*MINISTÉRIO DA CIÊNCIA E TECNOLOGIA*
***INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS***
*CURSO DE COMPUTAÇÃO APLICADA*

# Treinamento Populacional em Heurísticas

## Aplicações em Otimização

*Alexandre César Muniz de Oliveira*

São José dos Campos - SP

# Treinamento Populacional em Heurísticas: Aplicações em Otimização

Alexandre César Muniz de Oliveira

Proposta de Tese apresentada como parte dos requisitos para a obtenção do título de Doutor em Computação Aplicada no Instituto Nacional de Pesquisas Espaciais

Orientador: Prof. Dr. Luiz Antônio de Nogueira Lorena

São José dos Campos/SP
Maio/2002

# Sumário

# Lista de Figuras

# Lista de Tabelas

Capítulo

1

# Introdução

## 1.1 Conceitos em Otimização

Otimização é a busca da melhor solução, que minimiza custos ou maximiza ganhos, para um dado problema dentro de um conjunto finito ou infinito de possíveis soluções. O processo de busca pode partir de uma solução inicial ou de um conjunto delas, realizando melhoramentos progressivos até chegar a um outro conjunto que contenha uma ou todas as melhores soluções possíveis dentro do conjunto de possíveis soluções.

Na otimização combinatória, o universo de possíveis combinações (espaço de busca) é enumerável e finito. Por outro lado, a otimização numérica opera com elementos de um universo infinito de valores e combinações, uma vez que os parâmetros que se pretende otimizar podem assumir quaisquer valores numéricos (reais ou inteiros).

A solução para problemas de otimização, em geral, é obtida a partir de uma configuração inicial $I_0$ que contém uma solução $x_0$ (ou um conjunto $P_0$ delas) e controles $A_0$ específicos do algoritmo a ser empregado. Seguidas iterações são necessárias para se melhorar a qualidade dessa solução (ou soluções) até que se chegue a uma condição de término que pode ser a obtenção da solução ótima.

Um algoritmo que possui a capacidade de alcançar uma solução ótima a partir de um ponto qualquer do espaço de busca é considerado um algoritmo global. Por sua vez, algoritmos locais estão mais dependentes de configurações iniciais ou pontos de partida, uma vez que tendem a seguir superfícies de funções, ou vizinhanças de pontos e, portando, podem atingir pontos estacionários a partir dos quais não conseguem mais melhorar a solução. Portanto, algoritmos locais são menos robustos que algoritmos globais.

## 1.2 Algoritmos Evolutivos

Os mecanismos naturais que promovem a evolução dos seres vivos podem ser considerados processos inteligentes. A auto-organização, competição e cooperação são exemplos de estratégias bastante usadas em otimizadores e que são fruto da observação humana de como tais mecanismos funcionam na natureza.

Os algoritmos evolutivos são classes de otimizadores globais cujas estratégias de busca estão centradas na evolução natural e na genética. O comportamento comum desses algoritmos é o de evoluir uma população de indivíduos, que representam soluções candidatas de um determinado problema, ao longo de gerações, competindo entre si para sobreviverem e se reproduzirem. As estratégias evolutivas [1], a programação evolutiva [2], a programação genética [3], e os algoritmos genéticos [4] são classes de otimizadores evolutivos.

Os algoritmos evolutivos possuem características que os tornam aplicáveis a uma vasta quantidade de aplicações. Algumas características desejáveis são:

a) são métodos de otimização global, robustos e podem encontrar várias soluções;

b) podem otimizar um grande número de parâmetros discretos, contínuos ou combinações deles;

c) realizam buscas simultâneas em várias regiões do espaço de busca (paralelismo inerente);

d) utilizam informações de custo ou ganho e não necessitam obrigatoriamente de conhecimento matemático do problema (tais como derivadas);

e) podem ser eficientemente combinados com heurísticas e outras técnicas de busca local;

f) são modulares e facilmente adaptáveis a qualquer tipo de problema.

Como trabalham com população de soluções, podem ser mais lentos que outras alternativas e, mesmo as versões tradicionais mais simples, possuem alguns parâmetros que devem ser bem ajustados para obter eficácia.

No desenvolvimento de algoritmos eficientes para problemas de otimização, cada vez mais pesquisadores têm feito uso de informação especifica do problema para melhorar ou até viabilizar o desempenho dos seus algoritmos evolutivos. Os enfoques híbridos muitas vezes usam a estrutura geral evolutiva acrescida de otimizadores locais e/ou heurísticas encapsulados em operadores evolutivos unários.

É consenso entre muitos que tais otimizadores locais levam consigo o ônus de sua utilização, que é visível no aumento do custo computacional, e que torna necessário a formulação de estratégias que permitam uma *hibridização* eficaz.

Este trabalho propõe um nova forma de avaliação de indivíduos, usando informação sobre o problema em questão, obtida através de otimizadores locais e heurísticas específicas, possibilitando uma eficiente exploração do espaço de busca. A nova forma de avaliação expressa o quão treinados os indivíduos estão com relação a tais otimizadores locais e heurísticas específicas.

Por este enfoque, os indivíduos bem-treinados (ou bem-adaptados ao treinamento) são mais selecionados para as operações de cruzamento e mutação que irão promover a exploração do espaço de busca. Ao mesmo tempo, em que uma população dinâmica, controlada por um limiar de rejeição adaptativo, permite que esses indivíduos participem do processo evolutivo o máximo de tempo possível.

Algumas aplicações que utilizam o enfoque do treinamento populacional foram desenvolvidas para alguns tipos problemas de otimização e os resultados são promissores. Em vista disso, este trabalho propõe a sua extensão a outras áreas, assim como novos experimentos nas aplicações já desenvolvidas.

## 1.3 Objetivos e Organização do trabalho

Algoritmos de otimização eficientes têm conquistado mais e mais fóruns de discussão, devido às sempre crescentes áreas de aplicação em engenharia, física, genética, etc. O que seduz os pesquisadores dessa área é a possibilidade de construção de ferramentas robustas, velozes e aplicáveis a problemas nas mais variadas áreas do conhecimento. Embora essas qualidades, em geral, não sejam atributos presentes em uma mesma ferramenta, este trabalho propõe atividades que contribuem exatamente com essa linha de pesquisa.

O objetivo desta proposta é apresentar os primeiros resultados do Treinamento Populacional em Heurísticas e as próximas atividades a serem desenvolvidas. Para tanto, primeiramente é feita uma exposição de outros enfoques evolutivos que utilizam otimizadores locais, as estratégias de utilização desses operadores, bem como os problemas aos quais eles se destinam.

Este trabalho está dividido em capítulos, sendo este o primeiro e os subsequentes estão dispostos como se segue. No **capítulo 2**, são apresentados alguns enfoques evolutivos para problemas de otimização numérica que utilizam otimizadores locais. No **capítulo 3**, de forma genérica, são apresentados os alicerces do Treinamento Populacional. No **capítulo 4**, são descritas as primeiras aplicações e seus respectivos resultados. No **capítulo 5** são definidas as próximas atividades e um cronograma de trabalho proposto. Por último, são apresentadas as principais conclusões deste trabalho.

Capítulo

2

# Algoritmos Evolutivos Aplicados em Otimização Numérica

Neste capítulo, são apresentados alguns dos recentes enfoques evolutivos para problemas de otimização numérica, também chamada de otimização de parâmetros de função. Nesse contexto, pode-se incluir aplicações em redes neurais, problemas inversos, controle *fuzzy*, e outros onde o objetivo é encontrar valores para um conjunto de parâmetros, definidos em $\mathbb{R}$, que otimizam uma dada função.

São também apresentados alguns dos otimizadores locais de uso geral, bem como outros específicos e as estratégias utilização comumente empregadas na composição de algoritmos evolutivos híbridos.

## 2.1 Conceitos em Otimização Numérica

Um problema de otimização numérica pode ser formulado como:

$$\text{Otimize } \{f(x)\}, x = (x_1, x_2, x_3, ., x_m)^T \in \mathbb{R}^m \tag{2.1}$$

Sujeito ou não a restrições de igualdade na forma

$$c_j(x) = 0 \tag{2.2}$$

e/ou restrições de desigualdade na forma

$$c_k(x) \geq 0 \tag{2.3}$$

Problemas sujeitos a condições de restrição, $\mathbf{c(x)}$, do espaço de busca, são chamados de restritos (com restrições). Nos chamados problemas irrestritos, costuma-se delimitar o espaço de busca $\beta$ através de pares $L_i$ e $U_i$.

$$\text{Otimize } \{f(x)\}, x = (x_1, x_2, x_3, ., x_m)^T \in \mathbb{R}^m \text{ onde } L_i < x_i < U_i \tag{2.4}$$

Tanto a função objetivo, quanto as condições de restrição, se existirem, podem ser lineares ou não. Uma solução $x^*$ para um problema de minimização é chamada mínimo global quando não existir outra $x$, pertencente a $\beta$, de forma que $f(x) < f(x^*)$. Em problemas de maximização, o máximo global $x^*$ atende a $f(x^*) > f(x)$ para todo $x$ pertencente a $\beta$. Se a solução $x$ possui $f(x)$ mínimo apenas dentro de uma certa região em torno de $x$, chamada de vizinhança de $x$, diz-se que o mínimo é local. Os mínimos locais podem ser boas soluções, mas não são as melhores. Para certos métodos de busca esses pontos são indesejáveis, pois interrompem a busca por soluções melhores.

A função objetivo pode ter um ou muitos pontos de mínimos, o que define se ela é uno ou multimodal. Uma função unimodal apresenta apenas um ponto de mínimo ou máximo. Uma função multimodal, por sua vez, possui várias inflexões de sua superfície, o que caracterizam múltiplos pontos de mínimo ou máximo. Problemas irrestritos possuem mínimos globais ou locais em pontos chamados de estacionários (sem declive), quando o gradiente de $x$ nesse ponto é zero. Em problemas restritos, isso pode não acontecer. Nesse caso, um ponto mínimo ou máximo pode ser uma fronteira gerada por uma restrição. A Figura 2.1 mostra exemplos de pontos de mínimos em uma função multimodal.

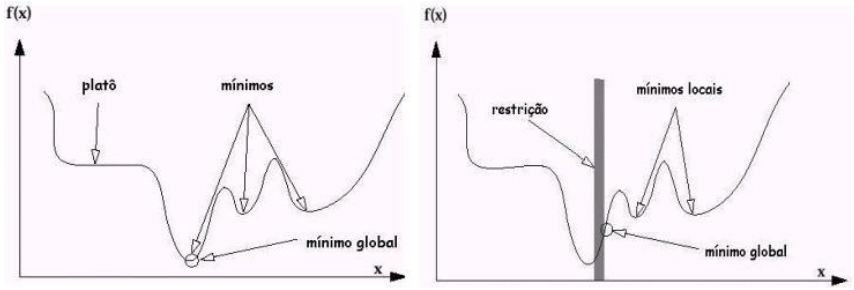


**Figura 2.1:** Espaços de busca multimodais sem restrição e com restrição

## 2.2 Algoritmos Evolutivos Híbridos

Otimizadores locais têm sido incorporados a algoritmos evolutivos, visando ganho de rapidez e precisão, ao mesmo tempo preservando a característica de rosbutez, comum nos algoritmos evolutivos. O desenvolvimento de dessas abordagens híbridas são motivadas

pelos bons resultados que algumas delas têm alcançado. Em geral, um operador evolutivo unário, tal como a mutação, é usado para encapsular um método de busca local.

Um exemplo é o Algoritmo Genético Simplex Híbrido (Simplex Genetic Algorithm Hybrid - SGAH) que utiliza uma arquitetura híbrida baseada em elitismo, onde o operador de busca local é aplicado somente aos melhores indivíduos da população [5].

A busca local usada no SGHA é uma variação probabilística do simplex *Down-Hill* [6], onde os passos de reflexão e contração, respectivamente, $\alpha$ e $\beta \in [0, 1]$, com distribuição triangular de pico em 0.5. A Figura 2.2 ilustra um movimento básico de reflexão do simplex na direção oposta ao pior ponto. O procedimento também admite outros dois movimentos, expansão (se o ponto refletido for promissor) e contração (se o ponto refletido não for promissor).

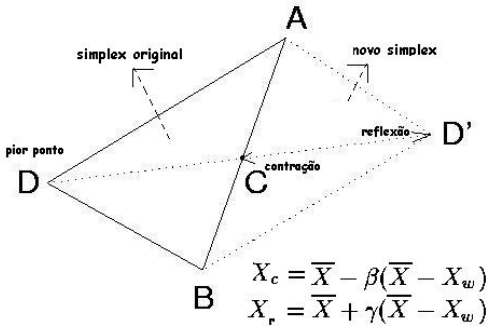


**Figura 2.2:** Simplex Down-Hill

No SGAH, toda a população é submetida aos operadores evolutivos tradicionais, mas apenas uma parcela S dela é submetida a operador de busca local simplex probabilístico. A nova população é composta também pelos $N$ melhores indivíduos. A arquitetura elitista do SGHA é mostrada na Figura 2.3.

A estratégia de se aplicar o otimizador local a apenas uma parcela da população parece ser uma das mais eficazes, pois promove melhoramentos apenas em indivíduos promissores, racionalizando o custo computacional inerente aos otimizadores locais e, ao mesmo tempo, permitindo que essa elite melhorada participe do processo evolutivo, transmitindo o material genético melhorado aos seus descendentes nas futuras recombinações.

Algoritmos evolutivos com busca local podem ter problemas com espaços de busca descontínuos se a busca opera com derivadas. Além de sua simplicidade o simplex Down-Hill é bastante utilizado também por trabalhar apenas com avaliações de função objetivo.

Um outro método de busca local igualmente simples e bem mais eficiente, o gradiente conjugado tem a desvantagem de trabalhar com derivadas da função objetivo, sendo, portanto, vulnerável a possíveis descontinuidades do espaço de busca.

O incremento no desempenho dos algoritmos híbridos, nem sempre justifica o aumento de complexidade causado pelo método de busca local. Quando computados a quantidade

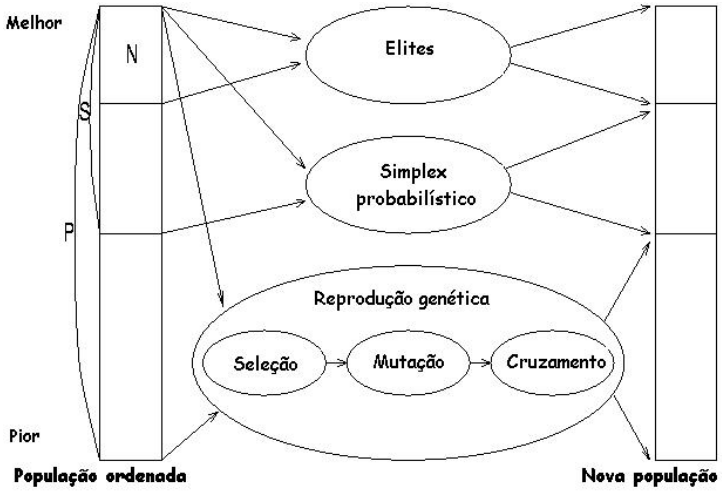


**Figura 2.3:** Arquitetura elitista do SGAH

de avaliações da função objetivo, esse número, dependendo do método local, chega a ser dez vezes maior que com o mesmo algoritmo sem a busca local. Em [7], um otimizador baseado em programação evolutiva é combinado com o método do gradiente conjugado. A conclusão do trabalho é que com dez vezes mais gerações, o algoritmo não-híbrido consegue encontrar soluções tão boas quanto o algoritmo híbrido, mas sem tantas avaliações da função objetivo [7].

Um outro enfoque evolutivo que emprega buscas locais, trabalhando com populações pequenas de indivíduos é o *Scatter Search*[8]. Esse enfoque também possui mecanismo explícito de manutenção de diversidade.

Inicialmente, é gerado um conjunto de vetores solução através de um sistema que garanta um espalhamento mais ou menos uniforme no espaço de busca. Depois, ao longo das gerações, essas soluções são melhoradas através de buscas locais e combinadas linearmente para produzir pontos tanto dentro quanto fora das regiões limitadas por estas soluções.

O *Scatter Search* pode ser detalhado através de 5 métodos:

a) Geração de Diversificação: é gerada uma coleção de soluções dentro do espaço de busca, seguindo uma distribuição uniforme entre os limites $L_i$ e $U_i$ de cada variável de controle $x_i$. São definidos contadores de freqüência para evitar que certas regiões do espaço de busca se tornem mais populosas que outras.

b) Melhoramento: buscas locais são usadas com o propósito de melhorar cada solução obtida. Para problemas de otimização numérica tem sido utilizado o já mencionado método simplex *Down-Hill*.

c) Atualização dos Conjuntos de Referência: são mantidos dois subconjuntos pequenos de indivíduos (em torno de 20% da população, cada), chamados de conjuntos de referência

$RefSet1$ e $RefSet2$. No primeiro são incluídos as soluções com melhor $f(x)$ e no outro são incluídos os que possuam a maior distância euclidiana mínima em relação às soluções de $RefSet1$. Indivíduos em $RefSet2$ possuem boa diversidade $d(x)$, mas não necessariamente se tratam de indivíduos ruins em termos de função objetivo.

d) Geração de Subconjuntos: a partir de $RefSet1$ e $RefSet2$ são gerados subconjuntos que serão combinados no subsequente Método de Combinação. Os indivíduos são tomados 2 a 2, depois 3 a 3, e assim por diante até o tamanho do conjunto de referência em questão ($RefSet1$ ou $RefSet2$).

e) Combinação de Soluções: a partir dos subconjuntos formados pelo procedimento anterior, pares de indivíduos são combinados para gerar novos indivíduos. O método de combinação é similar a um cruzamento aritmético [9] e é aplicado várias vezes gerando vários descendentes por combinação. Em [10], são apresentados basicamente 3 modos de combinação ($C1$, $C2$ e $C3$) e regras ($R1$, $R2$ e $R3$) para aplicação desses modos dependendo dos conjuntos aos quais pertençam as soluções pais.

C1:

$$x = \frac{x_{'} - r(x_{''} - x_{'})}{2} \tag{2.5}$$

C2:

$$x = \frac{x_{'} + r(x_{''} - x_{'})}{2} \tag{2.6}$$

C3:

$$x = \frac{x_{''} + r(x_{''} - x_{'})}{2} \tag{2.7}$$

onde $r$ é um número aleatório no intervalo $[0, 1]$.

R1: Se $x'$ e $x$" $\in RefSet1$, aplicar uma vez $C1$, $C3$ e duas vezes $C2$, gerando 4 soluções;

R2: Se $x'$ ou $x$" $\in RefSet1$ aplicar uma vez apenas $C1$, $C2$, $C3$, gerando 3 soluções;

R3: Se $x'$ e $x$" $\in RefSet1$ aplicar $C2$ uma vez e ou $C1$ ou $C3$ uma vez, gerando 2 soluções.

As novas soluções geradas, após melhoramento, são atualizadas em $RefSet1$ ou $RefSet2$ dependendo de duas regras:

U1: Em $RefSet1$, se $f(x)$ é melhor que a pior solução em $RefSet1$;

U2: Em $RefSet2$, se $d(x)$ é melhor que a pior solução em $RefSet2$;

O *Scatter Search* usa uma busca local sempre que um novo indivíduo é gerado pelo método de combinação. Essa estratégia não-elitista sobrecarrega o processo em termos de chamada à função objetivo. Um trabalho recente, onde o *Scatter Search* é usado para treinamento de redes Perceptron de Multi-Camadas, relata um custo computacional alto em termos de chamadas à função objetivo. Nesse mesmo trabalho, os autores sugerem o uso de uma busca local *quasi-Newton* aplicada aos 10 melhores indivíduos somente [11].

## 2.3 Conceitos em Problemas Inversos

Um problema posto de forma direta consiste em determinar-se os efeitos de uma ou diversas causas conhecidas. Esta é a forma mais comum de apresentação de problemas nas diversas áreas da ciência. Em alguns casos, equações matemáticas modelam um processo ou fenômeno físico presumidamente conhecido, e bem delimitado. A solução para essas equações, sujeitas a um conjunto de restrições e condições (dados) iniciais, constitui-se igualmente na solução para o problema direto.

Um problema indireto ou, mais comumente chamado, problema inverso é aquele em que o enfoque é o de determinar as causas que originaram um determinado efeito. A motivação para tal estudo vem necessidade de se conhecer estados passados, parâmetros de um sistema físico, ou como influenciar um sistema através de seu estado presente com o objetivo de dirigi-lo para um estado desejado no futuro.

Um problema direto pode ser posto na forma $y = Ax$, onde $x$ são os dados iniciais, $A$ é o operador de transformação, $y$ é a solução procurada, i.e., os efeitos da causa $x$. Um problema inverso, por sua vez, explora as duas outras opções:

a) a partir de um efeito $y$ e de um modelo conhecidos, pretende-se determinar as condições iniciais $x$;

b) a partir dos dados (ou condições) iniciais e finais, $x$ e $y$, pretende-se identificar o modelo $A$, responsável pela transformação.

Os problemas inversos tratados neste trabalho são aqueles relacionados a determinação das condições iniciais $x$ que originaram o estado atual $y$ de um processo físico cujo modelo $A$ é conhecido. Para este caso, a solução do problema inverso é:

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{y}. \tag{2.8}$$

Segundo a definição de Hardamard, um problema bem-posto [12]:

a) deve possuir solução,

b) única, e

c) estável.

Por estabilidade, entende-se que os dados finais devem depender continuamente dos dados iniciais, i.e., pequenas variações na entrada causam pequenas variações na saída. Por outro lado, para um problema ser considerado mal-posto, basta contrariar um desses critérios.

Os problemas inversos geralmente não dependem continuamente dos dados iniciais, ou seja, pequenos ruídos nos dados iniciais causam grandes diferenças na solução. Ruídos são facilmente inseridos, no processo de inversão, tanto por dados medidos experimentalmente, quanto gerados sinteticamente. Ambos sujeitos, portanto, a imprecisões, arredondamentos e truncamentos durante as computações matemáticas.

É comum o desenvolvimento de ferramentas matemáticas genéricas que possam dar suporte aos casos específicos de problemas mal-postos dentro das mais variadas áreas de aplicação. Por essa ótica, a pesquisa em problemas inversos está onipresente em várias áreas da ciência e engenharia.

Existem vários métodos considerados tradicionais para solução de problemas inversos: a inversão direta, mínimos quadrados e métodos de regularização. A inversão direta consiste na obtenção e aplicação da Eq. (2.8), analiticamente. O método dos mínimos quadrados, por sua vez, consiste em minimizar a diferença (ou resíduo).

$$\min\{\|\mathbf{Ax}-\mathbf{y}\|^2\} \tag{2.9}$$

As técnicas relativas à chamada teoria da regularização [13, 14, 15] têm grande importância dentro do contexto dos problemas inversos. A idéia básica envolvendo esta teoria é a de tornar o problema original bem-posto, restringindo o universo de possíveis soluções $x$, através de um estabilizador funcional que impõe restrições sobre a variação dos parâmetros do modelo físico. Tipicamente um problema inversos passa a ser formulado como :

$$\min p(y, A(x)) + \alpha R(x) \tag{2.10}$$

onde $p(y, A(x)$ é usualmente $\|\mathbf{Ax}-\mathbf{y}\|^2$.

Os métodos de regularização diferem entre si pelo termo regularizador que é adicionado à equação dos mínimos quadrados. Algumas das mais comuns são a regularização de Tikhonov [16], regularização de máxima entropia [17, 18], regularização de máxima entropia não-extensiva [19].

Considerando a regularização de Tikhonov de ordem 1, pode-se definir um problema inverso como uma minimização na forma:

$$Minf(x) = \|\mathbf{Ax}-\mathbf{y}\|^2 + \alpha(\mathbf{Lx}^T\mathbf{Lx}) \tag{2.11}$$

onde $\mathbf{L}$ é uma matriz que aproxima um operador de diferenciação:

$$L = \begin{pmatrix} -1 & 1 & 0 & \dots & 0 \\ 0 & -1 & 1 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & -1 & 1 \end{pmatrix} \tag{2.12}$$

As técnicas empregadas em problemas inversos trabalham no âmbito da minimização numérica sujeita a restrições de regularização, área esta, onde cada vez mais os algoritmos evolutivos têm sido satisfatoriamente aplicados, especialmente em espaços de buscas complexos, multimodais e não-convexos. Em tais espaços, os otimizadores locais, comumente usados em problemas inversos, como o método quasi-newton, podem ter fraco desempenho e/ou pouca robustez.

## 2.4 Algoritmos Evolutivos em Problemas Inversos

Algoritmos evolutivos têm sido aplicados para problemas inversos, em geral, como uma alternativa de otimizador global, robusta, voltada a espaços de busca multimodais e complexos [20, 21, 22]. Apesar de trabalharem com valores numéricos reais, na maioria dos casos, a abordagem evolutiva tradicional tem sido abandonada em favor de novas estruturas de indivíduos e de novos operadores que promovam a evolução, mas com melhor desempenho. Pesquisadores da área de problemas inversos tentam introduzir mais informação sobre o problema específico em que estão trabalhando e cada vez mais desenvolvem ferramentas menos gerais. Aparentemente, os algoritmos tradicionais não tiram proveito de algumas das peculiaridades de certos problemas inversos.

Genericamente, pode-se modelar um algoritmo evolutivo para um problema de determinação das condições iniciais, da seguinte forma:

a) Indivíduo: é a solução candidata $\mathbf{x}$;

b) Função de aptidão: $f(x) = \|\mathbf{A}(\mathbf{x}) - \mathbf{y}\|^2 + \alpha R(x)$, onde a matriz $y$ é gerada empírica ou sinteticamente, usando-se o modelo direto pré-construído; $A(x)$ operador de transformação aplicado à solução candidata $x$; $\alpha$ é a constante de regularização; $R(x)$ termo regularizador aplicado à solução candidata $x$;

c) Operadores evolutivos e os parâmetros de controle associados a eles.

Pode-se observar que a minimização do resíduo (2.9) foi acrescida de um termo que penaliza soluções candidatas que estejam fora de um perfil de regularização. O grau de penalização é controlado pelo parâmetro $\alpha$, que pode ser mantido constante ou variar ao longo do tempo, exatamente como sugere os enfoques de penalidades variáveis em problemas de otimização numérica com restrições [23]. Uma vez constante, $\alpha$ vai determinar o quanto a regularização vai influenciar a informação sobre o sistema.

1. Se $\alpha \to 0$: a informação sobre o sistema vai prevalecer, o que pode trazer resultados ruins com dados ruidosos.

2. $\alpha \to \infty$: a regularização vai prevalecer e a consistência com a informação pode ser perdida.

Em [22], é proposto um enfoque alternativo para o problema de inversão magnetotelúrico, que consiste em se obter a condutividade do subsolo a partir da condutividade da superfície. Nesta proposta, os indivíduos são estruturados em matrizes de números reais que representam uma fatia retangular de material de subsolo com diferentes níveis de condutibilidade. Neste trabalho é proposto ainda três novos tipos de operadores evolutivos: um cruzamento uniforme espacial, um operador unário de busca local do tipo *hill-climbing*, e um operador unário de homogeneização.

O cruzamento uniforme espacial implementa o conceito de vizinhanças vertical e horizontal, i.e., explora a vizinhança em duas dimensões. A cada 5 gerações, é aplicado o operador de busca local e o operador de homogeneização sobre a melhor solução encontrada. Este último explora outra peculiaridade do problema: o fato de regiões vizinhas provavelmente possuírem condutividades semelhantes.

Aplicar operadores de busca local apenas no melhor indivíduo, a cada conjunto de gerações, reduz significativamente o custo computacional da *hibridização*, mas, aparentemente, reduz também a possibilidade dos melhoramentos obtidos se disseminarem pela população através das recombinações.

Um outro aspecto dessa estratégia é a possibilidade de, em uma fase inicial, um indivíduo melhorado apresentar aptidão muito superior aos demais e dominar as recombinações, levando a população a uma perda de diversidade indesejada. Fenômenos como esse foram observados em experimentos realizados na atual fase deste trabalho.

## 2.5 Treinamento de Redes Neurais Artificiais

Redes Neurais Artificiais (RNAs) são arquiteturas computacionais compostas de unidades processadoras, que operam com um certo paralelismo, chamadas de neurônios. O modelo básico de um neurônio artificial pode ser descrito por:

$$y = f(\sum_{i=1}^{n} w_i x_i + \vartheta) \tag{2.13}$$

onde $x_i$ são entradas (estímulos elétricos) provenientes de outros neurônios ou do meio exterior; $w_i$ são pesos associados a cada uma das conexões (sinapses) que chegam ao neurônio; $\vartheta$ é um valor limiar de resistência a sinais elétricos; $f$ é uma função de ativação do neurônio; e $y$ é a saída do neurônio.

Semelhantemente aos neurônios biológicos, que emitem um pulso se a atividade elétrica interna superar um limiar, a atividade interna de um neurônio artificial passa por uma função de ativação para produzir em efeito de emissão de um pulso.

Através de diferentes formas de organizar os neurônios artificiais, pode-se gerar diferentes arquiteturas de redes neurais. Por exemplo, pode-se definir redes neurais de uma ou mais camadas de neurônios, com ou sem realimentação, com funções de ativação lineares

ou não. Uma rede multi-camadas é generalizada na Figura 2.4. O campo de estudo da neurocomputação, com suas diversas arquiteturas e aplicações é vasto e, em grande parte, bem sedimentado em termos teóricos [24].

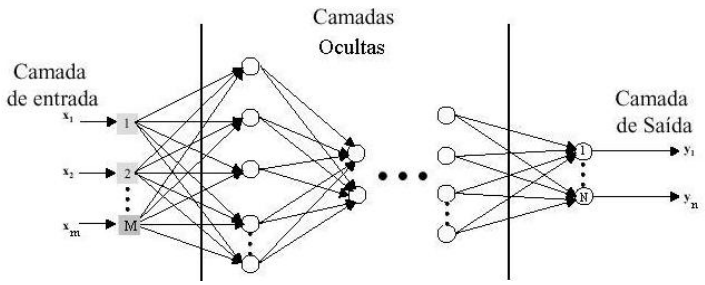


**Figura 2.4:** Rede neural multi-camada generalizada

O treinamento (ou aprendizado) de uma rede neural consiste no ajuste de seus parâmetros livres visando a assimilação de um comportamento desejado. Pode-se dizer que essa capacidade de "aprender" um determinado ambiente é uma das características marcantes das RNAs. Uma rede neural aprende, basicamente, através de um processo iterativo que ajusta progressivamente seus parâmetros livres, em geral, seu conjunto de pesos sinápticos. Os principais paradigmas de aprendizagem são: supervisionado e não-supervisionado.

No processo de aprendizagem não-supervisionada ou auto-organizada, não existe um conjunto de saídas esperadas para determinadas entradas que possa ser usado por um supervisor para avaliar o aprendizado. A rede se adapta a regularidades estatísticas dos dados de entrada, desenvolvendo a habilidade de criar representações internas para codificar características da entrada e, assim, gerar novas classes automaticamente. As chamadas redes ART (*Adaptative Resonance Theory*) [25] e de Kohonen [26] são exemplos que usam treinamento não-supervisionado.

A aprendizagem supervisionada é um paradigma de aprendizagem, no qual um supervisor possui conhecimento sobre o ambiente a ser assimilado e disponibiliza uma parte desse conhecimento à rede neural na forma de um conjunto de treinamento composto de amostras de entrada-saída. Entretanto, o ambiente completo é desconhecido para a rede neural. A diferença entre a resposta da rede e a resposta esperada produz um erro que é utilizado como referência para o ajuste dos parâmetros da rede. Espera-se que, pela representatividade do conjunto de treinamento, ao final do processo, a rede responda satisfatoriamente para elementos dentro e fora desse conjunto. As redes RBF (*Radial Basis Function*) [27] e Perceptron [28] são exemplos que usam o treinamento supervisionado.

Um algoritmo bastante utilizado em treinamentos supervisionados é o de Retropropagação (Backpropagation) [29] que utiliza a regra delta generalizada para ajustar os pesos

de uma rede multi-camada com funções de ativação não-lineares. Inicialmente, o erro $e$, medido no neurônio de saída $y_j$, na época de treinamento $t$, é definido por:

$$e_j(t) = d_j(t) - y_j(t) \tag{2.14}$$

onde $d_j$ é a saída desejada.

Associando-se o erro total da rede (somatório de todos os erros dos neurônios presentes na camada de saída $C_s$ a uma função de energia $E(t)$, tem-se:

$$E(t) = \frac{1}{2}\sum_{j \in C_s} e_j^2(t) \tag{2.15}$$

A medida de desempenho para o sistema é o erro médio quadrado sobre a amostra de treinamento, definido em função dos parâmetros livres da rede (pesos sinápticos). Esta função pode ser visualizada como uma superfície multidimensional de desempenho de erro, com os pesos sinápticos como coordenadas. Qualquer ponto de operação do sistema pode ser representado por um ponto sobre a superfície de erro. O objetivo do treinamento, nesse caso, é mover o ponto de operação para coordenadas (pesos) de menor valor de função energia (erro).

Segundo o algoritmo de Retropropagação, os pesos são ajustados através do vetor passo $\Delta w_{ji}$

$$\Delta w_{ji}(t) = \eta \delta_j(t) y_i(t) \tag{2.16}$$

onde $\eta$ é a taxa de aprendizado que determina o tamanho do passo; $y_i(t)$ é o valor de entrada do neurônio $j$; e $\delta_j(t)$ é o gradiente local que é dependente do tipo de neurônio em questão (camada de saída $C_s$ ou camada oculta $C_o$).

$$\delta_j(t) = \begin{cases} e_j(t)\varphi'_j(\sum_{i \in M} w_{ji}(t) y_i(t)) & \text{se } j \in C_s; \\ e_j(t)\varphi'_j(\sum_{i \in M} w_{ji}(t) y_i(t))(\sum_{k \in K} w_{kj}(t)\delta_k(t)) & \text{se } j \in C_o \end{cases} \tag{2.17}$$

onde $\varphi'_j$ é a derivada da função de ativação; $M$ é o conjunto de neurônios da camada anterior a $j$ e $K$ é o conjunto de neurônios da camada posterior a $j$.

Condensando 2.14 e 2.15, tem-se que o erro total, a ser minimizado, é função da saída atual da rede $y$, que é função $F$ da entrada $x$ e dos pesos $\mathbf{W}$ atuais.

$$min\{E = \frac{1}{2}[\sum_{j \in C_s}(d_j - F(x_j, W))]^2\} \tag{2.18}$$

O método do gradiente descendente, usado no algoritmo da Retropropagação possui alguns problemas de eficácia e robustez, inerentes aos otimizadores locais. A natureza

não-linear da função de ativação gera uma superfície de erro multimodal, que aliada a uma inicialização inadequada dos pesos (ponto de operação inicial), pode tornar o algoritmo lento e pouco robusto. Algumas heurísticas podem ser usadas para melhoria de desempenho [24]:

a) uso de função de ativação antissimétrica;

b) normalização de entradas e saídas;

c) inicialização de pesos com valores medianos;

d) taxa de aprendizado menor nas camadas posteriores.

No contexto de uma rede neural, as unidades ocultas fornecem um conjunto de funções que constituem uma base arbitrária para os padrões de entrada, quando eles são expandidos sobre o espaço oculto. As funções de ativação, de um modo genérico, podem ser chamadas de funções de base.

Funções radiais são uma classe de função caracterizada por sua forma, cujos valores de funcão $f(x)$ incrementam ou decrementam monotonicamente, proporcional à distância de $x$ para o ponto central de $f$. As redes RBF utilizam, em seus neurônios internos, funções de base com forma radial.

Uma rede RBF, em sua forma mais básica, a ser usada como exemplo neste capítulo, possui 3 camadas com papéis diferentes: uma camada de entrada que faz a *interface* com o ambiente exterior; uma única camada oculta que aplica uma transformação não-linear do espaço de entrada para o espaço oculto; e uma camada de saída linear, que fornece a resposta da rede.

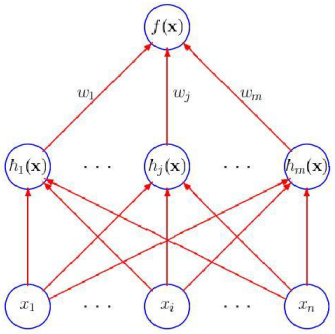


**Figura 2.5:** Exemplo de rede RBF de 3 camadas

Existem várias formas de se treinar uma rede RBF. Uma das possibilidades envolve as seguintes etapas:

a) aprendizagem auto-organizada, cujo o propósito é estimar localizações adequadas para os centros das funções de base radiais;

b) aprendizagem supervisionada, que calcula os pesos da camada de saída;

Um dos enfoques para se aplicar o treinamento supervisionado é usar o método dos mínimos quadrados. Entretanto, determinar a matriz-peso é um problema de modelar o processo responsável pela geração do mapeamento entrada-saída, ou seja, um problema de reconstrução de uma hipersuperfície[24] que pode ser um problema mal-posto, visto que os dados da amostra podem ser insuficientes ou ruidosos, causando provável violação dos critérios de unicidade e continuidade, respectivamente. Por esse motivo, a teoria da regularização pode ser aplicada para a construção de uma outra função de desempenho de treinamento que restrinja os possíveis valores de $\mathbf{W}$.

Para uma rede RBF com $N$ neurônios na camada de entrada (vetor de entrada N-dimensional), $M$ neurônios na camada oculta, e 1 neurônio na camada de saída, a função erro, medida para os $P$ padrões de treinamento, é:

$$\xi = \sum_{k \in P} \left\{ d_k - \sum_{j \in M} \sum_{i \in N} w_j G(\|x_{ki} - c_j\|) \right\}^2 + \alpha \sum_{j \in N} w_j^2. \tag{2.19}$$

onde $G$ é uma função de base radial, tipicamente uma função de *Green*, $c_i$ são os centros de $G$, previamente determinados, e $\alpha$ é o coeficiente de regularização [30].

## 2.6 Algoritmos Evolutivos em Treinamento Supervisionado

Algoritmos evolutivos podem ser alternativas eficientes para o treinamento de redes neurais. No caso do treinamento supervisionado, o enfoque evolutivo é particularmente bem interessante quando a função de erro, a ser minimizada, gera espaços de busca vastos, complexos, multimodais e não-diferenciáveis [31]. Outras motivações são a capacidade desses algoritmos encontrarem uma configuração ótima ou suficientemente próxima a ótima, e por não necessitarem de derivadas e gradientes para isso. Essa última característica é especialmente interessante para arquiteturas de redes neurais onde o cálculo de derivadas sejam razoavelmente complexo.

Uma aplicação bem simples, por exemplo, envolve o treinamento evolutivo aplicado a uma arquitetura de rede pré-estabelecida, fixa, cujos pesos sinápticos são os únicos parâmetros a serem ajustados no processo de treinamento. Nesse caso, a população de indivíduos, codificando a matriz-peso, evolui ao longo de gerações, por competição, onde os mais aptos são os que melhor minimizam a função erro para todos os padrões de treinamento [11].

Aplicações mais complexas, evoluem indivíduos que representam partes do problema de especificação completa de uma rede neural e que cooperam entre si, na medida em que a população inteira resolve o problema como um todo [32].

Algumas aplicações recentes têm integrado otimizadores locais para aumentar a eficiência dos treinamentos evolutivos. Em [11], as melhores soluções são melhoradas através do método *quasi-Newton*.

Uma estratégia mais simples é a que usa o próprio algoritmo de retropropagação como otimizador local. Em geral, nesses casos o algoritmo de retropropagação é chamado somente em uma fase final do treinamento, quando o melhor indivíduo está próximo do mínimo global e a superfície de função erro nesse ponto é suficientemente comportada a ponto do ajuste fino nos pesos ser feito de forma eficiente [31]. Essa estratégia de integração, apesar de considerada eficiente, não dissemina na população a informação obtida pelo otimizador local. O algoritmo evolutivo é apenas um mecanismo usado para estimar uma matriz-peso inicial não aleatória e de boa qualidade.

## 2.7 Considerações Finais

Foram apresentados neste capítulo enfoques evolutivos aplicados problemas de otimização numérica em áreas de problemas inversos e treinamento de redes neurais. Foi dada uma atenção especial às estratégias que foram usadas para integração de otimizadores locais aos algoritmos evolutivos, uma vez que a maioria dos trabalhos apresentados relatam ganhos de desempenho com a utilização de tais otimizadores.

Os otimizadores locais se utilizam de informações sobre o problema em questão para direcionar o processo evolutivo. A disseminação dessa informação na população deve ser feita de forma a não desequilibrar a exploração do espaço de busca em virtude de um indivíduo excessivamente melhorado.

Capítulo

3

# Treinamento Populacional em Heurísticas

Otimizadores locais, utilizando heurísticas específicas, podem ser usados como referência de treinamento para uma população de indivíduos. Neste capítulo, são introduzidos os fundamentos do Treinamento Populacional em Heurísticas e as linhas gerais das principais aplicações já desenvolvidas.

Inicialmente, são feitas analogias com o treinamento supervisionado usado em redes neurais, visando uma intuitiva exposição de idéias. A seguir, dois enfoques que utilizam o treinamento populacional são apresentados: o enfoque construtivo, que trabalha com indivíduos esquemas; e um outro, não-construtivo, recém testado para problemas de otimização numérica.

## 3.1 Definições

O termo treinamento é comumente usado para designar um processo no qual um modelo consegue assimilar ou se adaptar a um determinado ambiente inicialmente desconhecido para ele. Os parâmetros livres de modelos em treinamento são ajustados segundo alguma medida de desempenho ou comportamento esperado.

Como já foi visto no Capítulo 2, uma rede neural aprende, basicamente, através de um processo iterativo que ajusta progressivamente seus parâmetros livres. O aprendizado supervisionado é particularmente interessante para os propósitos deste capítulo por trabalhar com informação sobre o problema em questão.

Acompanhado por um supervisor, o processo de treinamento consiste em apresentar à rede um conjunto de padrões de entrada cujas saídas são conhecidas para se determinar um erro em relação a essas saídas. O erro é, então, corrigido progressivamente, através de um algoritmo que ajusta os pesos das conexões entre neurônios, até que a rede responda satisfatoriamente para todos os padrões de treinamento. Genericamente, diz-se que uma rede está treinada para um conjunto de treinamento **T** quando:

$$E = \frac{E_{p1} + E_{p2} + E_{p3} + \cdots + E_{pk}}{k} \leq \varepsilon \tag{3.1}$$

onde $E_{pk}$ são os erros quadráticos médios para cada um dos k pares $[\mathbf{x}, \mathbf{d}] \in \mathbf{T}$; e $\varepsilon$ valor máximo de erro requerido. Nesse caso, o treinamento é a otimização de um conjunto de parâmetros, minimizando-se uma função de erro.

Este trabalho propõe uma nova forma de se direcionar a evolução através de um treinamento populacional que utiliza sistematicamente informação sobre o ambiente (ou problema) a ser assimilado. Tal informação pode ser extraída através de heurísticas específicas.

**Tabela 3.1:** Comparação entre o treinamento de uma rede neural supervisionada e o treinamento populacional

| | Redes Neurais | Treinamento Populacional |
|---|---|---|
| Entidades | Camadas, neurônios, pesos | Indivíduos |
| Algoritmo | Correção do erro, etc | Evolução natural |
| Referência | Padrões de treinamento | Heurísticas |
| Função alvo | Minimizar o erro | Maximizar a adaptação |

Uma rede neural (conjunto neurônios, conexões e pesos) é treinada através de um algoritmo (regras) que usa, como referência, um conjunto de padrões de treinamento extraídos do ambiente (ou problema) em questão por um supervisor. De forma similar, pode-se dizer que uma população de indivíduos (soluções candidatas) é treinada segundo regras da evolução natural, usando, como referência, informações extraídas do ambiente (ou problema) em questão, através de heurísticas específicas.

A referência de treinamento, ou seja, a informação sobre o problema a ser atacado, é o conjunto de padrões, no caso da rede neural, e o conjunto de heurísticas específicas, no caso do treinamento populacional. Nos dois casos, a mudança nos elementos do conjunto de referência (padrões ou heurísticas) implica em mudança também no resultado do treinamento. Por outro lado, em geral, padrões de treinamento são estáticos, enquanto as heurísticas podem extrair informações sobre o problema dinamicamente.

Da mesma forma que o algoritmo de treinamento da rede neural preserva (ou fortalece) as melhores conexões e elimina (ou enfraquece) as piores, a evolução natural, como

regras de treinamento, premia os melhores indivíduos em detrimento dos piores. Existem diferentes tipos de algoritmos de treinamento, assim como existem diferentes formas de se organizar algoritmos evolutivos baseadas nas mesmas idéias da evolução natural.

Em alguns tipos de algoritmos de treinamento, o erro da rede neural é a medida de desempenho que, ao mesmo tempo, direciona o processo de treinamento. A chamada adaptação é usada pelo treinamento populacional como direcionador, pois as regiões do espaço de busca ocupadas por indivíduos bem-adaptados serão mais exploradas que as demais.

Um indivíduo bem-adaptado é aquele que não pode ser melhorado com relação à heurística de treinamento sendo utilizada. Portanto, a adaptação, diferentemente do erro, no caso de uma rede neural, é uma medida individual e não coletiva de desempenho em relação a um determinada heurística. Preliminarmente, pode-se dizer que a adaptação do indivíduo $S_i$ é:

$$\text{adaptação}(S_i) =\mid \text{aptidão}(S_i) - \text{aptidão}(\text{heurística}(S_i)) \mid \tag{3.2}$$

A aplicação do treinamento populacional é viabilizada por um algoritmo que promova a evolução de um conjunto de indivíduos privilegiando os mais adaptados com relação a alguma heurística específica para o problema. Entenda-se que o melhor indivíduo com relação a adaptação nem sempre é o melhor com relação a função objetivo.

As heurísticas se encaixam como ferramentas que extraem informações do problema, direcionando a exploração do espaço de busca, de posse dessa informação. Mas elas não enxergam o problema como um todo e podem não conseguir achar a solução ótima.

Otimizadores locais podem ser utilizados com a função de melhorar soluções, buscando no subespaço de busca próximo (vizinhança) por uma nova solução viável melhor que a original. Caso haja essa solução melhor, diz-se que o indivíduo original não está bem-adaptado à heurística empregada pelo otimizador local. Se o otimizador não lograr êxito em sua busca, significa que a solução original é a melhor solução dentro da vizinhança estabelecida pela heurística.

O significado para uma solução que não pode ser melhorada é que já se trata de um mínimo (ou máximo) ponto de energia, bem-adaptado a heurística usada e, portando, deve permanecer o máximo de épocas (gerações) possíveis participando do processo evolutivo.

Algumas implementações do Algoritmo Genético Construtivo (AGC)[33], já encampam bem essa filosofia de manter uma população de indivíduos bem-adaptados segundo alguma heurística específica. Mais detalhes do AGC são apresentados na próxima seção.

## 3.2 Algoritmo Genético Construtivo

O Algoritmo Genético Construtivo (AGC) tem sido aplicado a diversos tipos de problemas de combinatória tais como Localização [33], *Timetabling* [34], Leiaute de Matriz-Porta

[35]. O aspecto construtivo presente no AGC, reside no fato dele trabalhar inicialmente com uma população de esquemas, isto é, uma população de soluções candidatas parciais ou incompletas, que irão servir de base para a construção de uma população com soluções completas (chamadas de estruturas), ao longo do processo evolutivo.

É comum se representar a ausência de informação com o símbolo "#"(*do not care*). Um exemplo de esquema para um problema de permutação poderia ser:

$$4, \#, 6, 1, 3, \#, 8, \#, \#, \# \underset{\text{instanciação}}{\Longrightarrow} 4, 9, 6, 1, 3, 0, 8, 5, 7, 2 \tag{3.3}$$

onde as posições com "#"podem ser preenchidas com qualquer símbolo usado na representação, preservando a viabilidade da solução.

Os esquemas são indivíduos que não correspondem a uma solução viável, pelo menos na dimensão em que o problema está posto. Em princípio, não poderiam ser avaliados exatamente da mesma forma que as estruturas, mas é o que o AGC faz, por definição: ambos são avaliados de uma forma comum e competem entre si, durante o processo evolutivo, proporcionalmente a essa avaliação.

As estruturas são formadas, ao longo das gerações, a partir de recombinações sucessivas de esquemas. Espera-se que os fragmentos de soluções presentes nos esquemas sejam melhorados da mesma forma que as soluções presentes nas estruturas.

O AGC difere dos algoritmos genéticos Messy (m-GA's) [36, 37, 38] que também trabalham com esquemas, basicamente, for avaliá-los diretamente, enquanto que os m-GAs realizam buscas do tipo *hill-climbing* para gerar instâncias que serão avaliadas. Outras características do AGC que difere da maioria dos enfoques genéticos tradicionais são: população de tamanho variável e a possibilidade de se usar heurísticas para definição da função de avaliação de aptidão dos indivíduos.

Algumas aplicações do AGC utilizam heurísticas específicas para problemas de otimização combinatória. Aspectos da modelagem genético construtiva relativos a essas aplicações são detalhadas na próxima seção.

## 3.3 Modelagem Genético Construtiva

Uma aplicação do AGC pode ser dividida em duas fases: a construtiva e a ótima. A fase construtiva é usada para construir uma população de estruturas e esquemas bem-adaptadas através de operadores de seleção, recombinação e mutação. A fase ótima é conduzida simultaneamente e transforma os objetivos do problema original em um problema de minimização de intervalo que avalia estruturas e esquemas da mesma forma.

A primeira etapa na modelagem de problemas para o AGC é a definição das funções de aptidão. O AGC trabalha com uma dupla avaliação chamada de avaliação $f - g$, onde

($g \geq f$). Uma das funções de aptidão é a própria função objetivo, $f_{obj}$, e a outra retrata alguma o benefício de alguma manipulação sobre a solução original.

O primeiro requisito do AGC é que tanto as estruturas (indivíduos completos), quanto os esquemas (indivíduos incompletos) sejam avaliados da mesma forma, ou seja, a avaliação $f - g$ é aplicada da mesma forma para qualquer tipo de indivíduo $S_k$ na população. O segundo requisito é que o problema em questão seja modelado como um novo problema chamado de problema de otimização Bi-Objetivo (PBO):

$$\begin{cases} \min & \{g(S_k) - f(S_k)\} \\ \max & \{g(S_k)\} \\ & \textit{Sujeito a} \qquad g(S_k) \geq f(S_k) \end{cases} \tag{3.4}$$

Se o problema em questão é de maximização, a função $f$ reflete $f_{obj}$ e a avaliação $g$ pode ser uma otimização local aplicada ao indivíduo. Caso contrário, inverte-se a natureza das funções $f - g$. Para ambos os casos (minimização ou maximização), o PBO é formulado da mesma forma, ou seja, qualquer problema de otimização passa a ser descrito por esses dois problemas:

O1) minimização do intervalo $(f - g)$

O2) maximização da função objetivo $(f_{obj})$

O objetivo O1 está relacionado à fase ótima do AGC. Uma vez utilizada uma heurística $H$ qualquer para a definição da função $f$ (problema de minimização) ou da função $g$ (problema de maximização), O1 se refere à maximização da adaptação do indivíduo com relação a $H$ que foi preliminarmente sugerida na Eq. (3.2). Ou seja, minimizar o intervalo $(f - g)$ significa maximizar a adaptação do indivíduo.

O objetivo O2, por sua vez corresponde à fase construtiva do AGC. Uma vez que esquemas e estruturas são avaliados da mesma forma, vai existir uma diferença numérica em suas avaliações de função objetivo proporcional à quantidade de informação presente neles. Os esquemas, por terem menos informação que as estruturas, tendem apresentar valores para a função objetivo mais distantes de um pré-definido limitante superior chamado de $G_{max}$.

É desejável que estruturas sejam "construídas" a partir de esquemas ao longo das gerações e, portanto, haja uma maximização de função objetivo. Ao mesmo tempo que indivíduos bem-adaptados vão sendo gerados e haja uma minimização de intervalo $f - g$.

O processo evolutivo é, então, conduzido considerando um limiar de rejeição adaptativo que contempla ambos os objetivos do PBO. Seja um parâmetro $\alpha$ e uma constante $d \in [0, 1]$, a expressão

$$g(S_k) - f(S_k) > d \cdot G_{max} - \alpha \cdot d \cdot [G_{max} - g(S_k)] \tag{3.5}$$

apresenta uma condição para rejeição de um esquema ou estrutura $S_k$. O lado direito de (3.5) é o limiar, composto do valor esperado para a minimização do intervalo $d.G_{max}$, e a medida $[G_{max} - g(S_k)]$, que mostra a diferença entre a avaliação $g(S_k)$ e o limitante superior $G_{max}$.

A Eq. (3.5) pode ser examinada variando-se o valor de $\alpha$. Para $\alpha = 0$, ambos, esquemas e estruturas, são avaliados unicamente pela diferença $g - f$ (O1). A medida que $\alpha$ é incrementado, os esquemas são mais penalizados que as estruturas pela diferença $G_{max} - g$ (O2).

O parâmetro $\alpha$ está relacionado com o tempo de evolução. Considerando que os bons esquemas precisam ser preservados para serem recombinados, $\alpha$ inicia a partir de *zero* e é lentamente incrementado, de geração em geração. A população no tempo de evolução $\alpha$, denotada por $P_\alpha$, possui tamanho dinâmico de acordo com o valor de $\alpha$, e pode, inclusive, ser esvaziada durante o processo. O parâmetro $\alpha$ é isolado na Eq. (3.6) e o lado direito corresponde a um valor de *ranking* que é atribuído a cada indivíduo da população.

$$\alpha \geq \frac{d \cdot G_{max} - [g(S_k) - f(S_k)]}{d \cdot [G_{max} - g(S_k)]} = \delta(S_k) \tag{3.6}$$

Quando são criados, esquemas e estruturas recebem os seus correspondentes valores de *ranking* ($\delta$) que será comparado com o parâmetro $\alpha$, a cada geração, para efeito de eliminação doe indivíduos mal-adaptados ($\delta(S_k) \leq \alpha$). Dessa forma, os indivíduos com maior $\delta$ são melhores com relação ao $PBO$, sobrevivem por um número maior de gerações e, consequentemente, se reproduzem mais.

Os indivíduos são mantidos na população, em ordem descendente, de onde são selecionados, recombinados e, eventualmente, sofrem mutação. O procedimento de seleção privilegia os indivíduos do topo do *ranking* (melhores). A recombinação, por sua vez, constrói novos indivíduos a partir da informação contida (total ou parcial) nos indivíduos pais. Por último a mutação, em geral, é algum tipo de busca local agressiva que ocorre com alta probabilidade.

## 3.4 Detalhamento do Treinamento Populacional

O AGC trabalha com um conceito de adaptação que engloba dois objetivos, sendo que um deles, O2, se refere especificamente a fase construtiva. Para o AGC, os melhores indivíduos (bem-adaptados) são estruturas com $(f - g)$ igual a *zero*. O objetivo original do problema é contemplado por O1, na medida que estruturas que não podem ser melhoradas pela heurística $H$, se esta for capaz de encontrar o ótimo, podem ser soluções ótimas. Com

isso, o AGC maximiza ou minimiza o problema original através da minimização o intervalo $(f - g)$.

Em otimização combinatória, a fase construtiva do AGC funciona como esperado: a medida em que estruturas vão sendo "construídas", vão ocupando o topo do *ranking*, o que garante uma maior participação no processo evolutivo. Isso ocorre sobretudo por que o acréscimo de informação nos esquemas minimiza o intervalo $[G_{max} - g(S_k)]$.

Entretanto, em alguns tipos de problemas, estruturas não necessariamente têm valor de função objetivo maior que esquemas. Por exemplo, em problemas de otimização numérica, soluções incompletas não podem ser coerentemente avaliadas, como se fossem um subconjunto do espaço de busca. Muitas vezes os parâmetros das função objetivo estão interrelacionados e a eliminação de um deles poderia até causar descontinuidades no espaço de busca, como uma divisão por *zero*.

Métodos alternativos para proceder a avaliação de esquemas podem ser sugeridos, como por exemplo a avaliação de uma das instâncias do esquema. Contudo, até o presente momento, não existe um sentido físico para uma solução incompleta de uma função objetivo numérica. Em face disto, um algoritmo alternativo que contempla somente a fase ótima do AGC está sendo sugerido neste trabalho.

Sem perda de generalidade, o intervalo $d \cdot [G_{max} - g(S_k)]$ é eliminado da Eq. (3.6), gerando:

$$\delta(S_k) = d \cdot G_{max} - [g(S_k) - f(S_k)] \tag{3.7}$$

A Eq. (3.7) contempla apenas a minimização do intervalo $[g(S_k) - f(S_k)]$ e, dessa forma, os melhores indivíduos seriam aqueles melhor adaptados à heurística $H$ empregada avaliação $(f - g)$. Pode-se dizer que os melhores indivíduos são aqueles melhor treinados pela heurística $H$.

De forma genérica, a heurística $H$ é usada para gerar um conjunto de pontos (soluções) a partir de $S_k$ que serão avaliados para se escolher o melhor. A melhor avaliação encontrada nesse conjunto de pontos é usada como valor de $f(S_k)$ (em minimizações). O conjunto de pontos gerados por $H$ a partir de $(S_k)$ é chamado de vizinhança de $S_k$ ou $\varphi^H(S_k)$.

$$f(S_k) = g(S_v), S_v \in \{S_1, S_2, \ldots, S_V\} \subseteq \varphi^H, g(S_v) \leq g(S_k) \tag{3.8}$$

Uma característica típica dos problemas de otimização numérica é multimodalidade, i.e., a presença de vários, ou mesmo infinitos, mínimos locais. Por esse motivo, podem existir infinitos pontos no espaço de busca a partir dos quais a heurística $H$ não consegue encontrar um ponto com melhor valor de função objetivo. Considerando que $d \cdot G_{max}$ é um limitante superior fixo, é provável que um número significativo de indivíduos espalhados por um espaço de busca multimodal tenham a mesma avaliação de *ranking* $\delta$. Por esse motivo a Eq. (3.7) é modificada, para alcançar uma maior variabilidade nos valores de $\delta$.

$$\delta(S_k) = d \cdot [G_{max} - g(S_k)] - [g(S_k) - f(S_k)] \tag{3.9}$$

Pode-se observar que esta formulação é específica para problemas de minimização e, por isso, pode-se estabelecer que a função $g$ passa a ser o valor de função objetivo ($f_{obj}$) e $f$ é a avaliação do melhor indivíduo encontrado em $\varphi^H$. O *ranking* agora possui dois componentes:

I1) Intervalo $[g(S_k) - f(S_k)]$: mede o erro com relação a heurística de treinamento $H$;

I2) Intervalo $d \cdot [G_{max} - g(S_k)]$: contempla a maximização da diferença entre o limitante superior $G_{max}$ e a função objetivo $g$.

Da mesma forma que acontece com o AGC, antes da inicialização da população inicial, é gerado um pequeno conjunto aleatório de indivíduos e a maior avaliação de função objetivo dentre eles é atribuída ao limitante $G_{max}$. Durante o processo evolutivo, todo indivíduo com $g \geq G_{max}$ é descartado. A constante $d$ é um parâmetro a ser ajustado que significa o peso de $G_{max} - g$ na equação do *ranking*.

Cada indivíduo $S_k$ gerado, primeiramente recebe uma avaliação de função objetivo $g(S_k)$. A seguir, a heurística de treinamento $H$ é aplicada para avaliar a vizinhança $\varphi^H(S_k)$ e atribuir um valor para $f(S_k)$. Se $H$ falha, $S_k$ já está bem-treinada segundo $H$. Caso contrário, o valor de I1$\neq 0$ é subtraído do valor de I2, penalizando o indivíduo. Dessa forma, os indivíduos com melhores *rankings* são aqueles com baixos valores de função objetivo e que estejam bem-treinados com relação à heurística empregada. Novamente, o problema de minimização original é modelado como a minimização do intervalo I1 e a maximização do intervalo I2.

## 3.5 Heurísticas

O processo de modelagem de um algoritmo para treinamento populacional passa pela construção de procedimentos heurísticos específicos para os problemas em questão. É desejável que a heurística empregada tenha as seguintes propriedades:

a) seja capaz de encontrar a solução ótima:

$$\exists x_k / x^* \in \varphi^H(x_k), \exists x_{k-1} / x_k \in \varphi^H(x_{k-1}), \quad \cdots, \exists x_0 / x_1 \in \varphi^H(x_0), \quad \{x^*, x_k, x_{k-1}, \cdots, x_1, x_0\} \in \beta$$

onde $\beta$ é o espaço de busca e $x^*$ é a solução ótima para o problema, dentro dos critérios de optimalidade empregados;

b) seja configurável para um número máximo de passos (finita), o que define a precisão máxima do procedimento;

c) seja de baixo custo computacional, uma vez que será empregada na avaliação e cada indivíduo gerado.

O AGC para o problema de Leiaute de Matriz-Porta utiliza um procedimento que encontra a melhor solução dentro de uma vizinhança 2-Opt, i.e., dentro de um conjunto de pontos que podem ser obtidos através de movimentos do tipo 2-Opt [35]. Essa e outras aplicações são detalhadas posteriormente.

O treinamento de uma população para uma única heurística significa que toda a população é avaliada da mesma forma, ou seja, todos os indivíduos têm probabilidades de participarem do processo evolutivo proporcional a essa avaliação comum. Assim, a única variável envolvida na avaliação são os seus próprios genótipos.

Por outro lado, no treinamento populacional também podem ser empregadas mais de um tipo de heurística. Diferentes heurísticas estabelecem diferentes vizinhanças e, consequentemente, diferentes visões do mesmo espaço de busca. Múltiplas heurísticas também influenciam a avaliação do indivíduo. Heurísticas pouco eficientes poderiam privilegiar indivíduos aparentemente bem-treinados em detrimento de outros, cuja eficiência de suas heurísticas, acabaria por penalizá-los.

A utilização de coeficientes de pesos poderia equilibrar a participação de todos indivíduos no processo evolutivo de forma um pouco mais independente das heurísticas de cada um, objetivando, principalmente, manter as diferentes visões do espaço de busca dentro da população ao longo das gerações.

Heurísticas podem ser utilizadas em implementações separadas ou até mesmo em uma única implementação que contemple várias heurísticas cooperando ou competindo entre si através dos indivíduos associados a elas. Dessa forma, o enfoque multi-heurístico do treinamento populacional pode ser explorado de duas formas:

a) uma única população treinada em diferentes heurística (1:N);

b) múltiplas populações treinadas em diferentes heurísticas (N:N);

A primeira forma pode trabalhar com uma população de indivíduos definidos por pelo par $(s_i, h_j)$, onde $s_i \in \beta$ e $h_j \in \mathbf{H}$ (conjunto de heurísticas empregadas). Nesse caso, os coeficientes de pesos mencionados anteriormente seriam constantes relacionadas as suas respectivas heurísticas.

A segunda forma abre espaço para o estudo de algoritmos paralelos. Subpopulações paralelas sendo treinadas em diferentes heurísticas sem a necessidade de pesos. Procedimentos migratórios permitiriam a troca de material genético entre as diferentes subpopulações.

## 3.6 Algoritmo para Treinamento Populacional

Como foi dito anteriormente, a aplicação do treinamento populacional é viabilizado por um algoritmo que promova a evolução de um conjunto de indivíduos privilegiando os mais adaptados com relação a uma ou mais heurísticas específicas para o problema. Nesta seção é apresentado uma versão preliminar do Algoritmo para Treinamento Populacional (ATP) baseado no AGC e que serve de ilustração para o processo evolutivo como um todo.

```
Gmax=Define_Gmax();
P=Gera_Populacao();
Melhor=Guarda_Melhor(P);
alpha=Pior_Ranking(P);
t=1;
Enquanto (Não Fim)
    passo=1;
    t=t+1;
        Enquanto (passo < MAX_PASSOS)
                Base=Seleciona(P, BASE);
                Guia=Seleciona(P, GUIA);
                Novo=Cruzament(Base, Guia);
                AvaliaG(Novo);
                Melhor=Guarda_Melhor(Novo);
                Se (PERC_MUTACAO)
                    Mutacao(Novo);
                Se (Novo > Gmax)
                    Descarta(Novo);
                AvaliaF(Novo);
                AvaliaDelta(Novo);
                InsereNaPopulacao(P, Novo);
                passo = passo  1;
        Fim;
        PodaPopulacao(P, alpha);
        inc = V1*tamPop*(Melhor_Ranking(P)-Pior_Ranking(P))/GeracoesRestantes;
        Se (inc > V2)
            alpha = alpha + inc;
        Senao
            alpha = alpha + V2;
        t = t+1;
    Fim;
Resultados(P);
Fim;
```

A grande maioria do que já foi explicado neste trabalho aparece neste pseudocódigo do ATP: uma fase preliminar onde é definido um valor para $G_{max}$, gerada a população inicial, e inicializado o parâmetro $\alpha$ (*alpha*). Observe que o primeiro valor que *alpha* recebe é o valor do *delta* do pior indivíduo (*ranking*) na população (*alpha=Pior_Ranking(P)*). A cada geração, várias recombinações são realizadas, novos indivíduos são gerados e, ao final de cada uma delas, ocorre a *poda* de indivíduos mal-adaptados (*PodaPopulacao(P, alpha)*), i.e., indivíduos com $\alpha \geq \delta$ são eliminados da população $P$. Após a *poda*, o *alpha* é atualizado com um novo *ranking*, de acordo com a faixa de *rankings* na população.

Há sempre uma preocupação quanto a dinâmica da população que pode crescer demasiadamente ou esvaziar-se prematuramente. Em vista disso, o incremento de *alpha* não é mais linear como tem sido sugerido nos recentes trabalhos do AGC [33, 34, 35]. Diferentemente, este trabalho propõe um ajuste adaptativo seguindo a estratégia de incrementar *alpha*, considerando o tamanho da população, à faixa de valores dos *rankings* na população, e o número de gerações que restam para o término da simulação.

A medida em que a população vai crescendo, vai também aumentando a pressão para que os incrementos de *alpha* sejam maiores, mantendo-a controlada. Quando houverem poucos indivíduos na população, provavelmente eles estarão com *rankings* concentrados em uma pequena faixa o valores. Isso tende a tornar ínfimo os incrementos de *alpha*. Nesse caso o incremento mínimo aceitável é $V2$. Um outro parâmetro que ajusta a tamanho do incremento é $V1$, usado como um coeficiente uniformizador de incremento.

Na recombinação, primeiramente, dois indivíduos são selecionados da população através de um procedimento de seleção chamado de Base-Guia [33, 35]. Este procedimento consiste na escolha de um indivíduo dentro de um conjunto de elite da população, por exemplo, dentre os 20% melhores (indivíduo *Base*), enquanto que o outro é escolhido dentre a população inteira (indivíduo *Guia*). Este esquema de seleção é facilitado pelo fato da população dinâmica estar ordenada pelo *ranking* onde os melhores indivíduos estão no topo.

A recombinação é determinante para o bom desempenho de qualquer algoritmo evolutivo. O AGC tem usado variações de recombinações chamadas de recombinação Base-Guia para construir novos indivíduos a partir de esquemas ou estruturas. Um exemplo de recombinação Base-Guia é mostrado a seguir:

**Tabela 3.2:** Exemplos de cruzamentos Base-Guia

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Base | 1 | 5 | # | 8 | # | # | 3 | 4 | 1 | 5 | # | 8 | # | # | 3 | 4 |
| Guia | # | 2 | # | 4 | 3 | 8 | 5 | 7 | # | 2 | # | 4 | 3 | 8 | 5 | 7 |
| Novo | 1 | 5 | # | 8 | 3 | # | # | 4 | 1 | 5 | # | 8 | # | # | 3 | 4 |

Problemas de permutação trabalham com soluções onde cada *gene* significa um vértice (rótulo) que não pode aparecer mais de uma vez no cromossomo. A Tabela 3.2 mostra

duas formas diferentes de cruzamento Base-Guia para problemas de permutação que têm a mesma filosofia: compor um novo indivíduo viável a partir da soluções (completas ou parciais) de dois outros previamente selecionados. As formas privilegiam a informação contida no indivíduo *Base*, desde que esta não seja uma indeterminação #. Caso o gene do indivíduo *Base* seja # ou já esteja presente no indivíduo novo, o gene do indivíduo *Guia* é considerada na composição. Caso este também seja # ou já esteja presente no indivíduo novo, uma indeterminação é posta nessa posição.

A diferença entre as duas formas de cruzamento Base-Guia está na ordem em que os indivíduos são inspecionados. Na primeira, os indivíduos são inspecionados da esquerda para a direita, executando-se a regra citada anteriormente. Na segunda forma, toda a informação do indivíduo *Base* é copiada para o novo indivíduo e, só depois disso, os genes do indivíduo *Guia* são inspecionados, da esquerda para a direita, para serem postos no novo indivíduo, caso não haja perda de viabilidade. Essa diferença entre as duas formas pode causar diferenças leves entre os indivíduos gerados, como pode ser observado na Tabela 3.2.

O ATP não trabalha com esquemas e, por isso, recombinações do tipo Base-Guia não são necessárias. Para as primeiras aplicações em otimização numérica, tem sido usado cruzamentos para parâmetros reais, como o *Blend*[39], ou *Simulated Binary*[40].

Um outro operador evolutivo que pode ser usado é o de *mutação*. A primeira versão do ATP para otimização numérica utilizou uma mutação não-uniforme [9], mas podem ser empregadas buscas locais mais agressivas, como o simplex *Down-Hill*.

A aplicação do AGC para problemas de permutação utiliza a mesma heurística 2-Opt tanto para o treinamento, cálculo de $f$, quanto para a mutação. Sendo que para a mutação, a busca local inspeciona uma vizinhança 2-Opt maior e, por isso, tem um custo computacional maior [35].

Capítulo

4

# Treinamento Populacional em Heurísticas: Resultados Parciais

Neste capítulo são relatados os resultados dos primeiros experimentos com algoritmos genéticos que utilizam o treinamento em heurísticas. Em combinatória, o AGC foi aplicado a dois tipos de problemas de permutação relacionados com leiaute de matrizes (MOSP e GMLP), e a problemas SAT. Em otimização numérica, uma versão preliminar do ATP foi aplicada a um grupo de funções-teste irrestritas.

Inicialmente, é feita uma exposição de alguns aspectos teóricos do problema em questão; a seguir, são mostrados detalhes relativos ao treinamento populacional, como codificação, avaliação e heurísticas empregadas; e por último, os resultados obtidos são comparados com os de outros trabalhos recentes encontrados na literatura.

## 4.1 Problema Relacionados com Leiaute de Matrizes

O Problema de Leiaute de Matriz-Porta (*Gate Matrix Layout Problem*) está relacionado com *arrays* lógicos uni-dimensionais e *arrays* lógicos programáveis [41, 42]. Em projetos de circuitos VLSI (*Very Large Scale Integration*), o objetivo é arranjar o conjunto de nós de circuito (portas) em uma seqüência que minimize a área de leiaute, i.e., minimize o número de trilhas necessárias para cobrir a interconexão das portas.

A Figura 4.1 mostra exemplo de um circuito composto de 9 portas, dispostas em colunas, que se interconectam através de redes, dispostas em linhas. Para interconexão

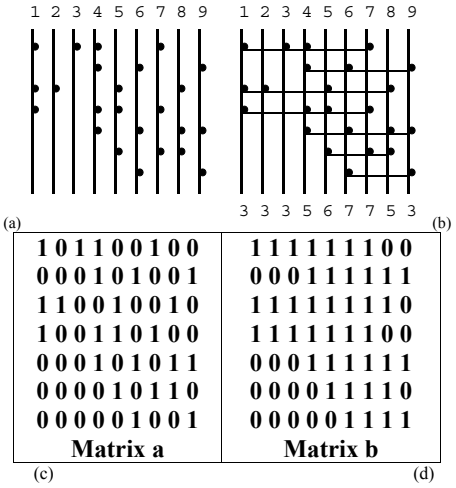


**Figura 4.1:** Circuito VLSI: (a)original, e (b)interconectado; representação matricial: (c)original, e (d)interconectada [35]

é utilizado o chamado metal, representado como uma seqüência de 1's nas linhas das matrizes (Fig. 4.1d). Essa seqüência de 1's entre a porta mais a esquerda e porta mais a direita da mesma rede é obtida através da aplicação da *propriedade de 1's consecutivos* [43].

Para conectar portas que fazem parte de uma mesma rede, muitas vezes é necessário passar por portas que não fazem parte da rede. Isso não inviabiliza o projeto, mas causa um provável desnecessário gasto de metal. Além disso, e mais importante, redes que não se sobrepõem podem compartilhar uma mesma trilha, ao passo que, aquelas que se sobrepõem tem que utilizar trilhas diferente. A minimização do número de trilhas é objetivo principal nesse tipo de problema.

A Figura 4.1b mostra uma seqüência de números que significam a quantidade de sobreposições de redes e que é usado para o cálculo do número de trilhas necessárias para interconectar todo o circuito. O número de trilhas é $TR = \max\{3, 3, 3, 5, 6, 7, 7, 5, 3\} = 7$, ou seja, é preciso alocar espaço no circuito para 7 trilhas. Uma permutação ótima para este problema exemplo é mostrada na Figura 4.2.

Recentemente, em [44] foram apresentados diversos aspectos do GMLP e sua relação com outros tipos de problemas, como o Problema de Minimização de Pilhas Abertas (*Minimization of Open Stacks Problem -MOSP*), incluindo o fato de ambos serem problemas *NP-hard*.

Um MOSP ocorre em certas situações da produção industrial, onde se tem vários padrões de itens com especificações diferentes que precisam ser cortados. Por exemplo, considere uma indústria de componentes (padrões) de madeira prensada que são formados por pedaços (itens) de madeira de diferentes tamanhos. A medida em que os padrões são processados, itens são cortados e colocados em pilhas de acordo com seus tamanhos: uma

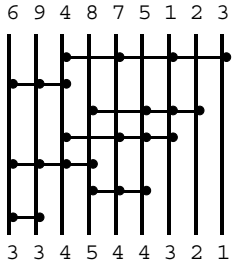


**Figura 4.2:** Solução ótima para o GMLP da Figura 4.1 [35]

pilha para cada tipo de item que fica aberta até que o último item daquele mesmo tipo seja cortado.

Deve-se determinar a seqüência de padrões de corte que minimize o máximo de pilhas abertas durante o processo de corte. Tipicamente este é um problema que ocorre devido à limitações de espaço físico, uma vez que o acúmulo de pilhas pode causar a necessidade de remoção temporária de uma ou outra pilha, retardando o processo de produção.

Um MOSP é dado por uma matriz de $I \times J$, representando padrões (linhas) e itens (colunas), onde $P_{ij} = 1$ se o padrão $i$ contém o item $j$, e $P_{ij} = 0$ caso contrário. Os padrões são seqüêncialmente processados, i.e., todos os itens de cada padrão são cortados por vez. A seqüência de padrões determina o número máximo de pilhas abertas ao mesmo tempo, aqui chamado de *mos*.

A Figura 4.3 mostra: (a) um exemplo de MOSP, com a matriz $P_{ij}$ e, (b) a matriz $Q_{ij}$, derivada de $P_{ij}$ pela aplicação da propriedade de 1's-consecutivos. Trata-se da mesma propriedade que ocorre na formulação do GMLP, exceto que em um GMLP, ela é aplicada por linha (entre a porta mais a esquerda e mais a direita de cada rede), enquanto que em um MOSP ela ocorre por coluna (entre o primeiro item e o último item cortado de um mesmo tipo).

| | pieces | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| $P_{ij}$ = | pattern 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| | pattern 2 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| | pattern 3 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| | pattern 4 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| | pattern 5 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |

(a)

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | Σ |
|---|---|---|---|---|---|---|---|---|---|
| $Q_{ij}$ = | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 2 |
| | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 5 |
| | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 3 |
| | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 4 |
| | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 2 |
| | | | | | | | | MOS= max {2,5,3,4,2} = | 5 |

(b)

**Figura 4.3:** Exemplo de MOSP: (a)matriz original $P_{ij}$; (b)matriz $Q_{ij}$, derivada pela *propriedade de 1's consecutivos*

A Figura 4.4 mostra a solução ótima correspondente a esse problema exemplo. Vários outros aspectos teóricos e práticos do MOSP podem ser encontrados em [44, 45, 46].

| pieces | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | ∑ |
|---|---|---|---|---|---|---|---|---|---|
| pattern 5 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 2 |
| pattern 3 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 2 |
| pattern 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 3 |
| pattern 2 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 4 |
| pattern 4 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 3 |
| | | | | MOS = max {2,2,3,4,3} = | | | | | 4 |

**Figura 4.4:** Solução ótima para o MOSP da Figura 4.3

### 4.1.1 Codificação e Avaliação de Indivíduos

O relacionamento entre GMLP's e MOSP's permitiram que o AGC fosse modelado para ambos de forma similar em vários aspectos como codificação, avaliação, recombinação e mutação. Como já foi dito anteriormente, o AGC trabalha com estruturas (soluções completas) e esquemas (soluções parciais). A Figura 4.5 mostra como as soluções para o GMLP foram codificadas. Os símbolos '?' aparecem nas colunas associadas a posições do indivíduo com indeterminações ('#').

| | |
|---|---|
| **1 0 0 0 1 1 0 1 0** | **1 ? 0 ? ? ? 0 1 ?** |
| **0 0 0 1 0 1 1 0 0** | **0 ? 0 ? ? ? 1 0 ?** |
| **0 1 1 0 0 0 0 1 1** | **0 ? 1 ? ? ? 0 1 ?** |
| **1 0 1 0 0 1 0 1 0** | **1 ? 1 ? ? ? 0 1 ?** |
| **0 0 0 1 0 1 1 0 1** | **0 ? 0 ? ? ? 1 0 ?** |
| **1 0 1 0 0 0 0 0 1** | **1 ? 1 ? ? ? 0 0 ?** |
| **0 0 0 1 0 0 1 0 0** | **0 ? 0 ? ? ? 1 0 ?** |
| **$s_j$ = (7 2 5 9 3 4 6 1 8)** | **$s_k$ = (7 # 5 # # # 6 1 #)** |

**Figura 4.5:** Codificação de estruturas e esquemas para o GMLP

De forma similar foi feito para o MOSP, exceto pelo fato que, neste, a solução (indivíduo) é uma permutação de linhas (padrões) e não de colunas.

Para o GMLP, a função de aptidão $g$ retrata o custo da solução codificada no indivíduo em termos de número de trilhas e de total de metal usado para cobrir todo o circuito. A minimização do total de metal é um objetivo secundário usado para diferenciar a avaliação dos indivíduos.

$$g(S_k) = I \cdot J \cdot \max_{j \in J} \left\{ \sum_{i \in I} b_{ij} \right\} + \sum_{j \in J} \sum_{i \in I} b_{ij} \tag{4.1}$$

onde $I$ é o número de redes, $J$ é o número de portas e são usados para dar um peso maior na primeira parte da equação (minimização do número de trilhas).

Para o MOSP, $g$ avalia o custo em termos de número máximo de pilhas abertas ao mesmo tempo, também acrescido de um termo que calcula o que pode ser chamado de tempo total de pilhas abertas.

$$g(S_k) = I \cdot J \cdot \max_{i \in I} \left\{ \sum_{j \in J} q_{ij} \right\} + \sum_{i \in I} \sum_{j \in J} b_{ij} \tag{4.2}$$

onde $I$ é o número de padrões, $J$ é o número de itens e são usados para dar um peso maior na primeira parte da equação (minimização do máximo de pilhas abertas).

A função $f$, para os dois problemas, avalia a vizinhança 2-Opt do indivíduo. Mais detalhes sobre a implementação da heurística 2-Opt são apresentados na próxima seção.

$$f(S_k) = g(S_v), S_v \in \{S_1, S_2, \ldots, S_V\} \subseteq \varphi^{2-Opt}, g(S_v) \leq g(S_k) \tag{4.3}$$

As funções $(f - g)$ se relacionam através do *ranking* calculado por:

$$\delta(S_k) = \frac{d \cdot G_{max} - [g(S_k) - f(S_k)]}{d \cdot [G_{max} - g(S_k)]} \tag{4.4}$$

### 4.1.2 Heurística 2-Opt

A heurística 2-Opt foi usada para o treinamento nos dois problemas de permutação. Seja um indivíduo com N genes, a vizinhança de pontos a partir desse indivíduo pode ser completamente gerada através de $N(N-1)/2$ movimentos. Se cada um desses pontos for avaliado para que seja escolhido o melhor deles ao final do processo, o custo computacional dessa heurística seria proibitivo.

É requisito do treinamento populacional que a heurística seja configurável para uma precisão máxima. Por isso, o procedimento 2-Opt implementado utiliza um parâmetro que indica a largura $nw$ do cromossomo que será usado para inspeção de vizinhos.

Primeiramente, uma posição inicial é escolhida ao acaso e um processo iterativo é iniciado, a partir daí, tomando duas a duas posições como referência para movimentos do tipo 2-Opt. Por exemplo, para um indivíduo com $N = 10$, largura $nw = 4$, a partir da posição 2, o seguinte conjunto de pontos de referência é gerado: {(2,3),(2,4),(2,5),(3,4),(3,5),(4,5)}, i.e., um total de $4 \cdot (3)/2$.

Cada par de pontos indica as duas posições a partir das quais serão removidas duas arestas, supondo que cada gene equivalha a um vértice e que genes consecutivos estejam ligados por arestas. Uma vez removidas as arestas seguintes a esses pontos, os vértices são reconectados em uma nova disposição, gerando um novo indivíduo, que é vizinho do anterior por um movimento do tipo 2-Opt. Esse novo indivíduo é então avaliado para efeito do cálculo da função $f$. A Figura 4.6 mostra duas formas de se aplicar movimentos do tipo 2-Opt para duas situações distintas: pontos de referência consecutivos e não-consecutivos.

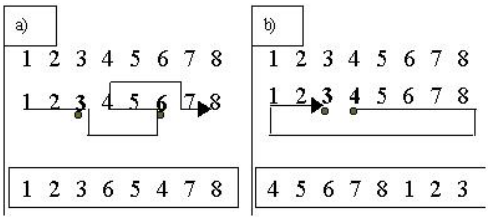


**Figura 4.6:** Exemplos de movimentos 2-Opt com pontos de referência: (a)não-consecutivos, e (b)consecutivos

### 4.1.3 Experimentos Computacionais

O CGA foi aplicado a várias instâncias de GMLP, obtendo as melhores soluções conhecidas para todas as instâncias [35]. A freqüência com que essas melhores soluções foram encontradas também se mostrou satisfatória. Para GMLP's considerados pequenos (com até 40 portas), como as instâncias wsn, wli, v4000, v4050 a freqüência de acerto do AGC foi de 100% (exceto para a instância v4090). Para GMLP's considerados grandes (acima de 70 portas), como w3 (70 portas) e w4 (141 portas), a freqüência ficou em 50% e 30%, respectivamente. A Tabela 4.1 mostra os 5 melhores resultados para as instâncias GMLP testadas.

**Tabela 4.1:** *Cinco* Melhores resultados (metal, trilhas) e respectivas freqüências de sucesso para as instâncias de GMLP

| | Melhores soluções do AGC | | | | | | | | | | % sucesso |
|---|---|---|---|---|---|---|---|---|---|---|---|
| wsn(25x17) | 104 | 8 | 104 | 8 | 105 | 8 | 107 | 8 | 113 | 8 | 100.00 |
| wli(10x11) | 18 | 4 | 18 | 4 | 18 | 4 | 18 | 4 | 18 | 4 | 100.00 |
| v4050(16x13) | 41 | 5 | 41 | 5 | 41 | 5 | 42 | 5 | 42 | 5 | 100.00 |
| v4000(17x10) | 53 | 5 | 53 | 5 | 53 | 5 | 54 | 5 | 55 | 5 | 100.00 |
| v4470(47x37) | 246 | 9 | 253 | 9 | 265 | 9 | 266 | 9 | 282 | 9 | 100.00 |
| v4090(27x23) | 95 | 10 | 96 | 10 | 96 | 10 | 98 | 10 | 99 | 10 | 90.00 |
| x0(48x40) | 303 | 11 | 304 | 11 | 305 | 11 | 306 | 11 | 308 | 11 | 80.00 |
| w1(21x18) | 39 | 4 | 39 | 4 | 39 | 4 | 39 | 4 | 43 | 4 | 100.00 |
| w2(33x48) | 235 | 14 | 236 | 14 | 267 | 14 | 273 | 14 | 275 | 14 | 100.00 |
| w3(70x84) | 677 | 18 | 687 | 18 | 834 | 18 | 843 | 18 | 717 | 18 | 50.00 |
| w4(141x202) | 1730 | 27 | 1836 | 27 | 1849 | 27 | 1745 | 28 | 1745 | 28 | 30.00 |

Com relação aos tempos de execução, o AGC foi comparado com um enfoque não-evolutivo, o *Microcanonical Optmization (MCO)* [47]. O tempo de uma execução do AGC foi superior ao do MCO. Entretanto, o AGC se mostrou bem mais robusto, necessitando de bem menos execuções para encontrar as melhores soluções conhecidas. Por esse prisma, o tempo total de 10 experimentos do AGC é inferior ao tempo total dos 1000 experimentos que foram realizados pelo MCO [35]. Além do que, em 1000 execuções, o MCO aparentemente obteve freqüências de acerto em torno de 40% para problemas pequenos [47].

Em relação ao MOSP, o AGC foi testado em 300 instâncias consideradas pequenas (menos de 40 padrões) e obteve 100% de freqüência de acerto. Em algumas instâncias, inclusive, obteve resultados melhores (valores negritos) que os divulgados na literatura como sendo os ótimos [48]. O AGC foi comparado com outros enfoques não-evolutivos, inclusive com um procedimento 2-Opt *standalone*. A heurística 2-Opt aplicada a uma população de 20 indivíduos gerados aleatoriamente mostrou um desempenho satisfatório para problemas pequenos. Entretanto, para instâncias consideradas grandes, como a GMLP w4 (141 portas) ela não conseguiu obter a melhor solução conhecida [48]. A Tabela 4.2 sumariza as execuções sobre o conjunto de 300 instâncias MOSP.

**Tabela 4.2:** Comparação do AGC com outros enfoques não-evolutivos para MOSP's

| I | J | OPT | COL | TS | GLS | CGA | 2Opt | I | J | OPT | COL | TS | GLS | CGA | 2Opt |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 10 | 10 | 5.5 | 5.5 | 5.5 | 5.5 | 5.5 | 5.5 | 25 | 10 | 8.0 | 8.0 | 8.0 | 8.0 | 8.0 | 8.0 |
| - | 20 | 6.2 | 6.2 | 6.2 | 6.2 | 6.2 | 6.2 | - | 20 | 9.8 | 9.8 | 9.8 | 9.9 | 9.8 | 9.8 |
| - | 30 | 6.1 | 6.1 | 6.1 | 6.2 | 6.1 | 6.1 | - | 30 | 10.5 | 10.6 | 10.7 | 10.6 | 10.5 | 10.5 |
| - | 40 | 7.7 | 7.7 | 7.7 | 7.7 | 7.7 | 7.7 | - | 40 | 10.4 | 10.4 | 10.7 | 10.6 | 10.4 | 10.5 |
| - | 50 | 8.2 | 8.2 | 8.2 | 8.2 | 8.2 | 8.2 | - | 50 | 10.0 | 10.0 | 10.1 | 10.2 | 10.0 | 10.0 |
| 15 | 10 | 6.6 | 6.6 | 6.6 | 6.6 | 6.6 | 6.6 | 30 | 10 | 7.8 | 7.8 | 7.8 | 7.8 | 7.8 | 7.8 |
| - | 20 | 7.2 | 7.2 | 7.2 | 7.5 | 7.2 | 7.2 | - | 20 | 11.1 | 11.2 | 11.2 | 11.2 | 11.1 | 11.1 |
| - | 30 | 7.4 | **7.3** | 7.4 | 7.6 | **7.3** | 7.6 | - | 30 | 12.2 | 12.2 | 12.6 | 12.2 | 12.2 | 12.2 |
| - | 40 | 7.3 | **7.2** | 7.3 | 7.4 | **7.2** | 7.3 | - | 40 | 12.1 | 12.1 | 12.6 | 12.4 | 12.1 | 12.2 |
| - | 50 | 7.6 | **7.4** | 7.6 | 7.6 | **7.4** | **7.4** | - | 50 | 11.2 | 11.2 | 12.0 | 11.8 | 11.2 | 11.2 |
| 20 | 10 | 7.5 | 7.5 | 7.7 | 7.5 | 7.5 | 7.5 | 40 | 10 | 8.4 | 8.4 | 8.4 | 8.4 | 8.4 | 8.4 |
| - | 20 | 8.5 | 8.5 | 8.7 | 8.6 | 8.5 | 8.5 | - | 20 | 13.0 | 13.0 | 13.1 | 13.1 | 13.0 | 13.0 |
| - | 30 | 8.8 | 8.8 | 9.2 | 8.9 | 8.8 | 8.9 | - | 30 | 14.5 | 14.5 | 14.7 | 14.6 | 14.5 | 14.5 |
| - | 40 | 8.6 | 8.6 | 8.6 | 8.7 | 8.6 | 8.6 | - | 40 | 15.0 | 15.0 | 15.3 | 15.3 | **14.9** | 15.0 |
| - | 50 | 7.9 | 7.9 | 8.0 | 8.2 | 7.9 | 8.0 | - | 50 | 14.6 | 14.6 | 15.3 | 14.9 | 14.6 | 14.9 |

As colunas da Tabela 4.2 contabilizam as médias das melhores soluções, de cada grupo de 10 instâncias de problemas MOSP, encontradas pelos enfoques que foram comparados com o AGC.

## 4.2 Problema de Satisfatibilidade (SAT)

O SAT é uma família de problemas NP-Completos [49, 50] que são centrais em um grande número de áreas na computação e engenharia. Uma instância SAT é uma expressão lógica (verdadeiro ou falso) composta de $m$ cláusulas distintas conectadas por operadores lógicos $\wedge's$ $(E)$, onde cada cláusula pode conter um número qualquer de literais conectados por operadores lógicos $\vee's$ $(OU)$. Um literal é qualquer uma das $n$ variáveis lógicas, eventualmente negadas. Essa formulação, chamada de Forma Conjuntiva Normal (*Conjuntive Normal Form - CNF*) é exemplificada a seguir:

$$(x_1 \vee x_5 \vee \overline{x_6}) \wedge (\overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_9) \wedge \cdots \wedge (x_4 \vee x_8 \vee x_5 \vee \overline{x_9}) \tag{4.5}$$

Instâncias SAT podem apresentar um número $k$ fixo de literais por cláusulas. Tais problemas são chamados de k-SAT.

O objetivo de um problema SAT é determinar se existe um conjunto de valores para as variáveis que satisfaça a fórmula, i.e., torne o resultado da expressão verdadeira. Se existir pelo menos uma solução, diz-se que o problema SAT é *satisfatível*, caso contrário, *não-satisfatível*.

### 4.2.1 Codificação e Avaliação de Indivíduos

A codificação de expressões lógicas em indivíduos esquemas é bastante natural. Os primeiros experimentos com instâncias SAT foram motivados pelo fato de uma solução incompleta poder ser avaliada como *verdadeiro* ou *falso*. Por exemplo, uma cláusula $(.V. \vee .F. \vee \#)$, apesar da presença de um símbolo #, pode ser avaliada como verdadeira. Isso garante a existência de soluções viáveis mesmo que incompletas.

Na codificação usada nos primeiros experimentos, cada gene codifica uma variável da expressão lógica que é avaliada em função dos valores dessas variáveis, através da soma do número de cláusulas satisfeitas. Uma solução viável é encontrada quando o número de cláusulas satisfeitas é igual ao número de cláusulas total da expressão.

O AGC tratou o SAT como um problema de maximização, onde a função de aptidão $f$ retrata o ganho de um dado conjunto de atribuições para as variáveis lógicas.

$$g(S_k) = \sum_{i=1}^{C} \{\vee_{j=1}^{N_i} a_{ij}\} \tag{4.6}$$

onde $C$ é o número de cláusulas na expressão, $N_i$ é o número de variáveis na i-ésima cláusula, $\vee$ é o operador lógico $OU$, e $a_{ij}$ é o valor lógico do j-ésimo literal na i-ésima cláusula.

O símbolo # é avaliado dentro de uma cláusula conforme a Tabela 4.3. No somatório final de todas as cláusulas avaliadas como *verdade*, o # é descartado, ou seja, é avaliado como *falso*.

**Tabela 4.3:** Tabela-verdade para operações lógicas com símbolo "#"

| $X$ | $Y$ | $X \vee Y$ |
|---|---|---|
| 0 | # | # |
| 1 | # | 1 |
| # | # | # |

A função $g$, por sua vez, avalia a vizinhança do indivíduo através de heurísticas específica para problemas SAT. Nos primeiros experimentos, foi utilizada a heurística proposta por Gent (1998) [51], explicada na próxima seção.

$$g(S_k) = f(S_v), S_v \in \{S_1, S_2, \ldots, S_V\} \subseteq \varphi^{Gent}, f(S_v) \leq f(S_k) \tag{4.7}$$

### 4.2.2 Heurísticas

Um outro ponto que tem motivado esta aplicação, é o número heurísticas desenvolvidas para o SAT. Existe a possibilidade de se usar mais uma heurística para o treinamento populacional. O procedimento baseado na heurística de Gent [51] é mostrado a seguir.

```
procedure Gent(S, I)
    Para um número fixo de variáveis v escolhidas aleatoriamente
        Se v = .F. and v ocorre mais vezes positivamente em  S
            I := I U {v = .V.}
            Guarda a avaliação de I' se for a melhor;
        Fim
        Se v = .V. and v ocorre mais vezes negativamente em S
            I' := I U {v = .F.}
            if  I  incrementa o número de cláusulas satisfeitas then
            Guarda a avaliação de I' se for a melhor;
        end
    end
    g = melhor avaliação de I';
end
```

Para efeito de avaliação $g$, o procedimento heurístico considera apenas uma parte das variáveis que compõem o indivíduo, ou seja, a vizinhança foi limitada da mesma forma que em outras aplicações do AGC. A heurística considera que variáveis com valor *.F.* e que aparecem mais vezes positivamente na expressão, podem ser *chaveadas*, o que provavelmente melhorará a avaliação da expressão, e vice-versa. É evidente que esse procedimento não garante que a solução ótima seja encontrada. Por isso, a primeira aplicação do AGC para SAT foi acrescida de uma mutação que utiliza o procedimento *WalkSAT*[52].

O WalkSAT incorpora movimentos randômicos quando não consegue incrementar o número de cláusulas satisfeitas. Ele usa uma heurística conhecida como *algoritmo guloso de Koutsoupias e Papadimitriou* [53] que, a partir de uma dada configuração de variáveis, chaveia aquela que causar um maior incremento no número de cláusulas satisfeitas. Esta heurística usada no WalkSAT possui um custo computacional maior, mas tem um desempenho bem superior.

### 4.2.3 Experimentos Computacionais

Foram usadas mais de 70 instâncias de problemas SAT em simulações para avaliar o desempenho do AGC. Todas sabidamente *satisfatíveis* e em formato CNF. Os experimentos foram divididos em 4 suites de teste e os resultados do AGC foram comparados com outros dois enfoques: um enfoque evolutivo a chamado de ASAP (*Adaptive evolutionary algorithm for the SAtisfiability Problem*) [54] e o procedimento WalkSAT. Um dos suites de teste é mostrado na Tabela 4.4.

**Tabela 4.4:** Comparação entre AGC, ASAP e WalkSAT para instâncias SAT *aim*

| Problema | Var | Cla | AGC(s) | WS(s) | ASAP(s) | %AGC | %WS | %ASAP |
|---|---|---|---|---|---|---|---|---|
| aim-50-1_6-yes1 | 50 | 80 | 49.50 | 40.00 | - | 50.00 | 25.00 | - |
| aim-50-2_0-yes1 | 50 | 100 | 17.00 | 38.75 | 18.76 | 100.00 | 50.00 | - |
| aim-50-3_4-yes1 | 50 | 170 | 8.25 | 0.00 | 0.05 | 100.00 | 100.00 | - |
| aim-50-6_0-yes1 | 50 | 300 | 45.50 | 0.00 | 0.01 | 100.00 | 100.00 | - |
| aim-100-1_6-yes1 | 100 | 160 | 44.00 | 50.50 | - | 0.00 | 0.00 | - |
| aim-100-2_0-yes1 | 100 | 200 | 36.50 | 50.25 | - | 50.00 | 0.00 | 0.00 |
| aim-100-3_4-yes1 | 100 | 340 | 13.75 | 0.25 | 0.33 | 100.00 | 100.00 | - |
| aim-100-6_0-yes1 | 100 | 600 | 86.50 | 0.00 | 0.02 | 100.00 | 100.00 | - |
| aim-200-1_6-yes1 | 200 | 320 | 91.25 | 52.00 | - | 0.00 | 0.00 | - |
| aim-200-2_0-yes1 | 200 | 400 | 62.75 | 57.00 | - | 0.00 | 0.00 | - |
| aim-200-3_4-yes1 | 200 | 680 | 106.50 | 0.75 | 20.70 | 75.00 | 100.00 | - |
| aim-200-6_0-yes1 | 200 | 1200 | 274.75 | 0.00 | 0.11 | 100.00 | 100.00 | - |

Os símbolos "-" significam que as referidas informações não estão disponíveis em [54].

## 4.3 Minimização de Funções Numéricas sem Restrição

Saindo do enfoque construtivo, foi desenvolvido um algoritmo similar ao AGC, mas que não trabalha com esquemas, apenas estruturas, e implementa unicamente o treinamento populacional em heurísticas. Esse algoritmo vai ser chamado neste trabalho de Algoritmo de Treinamento Populacional (ATP).

Otimização de funções numéricas é, em geral, a primeira etapa para se medir o desempenho de um otimizador. Um otimizador eficiente para funções numéricas pode ser facilmente adaptado para otimização de pesos de redes neurais supervisionadas, recuperação de condições iniciais em problemas inversos, otimização de conjuntos *fuzzy*.

Nesta primeira etapa, foram escolhidas *seis* funções-teste, com *cinco* e *dez* dimensões: *michalewicz, rosembrock, schwefel, langerman, griewangk*, e *rastringin*. Informações sobre essas funções podem ser encontradas em [55, 56]. A Figura 4.10 mostra seus respectivos gráficos em duas dimensões.

### 4.3.1 Avaliação e Heurísticas

Para a minimização das funções-teste, seus parâmetros foram codificados como valores reais. A função de aptidão $g$ retrata o valor da função objetivo, enquanto que a função $f$ avalia a vizinhança, segundo uma heurística que foi chamada de Busca Em-Linha.

Antes de implementar essa heurística, foram testados outros otimizadores locais, como o simplex *Down-Hill* e o método gradiente por diferenças finitas. Estes métodos possuem um custo computacional significativo, pois sua complexidade é função do número de dimensões do problema, ou seja, cada passo de otimização é proporcional ao número de parâmetros da função objetivo. O método gradiente ainda possui outras desvantagens além dessa, pois pode se tornar inviável para ser aplicado em funções com espaços de busca descontínuos.

A Busca Em-Linha inspeciona uma vizinhança de poucos pontos uniformemente espalhados entre o ponto em questão e um ponto aleatório escolhido dentre o conjunto dos indivíduos *Base*. Dessa forma, o grupo de elite da população, em termos de *ranking*, é usado como referência para estabelecer uma vizinhança para os demais indivíduos da população.

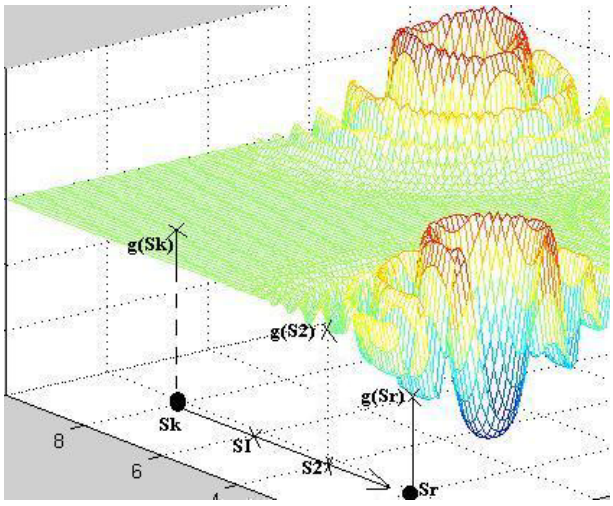


**Figura 4.7:** Exemplo bidimensional do procedimento de busca Em-Linha

A Figura 4.7 mostra um exemplo da busca Em-Linha para o caso bidimensional. Os pontos $S_k$ e $S_r$ são respectivamente o indivíduo a ser avaliado e o indivíduo referência, escolhido aleatoriamente dentro do conjunto de indivíduos *Base*. A linha entre $S_k$ e $S_r$ é dividida em intervalos iguais e pontos dentro desse intervalo são avaliados em função de $g$ (função objetivo). Eventualmente, alguns pontos na mesma linha, mas fora desse intervalo, também são avaliados. No exemplo da Figura 4.7, os limites do espaço de busca não permitem essa extrapolação. Também pelo exemplo, a melhor avaliação encontrada pela heurística, provavelmente, é a do ponto $S_2$. Essa avaliação é usada como valor de $f$.

Uma vez computadas as funções $f$ e $g$, o *ranking* é dado pela Eq. (3.9), aqui repetida:

$$\delta(S_k) = d \cdot [G_{max} - g(S_k)] - [g(S_k) - f(S_k)] \tag{4.8}$$

Na Figura 4.8, o efeito da avaliação dos intervalos $[g(S_k) - f(S_k)]$ (I1) e $d \cdot [G_{max} - g(S_k)]$ (I2) é mostrado separadamente. Indivíduos com I2 alto estão posicionados nas regiões mais promissoras (Fig. 4.8a), enquanto aqueles com I1 mínimo ($f - g = 0$) são posicionados em *plateaus* ou regiões que são prováveis mínimos locais (Fig. 4.8b). A população inteira preenche uma significante parte do espaço de busca. Os espaços não habitados por indivíduo algum são aqueles em que o valor de $g$ se aproxima de $G_{max}$ e indivíduos inicialmente gerados nessas regiões são progressivamente eliminados pelo limiar de rejeição adaptativo.

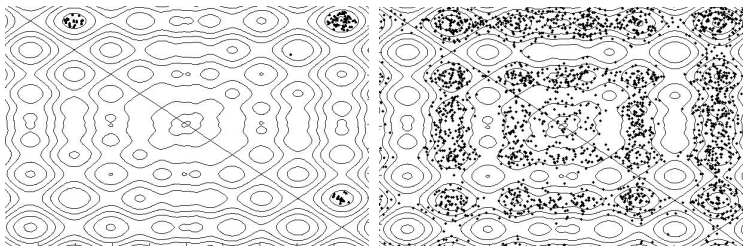

**Figura 4.8:** Indivíduos no mapa de contorno da função *schwefel* bidimensional: (a)indivíduos que maximizam I2 e, (b)indivíduos que minimizam I1

### 4.3.2 Experimentos Computacionais

Os experimentos computacionais foram realizados em duas etapas. Uma etapa para observar o comportamento do ATP no espaço bidimensional de algumas funções multimodais. A outra etapa para aferir o desempenho do algoritmo, comparando com um outro algoritmo genético.

Para comparação, foi implementado um algoritmo genético tradicional (AGT), também com os parâmetros da função codificados diretamente em real, mas como as seguintes diferenças em relação ao ATP: população fixa com *ranking* baseado simplesmente em função objetivo e atualização de população pelo método *steady-state* (ou não geracional). Na atualização *steady-state*, os filhos recém-gerados entram diretamente na população, passando a competir com seus próprios pais. Como a população é fixa, o novo indivíduo deve ocupar o local antes ocupado por um indivíduo ruim. É necessário, então, que haja um procedimento para se encontrar um novo indivíduo ruim a cada geração para ser o alvo da próxima substituição.

Foi observado nos experimentos com funções bidimensionais, que a população no ATP cresce sempre enquanto estiverem sendo gerados novos indivíduos com bons *rankings*.

Evidentemente, o limiar de rejeição adaptativo é usado para controlar a população, eliminando os indivíduos mal-adaptados.

A medida em que as regiões mais promissoras vão sendo encontradas e os indivíduos vão ficando bem-treinados em relação à heurística, a população pára de crescer e os piores indivíduos continuam a serem eliminados, a ponto da população se esvaziar ou permanecer apenas em torno dos mínimos globais (se existir mais de um).

Esse comportamento pode ser observado na seqüência de amostras visualizadas na Figura 4.9. Os gráficos mostram o mapa de contorno da função *langerman* bidimensional e as posições no espaço de busca ocupadas por indivíduos, após 100, 500, 1000 e 2000 gerações, respectivamente.

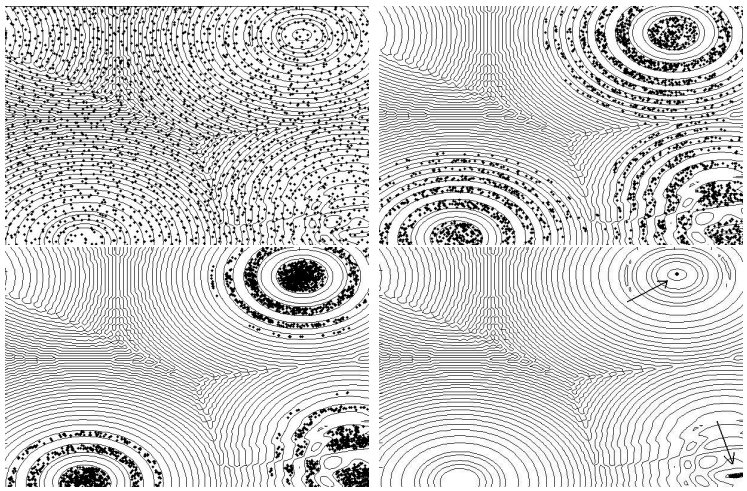

**Figura 4.9:** Indivíduos no mapa de contorno da função de *langerman* bidimensional depois de 100, 500, 1000 e 5000 gerações

Observe que no último quadro, apenas dois grupos de indivíduos se mantém na população. Um grupo em torno do mínimo global (canto inferior direito) e o outro em um ponto de mínimo local (canto superior direito). A outra região promissora foi descartada, pela eliminação dos indivíduos em torno dela.

Os desempenhos do AGT e ATP foram medidos através do erro medido entre as melhores soluções encontradas e as soluções ótimas esperadas. As funções utilizadas possuem mínimos conhecidos. Três delas, *rosenbrock, rastringi e griewank* possuem valor mínimo de função $g(x)^* = 0$. Por isso, um limiar de optimalidade foi fixado como sendo $0,001$ para estas funções. A Tabela 4.5 mostra apresenta um sumário dos resultados.

Em 5 dimensões, o ATP apresentou um melhor desempenho global, mesmo sem considerar o mal desempenho do AGT na função de *rosenbrock* (350% de média de erro). Dessa forma, o ATP obteve 7,66% de média de erro contra 15,88% obtido pelo AGT.

Em 10 dimensões, mais uma vez o AGT tem um mal desempenho na função de rosenbrock (melhor solução encontrada foi 6,10, i.e., 600% de erro em relação ao mínimo

**Tabela 4.5:** Comparação dos resultados do ATP e AGT para as 12 funções-teste

| | **Sol(med)** | | **Sol** | **Erro** | | **Aval(med)** | |
|---|---|---|---|---|---|---|---|
| | **ATP** | **AGT** | **Esperada** | **ATP** | **AGT** | **ATP** | **AGT** |
| Rosembrock(5) | 0.012 | 0.351 | 0.001 | 0.011 | 0.350 | 486024.50 | 534000.00 |
| Rastrigin(5) | 0.001 | 0.001 | 0.001 | 0.000 | 0.000 | 178334.30 | 98992.70 |
| Griewangh(5) | 0.009 | 0.006 | 0.001 | 0.008 | 0.005 | 304628.80 | 540362.45 |
| Langerman(5) | -0.963 | -0.358 | -1.400 | 0.437 | 1.042 | 103839.00 | 751024.00 |
| Michalewicz(5) | -4.682 | -4.687 | -4.687 | 0.005 | 0.000 | 83140.80 | 92533.80 |
| Schwefel(5) | -2094.672 | -2094.908 | -2094.914 | 0.243 | 0.006 | 359142.70 | 404562.20 |
| Rosembrock(10) | 0.102 | 6.640 | 0.001 | 0.101 | 6.639 | 703015.50 | 751000.00 |
| Rastrigin(10) | 0.007 | 0.001 | 0.001 | 0.006 | 0.000 | 142098.40 | 243424.20 |
| Griewangk(10) | 0.009 | 0.009 | 0.001 | 0.008 | 0.008 | 353668.30 | 574628.80 |
| Langerman(10) | -0.004 | -0.003 | -1.400 | 1.396 | 1.397 | 422314.40 | 751000.00 |
| Michalewicz(10) | -9.555 | -7.693 | -9.660 | 0.105 | 1.967 | 691504.10 | 751000.00 |
| Schwefel(10) | -3935.442 | -3952.952 | -4189.829 | 254.387 | 236.877 | 368686.50 | 544419.20 |
| | | | | | | **349699.775** | **503078.946** |

esperado, 0,001). Mais uma vez, excluindo o resultado da função de rosenbrock dos resultados de ambos, existe um equilíbrio de desempenho: ATP obteve 23,95% de erro, contra 26,75% obtido pelo AGT. Todos os experimentos realizados com o ATP e AGT para minimização de funções numéricas estão detalhados em [57].

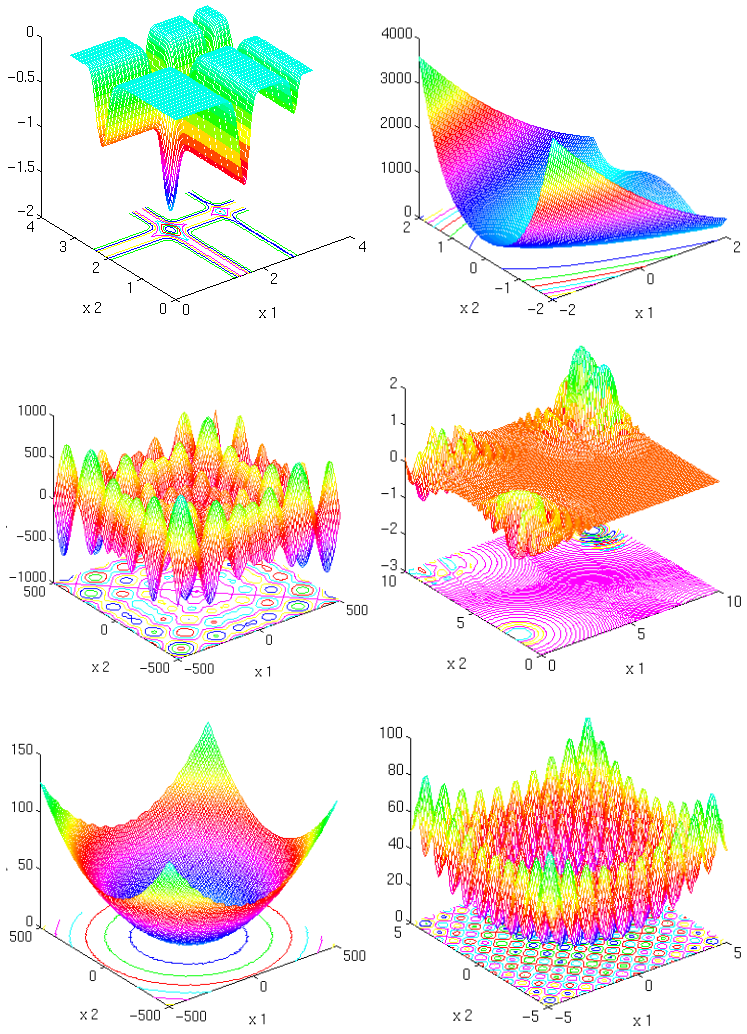


**Figura 4.10:** Gráficos em 3-D das Funções-teste

Capítulo

# 5

# Treinamento Populacional em Heurísticas: Próximas Etapas

Neste capítulo, outros aspectos das aplicações são levantados e novas etapas são sugeridas para cada aplicação. Ao final, as atividades relevantes são descritas e colocadas em um cronograma de atividades.

## 5.1 Minimização de Funções Numéricas

Pretende-se ampliar os experimentos, basicamente para tornar o algoritmo mais competitivo, possibilitando sua aplicação a problemas práticos que envolvam otimização numérica. As próximas etapas incluem o aprimoramento da forma como o treinamento populacional está implementado, principalmente no que tange o algoritmo em si e a formulação do *ranking*, que ainda podem ser melhorados.

Outras heurísticas de treinamento podem ser estudadas, como uma versão populacional do simplex *Down-Hill*, onde o indivíduo em questão forma um simplex com outros indivíduos da população.

Espera-se ao final desta etapa, ter-se um algoritmo competitivo, que possa ser usado em experimentos com funções-teste com maior dimensionalidade. e em outros tipos de problemas de otimização numérica nas áreas de problemas inversos e treinamento supervisionado de redes neurais.

Pretende-se ainda comprovar efetivamente os benefícios do treinamento populacional através de uma análise dos resultados que vierem a ser obtidos nos próximos experimentos

## 5.2 Enfoque Multi-heurísticas

Os resultados obtidos pelo AGC para MOSPs em problemas com menos de 70 padrões são considerados bons. Entretanto, novos experimentos podem ser realizados no sentido de se avaliar problemas maiores com novas heurísticas e até mesmo com novos algoritmos de treinamento populacional.

Algumas heurísticas gulosas têm obtido bons resultados como otimizadores locais [58, 59] e podem ser usadas nos próximos experimentos. Com relação aos novos algoritmos de treinamento, pretende-se confrontar os resultados do enfoque construtivo (AGC), com outros não-construtivos.

Apesar de existirem vários enfoques não evolutivos que conseguem bons resultados na maioria dos problemas SAT, devido também a possibilidade de se utilizar múltiplas heurísticas no treinamento populacional, esse campo pode ser promissor em termos de quantidade de experimentos que se pode realizar e de conclusões que podem ser tiradas. Por esses motivos, um algoritmo para treinamento populacional em uma ou mais heurísticas, será implementado e aplicado em instâncias grandes de MOSP's, GMLP's e problemas SAT.

## 5.3 Problemas Inversos e Redes Neurais

Uma vez que se tenha um algoritmo para treinamento populacional competitivo para problemas de otimização numérica, pretende-se aplicá-lo em recuperação de condições iniciais e em treinamento de redes neurais supervisionadas.

Dentre os vários problemas encontrados na literatura, o problema de difusão de calor em fractais bidimensionais, foi recentemente estudado e foi proposta uma metodologia para solução de sua equação de difusão [60]. A partir da implementação de seu modelo direto, pode-se desenvolver uma aplicação evolutiva, baseada em treinamento populacional, para a solução do problema inverso.

Um outro problema que pode ser resolvido com base na mesma aplicação é o de treinamento de redes RBF. Esse tipo de rede *feed-forward* apresenta algumas características que a tornam particularmente interessante para aplicação da teoria da regularização (vide Capítulo 2).

Todos os parâmetros livres de uma rede RBF, inclusive coeficientes de regularização e centros de função de base radial, podem ser codificados como soluções candidatas que evoluem de forma competitiva ou cooperativa [32].

A aplicação do treinamento populacional no aprendizado de redes RBF's deverá ser uma das etapas finais do trabalho de doutorado, em virtude de toda teoria necessária para que se obtenha bons resultados.

## 5.4 Cronograma de Atividades

As atividades estão divididas em específicas de cada aplicação e gerais. As atividades específicas estão relacionadas com as 5 principais aplicações: aplicação para MOSP's, otimização de funções numéricas, problemas inversos, treinamento de redes neurais, e problemas SAT. A revisão bibliográfica e a redação são atividades gerais e praticamente contínuas ao longo de todo o período de trabalho.

O cronograma geral está dividido em 3 etapas referentes aos períodos civis durante os quais os trabalhos deverão transcorrer: II semestre de 2002; I e II semestres de 2003; e I semestre de 2004. O cronograma mostrado a seguir apresenta cada período e suas respectivas atividades.

**Tabela 5.1:** Cronograma de atividades de Julho a Dezembro de 2002

| | 2002 | | | | | |
|---|---|---|---|---|---|---|
| Meses / Atividades | Jul | Ago | Set | Out | Nov | Dez |
| MOSP | | | | | | |
| " Múltiplas heurísticas | ■ | ■ | | | | |
| " Análise de resultados | | ■ | ■ | | | |
| Otimização Numérica | | | | | | |
| " Melhoramentos | | | ■ | ■ | ■ | |
| " Novas heurísticas | | | | ■ | ■ | |
| " Outras funções-teste | | | | | ■ | ■ |
| " Análise de resultados | | | | | ■ | ■ |
| Revisão bibliográfica | ■ | | ■ | ■ | | |
| Redação | | | | ■ | ■ | ■ |

**Tabela 5.2:** Cronograma de atividades de Janeiro a Dezembro de 2003

| | 2003 | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Meses / Atividades | Jan | Fev | Mar | Abr | Mai | Jun | Jul | Ago | Set | Out | Nov | Dez |
| Problemas inversos | | | | | | | | | | | | |
| " Desenvolvimento | | ■ | ■ | ■ | | | | | | | | |
| " Experimentos | | | ■ | ■ | ■ | | | | | | | |
| " Análise de resultados | | | | ■ | ■ | ■ | | | | | | |
| Redes Neurais | | | | | | | | | | | | |
| " Desenvolvimento | | | | | | | ■ | ■ | ■ | | | |
| " Experimentos | | | | | | | | ■ | ■ | ■ | | |
| " Análise de resultados | | | | | | | | | ■ | ■ | ■ | |
| Revisão bibliográfica | ■ | ■ | | | | | ■ | ■ | | | | |
| Redação | | | | ■ | ■ | ■ | ■ | | ■ | ■ | ■ | ■ |

**Tabela 5.3:** Cronograma de atividades de Janeiro a Junho de 2004

| | 2004 | | | | | |
|---|---|---|---|---|---|---|
| Meses / Atividades | Jan | Fev | Mar | Abr | Mai | Jun |
| Problemas SAT | | | | | | |
| ” Múltiplas heurísticas | ■ | ■ | | | | |
| ” Análise de resultados | | ■ | ■ | | | |
| Revisão bibliográfica | ■ | ■ | ■ | | | |
| Redação | | ■ | ■ | ■ | ■ | |
| Defesa | | | | | | ■ |

Capítulo

# 6

# Conclusão

Esta proposta de trabalho apresentou os primeiros resultados do Treinamento Populacional em Heurísticas e as próximas atividades a serem desenvolvidas. Inicialmente, foi feita uma exposição de outros enfoques evolutivos aplicados a otimização numérica em áreas como problemas inversos e redes neurais. Também foram discutidas as estratégias mais comuns para utilização de otimizadores locais para ganho de desempenho em algoritmos evolutivos.

O Treinamento Populacional em Heurísticas é uma nova forma de se direcionar a evolução através da utilização sistematicamente informação sobre o ambiente (ou problema) a ser assimilado, possibilitando uma eficiente exploração do espaço de busca.

Pelo enfoque proposto, os indivíduos são ordenados, considerando sua adaptação a heurísticas específicas. Indivíduos bem-treinados (ou bem-adaptados ao treinamento) servem de base para as operações de cruzamento e mutação que irão promover a exploração do espaço de busca. Ao mesmo tempo, em que uma população dinâmica, controlada por um limiar de rejeição adaptativo, permite que esses indivíduos participem do processo evolutivo o máximo de tempo possível.

Algumas aplicações que utilizam o enfoque do treinamento populacional foram desenvolvidas para alguns tipos problemas de otimização combinatória e otimização numérica. O enfoque construtivo, que opera com esquemas, foi modelado para problemas de permutação relacionados com leiaute de matrizes (MOSP e GMLP) e para problemas SAT. O conceito de esquemas não pôde ser empregado em otimização numérica e, por isso, uma versão não-construtiva do treinamento populacional foi desenvolvida e aplicada em funções-teste irrestritas.

Em vista dos bons resultados obtidos até o momento, foi proposto a extensão dos trabalhos a outras áreas, assim como novos experimentos nas aplicações já desenvolvidas, motivados pela possibilidade de utilização de diferentes heurísticas em uma mesma população. Espera-se, ao final das próximas etapas, contribuir efetivamente em diversas áreas do conhecimento relacionadas a otimização.

# Referências Bibliográficas


[1] Rechenberg, I. Evolutionstrategie: Optimierung technisher Systeme nach Prinzipien der biologischen Evolution. *Fromman-Holzboog Verlag*, Sttutgart. 1973.

[2] Fogel, L.J.; Owens, A.J.; Walsh, M.J; Artificial Intelligence Through Simulated Evolution *John Wiley*, New York. 1966.

[3] Koza, J.R. Genetic Programming Bradford Book, *The MIT Press*. 1992.

[4] Holland, J.H. Adaptation in Natural and Artificial Systems *University of Michigan Press*, Ann Arbor, 1975.

[5] Yen, J.; Lee, B. A Simplex Genetic Algorithm Hybrid, *In Proc. of the 1997 IEEE International Conference on Evolutionary Computation (ICEC97)*, pp. 175-180, 1997.

[6] Nelder, J.A.; Mead, R. A simplex method for function minimization. *Computer Journal*. 7(23), p. 308-313. 1965.

[7] Birru, H.K.; Chellapilla, K.; Rao, S.S. Local search operators in fast evolutionary programming. *Congress on Evolutionary Computation*,2, pp.1506-1513. 1999.

[8] Glover, F. A Template for Scatter Search and path Relinking. *Artificial Evolution, Lecture Notes in Computer Sciense*, Springer-Verlag, pp 13-54. 1998.

[9] Michalewicz, Z. Genetic Algorithms + Data Structures = Evolution Programs. *Springer-Verlag*, New York. 1996.

[10] Laguna, M.; Martí, R. Experimental Testing of Advanced Scatter Search Designs for Global Optimization of Multimodal Functions. 2000.

[11] Maaranen H.; Miettinen, K.; Mäkelä, M. M. Training MLP network using scatter search as a global optimizer, *IV Metaheuristics International Conference (MIC2001)*, pp. 725-772, Porto, Portugal. 2001.

[12] Hadamard, J. *Lectures on the Cauchy problem in linear partial differential equations*, Yale Universtity Press, New Haven, 1923.

[13] Phillips, D.L. A technique for the numerical solution of certain integral equations of the first kind. *J. Assoc. Comp. Mech.*, 9, 84-97. 1962.

[14] Tikhonov, A.N. Solution of incorrectly formulated problems and the regularization method, *Soviet Math. Dokl.*, 4, pp. 1035-1038, 1963.

[15] Twomey, S. On the numerical solution of Fredholm integral equations of the first kind by the inversion of the linear system produced by quadrature. *J. Assoc. Comp. Mech.*, 10, 97-101. 1963.

[16] Tikhonov, A.; Arsenin, V. Solutions of ill-posed problems. *Winston and Sons*. 1977.

[17] Jaynes E., *Information theory and statistical mechanics*, Phys.Rev., vol 106, pp 620-630, 1957.

[18] Shannon C.; Weaver W. The mathematical theory of communication, *University of Illinois Press*, 1949.

[19] Tsallis, C. *J. Stat. Phys.* 52, 479. 1988.

[20] Louis, S.J.; Chen, Q. Seismic Velocity Inversion with Genetic Algorithms. *In Proc. Congress on Evolutionary Computation*. 1999.

[21] Sofyan, E.; Trivailo, P. Solving Aerodynamic Load Inverse Problems Using a Hybrid FEM - Artificial Intelligence. - 3rd Australasian Matlab Users Conference [organised by CEANET and MATHWORKS], Crown Towers, Melbourne, Australia, 09-10 November, 8 pp. 2000.

[22] Navarro, P.L.K.G.; Oliveira, P.P.B.; Ramos, F.M.; Velho, H.F.C. An evolutionary approach in magnetotelluric inversion, *in Proc. of the 3rd International Conference on Inverse Problems in Engineering (CD-ROM)*, code IM06, Port Ludlow, USA. 1999

[23] Michalewicz, Z. A Survey of Constraint Handling Techniques in Evolutionary Computation Methods, *Proceedings of the 4th Annual Conference on Evolutionary Programming*, MIT Press, Cambridge, MA, pp. 135-155. 1995.

[24] Haykin, S. Neural networks, *Macmillan College Publishing Company*, 1994.

[25] Grossberg, S. How does the brain build a cognitive code. *Psycologial Review*, 87 pp. 1-51. 1980.

[26] Kohonen T. The self-organizing map, *Springer Verlag*, 1995.

[27] Broomhead D.S.; Lowe D. Multivariable function interpolation and adaptive networks. *Complex Systems*,2, pp. 321-355. 1988.

[28] Rosenblatt, F. The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *In Psychological Review*, Volume 65, pp. 386-408. 1958.

[29] Werbos, P. Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. PhD thesis, Harvard, Cambridge, MA, 1974.

[30] Orr, M. Introduction to Radial Basis Function Networks,available from http://www.anc.ed.ac.uk/ mjo/rbf.html.

[31] Yao, X. Evolving artificial neural networks. *in Proc of the IEEE*, 87(9):1423-1447, 1999.

[32] Topchy, A.P.; Lebedko, O.A.; Miagkikh, K.N. An Approach to Radial Basis Function Networks Training based on Cooperative Evolution and Evolutionary Programming, *in Proc. Int. Conf. on Neural Information Processing (ICONIP'97)*, New Zealand, pp. 253-258. 1997.

[33] Lorena, L.A.N.; Furtado J.C. Constructive genetic algorithm for clustering problems. *Evolutionary Computation*. 9(3): 309-327. 2001.

[34] Ribeiro Filho, G.; Lorena, L.A.N. A Constructive Evolutionary Approach to School Timetabling, In Applications of Evolutionary Computing, Boers, E.J.W., Gottlieb, J., Lanzi, P.L., Smith, R.E., Cagnoni, S., Hart, E., Raidl, G.R., Tijink, H., (Eds.) - *Springer Lecture Notes in Computer Science* vol. 2037, pp. 130-139. 2001

[35] Oliveira A.C.M.; Lorena L.A.N. A Constructive Genetic Algorithm for Gate Matrix Layout Problems. *Accepted to IEEE Transaction on Computer-Aided Designed of Integrated Circuits and Systems*. 2002.

[36] Goldberg, D.E.; Korb, B.; Deb, K. Messy genetic algorithms: motivation, analysis, and first results, *Complex Systems* v. 3: p. 493-530, 1989.

[37] Goldberg,D.E.; Deb, K.; Kargupta, H.; Harik, G. Rapid, accurate optimization of difficult problems using fast messy genetic algorithms, *IlliGAL Report No. 93004*, Illinois Genetic Algorithms Laboratory, Department of General Engineering, University of Illinois, Urbana, 1993.

[38] Kargupta, H. Search, polynomial complexity, and the fast messy genetic algorithm, Ph.D. thesis, *IlliGAL Report No. 95008*, Illinois Genetic Algorithms Laboratory, Department of General Engineering, University of Illinois, Urbana, 1995.

[39] Eshelman, L.J.; Schawer, J.D. Real-coded genetic algorithms and interval- schemata, *in Foundation of Genetic Algorithms-2*, L. Darrell Whitley (Eds.), Morgan Kaufmann Publishers: San Mateo, pp. 187-202. 1993.

[40] Deb, K.; Beyer, H.G. Self-Adaptive Genetic Algorithms with Simulated Binary Crossover. *Evolutionary Computation Journal*, 9 (2), 197-221. 2001

[41] Möhring, R. Graph problems related to gate matrix layout and PLA folding, *Computing*, Vol 7, pp. 17-51, 1990.

[42] Kashiwabara, T.; Fujisawa, T. NP-Completeness of the problem of finding a minimum clique number interval graph containing a given graph as a subgraph, *In Proc. Symposium of Circuits and Systems*. 1979.

[43] Golumbic, M. Algorithmic Graph Theory and Perfect Graphs. *Academic Press*, New York. 1980.

[44] Linhares, A. Industrial Pattern Sequencing Problems: Some Complexity Results And New Local Search Models. Doctoral Thesis, Instituto Nacional de Pesquisas Espaciais (INPE), São José dos Campos, Brazil, 2002.

[45] Becceneri, J.C. O problema de sequenciamento de padrões para minimização do número máximo de pilhas abertas em ambientes de corte industriais. Doctoral Thesis, Instituto Tecnológico de Aeronáutica, São José dos Campos, Brazil, 1999.

[46] Yanasse, H.H. Minimization of open orders-polynomial algorithms for some special cases. *Pesquisa Operacional*, v.16, p.1-26, 1996.

[47] Linhares, A.; Yanasse, H.; Torreão, J.R.A. Linear gate assignment: a fast statistical mechanics approach, *IEEE Transactions on Computer-Aided Designed of Integrated Circuits and Systems*. Vol. 18(12), pp. 1750-1758. 1999.

[48] Oliveira A.C.M.; Lorena L.A.N. 2-Opt Population Training for Minimization of Open Stack Problem. *Submitted to SBIA2002*. 2002.

[49] Garey, M.R.; Johnson, D.S. Computers and Intractability: A Guide to the Theory of NP-Completeness. *W.H. Freeman and Company*, San Francisco, 1979.

[50] Cook, S.A. The complexity of theorem-provingprocedures. *in Proc. of the Third ACM Symposium on Theory of Computing*, pages pp. 151-158, 1971.

[51] Gent, I.P. On the stupid algorithm for satisfiability. Technical report, Technical report APES-03-1998, available from http://www.cs.strath.ac.uk/ apes/reports/apes-03-1998.ps.gz. 1998.

[52] Selman, B.; Kautz, H.; Cohen, B. Noise Strategies for Improving Local Search. *in Proc. of the 12th National Conference on AI*, pages pp. 337-343. American Association for Artificial Intelligence, 1994.

[53] Koutsoupias E.; Papadimitriou, C.H. On the greedy algorithm for satisfability. *Information Processing Letters*, 43:53-55, 1992.

[54] Rossi, C.; Marchiori, E.; Kok, J. An Adaptive Evolutionary Algorithm for the Satisfiability Problem. *In Proceedings of the 14th Annual Symposium on Applied Computing(SAC 2000)*, pp. 463-469, 2000.

[55] Bersini, H.; Dorigo, M.; Langerman, S.; Seront G.; Gambardella, L.M. Results of the first international contest on evolutionary optimisation (1st ICEO), *in Proc. IEEE International Conference on Evolutionary Computation*, IEEE-EC96,IEEE Press, pp. 611-615. 1996.

[56] Digalakis, J.; Margaritis, K. An experimental study of benchmarking functions for Genetic Algorithms. *IEEE Systems Transactions*, p.3810-3815, 2000.

[57] Oliveira A.C.M.; Lorena L.A.N. Real-Coded Evolutionary Approaches to Unconstrained Numerical Optimization. *Submitted to LAPTEC2002*. 2002.

[58] Fraggioli, E.; Bentivoglio, C.A. Heuristic and exact methods for the cutting sequencing problem, *European Journal of Operational Research*, 110, pp. 564-575. 1998.

[59] Yuen, B.J; Richardson, K.V. Establishing the optimality of sequencing heuristics for cutting stock problems. *European Journal of Operational Research*, 84, 590-598, 1995.

[60] Ramos, F.M. Heat diffusion in two-dimensional fractals, *Journal of the Brazilian Society of Mechanical Sciences*, vol. 21, pp.133-143, 1999.