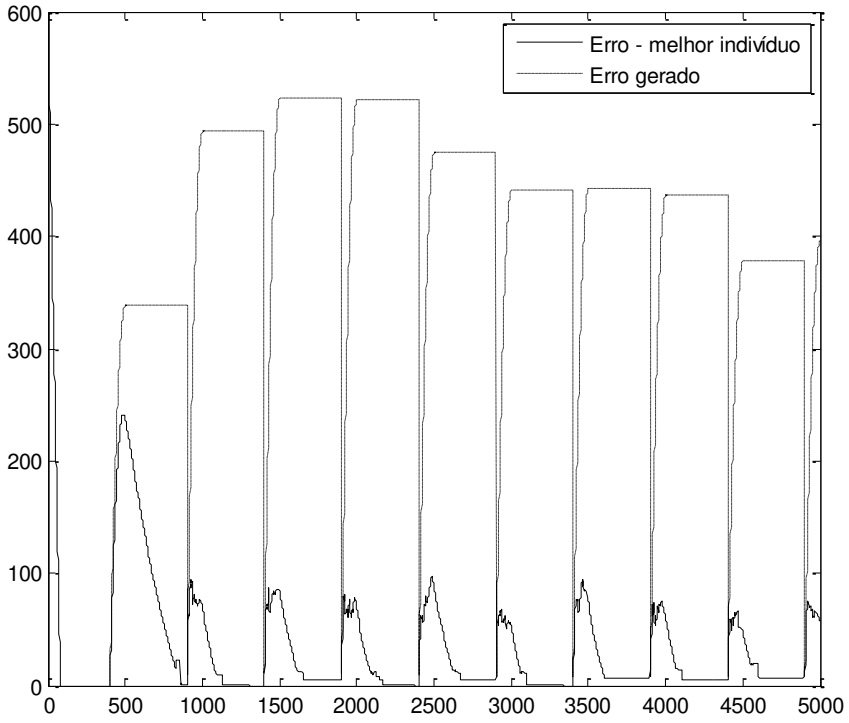


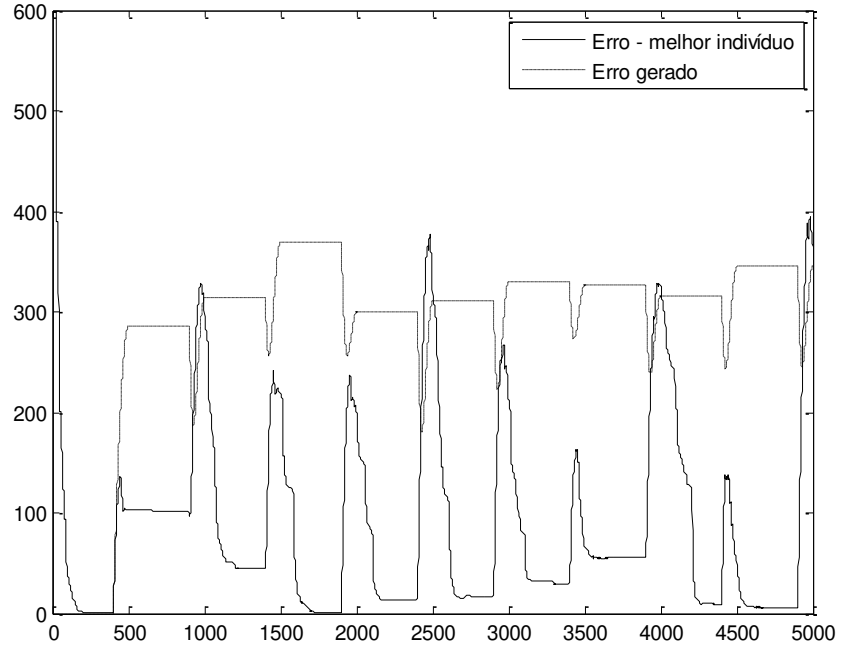
Figura 4.9: Aproximação de um ciclo de mudança da Figura 4.10.

Nas Figuras de 4.10 à 4.12, notamos que mesmo durante a mudança do ambiente, no caso *quadrático* a dopt-aiNet tem a capacidade de seguir o erro gerado causando pequenas oscilações nesse período e não permite que o erro aumente excessivamente. Já o desempenho obtido pelos algoritmos PSO e BCA é bem inferior, sendo que o BCA não foi capaz de responder ao deslocamento do ótimo.



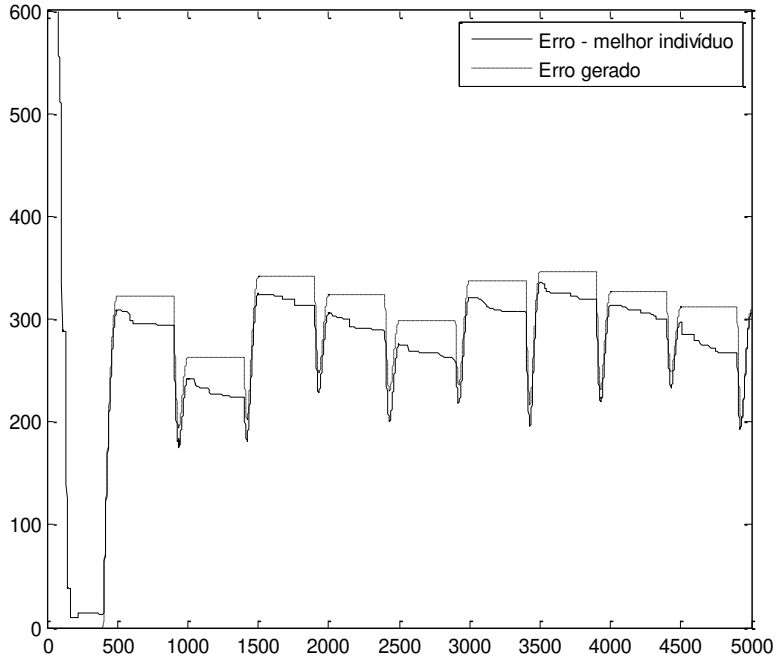
(a)

Figura 4.10: Ambiente dinâmico do tipo quadrático otimizando a função f_4 com o algoritmo dopt-aiNet.



(b)

Figura 4.11: Ambiente dinâmico do tipo quadrático otimizando a função f_4 com o algoritmo PSO.



(c)

Figura 4.12: Ambiente dinâmico do tipo quadrático otimizando a função f_4 com o algoritmo BCA.

As figuras de 4.13 à 4.15 ilustram o "tracking", ou seja, como cada componente da solução encontrada pela dopt-aiNet acompanha uma dentre as variáveis que compõe a solução ótima enquanto esta se altera em função do tempo. As figuras ilustram as variáveis que obtiveram menor erro, um caso médio de erro e o pior erro, respectivamente.

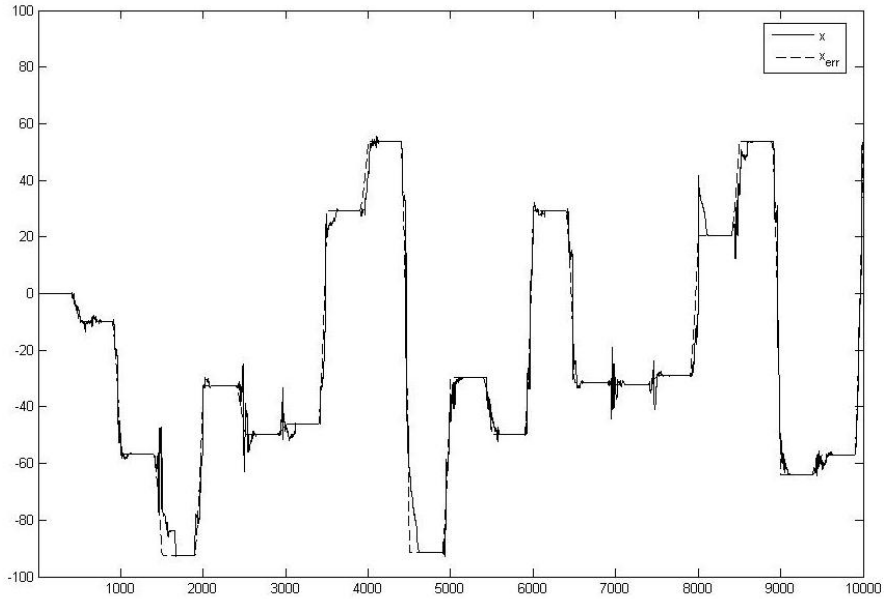


Figura 4.13: Tracking de uma variável do algoritmo dopt-aiNet (x) em relação à variável ótima (x_{err}) no melhor caso para f_4.

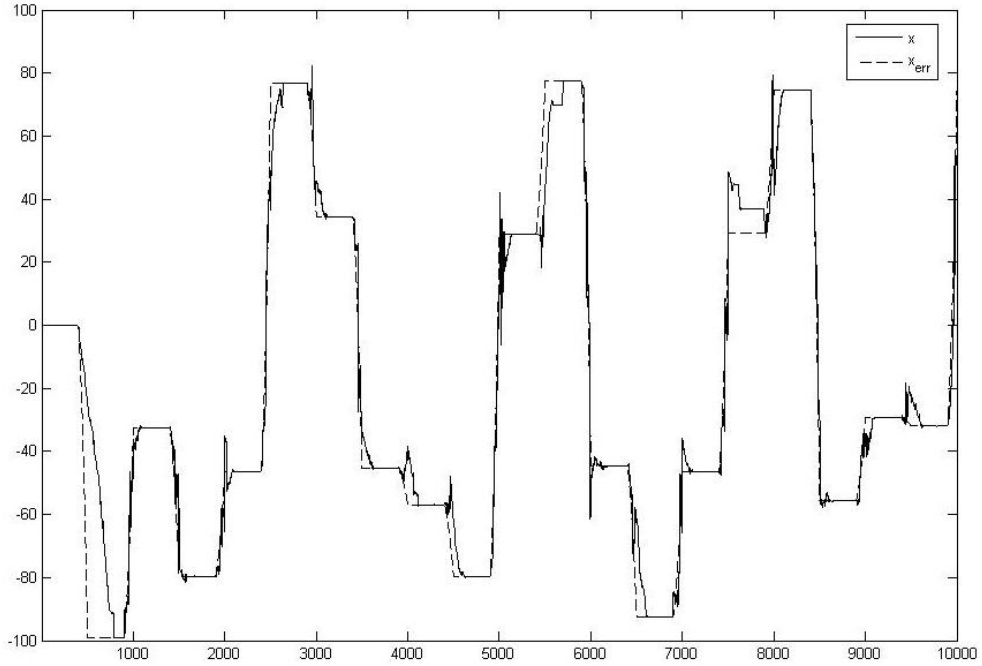


Figura 4.14: Tracking de uma variável do algoritmo dopt-aiNet (x) em relação à variável ótima (x_{err}) no caso médio para f_4.

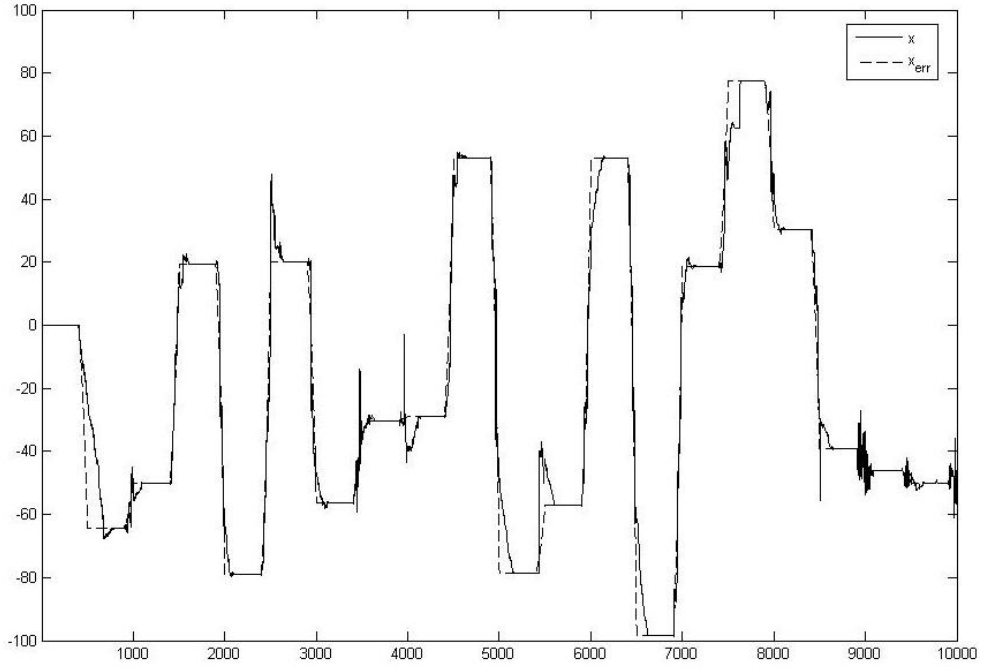


Figura 4.15: Tracking de uma variável do algoritmo dopt-aiNet (x) em relação à variável ótima (x_{err}) no pior caso para f_4.

Outros resultados de desempenho da dopt-aiNet, tanto em casos estáticos como em dinâmicos, podem ser conferidos em de França et al. (2005a), onde, para o caso estático, foram testadas 18 funções contendo características diversas e a comparação dos resultados foi feita com diferentes algoritmos da literatura. Vale mencionar também os resultados obtidos em Junqueira et al. (2005), no qual a dopt-aiNet foi utilizada em uma aplicação real e variante no tempo para um problema de processamento de sinais, produzindo bom desempenho.

4.9 Síntese do Capítulo

Neste capítulo, o algoritmo baseado em sistemas imunológicos para problemas de otimização, conhecido como opt-aiNet, foi analisado e reformulado para resolver problemas de otimização de forma mais rápida e eficiente, dando origem ao algoritmo dopt-aiNet (opt-aiNet para ambientes dinâmicos).

Foram feitos testes comparativos entre a dopt-aiNet e dois outros algoritmos: o PSO, baseado no comportamento dinâmico de sociedades e definido como um sistemas de partículas; e o BCA, outro algoritmo inspirado nos sistemas imunológicos e que vem sendo empregado na solução de problemas de otimização.

Nos ambientes estáticos, a dopt-aiNet teve uma larga vantagem em relação aos outros algoritmos, conseguindo obter o resultado ótimo em funções com características extremamente desafiadoras para ferramentas de otimização.

Em ambientes dinâmicos, a dopt-aiNet teve um desempenho superior aos outros algoritmos, mostrando uma maior capacidade de reação a mudanças. A dopt-aiNet foi capaz de detectar que houve alterações no ambiente e reagir imediatamente. Mesmo quando o ambiente continuava mudando, o algoritmo foi capaz de seguir o ótimo ao longo do espaço de busca. Dentre as vantagens que a dopt-aiNet teve em relação aos outros dois algoritmos, vale citar:

- Maior estabilidade e coerência quando ocorrem mudanças no ambiente;
- Devido à sua velocidade na otimização, a dopt-aiNet persegue o novo ótimo desde o início da mudança no ambiente, enquanto o PSO demora um pouco mais para começar a reagir e o BCA nem sequer reage;
- Sua capacidade de identificar múltiplos ótimos é também uma vantagem, pois ela tende a manter vários indivíduos em "pontos estratégicos";
- Embora o custo por iteração seja maior que aquele associado aos outros algoritmos, o número total de avaliações de função necessárias para atingir o novo ótimo é menor.

Parte III

Domínio Discreto

Capítulo 5

Otimização por Colônia de Formigas

Neste capítulo, será introduzido o conceito de Otimização por Colônia de Formigas (*Ant Colony Optimization* – ACO), assim como algumas melhorias para incremento de desempenho, como o chamado *Max-Min Ant System* (MMAS) (Stützle & Hoos, 1997), a estrutura hiper-cubo e o *Improved Max-Min Ant System* (IMP.MMAS) (de França et al., 2004a, 2004b, 2005b). Adicionalmente, seus parâmetros serão estudados e otimizados, e seu desempenho para o problema do caixeiro viajante, que é o objeto de estudos dessa parte da dissertação, será avaliado com instâncias conhecidas da literatura.

5.1 Inspiração no Comportamento das Formigas

Sistemas baseados em colônias de formigas foram propostos por Dorigo (1992), inspirados no estudo do comportamento de formigas argentinas (Goss et al., 1989). Neste estudo, foi verificada a capacidade destas formigas em encontrarem o menor caminho entre o ninho e a fonte de alimentos através de uma comunicação indireta, utilizando uma substância chamada *feromônio*. Esse comportamento se encaixa de forma natural no contexto do problema do caixeiro viajante, que também é um problema de rota mínima.

Devido ao sucesso inicial dos resultados obtidos pelo ACO, outros algoritmos baseados em colônias de insetos sociais foram desenvolvidos. Como exemplo, é possível citar o *algoritmo de agrupamento baseado em formigas* (*Ant Clustering Algorithm* - ACA), que se baseia no comportamento de limpeza e organização dos ninhos de algumas espécies de formigas (Bonabeau et al., 1999; Lumer & Faieta, 1994). Como este capítulo enfoca a capacidade das formigas em encontrarem alimento percorrendo caminhos mínimos, apenas o algoritmo ACO será abordado.

5.1.1 Em Busca de Alimento

Em seus estudos, (Goss et al., 1989) notaram que as formigas, durante o processo de coleta de alimentos, tendem a seguir um mesmo caminho, com apenas pequenas variações. Surpreendentemente, esse caminho é ótimo, ou quase ótimo, se comparado a todos os caminhos possíveis em uma arena bi-dimensional. Como experimento inicial, foi criado um ambiente consistindo de uma colônia de formigas em um dos lados da arena e uma fonte de alimentos no outro. Para ir do ninho à fonte de alimentos e retornar ao ninho, foram construídos dois caminhos A e B, sendo o caminho A mais curto que o B.

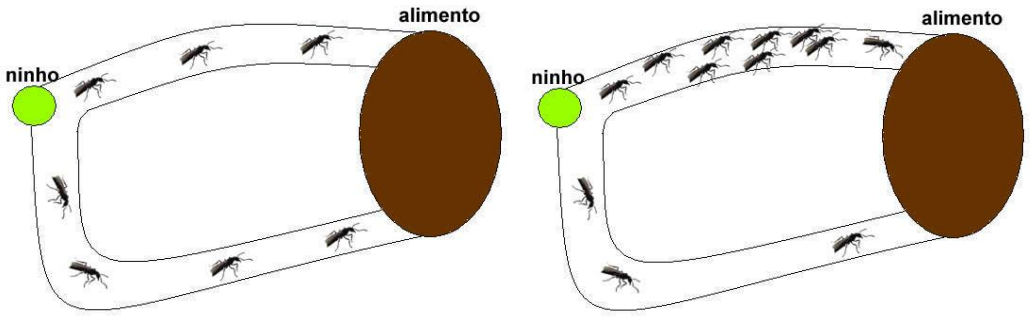


Figura 5.1: Comportamento das formigas durante vários instantes. (a) Início da busca e (b) preferência pelo menor caminho com o decorrer do experimento.

Goss et al. (1989) verificaram que inicialmente as formigas se distribuíam uniformemente pelos dois caminhos. Mas, com o passar do tempo, a densidade de formigas percorrendo o caminho A era muito maior do que a densidade associada ao caminho B. Em um determinado momento dos experimentos, um obstáculo foi colocado no caminho A, de modo que as formigas tivessem dificuldades em passar por ele. Após um determinado período de tempo, a densidade de formigas em cada caminho inverteu-se, tendo um maior número de formigas no caminho B, que passou a ser a melhor opção. Esse processo é ilustrado na Figura 5.1.

Em termos comportamentais, o que ocorre é que toda vez que uma formiga encontra uma fonte de alimento e retorna ao ninho ela secreta uma substância chamada de *feromônio*, que serve de aviso para as outras formigas de que por esse caminho elas irão encontrar alimento. Cada formiga tende a escolher um caminho proporcionalmente à concentração de feromônio percebida. Como o menor caminho será percorrido mais rapidamente, a concentração de feromônio tende a crescer mais rapidamente que sua evaporação, atraindo mais formigas para este caminho. Já em caminhos longos, a taxa de reposição do feromônio não é suficientemente rápida para evitar uma evaporação significativa da substância, causando assim um efeito de esquecimento.

5.2 Sistema de Formigas

Conceitualmente, o primeiro Sistema de Formigas (*ant system* – AS) é bastante simples (Dorigo, 1992) e pode ser resumido como apresentado no Algoritmo 5.1.

```
[S] = Função AS()
      Enquanto it < max_it faça,
            Para cada formiga faça,
                  construir_solução();
                  atualizar_feromonio();
            Fim
      Fim
Fim
```

Algoritmo 5.1: Algoritmo Ant System.

Embora o algoritmo pareça muito simples, existe um processo de comunicação implícito nas funções que cada formiga executa. A função `construir_solução()` gera uma solução factível para o problema a partir de informações locais do ambiente e da quantidade de feromônio depositado pelas formigas. Esse feromônio indica a qualidade de cada parte da solução. Após esse processo, a formiga executa a função `atualizar_feromonio()`, através da qual deposita uma certa quantidade de feromônio em cada parte dessa solução, quantidade esta proporcional à qualidade da própria solução.

5.2.1 Construindo Soluções e Atualizando o Feromônio

Os algoritmos de otimização por colônia de formigas foram introduzidos para resolver problemas de otimização discreta. Para isso, a maioria deles emprega uma representação do problema em forma de um grafo $G = \langle V,E \rangle$ ponderado, no qual V é o conjunto de vértices ou nós do grafo e E é o conjunto de arcos ou arestas do grafo. O peso de cada arco corresponde ao custo associado de ir de um nó a outro do grafo. No caso do exemplo do problema do caixeiro viajante, o custo de cada arco corresponde à distância entre as cidades do mapa.

Baseado nesta representação em grafo do problema do caixeiro viajante (PCV), o algoritmo AS constrói uma solução fazendo com que um conjunto de agentes móveis, denominados *formigas artificiais* ou simplesmente *formigas*, caminhe de um nó a outro do grafo até percorrerem uma rota completa. Assim, para construir uma solução para o PCV colocamos uma formiga k em um nó inicial i, escolhido aleatoriamente, e sucessivamente escolhemos o próximo nó j a ser visitado, seguindo a regra probabilística descrita na Eq. 5.1:

$$p_{i,j}^{k}(t) = \begin{cases} \dfrac{\tau_{i,j}(t)}{\sum_{j \in J^k} \tau_{i,k}(t)} & se\ j \in J^k \\ 0 & c.c. \end{cases}, \qquad (5.1)$$

onde $\tau_{i,j}(t)$ representa o nível de feromônio na aresta (i,j) no instante t e J^k é a lista de nós não visitados pela formiga k.

Com o intuito de ajudar na escolha do próximo passo, freqüentemente é utilizada uma informação do problema geralmente empregada em soluções de heurísticas construtivas. No caso do caixeiro viajante, pode ser utilizada a distância entre os vértices i e j, associada a uma heurística gulosa. Então, a Eq. 5.1 se torna:

$$p_{i,j}^{k}(t) = \begin{cases} \dfrac{\tau_{i,j}^{\alpha}(t) \cdot \eta_{i,j}^{\beta}}{\sum_{j \in J^k} \tau_{i,j}^{\alpha}(t) \cdot \eta_{i,j}^{\beta}} & se\ j \in J^k \\ 0 & c.c. \end{cases}, \tag{5.2}$$

onde η é a heurística de informação do problema, que nesse caso é $\frac{1}{d_{ij}}$. Com isso, a probabilidade de uma formiga *k*, que está em um nó *i*, se deslocar para um nó $j \in J^k$ aumenta proporcionalmente à quantidade relativa de feromônio na aresta (*i*, *j*) e ao inverso do comprimento da aresta (*i*, *j*).

Adicionalmente, também pode ser utilizada uma escolha determinística do próximo nó com base no feromônio e na heurística de informação:

$$p_{i,j}^{k}(t) = \begin{cases} \dfrac{\tau_{i,j}^{\alpha}(t) \cdot \eta_{i,j}^{\beta}}{\sum_{j \in J^k} \tau_{i,j}^{\alpha}(t) \cdot \eta_{i,j}^{\beta}} & se\ j \in J^k\ e\ q \le q_0 \\ \mathbf{arg\,max}\left\{\tau_{i,j}^{\alpha}(t) \cdot \eta_{i,j}^{\beta}\right\} & se\ j \in J^k\ e\ q > q_0 \\ 0 & c.c. \end{cases}, \tag{5.3}$$

onde *q* é um número aleatório no intervalo [0, 1] e q_0 é o limiar de escolha entre uma decisão determinística ou probabilística, também escolhido no intervalo [0, 1].

Após gerar um caminho completo (quando J^k for um conjunto vazio) o desempenho de cada formiga é avaliado medindo-se o comprimento do caminho percorrido. Uma vez que todas as formigas construíram um caminho e tiveram seus desempenhos avaliados, diferentes níveis de feromônio são depositados em cada aresta do grafo de acordo com esse desempenho.

$$\tau_{i,j}(t+1) = (1-\rho) \cdot \tau_{i,j}(t) + \rho \cdot \Delta\tau_{ij}, \tag{5.4}$$

onde $\Delta\tau_{ij}$ é o incremento de feromônio e ρ é uma constante de evaporação de feromônio definida pelo usuário no intervalo aberto (0,1).

Esse incremento, como dito anteriormente, é proporcional à qualidade da solução gerada pela formiga, como mostra a Eq. 5.5:

$$\Delta\tau_{i,j} = \begin{cases} 1/f(S) & se(i,j) \in S \\ 0 & c.c. \end{cases}, \tag{5.5}$$

onde *S* é a solução utilizada para atualizar a trilha de feromônio e *f*(*S*) reflete a qualidade dessa solução, de forma que quanto menor esse valor melhor a solução.

5.3 MMAS: Max-Min Ant System

Embora o ACO tenha obtido bons resultados em instâncias pequenas e médias de caixeiro viajante, alguns pontos prejudicam o desempenho geral do algoritmo. Conseqüentemente, muitos autores propuseram extensões para o algoritmo, sendo a mais efetiva a proposta de Stützle & Hoos (1997), denominada *Max-Min Ant System* (MMAS).

Esse algoritmo propõe basicamente duas mudanças no ACO original. A primeira é referente ao método de atualização da trilha de feromônio, onde no MMAS a atualização é feita apenas na melhor solução global, ou seja, melhor solução entre as melhores de todas as iterações, e/ou na melhor solução local, ou seja, melhor solução da iteração atual. Gera-se assim uma busca em torno de um ótimo local, uma vez que se incentiva a utilização apenas dos melhores caminhos encontrados até o momento. A segunda alteração é a limitação dos valores de feromônio. No algoritmo original, podia ocorrer a estagnação em poucas iterações, pois o nível de feromônio na aresta da melhor solução encontrada crescia indefinidamente, enquanto a concentração de feromônio das outras arestas tendia a zero. Foram então definidas duas novas constantes, τ_{min} e τ_{max}, que são os limites mínimo e máximo do nível de feromônio, respectivamente. Impondo este limite, há um favorecimento da exploração do espaço de busca, pois mesmo as arestas que não pertencem às melhores soluções terão chances de serem escolhidas.

Conforme discutido em Stützle & Dorigo (1999) e Stützle & Hoos (1997), o parâmetro τ_{min} tem maior influência que τ_{max} na convergência do algoritmo. O parâmetro τ_{max} é calculado como o valor máximo esperado em uma situação na qual a solução ótima global é sempre obtida pelo algoritmo durante infinitas iterações. Partindo da Eq. 5.4 de atualização de feromônio e, dado que uma das formigas no algoritmo sempre constrói a solução ótima S_{opt}, o feromônio de uma aresta $(i,j) \in S_{opt}$ na iteração t será:

$$\tau_{i,j}(t) = (1-\rho)^t \cdot \tau_0 + \rho \cdot \frac{1}{f(S_{opt})} \sum_{j=1}^{t} (1-\rho)^{t-j} . \tag{5.6}$$

Como $\rho < 1$ e para $t \to \infty$, temos:

$$\lim_{t\to\infty} \tau_{i,j}(t) = \lim_{t\to\infty} (1-\rho)^t \cdot \tau_0 + \lim_{t\to\infty} \rho \cdot \frac{1}{f(S_{opt})} \sum_{j=1}^{t} (1-\rho)^{t-j} , \tag{5.7}$$

$$\lim_{t\to\infty} (1-\rho)^t \cdot \tau_0 = 0 , \tag{5.8}$$

$$\lim_{t\to\infty} \rho \cdot \frac{1}{f(S_{opt})} \sum_{j=1}^{t} (1-\rho)^{t-j} = \rho \cdot \frac{1}{f(S_{opt})} \cdot \frac{1}{\rho} = \frac{1}{f(S_{opt})} , \tag{5.9}$$

$$\tau_{max} = \lim_{t\to\infty} \tau_{i,j}(t) = \frac{1}{f(S_{opt})} . \tag{5.10}$$

Como o valor de $f(S_{opt})$ não é conhecido, ele é substituído pelo melhor valor da função-objetivo encontrado até o momento.

Para encontrar o valor de τ_{min}, os autores definiram p_{dec} como a probabilidade de uma aresta pertencente à solução ótima S_{opt} ser escolhida. Então, a probabilidade p_{best} da solução S_{opt} ser inteiramente construída para um problema com n cidades é definida como:

$$p_{best} = (p_{dec})^{(n-1)} , \tag{5.11}$$

$$p_{dec} = e^{\frac{\ln p_{best}}{n-1}} . \tag{5.12}$$

Como, na média, uma formiga tem que escolher entre $n/2$ cidades a cada passo (n é o número de cidades do PCV), podemos ajustar o parâmetro τ_{min} de forma que, da Eq. 5.12, temos:

$$p_{dec} = \frac{\tau_{\max}}{\tau_{\max} + \frac{n}{2} \cdot \tau_{\min}}. \tag{5.13}$$

Geralmente, é utilizado $\tau_{\min}$ de forma que $\tau_{\max}/\tau_{\min} = 2n$, resultando em $p_{\text{dec}} = 0{,}8$. Adicionalmente, para cada solução construída, algumas iterações de busca local são executadas de forma a obter uma solução melhorada porém, apenas as arestas da solução obtida antes da busca local sofrerão a atualização do feromônio.

5.4 Estrutura Hiper-Cubo para o ACO

Embora o MMAS tenha gerado resultados com bom desempenho, esse algoritmo ainda apresenta dificuldades em resolver problemas de dimensões elevadas, ou seja, problemas que envolvem um grande número de cidades. Em Blum & Dorigo (2004) e Blum et al. (2001), foi criada a *estrutura hiper-cubo* para o ACO (*ACO Hyper-Cube Framework* - HCF-ACO), na qual foram definidas alterações no espaço limite da trilha de feromônio, que passou a pertencer ao intervalo [0, 1], e na atualização dessa trilha de forma a manter esse limite.

Partindo da Eq. 5.4 e normalizando $\Delta\tau$ de forma a resultar em um valor no intervalo [0, 1], temos:

$$\tau_{i,j}(t+1) = (1-\rho) \cdot \tau_{i,j}(t) + \rho \cdot \Delta\tau_{i,j}, \tag{5.14}$$

$$\Delta\tau_{i,j} = \begin{cases} F(S) \Big/ \sum_{\{s \in Satu | i,j \in S\}} F(s) & se(i,j) \in S \\ 0 & c.c. \end{cases}, \tag{5.15}$$

$$F(S) = \frac{1}{f(S)}, \tag{5.16}$$

onde S_{atu} é o conjunto de soluções a serem atualizadas (no caso do MMAS, a melhor global e/ou a melhor local).

Dessa forma, é possível reescrever a Eq. 5.14 como segue:

$$\tau_{i,j}(t+1) = \tau_{i,j}(t) + \rho \cdot \left(\Delta\tau_{i,j} - \tau_{i,j}(t)\right), \tag{5.17}$$

que representa o deslocamento da trilha de feromônio na direção $\Delta\tau$ em ρ unidades.

5.5 IMP.MMAS: MMAS Melhorado

Em de França et al. (2004a; b), foi proposto o algoritmo IMP.MMAS (do inglês *IMProved MMAS*) para resolver o problema das *p-medianas capacitadas* (PMC). O problema PMC consiste em, encontrar a melhor localização de *p* fábricas (medianas) em um espaço discreto com *m* possíveis localizações pré-definidas, de forma a minimizar o custo de atendimento de demandas de *n* clientes, seguindo uma tabela de custos. Além disso, é necessário atender as restrições de demanda de cada cliente e capacidade de cada mediana. Um exemplo de aplicação é ilustrado na Figura 5.2, na qual, dentro de um conjunto contendo as 186 cidades brasileiras mais populosas, 5 são escolhidas para a construção de indústrias (medianas) que abastecerão todas essas cidades (clientes). Curiosamente a cidade de Campinas, que é um ponto estratégico de ligação entre a cidade de São Paulo e diversas cidades do interior do estado, foi escolhida pelo algoritmo como uma das medianas.

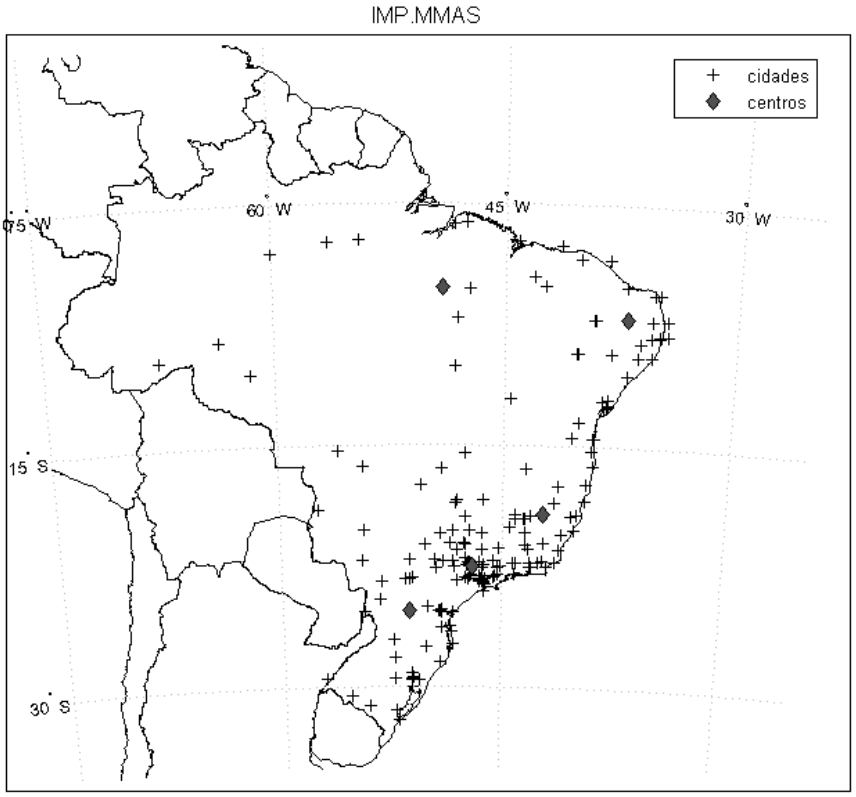


Figura 5.2: Exemplo de aplicação do problema PMC para um conjunto das 186 cidades brasileiras mais populosas. As cidades escolhidas para construção das indústrias foram: *Campinas (SP), Ipatinga (MG), Marabá (PA), Guarapuava (PR), Patos (PB)*.

Nestes trabalhos, foram propostas melhorias para o HCF-ACO que resultaram em soluções com qualidades ainda superiores. A primeira melhoria proposta refere-se ao cálculo da atualização do incremento do feromônio ($\Delta\tau$), sendo que na Eq. 5.15 a

distribuição dos valores de $\Delta\tau$ entre as soluções seria desigual, o que poderia resultar em valores muito pequenos que não influenciariam na trilha de feromônio, podendo levar a uma estagnação mais acelerada do algoritmo.

Calculando o valor da atualização do feromônio de maneira proporcional à relação entre a qualidade da solução e a melhor e a pior solução global, teremos:

$$\Delta\tau_{i,j} = \frac{F(S) - Fpior}{Fmelhor - Fpior}, \tag{5.18}$$

resultando em "1" caso $F(S)$ seja igual à melhor solução, e "0" caso ele represente a pior solução. Todas as soluções intermediárias, entre a melhor e a pior, têm seus valores atribuídos linearmente.

Outra alteração é referente à estagnação do algoritmo, que pode ocorrer quando o número de arestas com nível de feromônio que esteja a uma distância σ do valor limite máximo é igual ao número de vértices e o número de arestas com nível de feromônio que esteja a uma distância σ do valor limite mínimo é igual ao número de arestas menos o número de vértices, ou seja:

$$\sum_{\tau_{i,j} \geq \tau_{max} - \sigma} 1 = |V| \quad e \quad \sum_{\tau_{i,j} \leq \tau_{min} + \sigma} 1 = \left(|E| - |V|\right), \tag{5.19}$$

onde |V| é o número de vértices do grafo, |E| é o número de arestas e σ representa um valor de erro.

Nessa situação, a probabilidade de escolher o mesmo conjunto de arestas (as |V| arestas com nível de feromônio próximo ao valor máximo) é muito maior do que as outras arestas pertencentes ao grafo (as |E| - |V| arestas que têm o nível de feromônio próximo ao valor mínimo).

Toda vez que o algoritmo se encontra nesta condição estabelecida pela Eq. (5.19), é atribuído o valor τ_0 na trilha de feromônio de cada uma das arestas do grafo, restabelecendo o equilíbrio nas probabilidades de escolha de cada aresta. Embora os valores da trilha de feromônio são reinicializados, o procedimento de busca mantém os valores atuais de F_{melhor} e F_{pior}. Dessa forma os valores de $\Delta\tau_{i,j}$ calculado pela Eq. (5.18) tenderá a ser menor do que no início do algoritmo quando a diferença entre os valores de F_{melhor} e F_{pior} era menor. Isso faz com que, a cada nova reinicialização, o algoritmo tenda a fazer uma busca mais "cuidadosa" até a convergência.

5.6 Resultados em Ambiente Estáticos

O problema utilizado para realizar os testes no caso de domínio discreto será o problema do caixeiro viajante descrito anteriormente na seção 2.3.1. A escolha desse problema é devida à sua simplicidade de formulação e representação e por ser um problema amplamente empregado na avaliação de desempenho de algoritmos em problemas combinatórios.

Para realizar os testes iniciais em ambientes estáticos, quatro instâncias retiradas do TSPLIB foram utilizadas: att48, eil76, kroC100, ch150, contendo 48, 76, 100 e 150 cidades, respectivamente. O TSPLIB é um repositório público de instâncias para o PCV na Internet onde, além das instâncias, consta também o ótimo global para algumas delas. Este repositório se encontra no endereço:

http://www.iwr.uni-heidelberg.de/groups/comopt/soft/TSPLIB95/TSPLIB.html.

Primeiramente, foram feitos experimentos com diversas combinações de parâmetros dos algoritmos, de forma que aqueles parâmetros que apresentassem melhores resultados fossem utilizados nos experimentos finais e nos próximos capítulos. Para isso, 30 rodadas de cada experimento foram executadas para as três primeiras instâncias e os critérios de parada do algoritmo foram a obtenção de um ótimo global conhecido ou 60 iterações, o que ocorresse primeiro. A busca local utilizada foi o 3-opt, descrito na Seção 2.4.2, em no máximo 3 iterações para cada solução construída.

A primeira série de experimentos serviu para medir a sensibilidade do algoritmo aos parâmetros α e β, e é ilustrada resumidamente na Tabela 5.1.

Esses valores foram escolhidos de forma que fosse testado o comportamento do algoritmo utilizando apenas a informação do feromônio, e, em seguida, proporções diferentes entre a informação heurística e o feromônio.

Tabela 5.1: Experimentos de 1 a 6 para regular os valores de α e β.

Experimento	No. de formigas	α	β	ρ	q_0
1	10	3	0	0,1	0,4
2	10	3	1	0,1	0,4
3	10	3	2	0,1	0,4
4	10	1	3	0,1	0,4
5	10	2	3	0,1	0,4
6	10	1	1	0,1	0,4

Os resultados são apresentados nas Tabelas 5.2 a 5.4. Da esquerda para a direita, as colunas mostram o número do experimento, a média do melhor valor da função-objetivo encontrada durante os 30 experimentos, o seu desvio padrão, o valor mínimo obtido, o valor máximo, a média de iterações para se atingir o melhor valor e porcentagem referente à quantidade de ótimos globais obtidos nos experimentos.

Como pode ser visto nas Tabelas 5.2, 5.3 e 5.4, os parâmetros que resultaram nos melhores desempenhos foram $\alpha = 2$ e $\beta = 3$ (experimento 5). Definidos os valores de α e β, o parâmetro ρ deve ser investigado. Foi definido um domínio $\rho \in [0{,}1..0{,}4]$ que indica a taxa de evaporação de feromônio: quanto maior o valor de ρ, mais rápido o feromônio irá se dissipar caso não seja reforçado. A Tabela 5.5 detalha esses experimentos.

Tabela 5.2: Resultados dos experimentos de 1 a 6 para a instância att48. O melhor resultado é apresentado em negrito.

att48						
Experimento	**Média**	**Desvio Padrão**	**Mínimo**	**Máximo**	**Média de Iterações**	**% de sucesso na localização do ótimo global**
1	10719,53	92,25	10628	11023	59,26	10,00
2	10693,73	53,25	10628	10840	58,26	13,30
3	10665,27	46,15	10628	10840	51,20	**30,00**
4	10645,57	25,95	10628	10738	52,53	**50,00**
5	**10634,17**	**10,47**	**10628**	**10653**	**41,53**	**73,30**
6	10961,17	178,48	10684	11322	60,00	**0,00**

Tabela 5.3: Resultados dos experimentos de 1 a 6 para a instância eil76. O melhor resultado é apresentado em negrito.

eil76						
Experimento	**Média**	**Desvio Padrão**	**Mínimo**	**Máximo**	**Média de Iterações**	**% de sucesso na localização do ótimo global**
1	591,10	14,89	568	618	60,00	0,00
2	552,73	6,91	542	571	60,00	0,00
3	543,23	4,30	538	553	57,00	20,00
4	549,26	6,59	539	571	60,00	0,00
5	**540,73**	**3,43**	**538**	**551**	**54,33**	**36,60**
6	622,33	15,84	597	654	60,00	0,00

Tabela 5.4: Resultados dos experimentos de 1 a 6 para a instância kroC100. O melhor resultado é apresentado em negrito.

kroC100						
Experimento	**Média**	**Desvio Padrão**	**Mínimo**	**Máximo**	**Média de Iterações**	**% de sucesso na localização do ótimo global**
1	27321,63	1336,29	24930	30353	60,00	0
2	21425,60	474,00	20753	22731	60,00	0
3	20851,67	112,79	20749	21120	57,00	30
4	21055,20	171,76	20780	21613	60,00	0
5	**20826,07**	**104,00**	**20749**	**21197**	**56,06**	**40**
6	25440,77	999,32	23681	27684	60,00	0

Tabela 5.5: Experimentos de 7 a 10 para determinar o valor de ρ.

Experimento	**No. de formigas**	α	β	ρ	q_0
7	10	2	3	0,1	0,4
8	10	2	3	0,2	0,4
9	10	2	3	0,3	0,4
10	10	2	3	0,4	0,4

As Tabelas 5.6 a 5.8 mostram os resultados obtidos nesses experimentos. Em seqüência, as colunas mostram o número do experimento, a média do melhor valor da função-objetivo encontrado durante os 30 experimentos, o seu desvio padrão, o valor mínimo obtido, o valor máximo, a média de iterações para se atingir o melhor valor e porcentagem referente à quantidade de ótimos globais obtidos nos experimentos.

A ordem de prioridade na escolha do experimento com os melhores resultados ficou definida, de forma decrescente, como: porcentagem de ótimos encontrados (o número de acertos é importante), média do valor da função-objetivo (pois uma média baixa indica resultados de boa qualidade quando o ótimo global não é encontrado) e, finalmente, média de iterações, que define a velocidade com que se converge para o ótimo global. Nas duas primeiras instâncias, o experimento 7 teve melhores resultados e na terceira instância o experimento 8 apresentou melhores resultados. Como nessa última instância o experimento 7 não apresentou grande diferença de desempenho para o melhor caso, este foi o escolhido para seguir com os experimentos.

Tabela 5.6: Resultados dos experimentos de 7 a 10 para a instância att48, melhor resultado em negrito.

att48						
Experimento	**Média**	**Desvio Padrão**	**Mínimo**	**Máximo**	**Média de Iterações**	**% de sucesso na localização do ótimo global**
7	**10634,17**	**10,47**	**10628**	**10653**	**41,53**	**73,30**
8	10647,07	25,45	10628	10711	39,86	56,60
9	10653,20	30,24	10628	10738	44,73	46,60
10	10660,93	32,91	10628	10767	47,53	33,30

Tabela 5.7: Resultados dos experimentos de 7 a 10 para a instância eil76, melhor resultado em negrito.

eil76						
Experimento	**Média**	**Desvio Padrão**	**Mínimo**	**Máximo**	**Média de Iterações**	**% de sucesso na localização do ótimo global**
7	**540,73**	**3,43**	**538**	**551**	**54,33**	**36,60**
8	540,70	3,32	538	551	54,26	33,30
9	541,23	3,43	538	552	53,26	20,00
10	543,36	5,25	538	554	53,73	20,00

Tabela 5.8: Resultados dos experimentos de 7 a 10 para a instância kroC100, melhor resultado em negrito

kroC100						
Experimento	**Média**	**Desvio Padrão**	**Mínimo**	**Máximo**	**Média de Iterações**	**% de sucesso na localização do ótimo global**
7	20826,07	104,01	20749	21197	56,06	40,00
8	**20803,60**	**86,40**	**20749**	**21073**	**46,86**	**50,00**
9	20821,43	92,39	20749	21010	44,80	46,60
10	20827,93	119,87	20749	21194	40,86	50,00

Definido então o valor ótimo de ρ, resta apenas definir $q_0 \in$ [0,1; 0,9], como descrito na Tabela 5.9.

Tabela 5.9: Experimentos de 11 a 19 para regular o valor de q_0.

Experimento	No. de formigas	α	β	ρ	q_0
11	10	2	3	0,1	0,1
12	10	2	3	0,1	0,2
13	10	2	3	0,1	0,3
14	10	2	3	0,1	0,4
15	10	2	3	0,1	0,5
16	10	2	3	0,1	0,6
17	10	3	2	0,1	0,7
18	10	3	2	0,1	0,8
19	10	3	2	0,1	0,9

Os resultados são apresentados nas tabelas 5.10 a 5.12. Em seqüência, as colunas mostram o número do experimento, a média do melhor valor da função-objetivo encontrada durante os 30 experimentos, o seu desvio padrão, o valor mínimo obtido, o valor máximo, a média de iterações para se atingir o melhor valor e porcentagem referente à quantidade de ótimos globais obtidos nos experimentos.

Tabela 5.10: Resultados dos experimentos de 11 a 19 para a instância att48, melhor resultado em negrito.

att48						
Experimento	**Média**	**Desvio Padrão**	**Mínimo**	**Máximo**	**Média de Iterações**	**% de sucesso na localização do ótimo global**
11	10644,00	27,64	10628	10738	47,20	20
12	10646,97	25,88	10628	10738	50,33	15
13	10658,57	39,20	10628	10767	46,93	15
14	10634,17	10,48	10628	10653	41,53	22
15	10637,87	22,87	10628	10738	40,73	22
16	10637,87	14,40	10628	10684	39,67	19
17	10635,43	20,79	10628	10725	33,53	25
18	10632,87	12,42	10628	10684	31,20	25
19	**10632,53**	**11,92**	**10628**	**10684**	**25,53**	**25**

Tabela 5.11: Resultados dos experimentos de 11 a 19 para a instância eil76, melhor resultado em negrito.

eil76						
Experimento	**Média**	**Desvio Padrão**	**Mínimo**	**Máximo**	**Média de Iterações**	**% de sucesso na localização do ótimo global**
11	543,17	4,21	538	553	59,47	3
12	541,90	2,94	538	548	59,00	3
13	541,53	3,15	538	548	58,00	8
14	**540,73**	**3,43**	**538**	**551**	**54,33**	**11**
15	540,43	2,14	538	545	56,84	8
16	541,26	2,64	538	548	57,60	5
17	540,96	2,80	538	551	55,73	7
18	540,33	2,04	538	545	55,20	5
19	540,57	2,57	538	549	58,60	2

Tabela 5.12: Resultados dos experimentos de 11 a 19 para a instância kroC100, melhor resultado em negrito.

kroC100						
Experimento	**Média**	**Desvio Padrão**	**Mínimo**	**Máximo**	**Média de Iterações**	**% de sucesso na localização do ótimo global**
11	20872,97	133,31	20749	21241	58,67	6
12	20835,33	85,87	20749	21021	59,40	6
13	20804,23	56,09	20749	20886	57,93	11
14	20826,07	104,01	20749	21197	56,07	12
15	20791,37	79,19	20749	21126	52,93	17
16	20783,63	51,91	20749	20880	50,93	18
17	**20789,57**	**77,34**	**20749**	**21010**	**44,60**	**21**
18	20790,07	71,24	20749	21010	46,67	19
19	20783,53	55,52	20749	20880	38,80	20

Como, neste caso, os resultados diferem de uma instância para outra e se torna difícil determinar um único valor de q_0 que seja o melhor para as três instâncias, foi efetuado uma média ponderada dos valores, seguindo o seu grau de importância, da seguinte forma:

$$ponto = \frac{4.(100 - pa) + 3.med + 2.medit + 1.desv}{10}, \tag{5.20}$$

onde *pa* é a porcentagem de acertos, *med* é a média do valor da função-objetivo, *medit* é a média de iterações e *desv* é o desvio padrão do número de iterações. Essa ponderação foi feita levando em conta o que é desejável no algoritmo, em primeiro lugar que ele obtivesse um maior número de acertos, em segundo que a média dos valores obtidos fosse a menor

possível (indicando uma proximidade maior com o ótimo global), em terceiro a média de iterações utilizadas para obter os resultados e, por último, o desvio padrão que indica a oscilação de resultados obtidos. Note que a fórmula foi ajustada para que quanto menor a pontuação melhor o resultado do experimento. Os resultados são mostrados na Tabela 5.13.

Tabela 5.13: Pontuação dos experimentos de 11 a 19 para cada uma das instâncias, melhores resultados em negrito.

Experimento	att48	eil76	kroC100
11	3218,737	211,2644	6318,955
12	3226,745	210,664	6303,067
13	3230,876	203,7082	6283,799
14	3210,271	**198,7633**	6293,434
15	3212,46	203,0511	6273,249
16	3215,4	207,4972	6266,468
17	3206,083	204,3831	6265,524
18	3204,009	206,6773	6268,144
19	**3202,725**	211,4802	**6261,705**

Note que, embora permaneça um impasse entre as instâncias att48 e kroC100, houve um consenso no experimento 19, mas a pontuação obtida nesse experimento pela instância eil76 está muito alta. Verificando os experimentos com valores de pontuação próximos dos melhores, verificou-se que o experimento 17 traz um bom desempenho para todas as instâncias.

Finalmente, definidos todos os parâmetros de entrada do algoritmo, os experimentos finais serão executados com um número maior de formigas e de iterações, como descrito na Tabela 5.14.

Tabela 5.14: Parâmetros utilizados para os testes finais.

No. de formigas	α	β	ρ	q_0
40	2	3	0,1	0,7

Adicionalmente, cinco iterações da busca local 3-opt foram executadas para cada solução construída.

Tabela 5.15: Resultados finais para as 4 instâncias.

Resultados Finais							
Instância	**Ótimo Global**	**Média**	**Desvio Padrão**	**Mínimo**	**Máximo**	**Média de Iterações**	**% de sucesso na localização do ótimo global**
Att48	10628	10628,00	0,00	10628	10628	8,20	100,00
Eil76	538	538,03	0,18	538	539	22,80	96,67
kroC100	20749	20749,00	0,00	20749	20749	17,53	100,00
Cho150	6528	6552,97	9,32	6528	6566	97,87	3,33

Na Tabela 5.15, é possível verificar que a porcentagem da quantidade de vezes que o ótimo global foi encontrado está em 100% ou próximo disso, com exceção da última instância. Mas mesmo neste caso, embora tenha sido obtida uma porcentagem muito baixa de sucesso, o valor médio obtido ficou abaixo de 0,5% em relação ao ótimo global.

5.7 Síntese do Capítulo

Neste capítulo, foram apresentados conceitos de otimização inspirada no comportamento de colônias de formigas e, em seguida, foram apresentadas melhorias que procuram suprir deficiências do algoritmo padrão de modo a melhorar seu desempenho, chegando no algoritmo final denominado IMP.MMAS.

Em seguida, diversos experimentos foram feitos com instâncias conhecidas da literatura, de forma a ajustar os parâmetros do algoritmo para obter o melhor resultado possível. Após esta análise de sensibilidade paramétrica, os testes finais foram feitos. Como resultado, as três instâncias menores obtiveram quase 100% de resultados ótimos em todos os experimentos e a quarta instância obteve, na média, um resultado muito próximo do ótimo, sugerindo que o algoritmo está otimizado para enfrentar os problemas dinâmicos que serão abordados no Capítulo 7.

Capítulo 6

copt-aiNet

Neste capítulo, será apresentado o algoritmo derivado da opt-aiNet para resolver problemas combinatórios veiculados a espaços de busca discretos, denominado copt-aiNet (do inglês *Artificial immune Network for Combinatorial Optimization*). Cada detalhe do algoritmo será explicado e as características comuns aos sistemas imunológicos artificiais serão apontadas. Para avaliação de desempenho, serão executados os mesmos testes feitos no capítulo anterior com o algoritmo IMP.MMAS.

6.1 copt-aiNet: opt-aiNet para Problemas Combinatórios

O algoritmo copt-aiNet, proposto em de Sousa et al. (2003), foi inicialmente aplicado ao problema de *reordenação de dados de expressão gênica,* e seu objetivo era estender a opt-aiNet para resolver problemas de otimização combinatória. Para tanto, ele deve possuir características específicas definidas na opt-aiNet, como controle e manutenção de diversidade, controle dinâmico do tamanho da população e convergência para múltiplos ótimos locais (busca multimodal). Todas essas características são controladas, principalmente, pela função de supressão do algoritmo que, neste caso, é definida como a quantidade de passos necessários para transformar uma solução em outra. Este mecanismo obedece a todas as propriedades de uma métrica de distância: simetria, desigualdade triangular e não-negatividade. Como resultado, o algoritmo retorna várias soluções com valores similares da função-objetivo, mas com características diferentes, conforme ilustrado na Figura 6.1.

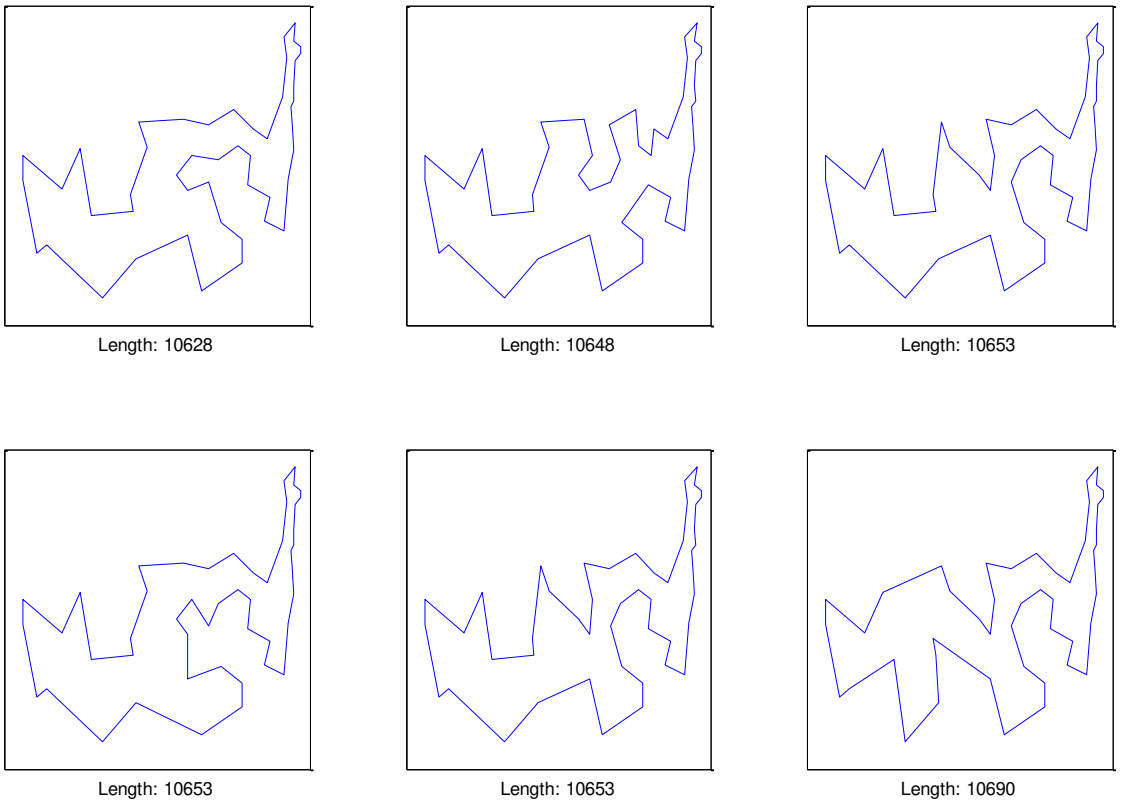


Figura 6.1: Múltiplas soluções encontradas pelo algoritmo copt-aiNet em uma rodada. Note que, embora as distâncias totais dos percursos sejam muito próximas entre si, o caminho adotado sofre algumas alterações visíveis.

Como é possível observar, todas as alternativas de solução ilustradas na Figura 6.1 apresentam bom desempenho, enquanto os percursos dessas soluções são visivelmente distintos, a menos da região mais à direita, com uma seqüência vertical de cidades. A existência de múltiplas propostas de solução é desejável quando avaliações a posteriori são realizadas com base em critérios que não foram necessariamente considerados durante a busca. Nestes casos, poderia ser preferível percorrer um caminho um pouco mais longo em troca de algum outro benefício. Uma situação típica seria a adoção de uma rota alternativa para evitar engarrafamentos, uma via interditada, etc.

O pseudocódigo apresentado no Algoritmo 6.1 descreve a copt-aiNet de forma similar à opt-aiNet (Algoritmo 3.4).

```
[Ab,S] = Função copt-aiNet(N0,L,Nc,m,d,k,M,max);
      Ab := gera(N0,L);
      fit = f(Ab);
      Enquanto critério_convergencia é falso faça,
            C = clona(Ab,1,Nc);
            C* = mutação(C);
            fit* = f(C*);
            M = seleciona(C*,fit*);
            Se media de fitness não melhorou,
                  [Ab,fit] = suprime(Ab);
            Fim
            Se tamanho(Ab) < m,
                  [Ab,fit] = insere(Ab,fit,d);
            Fim
            Se não houve melhoras das k melhores células por M iterações,
                  [Ab,fit] = maturação(Ab,fit);
            Fim
            Se nenhum das k melhores células melhorou por max iterações,
                  critério_convergencia = verdadeiro;
            Fim
      Fim
Fim
```

Algoritmo 6.1: Pseudo-código do algoritmo copt-aiNet.

Note que este algoritmo contém muitas semelhanças com a opt-aiNet. As principais diferenças estão na representação de cada anticorpo e nos processos internos de mutação, maturação e supressão. A representação dos anticorpos irá depender do problema a ser tratado. Por exemplo, no caso do caixeiro viajante uma permutação de inteiros representando a seqüência de vértices da solução é empregada. Nas próximas seções, cada passo do algoritmo será descrito detalhadamente.

6.1.1 Criação da População Inicial

A população inicial é construída aleatoriamente, partindo de uma solução factível e trocando sucessivamente os elementos do vetor solução de forma aleatória. Esse processo é ilustrado no Algoritmo 6.2.

```
[Ab] = Função gera(N0,L);
      Para i = 1 .. N0,
            Para j = 1 .. L,
                  Ab[i][j] = j;
            Fim
            Para j = 1 .. L,
                  troca = rand(1,L);
                  temp = Ab[i][j];
                  Ab[i][j] = Ab[i][troca];
                  Ab[i][troca] = temp;
            Fim
      Fim
Fim
```

Algoritmo 6.2: Criação da população inicial na copt-aiNet.

Inicialmente, o vetor solução é inicializado com os vértices de forma seqüencial. Em seguida, são executadas *L* trocas aleatórias, sendo *L* o número de arestas, gerando assim um indivíduo factível e aleatório.

6.1.2 Mutação

No processo de mutação, cada clone sofre um determinado número de modificações proporcional ao valor da função-objetivo de sua solução. Há três tipos possíveis de modificações que podem ser escolhidos aleatoriamente, conforme descritos em Herdy (1990): i) mutação por inserção; ii) mutação inversiva; e iii) mutação por troca cruzada.

Mutação por Inserção (*Insertion Mutation*)

Neste tipo de mutação um elemento do vetor solução é removido de sua posição atual e recolocado em outra posição escolhida aleatoriamente (Figura 6.2).

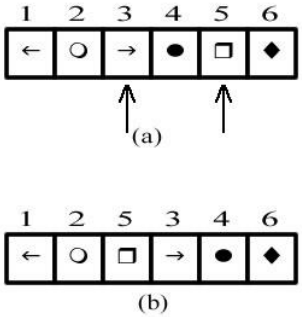


Figura 6.2: Mutação por Inserção: (a) o elemento na posição 5 é escolhido para ser inserido na posição 3, (b) o processo é efetuado deslocando todos os elementos que se situam após a posição escolhida.

Mutação Inversiva (*Exchange Mutation*)

A mutação inversiva simplesmente escolhe aleatoriamente dois elementos do vetor solução, permutando suas posições (Figura 6.3).

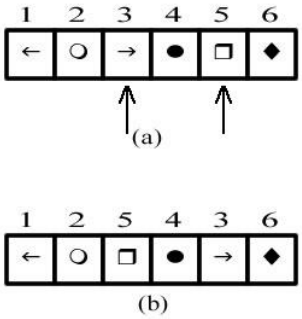


Figura 6.3: Mutação Inversiva: (a) os elementos das posições 3 e 5 são escolhidos para efetuar a troca, (b) estes elementos são trocados sem que nenhum outro seja afetado.

Mutação por Troca Cruzada (*Cross-Exchange Mutation*)

Na mutação por troca cruzada, uma faixa de elementos do vetor solução é invertida aleatoriamente (Figura 6.4).

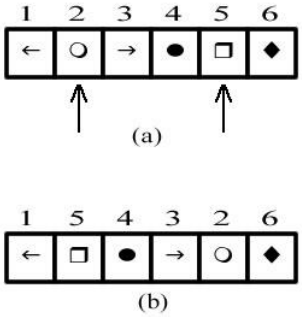


Figura 6.4: Mutação por Troca Cruzada: (a) a faixa de elementos que será invertida é selecionada e (b) suas posições são rotacionadas.

6.1.3 Supressão

A função de supressão, como explicado anteriormente, necessita de uma medida de similaridade entre os anticorpos que permita verificar se dois indivíduos estão próximos entre si, ou seja, quais as chances dos dois convergirem para o mesmo ótimo local. No caso de problemas combinatórios, não há uma superfície de fitness contínua e com ótimos locais bem definidos, dificultando essa tarefa.

Para resolver este problema, o cálculo de similaridade entre dois anticorpos da copt-aiNet será feito comparando o número de trocas necessárias para transformar um dos anticorpos em outro. Conceitualmente o procedimento é descrito no Algoritmo 6.3.

```
[num_ops] = Função calcula_similaridade(cell1, cell2);
      k = 1;
      num_ops = 0;
      cur_elem = cell1[k];
      final = falso;
      m = índice de cell2 tal que cell2[m] = cur_elem;
      Enquanto final = falso faça.
            k = k + 1;
            cur_elem = cell1[k];
            n = índice de cell2 tal que cell2[n] = cur_elem;
            Se m+1 > num_vertices então,
                  idx1 = 1;
            Senão
                  idx1 = m+1;
            Fim
            Se m-1 < 1 então,
                  idx2 = num_vertices;
            Senão
                  idx2 = m-1;
            Fim
            Se n ≠ idx1 E n ≠ idx2 então,
                  num_ops = num_ops + 1;
            Fim
            m = n;
            Se k = num_vertices então,
                  final = verdadeiro;
            Fim
      Fim
Fim
```

Algoritmo 6.3: Cálculo de similaridade entre duas células.

O algoritmo consiste basicamente em verificar quantas trocas de aresta são necessárias para transformar uma solução em uma outra. Dados dois anticorpos, $cell_1$ e $cell_2$, inicialmente o primeiro elemento de $cell_1$ é guardado, assim como a respectiva posição m deste mesmo elemento em $cell_2$. Em seguida, é verificado se o elemento de $cell_2$ na posição j+1 ou j−1 coincide com o próximo elemento de $cell_1$. O processo é então repetido para as outras arestas de $cell_1$.

Com o cálculo de similaridade definido, existem duas opções para eliminar anticorpos da população: eliminar elementos com similaridade menor ou igual a um limiar pré-definido, como é feito tradicionalmente na opt-aiNet; ou eliminar os n elementos mais similares.

6.1.4 Inserção de Novos Indivíduos

O processo de inserção de novos indivíduos na população é feito através de um processo de recombinação dos melhores anticorpos, preservando assim a memória da população ao mesmo tempo em que a diversidade é mantida.

Inicialmente, dois anticorpos, dentre os melhores, são escolhidos aleatoriamente para gerar o novo anticorpo. Partindo de um vértice inicial, é verificado se existem arestas

em comum entre os dois anticorpos e que partem deste vértice. Em caso afirmativo, esta aresta passa a fazer parte da nova solução, caso contrário a aresta factível de menor distância é utilizada.

```
[Ab] = Função insere_celula();
      [cell1, cell2] = seleciona_aleat(n_best);
      final = falso;
      vert = random(1,tamanho_vertices);
      i = n_cells;
      idx = 1;
      Ab[i][idx] = vert;
      Enquanto final = falso faça,
            idx = idx + 1;
            Se aresta1(cell1[vert]) = aresta1(cell2[vert]) então,
                  Ab[i][idx] = aresta1(cell[vert]);
            Senão Se aresta1(cell1[vert]) = aresta2(cell2[vert]) então,
                  Ab[i][idx] = aresta1(cell[vert]);
            Senão Se aresta2(cell1[vert]) = aresta1(cell2[vert]) então,
                  Ab[i][idx] = aresta2(cell[vert]);
            Senão Se aresta2(cell1[vert]) = aresta2(cell2[vert]) então,
                  Ab[i][idx] = aresta2(cell[vert]);
            Senão
                  Ab[i][idx] = vértice_mais_próximo(vert);
            Fim
            Se tamanho(Ab[i]) = tamanho_vertices então,
                  final = verdadeiro;
            Fim
      Fim
Fim
```

Algoritmo 6.4: Criação da população inicial.

Inicialmente, a função `seleciona_aleat()` seleciona dois anticorpos aleatoriamente dentro do conjunto de melhores indivíduos. Em seguida, o vértice inicial (*vert*) é escolhido aleatoriamente e inserido como primeiro elemento da nova solução. Dentro do laço principal, é verificado se alguma aresta das duas soluções partindo de *vert* coincide. Em caso afirmativo, essa aresta é inserida na nova solução. Caso contrário, o vértice mais próximo é escolhido.

6.1.5 Maturação

O processo de maturação nada mais é que uma busca local ou alguma meta-heurística não populacional. Na copt-aiNet original, é aplicada a busca tabu (Glover & Kochenberger, 2002; Michalewicz & Fogel, 2000).

A busca tabu consiste em uma busca local com memória de curta duração utilizada de forma a evitar explorar o mesmo local mais de uma vez por determinado tempo. Isso proíbe que um mesmo movimento se repita por um determinado número de iterações, ou seja, esse movimento se torna *tabu* durante um certo período de tempo.

No caso do problema do caixeiro viajante, são aplicados seqüencialmente os 3 movimentos descritos no processo de mutação, com a diferença de que os elementos

escolhidos não são aleatórios, mas sim o melhor movimento possível e que não esteja marcado como tabu.

6.2 Resultados em Ambientes Estáticos

Para realizar os testes em ambientes estáticos, quatro instâncias foram retiradas do TSPLIB, um repositório público de instâncias do problema do caixeiro viajante utilizadas para testes: att48, eil76, kroC100, ch150, contendo 48, 76, 100 e 150 cidades, respectivamente.

Os testes foram feitos seguindo a mesma metodologia experimental do capítulo anterior. Portanto, foram executadas 30 rodadas de cada experimento até que se atingisse o ótimo ou que um máximo de 3.500 iterações fosse atingido. O número de iterações foi muito maior do que nos experimentos do ACO, pois o tempo de cada iteração na copt-aiNet é muito menor. Os resultados são apresentados na Tabela 6.1.

Tabela 6.1: Resultados iniciais da copt-aiNet para as 4 instâncias.

Resultados Finais						
Instância	**Média**	**Desvio Padrão**	**Mínimo**	**Máximo**	**Média de Iterações**	**% de ótimos globais encontrados**
att48	10629,17	4,86	10628	10653	1743,53	93,33
eil76	544,36	2,78	539	552	3500,00	0,00
kroC100	21075,20	131,19	20749	21356	3496,63	3,33
cho150	6682,50	49,36	6572	6743	3500,00	000

Os resultados iniciais mostram um baixo desempenho na otimização das instâncias escolhidas, se comparado aos resultados obtidos com o ACO no capítulo anterior. Com exceção da primeira instância, todas as outras tiverem uma porcentagem de ótimos globais encontrados próximos ou iguais a zero.

6.2.1 Aumentando a Vizinhança de Busca

Devido ao baixo desempenho na qualidade das soluções encontradas pela copt-aiNet quando comparado ao ACO, sentiu-se a necessidade de propor modificações no algoritmo capazes de promover uma melhora na qualidade das soluções. Para tanto, o mesmo artifício usado no IMP.MMAS foi utilizado aqui: a cada mutação os anticorpos passaram por um processo de busca local 3-opt com duas iterações. O número reduzido de iterações da busca local é importante para manter ao máximo a diversidade da população enquanto se busca soluções de qualidade. Os resultados são apresentados na Tabela 6.2.

Tabela 6.2: Resultados iniciais do copt-aiNet para as 4 instâncias.

Resultados Finais						
Instância	**Média**	**Desvio Padrão**	**Mínimo**	**Máximo**	**Média de Iterações**	**% de ótimos globais encontrados**
att48	10628,00	0,00	10628	10628	12,90	100
eil76	538,00	0,00	538	538	45,80	100
kroC100	20749,00	0,00	20749	20749	24,53	100
cho150	6531,13	6,82	6528	6549	188,93	80

Na Tabela 6.2, podemos verificar uma melhora significativa na qualidade dos resultados, obtendo 100% de ótimos globais nas 3 primeiras instâncias e 80% na última, um resultado excelente quando comparado ao ACO. Adicionalmente, podemos observar que a média de iterações necessárias para obtenção do ótimo global é aproximadamente o dobro da média do ACO, lembrando que cada iteração da copt-aiNet leva menos tempo que no ACO, desqualificando o número de iterações como um fator de comparação. Essa diferença é ilustrada na Tabela 6.3, onde a média do tempo por iteração que cada algoritmo utiliza para cada uma das instâncias estudadas é apresentada. Os testes foram feitos em um Athlon XP 2000+ (~1.67 GHz) com 512 MB de RAM rodando em ambiente Linux Slackware 10.1 e os códigos compilados com GCC 3.2.

Tabela 6.3: Comparação de tempo médio por iteração e tempo médio total de cada um dos algoritmos estudados.

tempo médio por iteração (seg.) / tempo médio total (seg.)		
	ACO	**copt-aiNet**
att48	0,52/5,94	0,03/5,62
eil76	1,77/64,31	0,10/54,51
kroC100	4,46/165,55	0,12/127,27
cho150	11,17/1205,23	0,29/634,26

6.3 Síntese do Capítulo

Nesse capítulo, foi apresentado o algoritmo denominado copt-aiNet, similar à opt-aiNet e à dopt-aiNet, mas projetado especificamente para resolver problemas de otimização combinatória. Os mesmos testes efetuados com o algoritmo ACO no capítulo anterior foram repetidos com a copt-aiNet para que fosse feita uma comparação de desempenho. Os resultados iniciais produziram um bom desempenho médio, principalmente considerando que o algoritmo encontra múltiplas soluções ótimas com valor da função-objetivo parecido, mas houve uma dificuldade em encontrar os ótimos globais dos problemas propostos. Para melhorar seu desempenho, foi utilizada a busca local 3-opt descrita anteriormente. Com esta modificação, a copt-aiNet obteve uma melhora significativa na qualidade das soluções, superando o ACO, ao mesmo tempo em que mantém a diversidade de soluções. Esta propriedade de manutenção de diversidade será melhor explorada ao longo do próximo capítulo, quando serão abordados problemas de otimização dinâmica.

Capítulo 7

IMP.MMAS × copt-aiNet: Resultados em Ambientes Dinâmicos

Neste capítulo, toda a metodologia utilizada para introduzir dinâmica nas instâncias do problema do caixeiro viajante será apresentada, assim como o método utilizado para comparar o desempenho dos dois algoritmos estudados nesta dissertação. Por fim, serão apresentados os resultados obtidos, as análises e conclusões.

7.1 Caixeiro Viajante Dinâmico

O tipo de dinâmica estudada neste trabalho para o caixeiro viajante será na forma de mudanças nos valores dos pesos das arestas. Esse tipo de mudança se justifica, pois os problemas de mundo real mais comuns, como roteamento de redes e congestionamento de tráfego, são representados justamente por um aumento ou redução no custo de certos trechos do percurso. Além disso, os casos de remoção ou adição de arestas e vértices são facilmente representáveis, alterando os valores de custos: um custo muito alto pode, por exemplo, ser equivalente a remover uma aresta do grafo.

Para realizar testes concisos de velocidade de reação a mudanças do ambiente, é necessário garantir que as mudanças realizadas nos pesos das arestas sejam feitas de tal forma que o novo ótimo seja diferente do ótimo atual. Uma maneira de fazer isso, supondo que o ótimo global das instâncias é conhecido, é efetuar uma troca entre algumas arestas ótimas por outras arestas escolhidas aleatoriamente, o processo é similar a um *k*-opt. O processo é ilustrado nas Figuras de 7.1 à 7.3, com um movimento 7-opt.

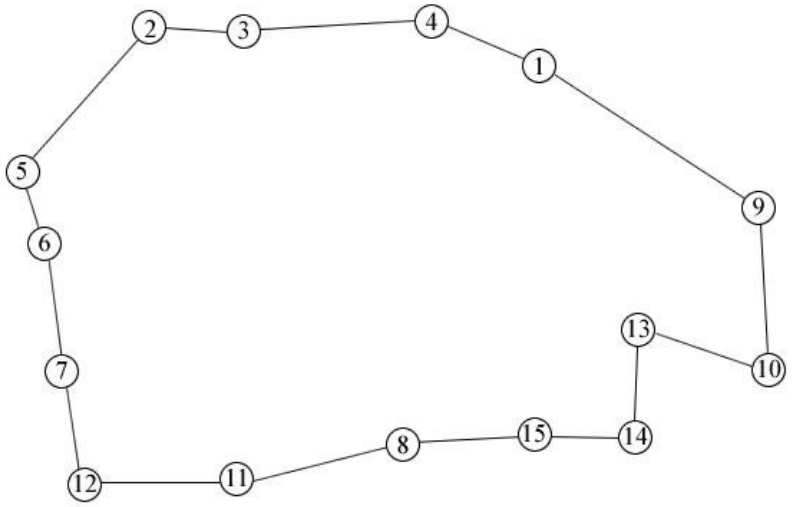


Figura 7.1: Solução ótima inicial em um grafo com 15 vértices.

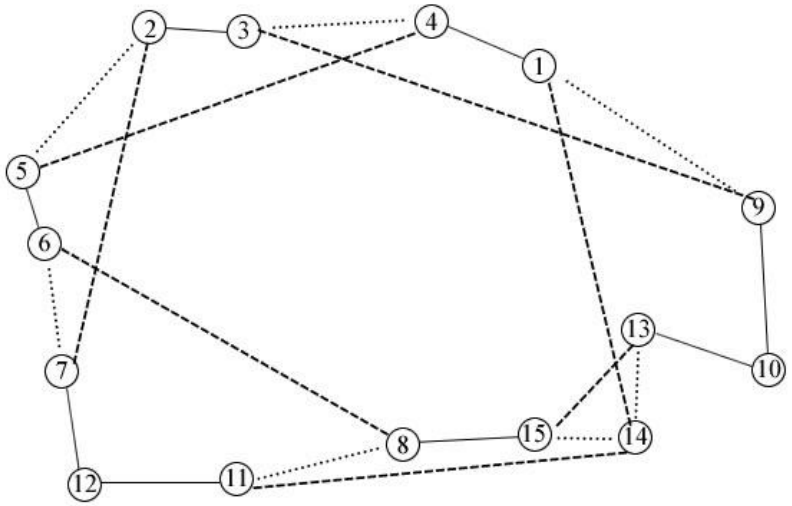


Figura 7.2: Gerando a mudança no grafo de forma a alterar a posição dos ótimos globais. Os valores dos pesos das arestas pontilhadas serão trocados com os valores das arestas serrilhadas, estas arestas são escolhidas de forma a manter uma solução factível de mesmo valor da função-objetivo ótima inicial, mas com uma rota diferente.

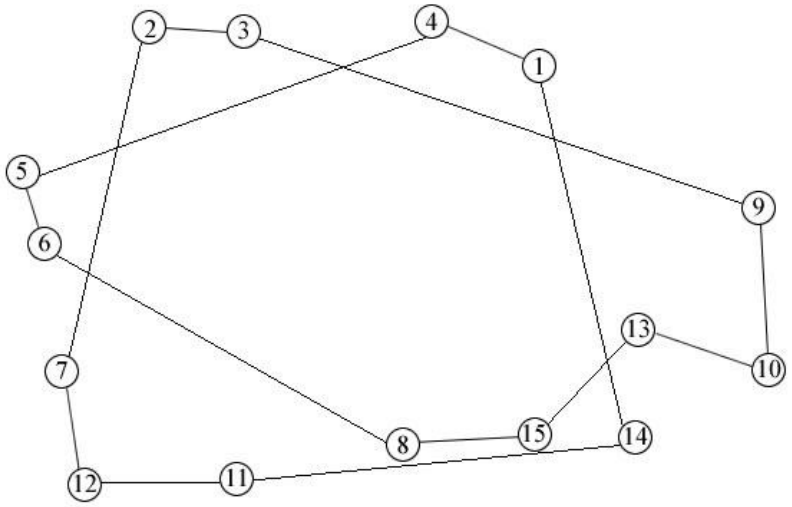


Figura 7.3: Após o processo, o valor da função-objetivo da rota ótima inicial irá aumentar enquanto essa nova rota terá o valor da função-objetivo da rota ótima original. No entanto, essa rota não necessariamente será o ótimo global.

Para os experimentos, foram geradas 10 mudanças com movimentos 11-opt, conforme descrito acima. Esse número de trocas foi escolhido de forma que a mudança não perturbasse muitas arestas em relação ao total (a menor instância possui 48 arestas por solução) e que o número seja ímpar para facilitar a escolha das arestas a substituir. A cada troca, o novo ótimo global é estimado através do método Lin-Kernighan descrito no Capítulo 2. Note que o método Lin-Kernighan possui um alto desempenho, conseguindo obter resultados muito próximos do ótimo em poucas iterações para instâncias pequenas. A comparação desse método com outros algoritmos se torna necessária apenas para instâncias com um número elevado de vértices.

7.1.1 Experimento Realizado

Para que as comparações sejam feitas de forma justa, foi determinado que as mudanças dos pesos nas arestas ocorressem em certos períodos de tempo, independentemente do número de iterações, pois cada iteração do ACO é equivalente a muitas iterações da copt-aiNet. Os tempos foram ajustados baseado no maior tempo entre os dois algoritmos necessário para otimizar cada uma das instâncias. Esses tempos estão representados na tabela a seguir:

Tabela 7.1: Período de mudanças no ambiente do PCV.

Instância	Período de Mudança (seg.)
Att48	20
Eil76	70
kroC100	180
Ch150	450

Cada experimento foi executado 30 vezes e, em seguida, foi calculada a média do menor valor obtido da função-objetivo, o menor valor, o maior valor, a média de tempo para se obter esse melhor valor, a porcentagem de ótimos globais encontrados e o desvio da média em relação ao ótimo global calculado por (Média – Valor Ótimo)/(Valor Ótimo). A coluna denominada "mudança" é referente a cada período de tempo em que o ambiente permanece estático. Portanto, o ótimo global entre cada linha é diferente.

Tabela 7.2: Resultados para a instância att48 obtidos com o algoritmo copt-aiNet.

att48						
Mudança	**Média**	**Mínimo**	**Máximo**	**Tempo médio**	**% de acertos**	**Distância em relação ao ótimo (%)**
1	10628,00	10628	10628	2,26	100,00	0,00
2	9819,00	9819	9819	1,36	100,00	0,00
3	10323,00	10323	10323	0,60	100,00	0,00
4	10225,03	10222	10306	10,00	90,00	0,03
5	10133,00	10133	10133	3,13	100,00	0,00
6	10306,40	10302	10367	0,83	86,66	0,04
7	10346,33	10346	10351	1,93	93,33	0,00
8	10307,03	10283	10337	11,53	3,33	0,23
9	10434,07	10415	10461	6,83	36,66	0,18
10	10436,37	10432	10446	2,76	43,33	0,04

Tabela 7.3: Resultados para a instância att48 obtidos com o algoritmo IMP.MMAS.

att48						
Mudança	**Média**	**Mínimo**	**Máximo**	**Tempo médio**	**% de acertos**	**Distância em relação ao ótimo (%)**
1	10628,00	10628	10628	2,63	100,00	0,00
2	9819,00	9819	9819	5,76	100,00	0,00
3	10323,00	10323	10323	2,70	100,00	0,00
4	10222,03	10222	10223	15,00	90,00	0,00
5	10133,00	10133	10133	7,46	100,00	0,00
6	10371,00	10303	10379	9,80	0,00	0,67
7	10356,80	10351	10380	8,53	0,00	0,10
8	10313,17	10289	10357	15,73	0,00	0,29
9	10445,20	10415	10470	10,76	23,33	0,29
10	10442,73	10432	10460	5,33	30,00	0,10

Tabela 7.4: Resultados para a instância eil76 obtidos com o algoritmo copt-aiNet.

eil76						
Mudança	**Média**	**Mínimo**	**Máximo**	**Tempo médio**	**% de acertos**	**Distância em relação ao ótimo (%)**
1	538,10	538	539	30,83	90,00	0,019
2	538,10	536	542	24,50	16,66	0,39
3	531,47	531	534	28,13	73,33	0,09
4	534,80	534	535	22,87	20,00	0,15
5	536,27	536	538	39,73	80,00	0,05
6	531,43	531	534	18,30	80,00	0,08
7	547,43	547	548	21,20	56,66	0,08
8	549,43	549	551	17,07	66,66	0,08
9	553,63	552	557	30,17	3,33	0,29
10	545,83	544	549	25,07	3,33	0,34

Tabela 7.5: Resultados para a instância eil76 obtidos com o algoritmo IMP.MMAS.

eil76						
Mudança	**Média**	**Mínimo**	**Máximo**	**Tempo médio**	**% de acertos**	**Distância em relação ao ótimo (%)**
1	538,03	538	539	29,07	96,66	0,0062
2	541,53	537	544	49,70	0,00	1,0323
3	540,43	537	544	57,77	0,00	1,7765
4	542,67	539	547	63,70	0,00	1,623
5	546,10	541	558	66,90	0,00	1,8843
6	541,70	537	547	54,13	0,00	2,0151
7	552,43	550	558	52,33	0,00	0,9933
8	561,27	555	569	63,10	0,00	2,2344
9	563,17	557	571	62,63	0,00	2,0229
10	558,80	551	569	55,40	0,00	2,7206

Tabela 7.6: Resultados para a instância kroC100 obtidos com o algoritmo copt-aiNet.

kroC100						
Mudança	**Média**	**Mínimo**	**Máximo**	**Tempo médio**	**% de acertos**	**Distância em relação ao ótimo (%)**
1	20749,00	20749	20749	49,37	100,00	0,00
2	20337,97	20335	20424	65,30	0,00	0,10
3	20373,93	20201	20538	42,53	40,00	0,86
4	20220,03	20090	20367	35,83	3,33	0,65
5	20083,47	20010	20215	17,37	23,33	0,37
6	20073,43	19852	20276	22,33	3,33	1,11
7	20028,63	19800	20191	35,33	3,33	1,15
8	19961,63	19914	20063	63,93	0,00	0,46
9	19869,63	19807	19930	50,87	0,00	0,79
10	20344,13	20172	20488	51,87	3,33	0,85

Tabela 7.7: Resultados para a instância kroC100 obtidos com o algoritmo IMP.MMAS.

kroC100						
Mudança	**Média**	**Mínimo**	**Máximo**	**Tempo médio**	**% de acertos**	**Distância em relação ao ótimo (%)**
1	20749,00	20749	20749	60,30	100	0,00
2	20420,40	20335	20621	16,80	0	0,50
3	20630,17	20385	21033	28,53	0	2,12
4	20501,87	20157	20709	7,03	0	2,05
5	20960,47	20658	21377	20,50	0	4,75
6	21206,03	20824	21977	18,30	0	6,82
7	21761,97	21122	22586	5,63	0	9,91
8	22239,47	21504	23270	6,47	0	11,92
9	22234,97	21565	23019	15,30	0	12,79
10	22486,97	22130	23076	4,70	0	11,48

Tabela 7.8: Resultados para a instância ch150 obtidos com o algoritmo copt-aiNet.

ch150						
Mudança	**Média**	**Mínimo**	**Máximo**	**Tempo médio**	**% de acertos**	**Distância em relação ao ótimo (%)**
1	6543,00	6528	6559	341,32	26,60	0,23
2	6474,68	6463	6490	242,78	23,30	0,18
3	6494,14	6463	6535	325,21	6,66	0,48
4	6459,36	6435	6495	313,67	0,00	0,53
5	6463,11	6400	6524	266,28	0,00	1,41
6	6447,61	6371	6530	263,07	0,00	1,54
7	6478,89	6427	6558	202,21	0,00	1,58
8	6532,68	6473	6563	223,50	0,00	1,74
9	6562,79	6505	6606	188,57	0,00	1,81
10	6606,96	6539	6661	213,64	0,00	1,74

Tabela 7.9: Resultados para a instância ch150 obtidos com o algoritmo IMP.MMAS.

ch150						
Mudança	**Média**	**Mínimo**	**Máximo**	**Tempo médio**	**% de acertos**	**Distância em relação ao ótimo (%)**
1	6551,75	6528	6570	309,17	6,66	0,36
2	6587,00	6484	6830	431,28	0,00	1,92
3	6758,36	6592	6993	416,28	0,00	4,57
4	6961,93	6776	7160	378,88	0,00	8,36
5	7059,44	6835	7398	401,44	0,00	10,77
6	7089,37	6797	7355	423,77	0,00	11,64
7	7151,33	6947	7541	390,55	0,00	12,12
8	7128,70	6997	7384	428,33	0,00	11,02
9	7218,93	7019	7432	397,25	0,00	11,99
10	7377,18	7090	7788	374,51	0,00	13,60

Como pode ser observado nas tabelas acima, o algoritmo copt-aiNet apresenta um desempenho superior ao algoritmo IMP.MMAS em todos os itens utilizados na comparação (com exceção ao tempo na instância kroC100, onde o IMP.MMAS obteve resultados mais rápidos, mas, em contrapartida, em média, mais distantes do ótimo). É importante salientar que, embora a porcentagem de sucesso na localização do ótimo global nas duas últimas instâncias tenha sido baixo, o desvio da média da função-objetivo em relação ao ótimo obtido pela copt-aiNet é muito baixo para todas as instâncias. Isso significa que, embora o algoritmo não tivesse atingido o ótimo global, a solução está muito próxima de conseguir e continua sendo uma solução excelente, dado o pouco tempo de processamento. Embora o desempenho do ACO também possa ser considerado satisfatório, ele sofre o efeito de sua memória de longo prazo piorando sua resposta final a cada nova mudança do ambiente.

O sucesso da copt-aiNet se dá justamente por causa da manutenção de diversidade e controle da população, que permitem manter soluções igualmente boas e otimizá-las em paralelo. Ou seja, a cada mudança do ambiente, a copt-aiNet não fica presa a uma única solução ótima e isso permite que ela busque novos caminhos mais facilmente. Ao contrário do ACO, que sofre o efeito da convergência prematura devido à trilha de feromônio.

7.2 Síntese do Capítulo

Neste capítulo, a metodologia utilizada para medir o desempenho dos algoritmos em ambientes dinâmicos foi introduzida e, em seguida, os experimentos foram executados. O problema utilizado para isso foi o caixeiro viajante. O sistema utilizado para efetuar a mudança no ambiente do PCV foi uma troca de peso das arestas ótimas utilizando um algoritmo *k*-opt.

Ao final dos experimentos, foi verificado um desempenho superior da copt-aiNet em relação ao ACO por causa de suas características, como ajuste automático do tamanho da população e manutenção de diversidade. Essas características permitem à copt-aiNet reagir mais rapidamente quando exposta a um novo ambiente, ao mesmo tempo em que aproveita as informações do ambiente anterior.

Capítulo 8

Conclusões e Trabalhos Futuros

Este trabalho teve como objetivo estudar algoritmos de otimização bio-inspirados e estendê-los para operar em ambientes variantes no tempo, abrangendo tanto problemas contínuos como discretos.

Nos capítulos iniciais, foram apresentadas técnicas de otimização amplamente utilizadas, onde assuntos como métodos exatos, buscas locais, métodos heurísticos e meta-heurísticas foram abordados, para que o entendimento dos capítulos remanescentes fosse facilitado. Em seguida, o problema de otimização contínua (especificamente não-linear) foi brevemente revisado usando o algoritmo conhecido como "opt-aiNet", originalmente proposto para operar em ambientes estáticos. Uma vez que o desempenho deste algoritmo não foi satisfatório em determinadas situações, ele foi estendido e denominado dopt-aiNet, particularmente projetado para obter bons desempenhos em ambientes dinâmicos.

O algoritmo dopt-aiNet evidenciou características dos algoritmos inspirados nos sistemas imunológicos como, manutenção de diversidade e controle dinâmico do tamanho da população. Boa parte deles é capaz de espalhar vários indivíduos pelo espaço de busca e encontrar múltiplas soluções diferentes. Outra característica marcante, também evidenciada, é o controle dinâmico do tamanho da população, que ajuda a identificar os principais ótimos locais da função otimizada e reduzir o custo computacional da ferramenta.

Os resultados em ambientes dinâmicos foram comparados com mais dois algoritmos bio-inspirados: o PSO e o BCA. A dopt-aiNet foi a abordagem que obteve melhor desempenho entre todos os três algoritmos, com uma resposta muito mais rápida a mudanças no ambiente.

Na segunda parte da dissertação, foram estudados problemas discretos utilizando para isso duas ferramentas também inspiradas na natureza: i) o ACO, inspirado no comportamento de colônias de formigas; e ii) a copt-aiNet, uma extensão da opt-aiNet para problemas combinatórios. Cada um desses algoritmos foi estudado em separado e aperfeiçoado para obter um melhor desempenho para as classes de problemas consideradas. Na seqüência, os algoritmos foram comparados em uma situação dinâmica do problema do caixeiro viajante e, novamente, o algoritmo imuno-inspirado apresentou os melhores resultados.

Diante dos resultados apresentados, é possível concluir que os algoritmos imuno-inspirados têm uma boa adaptabilidade a ambientes dinâmicos. Isso se deve a suas características essenciais, como geração e manutenção de diversidade, memória de soluções promissoras passadas, e natureza populacional da busca, sendo que cada solução candidata

(anticorpo) se adapta de forma independente das demais e o tamanho da população é adaptável às peculiaridades de cada aplicação.

Para trabalhos futuros, alguns aspectos do algoritmo dopt-aiNet podem ser melhorados, tanto em desempenho computacional, como em qualidade de resultados. Alguns destes pontos são: uma mutação única que possua um maior número de direções possíveis, e que contemple as mesmas direções das duas mutações já utilizadas, e a definição de algum critério para selecionar quais células serão comparadas entre si no algoritmo de supressão, reduzindo o seu custo computacional total.

No algoritmo copt-aiNet, dois pontos que merecem um estudo à parte são: o critério de supressão de duas soluções, ou seja, o quão parecidas duas soluções devem ser para uma delas ser eliminada, e em quais situações a busca local 3-opt deve ser executada de forma a reduzir o impacto na diversidade de soluções.

Um outro estudo poderia levar para a união dos algoritmos IMP.MMAS e copt-aiNet, reunindo duas características importantes como o mecanismo de memória bem elaborado do IMP.MMAS e a manutenção de diversidade da copt-aiNet.

Trabalhos Publicados

- IMP.MMAS:

de França, F. O., Von Zuben, F. J., & de Castro, L. N. (2004a). *Definition of Capacited p-Medians by a Modified Max Min Ant System with Local Search*. Proceedings of ICONIP - 2004 11th International Conference on Neural Information Processing - SPECIAL SESSION ON ANT COLONY AND MULTI-AGENT SYSTEMS, Calcutta.

de França, F. O., Von Zuben, F. J., & de Castro, L. N. (2004b). *A Max Min Ant System Applied To The Capacitated Clustering Problem*. Proceedings of 2004 IEEE Workshop on Machine Learning for Signal Processing, São Luiz, Brasil.

de França, F. O., Von Zuben, F. J., & de Castro, L. N. (2005b). Max Min Ant System and Capacitated p-Medians: Extensions and Improved Solutions. *Informatica, 29*(2), 163-171.

- dopt-aiNet:

de França, F. O., Von Zuben, F. J., & de Castro, L. N. (2005a). *An Artificial Immune Network for Multimodal Function Optimization on Dynamic Environments*. Proceedings of the 2005 conference on Genetic and Evolutionary Computation Conference (GECCO'05), Washington DC, USA.

Junqueira, C., de França, F. O., Attux, R. R. F., Suyama, R., de Castro, L. N., Von Zuben, F. J., & Romano, J. M. T. (2005). *A Proposal for Blind FIR Equalization of Time-Varying Channels*. Proceedings of the IEEE International Workshop on Machine Learning for Signal Processing

Referências

Angeline, P. J. (1997). *Tracking Extrema in Dynamic Environments*. Proceedings of the 6th International Conference on Evolutionary Programming, Indianapolis, Indiana, USA.

Bazaraa, M. S., Jarvis, J. J., & Sherali, H. D. (1990). *Linear programming and network flows* (2nd ed.). New York: Wiley.

Bazaraa, M. S., Sherali, H. D., & Shetty, C. M. (1993). *Nonlinear programming : theory and algorithms* (2nd ed.). New York: Wiley.

Bazaraa, M. S., & Shetty, C. M. (1976). *Foundations of optimization*. Berlin ; New York: Springer-Verlag.

Beasley, D. (2001, 03/08/2005). AI FAQ/genetic. Retrieved from http://www.faqs.org/faqs/ai-faq/genetic/

Bersini, H. (2002). *Self-Assertion versus Self-Recognition: A Tribute to Francisco Varela*. Proceedings of the Int. Conf. on Artificial Immune Systems

Besendovsky, H. O., & del Rey, A. (1996). Immune-Neuro-Endocrine Interactions: Facts and Hypotheses. *Endocrine Reviews, 17*(1), 64-102.

Blum, C., & Dorigo, M. (2004). The Hyper-Cube Framework for Ant Colony Optimization. *IEEE Transactions on Systems, Man and Cybernetics, Part B, 34*(2), 1161-1172.

Blum, C., Roli, A., & Dorigo, M. (2001, 2001). *HC-ACO: The hyper-cube framework for Ant Colony Optimization*. Proceedings of MIC'2001 - Meta-heuristics International Conference, Porto, Portugal.

Bonabeau, E., Dorigo, M., & Theraulaz, G. (1999). *Swarm Intelligence: From Natural to Artificial Systems*. New York, NY: Oxford University Press,.

Bretscher, P., & Cohn, M. (1970). A theory of self-nonself discrimination. *Science, 169*, 1042-1049.

Burnet, F. M. (1955). A modification of Jerne's theory of antibody production using the concept of clonal selection. *Aust J. Sci, 20*, 67-77.

Carlisle, A. J. (2002). *Applying the Particle Swarm Optimizer to Non-Stationary Environments*, Auburn University.

Churchman, C. W., Ackoff, R. L., & Arnoff, E. L. (1957). *Introduction to Operations Research*. New York: J. Wiley and Sons.

Dasgupta, D. (1998). *Artificial immune systems and their applications*. Berlin ; New York: Springer.

Davidon, W. (1959). *Variable metric method for minimization*. Argonne: ANL-5990.

de Castro, L. N. (2001). *Engenharia Imunológica: Desenvolvimento e Aplicação de Ferramentas Computacionais Inspiradas em Sistemas Imunológicos Artificiais*, UNICAMP, Campinas).

de Castro, L. N. (2003). Immune Cognition, Micro-evolution, and a Personal Account on Immune Engineering. *S.E.E.D. Journal (Semiotics, Evolution, Energy, and Development), 3*(3), 134-155.

de Castro, L. N., & Timmis, J. (2002a). *An Artificial Immune Network for Multimodal Function Optimization*. Proceedings of the 2002 Congress on Evolutionary Computation CEC2002

de Castro, L. N., & Timmis, J. (Eds.). (2002b). *Artificial immune systems: a new computational intelligence approach* (Vol. 16). London: Springer.

de Castro, L. N., & Von Zuben, F. J. (2000). *An Evolutionary Immune Network for Data Clustering*. Proceedings of IEEE SBRN

de Castro, L. N., & Von Zuben, F. J. (2001a). aiNet: An Artificial Immune Network for Data Analysis. In H. A. Abbas, R. A. Sarker, & C. S. Newton (Eds.), *Data Mining: A Heuristic Approach* (1 ed., pp. 231-259): Idea Group Publishing.

de Castro, L. N., & Von Zuben, F. J. (2001b). Automatic Determination of Radial Basis Function: An Immunity-Based Approach. *International Journal of Neural Systems (IJNS), Special Issue on Non-Gradient Learning Techniques*.

de Castro, L. N., & Von Zuben, F. J. (2002). Learning and Optimization Using the Clonal Selection Principle. *IEEE Transactions on Evolutionary Computation, Special Issue on Artificial Immune Systems, 6*(3), 239-251.

de Castro, L. N., & Von Zuben, F. J. (2005). *Recent Developments in Biologically Inspired Computing*: Idea Group Inc.

de França, F. O., Von Zuben, F. J., & de Castro, L. N. (2004a). *Definition of Capacited p-Medians by a Modified Max Min Ant System with Local Search*. Proceedings of ICONIP - 2004 11th International Conference on Neural Information Processing - SPECIAL SESSION ON ANT COLONY AND MULTI-AGENT SYSTEMS, Calcutta.

de França, F. O., Von Zuben, F. J., & de Castro, L. N. (2004b). *A Max Min Ant System Applied To The Capacitated Clustering Problem*. Proceedings of 2004 IEEE Workshop on Machine Learning for Signal Processing, São Luiz, Brasil.

de França, F. O., Von Zuben, F. J., & de Castro, L. N. (2005a). *An Artificial Immune Network for Multimodal Function Optimization on Dynamic Environments*. Proceedings of the 2005 conference on Genetic and Evolutionary Computation Conference (GECCO'05), Washington DC, USA.

de França, F. O., Von Zuben, F. J., & de Castro, L. N. (2005b). Max Min Ant System and Capacitated p-Medians: Extensions and Improved Solutions. *Informatica, 29*(2), 163-171.

de Sousa, J. S., Gomes, L. d. C. T., Bezerra, G. B., de Castro, L. N., & Von Zuben, F. J. (2003). An Immune-Evolutionary Algorithm for Multiple Rearrangements of Gene Expression Data. *Genetic Programming and Evolvable Machines, 5*(2), 157-179.

Dorigo, M. (1992). *Optimization, Learning and Natural Algorithms*, Politecnico di Milano, Italy).

Eberhart, R. C., & Kennedy, J. (1995). *A new optimizer using particle swarm theory.* Proceedings of the Sixth International Symposium on Micromachine and Human Science, Nagoya, Japan.

Everitt, B. S., Landau, S., & Leese, M. (2001). *Cluster Analysis* (4th edition ed.). London: Hodder Arnold.

Farina, M., Deb, K., & Amato, P. (2004). Dynamic multiobjective optimization problems: test cases, approximations, and applications. *IEEE Transactions on Evolutionary Computation, 8*, 425-442.

Fogel, L. J., Owens, A. J., & Walsh, M. J. (1966). Artificial Intelligence through Simulated Evolution.

Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability : a guide to the theory of NP-completeness*. San Francisco: W. H. Freeman.

Glover, F. (1986). Future paths for Integer Programming and Links to Artificial Intelligence. *Computers and Operations Research, 5*, 533-549.

Glover, F., & Kochenberger, G. A. (2002). *Handbook of Metaheuristics*: Kluwer Academic Publishers.

Goss, S., Aron, S., & J. L. Deneubourg. (1989). Self-organized shortcuts in the Argentine ant. *Naturwissenschaften, 76*, 579-581.

Herdy, M. (1990). *Application of the Evolution Strategy to Discrete Optimization Problems*. Proceedings of the First International Conference on Parallel Problem Solving from Nature (PPSN), Lecture Notes in Computer Science

Holland, J. H. (1992). *Adaptation in natural and artificial systems : an introductory analysis with applications to biology, control, and artificial intelligence* (1st MIT Press ed.). Cambridge, Mass.: MIT Press.

Holland, P. W. H., Garcia-Fernandez, J., Williams, N. A., & Sidow, A. (1994). Gene duplications and origins of verte-brate development. . *Development Supplement*, 125-133.

Jin, Y., & Branke, J. (2005). Evolutionary Optimization in Uncertain Environments—A Survey. *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, 9*(3).

Junqueira, C., de França, F. O., Attux, R. R. F., Suyama, R., de Castro, L. N., Von Zuben, F. J., & Romano, J. M. T. (2005). *A Proposal for Blind FIR Equalization of Time-Varying Channels*. Proceedings of the IEEE International Workshop on Machine Learning for Signal Processing

Kelsey, J., & Timmis, J. (2003). *Immune Inspired Somatic Contiguous Hypermutation for Function Optimisation*. Proceedings of Genetic and Evolutionary Computation Conference (GECCO'03), Chicago, USA.

Kennedy, J., & Eberhart, R. C. (1995). *Particle swarm optimization.* Proceedings of IEEE International Conference on Neural Networks, Piscataway, NJ.

Koza, J. R. (1992). Genetic Programming: On the Programming of Computers by Means of Natural Selection. 840.

Lawer, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G., & Shmoys, D. B. (1985). *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. Wiley-Interscience.

Lin, S., & Kernighan, B. W. (1973). An Effective Heuristic Algorithm for the Travelling-Salesman Problem. *Operations Research, 21*, 498-516.

Luenberger, D. G. (1984). *Linear and Nonlinear Programming*. Reading, MA: Addison Wesley.

Lumer, E. D., & Faieta, B. (1994). Diversity and Adaptation in Populations of Clustering Ants. In D. In Cliff, P. Husbands, J. Meyer, & W. S. (Eds.), *From Animals to Animats 3*. Cambridge, MA: The MIT Press/Bradford Books.

Matzinger, P. (1994). Tolerance, danger, and the extended family. *Annual Review of Immunology, 12*, 991-1045.

McCulloch, W. S., & Pitts, W. H. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics, 5*, 115-133.

Michalewicz, Z., & Fogel, D. B. (2000). *How to solve it : modern heuristics*. Berlin ; New York: Springer.

Ohno, S. (1970). Evolution by Gene Duplication. *Allen and Unwin*.

Orman, A. J., & Williams, H. P. (2004). *A Survey of Different Integer Programming Formulations of the Travelling Salesman Problem*. London: The London School of Economics and Political Science,.

Paton, R. (Ed.). (1994). *Computing with Biological Metaphors*: Chapman & Hall.

Rechenberg, I. (1965). Cybernetic Solution Path of an Experimental Problem. *Royal Aircraft Establishment Translation, B. F. Toms, Trans.*(1122).

Reynolds, C. W. (1987). *Flocks, herds, and schools: A distributed behavioral model.* Proceedings of Computer Graphics (SIGGRAPH '87 Proceedings)

Rosenblatt, F. (1957). *The Perceptron - a Perceiving and Recognizing Automaton* (85-460-1). Ithica: Cornell Aeronautical Laboratory.

Rui, X., & Wunsch, D., II. (2005). Survey of clustering algorithms. *Neural Networks, IEEE Transactions on, 16*(3), 645-678.

Schwefel, H. P. (1965). *Kybernetische Evolution als Strategie der Experimentellen*, Technical University of Berlim.

Stützle, T., & Dorigo, M. (1999). ACO algorithms for the Quadratic Assignment Problem. In M. D. D. Corne, and F. Glover (Ed.), *New Ideas in Optimization* (pp. 33-50): McGraw-Hill.

Stützle, T., & Hoos, H. H. (1997). *The MAX-MIN ant system and local search for the traveling salesman problem*. Proceedings of the IEEE International Conference on Evolutionary Computation (ICEC'97), Piscataway, NJ, USA.

Zahn, C. T. (1971). Graph-Theoretical Methods for Detecting and Describing Gestalt Clusters. *IEEE Transactions on Computers, C-20*(1), 68-86.