Universidade Estadual de Campinas
Instituto de Matemática Estatística e Computação Científica
Departamento de Matemática Aplicada

Algoritmo duas fases em otimização global

Gabriel Haeser
Mestrado em Matemática Aplicada

Orientadora: **Prof.ª Dr.ª Márcia A. Gomes Ruggiero**

Trabalho financiado pela FAPESP

Campinas
Março de 2006

Algoritmo duas fases em Otimização Global

Este exemplar corresponde à redação final da tese devidamente corrigida e defendida por **Gabriel Haeser** e aprovada pela comissão julgadora.

Campinas, 9 de março de 2006.

Prof.ª Dr.ª **Márcia A. Gomes Ruggiero**
Orientadora

Banca examinadora:
Prof.ª Dr.ª Márcia A. Gomes Ruggiero (IMECC/UNICAMP)
Prof. Dr. José Mario Martínez (IMECC/UNICAMP)
Prof. Dr. Ernesto Julián Goldberg Birgin (IME/USP)

Dissertação apresentada ao **Instituto de Matemática Estatística e Computação Científica, UNICAMP**, como requisito parcial para a obtenção do título de **Mestre em Matemática Aplicada**.

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP
Bibliotecária: Maria Júlia Milani Rodrigues – CRB8a / 2116

Haeser, Gabriel
H119a Algoritmo duas fases em otimização global / Haeser Gabriel -- Campinas, [S.P. :s.n.], 2006.

Orientadora : Márcia Aparecida Gomes Ruggiero
Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Matemática, Estatística e Computação Científica.

1. Otimização. 2. Simulated annealing (Matemática). 3. Algoritmos genéticos. I. Ruggiero, Márcia Aparecida Gomes. II. Universidade Estadual de Campinas. Instituto de Matemática, Estatistica e Computação Científica. III. Título.

Título em inglês: Two-phase algorithm for global optimization

Palavras-chave em inglês (Keywords): 1. Optimization. 2. Simulated annealing (Mathematics). 3. Genetic algorithms.

Área de concentração: Otimização

Titulação: Mestre em Matemática Aplicada

Banca examinadora: Profa. Dra. Márcia Aparecida Gomes Ruggiero (IMECC-UNICAMP)
Prof. Dr. José Mario Martínez (IMECC-UNICAMP)
Prof. Dr. Ernesto Julián Goldberg. Birgin (IME-USP)

Data da defesa: 09/03/2006

Dissertação de Mestrado defendida em 09 de março de 2006 e aprovada

Pela Banca Examinadora composta pelos Profs. Drs.

Prof. (a). Dr (a). MÁRCIA APARECIDA GOMES RUGGIERO

Prof. (a). Dr (a). JOSÉ MARIO MARTINEZ PEREZ

Prof. (a). Dr (a). ERNESTO JULIÁN GOLDBERG BIRGIN

à Etiene.

Agradecimentos

À FAPESP e CNPq.

À Etiene por diversos motivos.

Ao Eloi, Neuza, Lucas e Karina.

À prof. Márcia pela orientação e amizade desde a graduação.

Ao Ricardo e Moiseis pela companhia, amizade, discussões e opiniões que contribuiram neste trabalho e na minha formação.

À você, por ter lido pelo menos até aqui (a não ser que você já tenha sido citado anteriormente).

Resumo

Neste trabalho estudamos a teoria de algumas heurísticas para otimização global, e também a generalização do algoritmo genético de Aarts, Eiben e van Hee. Propomos um algoritmo para otimização global de problemas canalizados e diferenciáveis utilizando *simulated annealing* e o solver local GENCAN. Experimentos numéricos com o problema OVO (*Order-Value Optimization*) são apresentados, e também com 28 problemas clássicos da literatura. Para problemas de otimização com restrições, apontamos idéias de como utilizar solvers locais e heurísticas globais em busca de bons algoritmos para otimização global, e propomos um algoritmo baseado em *simulated annealing* com solver local ALGENCAN.

Palavras chave: otimização global; otimização local; simulated annealing; algoritmo genético

Abstract

In this work we study the theory behind some classical heuristics for global optimization, and a generalization of genetic algorithms from Aarts, Eiben and van Hee. We propose an algorithm for global optimization of box-constrained differentiable problems, using simulated annealing and the local solver GENCAN. Numerical experiments are presented for the OVO problem (Order-Value Optimization) and 28 classical problems. For general nonlinear programming problems, we mention some ideas of how to use local solvers and global heuristics towards good algorithms for global optimization, we also propose an algorithm based on simulated annealing with local solver ALGENCAN.

Key words: global optimization; local optimization; simulated annealing; genetic algorithms

Sumário

Capítulo 1

Introdução

Em pesquisas na literatura especializada sobre aplicações de técnicas em otimização contínua, principalmente na área de Engenharia Química, é possível constatar que um grande número de problemas resulta em modelos não lineares com funções não convexas. As não convexidades levam a múltiplos ótimos locais tornando difícil a tarefa de identificar o ótimo global, que é a solução de interesse nestas aplicações. A dificuldade central da busca pelo ótimo global resulta do fato de que os algoritmos usuais em otimização dependem fortemente do ponto inicial. No caso de convergência, é obtido um ponto estacionário, que pode não ser ótimo local do problema, muito menos ótimo global.

A proposta nesta pesquisa é estudar como usar algoritmos de otimização local em conjunto com heurísticas globais em busca de algoritmos de otimização global para problemas de programação não linear com restrições de caixa e com restrições gerais.

As heurísticas têm por objetivo:

- identificar boas regiões de busca que contenham boas aproximações iniciais para um otimizador local;
- evitar que um mesmo ótimo local seja obtido repetidas vezes pelo otimizador local;
- identificar sub– e super–soluções e incorporar estas informações no processo de resolução;
- gerar sequências que não caiam em armadilhas de ótimos locais e busquem com mais intensidade as direções que conduzem ao ótimo global.

Identificada uma boa aproximação inicial, a investigação será realizada através do solver local GENCAN [12], no caso de problemas canalizados e ALGENCAN [5, 6] no caso de problemas com restrições.

Destacamos as principais contribuições deste trabalho:

- apresentamos uma proposta de um solver global para problemas canalizados, utilizando uma heurística de otimização global e um solver local;
- apontamos idéias de como utilizar uma heurística de otimização global e um solver local no caso de problemas de otimização com restrições, e propomos um algoritmo utilizando simulated annealing e o solver local ALGENCAN.

O trabalho se divide nos seguintes capítulos:

Capítulo 2

Apresentamos uma descrição de algoritmos genéticos e esquemas evolutivos, em seguida descrevemos a teoria para o *simulated annealing* e apresentamos uma generalização de algoritmos genéticos proposta em [1, 18].

Capítulo 3

Apresentamos o método DBMS (*Darwin and Boltzmann Mixed Strategy*) para problemas canalizados, proposto em [26] que envolve *simulated annealing* e uma busca local, e propomos SA–GENCAN, onde utilizamos o solver local GENCAN juntamente com *simulated annealing*. O desempenho computacional deste algoritmo é analisado através da resolução de 28 problemas clássicos. Apresentamos também o desempenho computacional das heurísticas do Capítulo 2 para o problema OVO (*Order-Value Optimization*) descrito em [7, 8, 9, 10].

Capítulo 4

Utilizamos o solver local ALGENCAN, [5, 6] juntamente com *simulated annealing* para propormos o SA–ALGENCAN. Experimentos numéricos são apresentados através da resolução de 12 problemas clássicos.

Capítulo 5

Realizamos uma análise dos resultados obtidos.

Capítulo 2

Heurísticas para otimização global

Há uma grande variedade de problemas em otimização para os quais nenhum algoritmo eficiente foi desenvolvido que consiga obter o ótimo global, mas em muitos casos é possível aplicar técnicas eficientes que retornam respostas satisfatórias. Estas técnicas são denominadas heurísticas. A otimalidade não é garantida, mas a execução por um tempo suficientemente grande, garante que o erro cometido pode ser feito tão pequeno quanto se queira. Neste capítulo apresentamos os aspectos teóricos de algumas heurísticas clássicas.

Os algoritmos em otimização global [30] podem ser classificados em técnicas estocásticas e determinísticas. Métodos estocásticos, apresentam a vantagem de não fazer qualquer exigência sobre as funções que definem o problema e requerem heurísticas em seus procedimentos de busca. Entre os métodos estocásticos destacam–se métodos de busca aleatória, clustering e as metaheurísticas: simulated annealing [13], [34], algoritmos genéticos e tabu search [16], [22] e [23]. As técnicas determinísticas procuram tirar proveito da estrutura do problema, das particularidades das funções envolvidas e em geral garantem convergência finita a um nível pré–estabelecido de precisão. Entre os processos determinísticos podemos citar os métodos do tipo branch-and-bound e algoritmos de decomposição, [19], [31].

2.1 Algoritmo Genético

Apresentamos o algoritmo genético aplicado na resolução de problemas do tipo:

$$\begin{aligned} &\max_{x} f(x) \\ &\text{s.a.} \quad l \leq x \leq u. \end{aligned}$$

Onde $f\colon \mathbb{R}^n \to \mathbb{R}$ é contínua e $f(\cdot) > 0$.

Note que um problema geral canalizado pode ser transformado para o tipo descrito acima, basta trocar $f(x)$ por $-f(x)$ no caso de problemas de minimização, e/ou somar $M > 0$, suficientemente grande, à função objetivo, de modo que valha $f(x) > 0 \quad \forall\, x \in \mathbb{R}^n$.

A idéia é representar cada candidato à solução como uma seqüência de zeros e uns e efetuar operações de cruzamento e mutação com o intuito de evoluir uma população de candidatos em direção a um maximizador global. Abaixo esquematizamos estas idéias:

1. `Inicializar população.`
2. `Se "critério de parada" então terminar execução.`
3. `Realizar cruzamento.`
4. `Realizar mutação.`
5. `Selecionar os melhores candidatos.`
6. `Voltar para 2.`

Representação

Escolhemos h_i, o tamanho da discretização do espaço de busca na i–ésima coordenada, para $i = 1, \ldots, n$. Em seguida encontramos m_i, o menor inteiro que satisfaz

$$\frac{u_i - l_i}{h_i} \leq 2^{m_i} - 1,$$

assim podemos representar os números $l_i + k\dfrac{u_i - l_i}{2^{m_i} - 1}$ para $k = 0, 1, \ldots, 2^{m_i} - 1$ através de seqüências binárias v_i de m_i dígitos, utilizando a conversão

$$x_i = l_i + \texttt{bin2dec}(v_i)\frac{u_i - l_i}{2^{m_i} - 1},$$

onde `bin2dec` é uma função que transforma um número binário em seu correspondente em decimal.

Cada candidato à solução é representado por uma seqüência binária de tamanho $m = \sum_{i=1}^{n} m_i$, formada pela concatenação dos vetores binários que representam cada componente. Cada seqüência deste tipo, que representa um ponto do espaço de busca, é chamada de cromossomo ou indivíduo, enquanto um conjunto de cromossomos é chamado de população.

Inicialização

Escolhemos o tamanho da população, denotado por $n_{\texttt{pop}}$, que se manterá constante ao longo da execução do algoritmo. Em seguida geramos aleatoriamente $n_{\texttt{pop}}$ seqüências binárias de tamanho m, que irão compor a população inicial. Se possuirmos algum conhecimento sobre a distribuição dos maximizadores, poderemos utilizar tal informação para gerar de maneira mais eficiente a população inicial.

Cruzamento

Antes de iniciarmos a execução do algoritmo, definimos um de seus parâmetros, a probabilidade de cruzamento p_c, que nos fornece um número esperado de $p_c n_{\texttt{pop}}$ cruzamentos.

Para cada cromossomo v_k, $k = 1, \ldots, n_{\texttt{pop}}$, geramos um número aleatório r_k no intervalo $[0, 1]$ e escolhemos v_k para cruzamento se $r_k < p_c$.

O número de cromossomos escolhidos para cruzamento deve ser par, caso contrário excluímos ou incluímos um cromossomo, com igual probabilidade.

Montamos os pares de maneira aleatória. Para cada par, escolhemos aleatoriamente uma posição da seqüência de m dígitos e produzimos dois novos descendentes que substituirão os dois indivíduos na próxima população.

O cruzamento é feito da seguinte maneira: sendo

$$(b_1 b_2 \ldots b_{pos} b_{pos+1} \ldots b_m) \quad \text{e} \quad (c_1 c_2 \ldots c_{pos} c_{pos+1} \ldots c_m)$$

os indivíduos selecionados e *pos* a posição aleatória do cromossomo, os dois novos indivíduos que substituirão os anteriores são:

$$(b_1 b_2 \ldots b_{pos} c_{pos+1} \ldots c_m) \quad \text{e} \quad (c_1 c_2 \ldots c_{pos} b_{pos+1} \ldots b_m)$$

Mutação

Após a mudança da população através dos cruzamentos, aplicamos um processo de mutação nos cromossomos: para cada um dos m dígitos de cada um dos $n_{\texttt{pop}}$ cromossomos, geramos um número aleatório r. Se $r < p_m$, trocamos o dígito correspondente, onde p_m é a probabilidade de mutação, outro parâmetro do algoritmo.

Seleção

Calculamos o valor da função objetivo f_k de cada cromossomo v_k, para $k = 1, \ldots, n_{\texttt{pop}}$, avaliando a função f no ponto de $\mathbb{R}^n$ que v_k representa. Em seguida calculamos a probabilidade de seleção p_k de cada cromossomo:

$$p_k = \frac{f_k}{\sum_{i=1}^{n_{\texttt{pop}}} f_i},$$

note aqui a necessidade de f ser positiva. Em seguida calculamos a probabilidade acumulada q_k para cada cromossomo v_k:

$$q_k = \sum_{i=1}^{k} p_i.$$

Geramos um número aleatório r, no intervalo $[0, 1]$, e escolhemos v_1 se $r \leq q_1$ senão escolhemos v_j tal que $q_{j-1} < r \leq q_j$. Repetimos o processo $n_{\texttt{pop}}$ vezes de modo a obtermos uma nova população de tamanho $n_{\texttt{pop}}$, com possivelmente indivíduos duplicados. Este processo está baseado na idéia de seleção natural de Charles Darwin, onde os melhores indivíduos têm maiores probabilidades de sobreviver.

Um desenvolvimento mais detalhado do algoritmo genético pode ser encontrado em [28].

Exemplo

A idéia do algoritmo genético pode ser aplicada em outros contextos, para exemplificar esta característica, apresentamos uma aplicação extraída de [28].

O dilema do prisioneiro é um jogo para duas pessoas, onde a cada rodada os jogadores decidem simultaneamente se vão trair ou cooperar. Os pontos para cada possível resultado são mostrados na Tabela 2.1 onde o primeiro elemento corresponde à linha e o segundo à coluna.

Tabela 2.1: Dilema do Prisioneiro

	Trair	Cooperar
Trair	$(1, 1)$	$(5, 0)$
Cooperar	$(0, 5)$	$(3, 3)$

Considere o problema de encontrar uma estratégia ótima para o dilema do prisioneiro. Vamos considerar apenas estratégias baseadas nas três últimas jogadas. Uma estratégia pode ser vista como um vetor de $4^3 = 64$ posições, onde cada posição indica o que deve ser feito na próxima rodada (T representa trair e C cooperar) dependendo do resultado das três rodadas anteriores. O algoritmo segue de acordo com o esquema abaixo.

- Criar aleatoriamente a população inicial, n_{pop} vetores de 64 posições formados por T's e C's.
- Simular jogos entre as estratégias. A eficácia é dada pela pontuação média de cada estratégia.
- Selecionar as estratégias de acordo com as probabilidades de seleção.
- Criar novas estratégias por cruzamento e mutação.

De acordo com [28], alguns padrões interessantes aparecem dentre as melhores estratégias:

- C depois de $(CC)(CC)(CC)$ "Manter cooperação"
- T depois de $(CC)(CC)(CT)$ "Revidar"
- C depois de $(TC)(CT)(CC)$ "Aceitar trégua"
- C depois de $(TC)(CC)(CC)$ "Esquecer ganho anterior"
- T depois de $(TT)(TT)(TT)$ "Manter traição"

Onde (CT) indica que nesta rodada o primeiro jogador cooperou e o segundo traiu.

2.2 Esquemas Evolutivos

A idéia dos esquemas evolutivos é basicamente a mesma do algoritmo genético, porém evitando a representação binária. Para isto devemos dar um novo sentido para mutação e cruzamento.

Representação

Cada indivíduo em uma população é um par de vetores, isto é, $v = (x, \sigma)$, onde x é um ponto no espaço de busca e σ é um vetor com os desvios padrões para cada componente de x. O vetor σ é utilizado na mutação, onde a cada componente x_i de x, é somada uma perturbação normal de média zero e desvio padrão relacionado com σ_i, a i–ésima componente do vetor σ.

Inicialização

Inicializamos a população inicial com $n_{\texttt{pop}}$ vetores, de maneira aleatória, juntamente com seus respectivos desvios padrões.

Cruzamento

Fixamos o inteiro positivo $\mu \leq n_{\texttt{pop}}/2$ de novos elementos que gostaríamos de gerar. Selecionamos aleatoriamente 2μ indivíduos da população e montamos os μ casais, também de maneira aleatória. Sendo

$$(x^1, \sigma^1) = ((x_1^1, \ldots, x_n^1), (\sigma_1^1, \ldots, \sigma_n^1)) \quad \text{e} \quad (x^2, \sigma^2) = ((x_1^2, \ldots, x_n^2), (\sigma_1^2, \ldots, \sigma_n^2))$$

um casal selecionado. Há dois tipos de cruzamentos:

Discreto: o novo indivíduo é

$$(x, \sigma) = ((x_1^{q_1}, \ldots, x_n^{q_n}), (\sigma_1^{q_1}, \ldots, \sigma_n^{q_n})),$$

onde $q_i = 1$ ou $q_i = 2$ com igual probabilidade, para $i = 1, \ldots, n$.

Intermediário: o novo indivíduo é

$$(x^1, \sigma^1) = (((x_1^1 + x_1^2)/2, \ldots, (x_n^1 + x_n^2)/2), ((\sigma_1^1 + \sigma_1^2)/2, \ldots, (\sigma_n^1 + \sigma_n^2)/2))$$

Diferentemente do algoritmo genético, os novos indivíduos produzidos não substituem os anteriores. Estes são acrescentados à população e competirão com os demais pela sobrevivência no processo de seleção.

Mutação

Nos esquemas evolutivos, cruzamento e mutação podem ser vistos como partes de uma mesma operação, pois cada indivíduo produzido pelo cruzamento sofre obrigatoriamente a mutação.

A mutação do indivíduo (x, σ) consiste em sua substituição pelo indivíduo (x', σ') dado por:

$$\sigma' = \sigma.e^{N(0,\Delta\sigma)} \quad \text{e} \quad x' = x + N(0, \sigma'),$$

onde $\Delta\sigma > 0$ é um parâmetro do método e $N(0, t)$ é uma variável aleatória de distribuição normal multivariada com vetor de médias nulo e matriz de covariância diagonal, cujos elementos são dados pelo quadrado de cada componente de t. Note que como $\Delta\sigma \in \mathbb{R}$, $N(0, \Delta\sigma)$ se reduz à normal univariada de média zero e desvio–padrão $\Delta\sigma$. A idéia por trás deste procedimento vem da natureza, onde pequenas mudanças ocorrem com maior probabilidade do que grandes mudanças.

Seleção

Dos μ novos indivíduos produzidos, descartamos os que não satisfazem as restrições. Dentre os restantes e a população anterior escolhemos os $n_{\mathtt{pop}}$ melhores, de acordo com o valor da função objetivo.

O algoritmo segue repetindo estes passos com a nova população construída, até a satisfação de algum critério de parada conveniente.

Um desenvolvimento mais detalhado de esquemas evolutivos pode ser encontrado em [11, 28, 33].

2.3 Simulated Annealing

Annealing é o processo utilizado para fundir um metal, onde este é aquecido a uma temperatura elevada e em seguida é resfriado lentamente, de modo que o produto final seja uma massa homogênea.

O *simulated annealing* surgiu no contexto da mecânica estatística, desenvolvido por Kirkpatrick, Gelatt e Vecchi em 1983 [24] e independentemente por Černy em 1985 [15], utilizando o algoritmo de Metropolis de 1953 [27].

Esta técnica é utilizada em problemas de otimização combinatória:

$$\begin{aligned} &\min_x f(x) \\ &\text{s.a.} \quad x \in S \end{aligned}$$

Onde $f\colon S \to \mathbb{R}$, S finito.

O desenvolvimento a seguir é baseado em [2, 36, 25].

O algoritmo consiste em inicializar uma temperatura inicial T e um ponto inicial $i \in S$. A partir de i geramos $j \in S$, isto é, geramos j na vizinhança de i e aceitamos ou rejeitamos j de acordo com o critério de Metropolis (que depende de T). Repetimos este processo L_k vezes, de modo que um certo equilíbrio é atingido, em seguida diminuímos o valor de T e repetimos o processo. De forma esquemática temos:

`INICIALIZAR` i, T_0, L_0
$k := 0$
`REPETIR`

PARA $l = 1$ ATÉ L_k
GERAR j de $V(i)$
SE $f(j) \leq f(i)$ ENTÃO $i := j$
SENÃO SE random$[0,1) < \exp\left(\frac{f(i) - f(j)}{T_k}\right)$
ENTÃO $i := j$
$k := k + 1$
CALCULAR L_k e T_k
ATÉ ''critério de parada''

A função vizinhança V é tal que $V : S \to \mathbb{P}(S), i \mapsto V(i)$. Dizemos que $V(i)$ é a vizinhança de i e se $j \in V(i)$, dizemos que j é vizinho de i. Além disso, assumimos que $j \in V(i) \Leftrightarrow i \in V(j)$.

A idéia do algoritmo é inicialmente aceitar quase todas as transições propostas, a fim de escapar de um minimizador local (para isso, T_0 deve ser suficientemente grande) e em seguida aceitar cada vez menos pontos que pioram o valor da função objetivo, sendo que no limite $T \to 0$, só aceitamos pontos que melhoram o valor da função objetivo. Na Figura 2.1, temos a probabilidade de aceitarmos j a partir de i, dado que $f(j) > f(i)$, onde $\Delta f = f(j) - f(i)$ é fixo. Vemos que a probabilidade tende a 1 conforme $T \to +\infty$ e tende a 0 conforme $T \to 0$

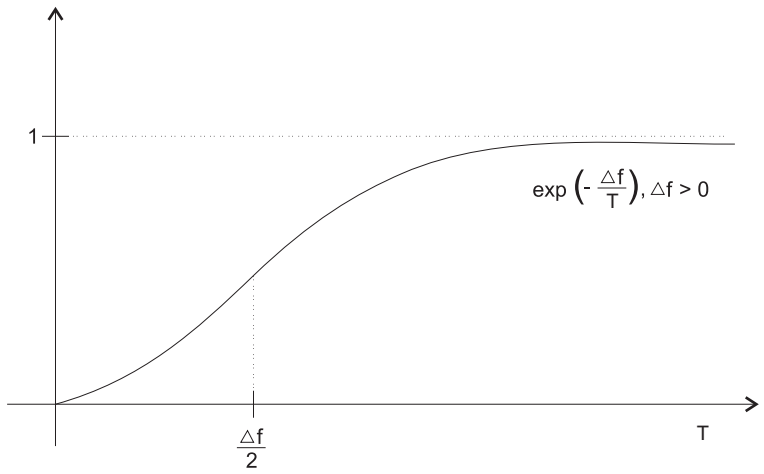

Figura 2.1: Gráfico de $\exp(-\frac{\Delta f}{T}), \Delta f > 0$ para T positivo. $\frac{\Delta f}{2}$ é o ponto que anula a segunda derivada.

Para descrevermos a teoria por trás deste algoritmo e demonstrarmos a sua convergência, precisamos entender o que são cadeias de Markov e o funcionamento do algoritmo de Metropolis.

2.3.1 Cadeias de Markov

Definição 2.3.1 *Uma cadeia de Markov é uma seqüência de variáveis aleatórias X_k que assume valores em S tal que:*

$$\mathbf{P}\{X_k = i_k | X_{k-1} = i_{k-1}, \ldots, X_0 = i_0\} = \mathbf{P}\{X_k = i_k | X_{k-1} = i_{k-1}\},$$

além disso, a cadeia é dita finita se S é finito.

Ou seja, em uma cadeia de Markov, a probabilidade de mudarmos de um estado i para um estado j não depende dos estados anteriores.

Assim, em uma cadeia de Markov finita, podemos definir para cada k, a *matriz de transição* $P^{(k)}$, onde

$$P_{ij}^{(k)} = \mathbf{P}\{X_k = j | X_{k-1} = i\}.$$

Seja $a_i^{(k)}$ a probabilidade da k–ésima variável aleatória assumir o valor i, ou seja,

$$a_i^{(k)} = \mathbf{P}\{X_k = i\}.$$

Podemos encontrar uma fórmula recursiva para $a_i^{(k)}$ em função de $P^{(k)}$:

$$\begin{aligned} a_i^{(k)} &= \mathbf{P}\{X_k = i\} \\ &= \sum_l \mathbf{P}\{X_k = i, X_{k-1} = l\} \\ &= \sum_l \mathbf{P}\{X_k = i | X_{k-1} = l\}\mathbf{P}\{X_{k-1} = l\} \\ &= \sum_l a_l^{(k-1)} P_{li}^{(k)}. \end{aligned}$$

Temos então que o vetor $a^{(k)} = (a_1^{(k)}, a_2^{(k)}, \ldots)^{\mathtt{T}}$ é dado por:

$$\begin{aligned} a^{(k)\,\mathtt{T}} &= a^{(k-1)\,\mathtt{T}} P^{(k)} \\ &= a^{(0)\,\mathtt{T}} \prod_{l=1}^{k} P^{(l)}, \end{aligned}$$

onde $a^{\mathtt{T}}$ é o transposto de a.

Definição 2.3.2 *Dizemos que a cadeia de Markov é homogênea se $P^{(k)}$ não depende de k, neste caso escrevemos apenas P, para a matriz de transição. Caso contrário dizemos que a cadeia é heterogênea.*

Definição 2.3.3 *Em uma cadeia de Markov finita e homogênea, dizemos que existe a distribuição limite se, para cada $i \in S$ existe o limite*

$$q_i = \lim_{k \to +\infty} \mathbf{P}\{X_k = i | X_0 = j\},$$

além disso q_i independe de j. O vetor q é chamado de distribuição limite da cadeia de Markov.

Se existe a distribuição limite, então temos

$$\begin{aligned}
\lim_{k \to +\infty} a_i^{(k)} &= \lim_{k \to +\infty} \mathbf{P}\{X_k = i\} \\
&= \lim_{k \to +\infty} \sum_j \mathbf{P}\{X_k = i, X_0 = j\} \\
&= \lim_{k \to +\infty} \sum_j \mathbf{P}\{X_k = i | X_0 = j\} \mathbf{P}\{X_0 = j\} \\
&= q_i \sum_j \mathbf{P}\{X_0 = j\} \\
&= q_i
\end{aligned}$$

Logo a distribuição limite é a distribuição de probabilidade de cada $i \in S$ após um número infinito de transições da cadeia de Markov. Além disso,

$$\begin{aligned}
q^{\mathtt{T}} &= \lim_{k \to +\infty} {a^{(0)}}^{\mathtt{T}} \prod_{l=1}^{k} P^{(l)} \\
&= \lim_{k \to +\infty} {a^{(0)}}^{\mathtt{T}} P^k \\
&= \lim_{k \to +\infty} {a^{(0)}}^{\mathtt{T}} P^{k-1} P \\
&= q^{\mathtt{T}} P.
\end{aligned}$$

Ou seja, q é um autovetor de $P^{\mathtt{T}}$ associado ao autovalor 1.

Definição 2.3.4 *Em cadeias de Markov finitas e homogêneas, definimos a probabilidade de transição em n estados por*

$$P_{ij}^n = \mathbf{P}\{X_n = j | X_0 = i\}.$$

A matriz P^n formada pelos elementos P_{ij}^n é chamada matriz de transição em n estados.

Devido à cadeia de Markov ser homogênea, podemos escrever $P_{ij}^n = \mathbf{P}\{X_{m+n} = j | X_m = i\}$, qualquer que seja m.

Observe a diferença de notação entre uma matriz de transição em n estados P^n em uma cadeia de Markov homogênea, e uma matriz de transição $P^{(k)}$ em uma cadeia de Markov heterogênea.

Além disso, para cadeias de Markov homogêneas não há perigo de confusão entre P^n, a matriz de transição em n estados e P^n, a n–ésima potência da matriz de transição P, pois felizmente elas coincidem, conforme vemos no Teorema 2.3.5 abaixo.

Teorema 2.3.5 *Equações de Chapman–Kolmogorov*

$$P_{ij}^n = \sum_{l \in S} P_{il}^r P_{lj}^{n-r} \quad \forall\, 0 < r < n \ e \ \forall\, i, j \in S.$$

Demonstração:

$$\begin{aligned} P_{ij}^n &= \mathbf{P}\{X_n = j | X_0 = i\} \\ &= \sum_{l \in S} \mathbf{P}\{X_n = j, X_r = l | X_0 = i\} \\ &= \sum_{l \in S} \frac{\mathbf{P}\{X_n = j, X_r = l, X_0 = i\}}{\mathbf{P}\{X_r = l, X_0 = i\}} \frac{\mathbf{P}\{X_r = l, X_0 = i\}}{\mathbf{P}\{X_0 = i\}} \\ &= \sum_{l \in S} \mathbf{P}\{X_n = j | X_r = l, X_0 = i\} \mathbf{P}\{X_r = l | X_0 = i\} \\ &= \sum_{l \in S} P_{lj}^{n-r} P_{il}^r. \end{aligned}$$

■

Em notação matricial temos $P^n = P^r P^{n-r}$, para todo $0 < r < n$ inteiro.

Usando este resultado para $r = 1$, provamos facilmente por indução em n que a matriz de transição em n estados P^n, é igual a n–ésima potência da matriz de transição P.

A seguir apresentaremos uma condição suficiente para que q seja a distribuição limite de uma cadeia de Markov, mas antes de enunciarmos o teorema precisamos das seguintes definições:

Definição 2.3.6 *Uma cadeia de Markov finita e homogênea com matriz de transição P é irredutível, se para cada $i, j \in S$ existe uma probabilidade positiva de atingir j a partir de i em um número finito de passos, isto é*

$$\forall\, i, j \in S \quad \exists\, n \geq 1 \quad | \quad P_{ij}^n > 0.$$

Definição 2.3.7 *Uma cadeia de Markov finita e homogênea com matriz de transição P é aperiódica se, para cada $i \in S$, o máximo divisor comum dos inteiros $n > 0$ tal que $P_{ii}^n > 0$ é igual a 1.*

Este máximo divisor comum é chamado de *período* de i. Logo a cadeia é aperiódica se, e só se o período de cada $i \in S$ é igual a 1.

Definição 2.3.8 *Um vetor q é dito estocástico se suas componentes q_i satisfazem as seguintes condições:*

$$q_i \geq 0 \quad \forall\, i, \;\; e \;\; \sum_i q_i = 1.$$

Definição 2.3.9 *Uma matriz P é dita estocástica se suas componentes P_{ij} satisfazem as seguintes condições:*

$$P_{ij} \geq 0 \quad \forall\, i, j, \;\; e \;\; \sum_j P_{ij} = 1.$$

Ou seja, uma matriz é estocástica se suas linhas são vetores estocásticos.

O próximo teorema apresenta uma condição suficiente para a existência da distribuição limite:

Teorema 2.3.10 *Seja P a matriz de transição associada a uma cadeia de Markov finita, homogênea, irredutível e aperiódica. Se um vetor estocástico q satisfaz*

$$q_i P_{ij} = q_j P_{ji}, \quad \forall\, i, j \in S,$$

então q é a distribuição limite desta cadeia de Markov.

A demonstração deste teorema pode ser encontrada em [2], página 38.

A seguir definimos o algoritmo de Metropolis.

2.3.2 Algoritmo de Metropolis

Dado um conjunto finito S e uma distribuição de probabilidade q em S, como construir uma cadeia de Markov em S com distribuição limite q? O algoritmo de Metropolis, responde a esta pergunta.

Além da aplicação deste resultado no *simulated annealing*, o algoritmo de Metropolis pode ser usado para calcular aproximadamente a esperança de variáveis aleatórias. Sendo

$g\colon S \to \mathbb{R}$

$$\begin{aligned} E_q(g(X)) &= \sum_{i\in S} g(i)q_i \\ &\approx \frac{1}{N-m}\sum_{k=m}^{N} g(X_k), \end{aligned}$$

onde $N > m \gg 0$ e X_k é a cadeia de Markov construída pelo algoritmo de Metropolis com distribuição limite q.

A seguir definimos o algoritmo.

Seja $G \in \mathbb{R}^{n\times n}$ uma matriz simétrica e estocástica, onde $n < +\infty$ é a cardinalidade de S. A matriz G é chamada *matriz de geração* e G_{ij} é a probabilidade de gerar j a partir de i.

Seja $\alpha_{ij} = \min\left\{\frac{q_j}{q_i}, 1\right\}$, a *matriz de aceitação* α. α_{ij} é chamado de *critério de Metropolis*.

O algoritmo consiste em: dado $X_k = i$, gerar j a partir de i de acordo com as probabilidades $G_{it}, \quad \forall\, t \in S$. Aceitar $X_{k+1} = j$ com probabilidade α_{ij}, ou rejeitar j, isto é, $X_{k+1} = i$ com probabilidade $1 - \alpha_{ij}$.

Deste modo X_k é uma cadeia de Markov finita e homogênea com

$$P_{ij} = \alpha_{ij}G_{ij}, \text{ se } i \neq j \text{ e } P_{ii} = 1 - \sum_{j\in S, j\neq i} P_{ij}.$$

Note que no algoritmo de Metropolis não é necessário conhecermos o valor da constante normalizadora da distribuição q, pois o algoritmo só depende de q na razão $\frac{q_j}{q_i}$.

Lema 2.3.11 *Uma cadeia de Markov irredutível é aperiódica se*

$$\exists\, j \in S \quad | \quad P_{jj} > 0.$$

Demonstração: Pela irredutibilidade temos que

$$\forall\, i, j \in S, \exists\, k, l \geq 1 \quad | \quad P_{ij}^k > 0 \text{ e } P_{ji}^l > 0.$$

Sendo $m = k + l$, temos pelo Teorema 2.3.5 $P_{ii}^m \geq P_{ij}^k P_{ji}^l > 0$, além disso, aplicando duas vezes o Teorema 2.3.5 obtemos $P_{ii}^{m+1} \geq P_{ij}^k P_{jj} P_{ji}^l > 0$. Como $\text{mdc}(m, m+1) = 1$, temos que a cadeia é aperiódica. ■

Teorema 2.3.12 *Se a distribuição q não é uniforme em S, $q_i > 0 \quad \forall\, i \in S$ e a matriz de geração G_{ij} é tal que*

$$\forall\, i, j \in S \quad \exists\, p \geq 1, \quad \exists\, l_0, l_1, \ldots, l_p \in S,$$
$$\textit{tal que } l_0 = i, l_p = j \textit{ e } G_{l_k l_{k+1}} > 0, \textit{ para } k = 0, 1, \ldots, p-1.$$

Então a cadeia de Markov construída pelo algoritmo de Metropolis é irredutível e aperiódica.

Demonstração: Podemos supor que $l_1, \ldots, l_{p-1}$ são distintos e diferentes de i e j. Usando o Teorema 2.3.5 $p-1$ vezes com $r = 1$, temos

$$\begin{aligned} P_{ij}^p &= \sum_{k_1 \in S} \sum_{k_2 \in S} \cdots \sum_{k_{p-1} \in S} P_{ik_1} P_{k_1 k_2} \cdots P_{k_{p-1} j} \\ &\geq P_{il_1} P_{l_1 l_2} \ldots P_{l_{p-1} j} \\ &= G_{il_1} \alpha_{il_1} G_{l_1 l_2} \alpha_{l_1 l_2} \cdots G_{l_{p-1} j} \alpha_{l_{p-1} j} \\ &> 0, \end{aligned}$$

já que $\alpha_{ij} > 0, \forall\, i, j \in S$.

O que mostra que a cadeia é irredutível.

Sejam $i, j \in S$ com $q_j < q_i$ e $G_{ij} > 0$, devido às hipóteses, o par i, j sempre existe. Logo $\alpha_{ij} < 1$, e

$$\begin{aligned} P_{ii} &= 1 - \sum_{l \in S, l \neq i} G_{il} \alpha_{il} \\ &= 1 - G_{ij} \alpha_{ij} - \sum_{l \in S, l \neq i, j} G_{il} \alpha_{il} \\ &> 1 - G_{ij} - \sum_{l \in S, l \neq i, j} G_{il} \\ &= 1 - \sum_{l \in S, l \neq i} G_{il} \\ &= G_{ii} \geq 0. \end{aligned}$$

Como a cadeia é irredutível, temos pelo Lema 2.3.11 que a cadeia é aperiódica. ■

Além disso, usando a simetria de G, temos que, para $i \neq j$:

$$
\begin{aligned}
q_i P_{ij} &= q_i \alpha_{ij} G_{ij} \\
&= q_i \min\left\{\frac{q_j}{q_i}, 1\right\} G_{ij} \\
&= \begin{cases} q_j G_{ij} & \text{se } q_j \leq q_i \\ q_i G_{ij} & \text{se } q_j > q_i \end{cases} \\
&= \begin{cases} q_i G_{ji} & \text{se } q_i < q_j \\ q_j G_{ji} & \text{se } q_i \geq q_j \end{cases} \\
&= q_j \min\left\{\frac{q_i}{q_j}, 1\right\} G_{ji} \\
&= q_j \alpha_{ji} G_{ji} \\
&= q_j P_{ji}.
\end{aligned}
$$

Este resultado e o Teorema 2.3.12 garantem as hipóteses do Teorema 2.3.10. Com isso temos que a cadeia de Markov construída pelo algoritmo de Metropolis possui distribuição limite q, se q não é uniforme em S e $q_i > 0 \quad \forall\, i \in S$.

Para a descrição do *simulated annealing*, é realizada uma escolha conveniente para a distribuição limite q.

2.3.3 Distribuição de Boltzmann

Dado $f : S \to \mathbb{R}$, S finito, a distribuição de Boltzmann é definida por:

$$
q_i(T) = \frac{1}{N_0(T)} \exp\left(-\frac{f(i)}{T}\right) \quad \forall\, i \in S,
$$

onde $N_0(T)$ é uma constante normalizadora, isto é,

$$
N_0(T) = \sum_{j \in S} \exp\left(-\frac{f(j)}{T}\right).
$$

A escolha desta distribuição se justifica pela seguinte propriedade:

$$
\lim_{T \to 0^+} q_i(T) = q_i^* \stackrel{\text{def}}{=} \begin{cases} \frac{1}{|S_{\mathtt{opt}}|} & \text{se } i \in S_{\mathtt{opt}} \\ 0 & \text{caso contrário} \end{cases},
$$

onde $S_{\mathtt{opt}} = \{i \in S | f(i) \leq f(j), \quad \forall\, j \in S\}$ e $|S_{\mathtt{opt}}|$ é a cardinalidade de $S_{\mathtt{opt}}$.

De fato, sendo $f_{\mathtt{opt}}$ o mínimo global de f em S e usando o fato de que $\lim_{x\to 0^+} \exp\left(\frac{a}{x}\right)$ vale 0 se $a<0$ e 1 se $a=0$, temos que:

$$\begin{aligned}
\lim_{T\to 0^+} q_i(T) &= \lim_{T\to 0^+} \frac{\exp\left(-\frac{f(i)}{T}\right)}{\sum_{j\in S}\exp\left(-\frac{f(j)}{T}\right)} \\
&= \lim_{T\to 0^+} \frac{\exp\left(\frac{f_{\mathtt{opt}}-f(i)}{T}\right)}{\sum_{j\in S}\exp\left(\frac{f_{\mathtt{opt}}-f(j)}{T}\right)} \\
&= \lim_{T\to 0^+} \frac{1}{\sum_{j\in S}\exp\left(\frac{f_{\mathtt{opt}}-f(j)}{T}\right)}\mathbb{I}(i\in S_{\mathtt{opt}}) \\
&\quad + \lim_{T\to 0^+} \frac{\exp\left(\frac{f_{\mathtt{opt}}-f(i)}{T}\right)}{\sum_{j\in S}\exp\left(\frac{f_{\mathtt{opt}}-f(j)}{T}\right)}\mathbb{I}(i\in S-S_{\mathtt{opt}}) \\
&= \frac{1}{|S_{\mathtt{opt}}|}\mathbb{I}(i\in S_{\mathtt{opt}})+0,
\end{aligned}$$

onde $\mathbb{I}(i\in A)$ vale 1 se $i\in A$ e 0 se $i\notin A$.

Isto é, conforme $T\to 0$ a distribuição de Boltzmann tende para uma distribuição uniforme nos minimizadores globais.

Note que $q_i(T)>0 \quad \forall i\in S$ e $\forall T>0$, assim obtemos o seguinte resultado:

$$\lim_{T\to 0^+}\lim_{k\to+\infty}\mathbf{P}_T\{X_k=i\}=\lim_{t\to 0^+} q_i(T)=q_i^*,$$

ou seja,

$$\lim_{T\to 0^+}\lim_{k\to+\infty}\mathbf{P}_T\{X_k\in S_{\mathtt{opt}}\}=1.$$

Note que este resultado também se verifica no caso particular em que a distribuição de Boltzmann é a distribuição uniforme em S, pois neste caso f é constante em S e o resultado é trivial.

A idéia do *simulated annealing* é obter $i\in S$ com distribuição q^*. Para isso construímos a cadeia de Markov homogênea com T fixo até atingirmos a distribuição limite, em seguida diminuímos T e restauramos o equilíbrio atingindo novamente a distribuição limite para a nova temperatura, repetimos o processo até $T\approx 0$.

Em [2] é obtido um resultado de convergência como este, para um algoritmo onde o número de iterações para cada T fixo é finito, com a hipótese adicional de que o decréscimo da temperatura deve ser suficientemente lento.

Surgem então alguns parâmetros em aberto quanto à implementação do algoritmo:

- O valor inicial para a temperatura (T_0).
- Uma função para ditar o decréscimo da temperatura.
- Um valor final para a temperatura, ou seja, um critério de parada.
- Um número finito de transições para cada temperatura (L_k).

Para implementarmos o *simulated annealing* devemos ter um *esquema de resfriamento*, que especifica os parâmetros acima.

2.3.4 Esquemas de Resfriamento

Temperatura inicial

Na literatura existem diversos procedimentos para inicializar a temperatura, abaixo destacamos três deles:

De acordo com [4] inicia-se a temperatura com um número pequeno, tipicamente 10^{-5}, e executamos um número m_0 de transições. Em seguida, calculamos a porcentagem de aceitação χ dos pontos gerados. Se $\chi < \chi_0$, com χ_0 previamente definido (tipicamente $\chi_0 = 0.95$), então incrementamos a temperatura multiplicando-a por uma constante maior do que 1, caso contrário aceitamos a temperatura atual como temperatura inicial para o *simulated annealing*.

Em [2, 14] executa-se m_0 transições com temperatura inicial alta, dada pelo valor da função objetivo em um ponto inicial, multiplicado por uma constante positiva (tipicamente 10^5). Calcula-se m_1, o número de transições onde o vizinho gerado possui menor valor da função objetivo, e m_2, o número de transições aceitas mas com maior valor da função objetivo. Define-se χ_0, a taxa de aceitação inicial (tipicamente $\chi_0 = 0.95$), e calcula-se a temperatura T a partir de

$$\chi_0 = \frac{m_1 + m_2 \exp\left(\frac{-\Delta f^+_{\texttt{inicial}}}{T}\right)}{m_1 + m_2},$$

onde $\Delta f^+_{\texttt{inicial}}$ é o aumento médio do valor da função objetivo dos m_2 pontos aceitos que pioram o valor da função objetivo.

Um terceiro método descrito em [35], consiste em gerar m_0 transições aleatórias calculando $\Delta f^+_{\texttt{inicial}}$, a média da variação absoluta da função objetivo no caso em que o valor da função objetivo aumenta. Sendo $P_{\texttt{inicial}}$ a probabilidade inicial de aceitação, previamente definida, calculamos T a partir de

$$P_{\texttt{inicial}} = \exp\left(\frac{-\Delta f^+_{\texttt{inicial}}}{T}\right).$$

Função para decréscimo da temperatura

Uma maneira frequentemente utilizada na literatura é decrescer a temperatura multiplicando-a por um fator constante, mas em [2] a função é reduzida através de:

$$T = \frac{T}{1 + \frac{T\log(1+\delta)}{3\sigma}},$$

onde σ é o desvio–padrão dos valores da função objetivo dos pontos gerados nesta iteração com a temperatura fixa, e δ é um parâmetro (tipicamente δ varia entre 10^{-4} e 10^{-2}).

Em [35] o decréscimo é feito calculando a razão entre o valor da função objetivo do melhor ponto encontrado nesta iteração (com a temperatura fixa), e a média do valor da função objetivo dos pontos gerados nesta iteração. A nova temperatura é a projeção desta razão no intervalo $[0.1, 0.9]$, multiplicado pela temperatura atual. Desta maneira a temperatura apresenta um decréscimo maior durante as iterações iniciais, pois a melhora da função objetivo tende a ser maior no início.

Valor final para a temperatura e outros critérios de parada

Em [35] os critérios de parada utilizados são:

- nenhum ponto foi gerado de modo a melhorar o valor da função objetivo durante as últimas μ trocas de temperatura (tipicamente $\mu = 4$);

- o número máximo de avaliações da função objetivo foi atingido. Tipicamente este número depende linearmente da dimensão do problema;

- baseado no cálculo da temperatura inicial apresentado em [35] temos que a temperatura final deve depender da variação média final da função objetivo em movimentos aceitos que aumentam o valor da função objetivo e da probabilidade final de aceitação destes pontos. Uma maneira de calcular a média final e a probabilidade final é baseado na média inicial e probabilidade inicial:

$$\Delta f^+_{\texttt{final}} = \varepsilon_1 \Delta f^+_{\texttt{inicial}} + \varepsilon_2$$

e

$$P_{\texttt{final}} = \varepsilon_1 P_{\texttt{inicial}} + \varepsilon_2,$$

onde $\Delta f^+_{\texttt{inicial}}$ e $P_{\texttt{inicial}}$ foram definidos no cálculo da temperatura inicial. Tipicamente $\varepsilon_1 = 10^{-6}$ e $\varepsilon_2 = 10^{-8}$, assim a temperatura final $T_{\texttt{final}}$ é calculada por

$$P_{\texttt{final}} = \exp\left(\frac{-\Delta f^+_{\texttt{final}}}{T_{\texttt{final}}}\right).$$

Assim, a execução é abortada se a temperatura atual T é menor do que a temperatura final $T_{\texttt{final}}$.

Podemos definir também um número máximo de trocas de temperatura.

Em [2, 14] apresenta-se um critério de parada que consiste em calcular a média dos valores funcionais dos p últimos pontos gerados e calcular a diferença com os p anteriores, normalizando este valor com a média dos p últimos pontos e por p. A iteração é interrompida se este número for suficientemente pequeno.

Número de transições para cada temperatura

O número de transições deve ser suficiente para garantir o equilíbrio, em [35] a temperatura é atualizada se o número de movimentos aceitos for maior do que $N_1 n$, onde n é a dimensão do problema, tipicamente $N_1 = 12$. Além disso a temperatura é atualizada se o número de transições nesta temperatura for maior do que $N_2 n$, onde tipicamente $N_2 = 100$.

Vizinhanças

A maneira como os novos pontos são gerados é uma característica essencial do *simulated annealing*. Em [26] exatamente uma das coordenadas do ponto é modificada, ou seja é gerada uma direção de busca paralela à um dos eixos.

Podemos gerar uniformemente pontos que distam menos do que uma distância r pré determinada do ponto atual. Para isso, a distância mais conveniente é a norma infinito, pois podemos reescrever as restrições em forma de caixa e a geração pode ser feita por componentes. Utilizando outras normas necessitaríamos de outras ferramentas estatísticas, como o método da Aceitação e Rejeição.

Em [17, 35] a distância r utilizada varia a cada iteração. Usa–se um vetor v, onde cada componente k do vizinho a ser gerado de x é gerado uniformemente no intervalo

entre $x_k - v_k$ e $x_k + v_k$. Em [35] v_k aumenta em módulo se os pontos gerados estão sendo aceitos em excesso ou diminui em módulo se os pontos gerados estão sendo pouco aceitos. Em [17] o vetor v é atualizado com o intuito de manter a porcentagem de pontos aceitos em 50%.

2.4 Algoritmo Genético Generalizado

Em 1989, Eiben, Aarts e van Hee [1, 18] generalizam as idéias de algoritmo genético, esquemas evolutivos e *simulated annealing*, com o que chamam de "algoritmo genético abstrato" e fornecem teoremas de convergência.

Incluímos aqui um resumo desta teoria, uma vez que unifica as idéias dos algoritmos anteriormente descritos.

Algoritmo genético generalizado é aplicado em problemas de otimização combinatória:

$$\min_x f(x)$$
$$\text{s.a.} \quad x \in S$$

Onde $f : S \to \mathbb{R}$, S finito.

Os elementos de S são chamados de indivíduos, e um conjunto de indivíduos é uma população.

Segue abaixo um esquema do algoritmo:

```
Construir população inicial
Enquanto ‘‘critério de parada’’ não for satisfeito
    ‘‘escolher’’ os pais a partir da população
    ‘‘produzir’’ os filhos destes pais
    estender a população adicionando os filhos a ela
    ‘‘selecionar’’ elementos da população estendida para sobreviverem
Exibir o melhor elemento da população.
```

O algoritmo depende essencialmente da escolha da *função escolha, função produção* e *função seleção*, mas antes de definirmos suas características precisamos das seguintes definições:

Definição 2.4.1 *Uma função vizinhança $V : S \to \mathbb{P}(S)$, é uma função que a cada $i \in S$ associa $V(i)$, a vizinhança de i. Se $j \in V(i)$ dizemos que j é vizinho de i.*

O conjunto $P \subseteq \mathbb{P}(S)$ denominado *férteis* é a coleção dos subconjuntos de S capazes de produzir descendentes. No caso do algoritmo genético padrão temos apenas cruzamento e mutação, assim P seria:

$$\{\{i, j\} | i, j \in S\}.$$

Note que P é a coleção dos subconjuntos de S com um ou dois elementos.

Os elementos de P são denominados *conjuntos–pais*, não havendo restrições para a cardinalidade desses conjuntos. Os elementos dos conjuntos–pais são denominados *pais*.

Introduzimos os conjuntos A, B, C e três parâmetros $\alpha \in A, \beta \in B, \gamma \in C$ escolhidos aleatoriamente que serão os responsáveis pela aleatoriedade em cada geração.

Definição 2.4.2 *Uma função escolha* $f_E \colon A \times \mathbb{P}(S) \to \mathbb{P}(P)$*, associa a cada população uma coleção de conjuntos–pais contidos na população, ou seja*

$$\forall\, \alpha \in A, \forall\, x \in \mathbb{P}(S), \forall\, y \in f_E(\alpha, x) \Rightarrow y \subseteq x.$$

Definição 2.4.3 *Uma função produção* $f_P : B \times P \to \mathbb{P}(S)$*, produz os filhos de um conjunto–pai, tal que cada filho está na união das vizinhanças dos pais, ou seja*

$$\forall\, \beta \in B, \forall\, x \in P \Rightarrow f_P(\beta, x) \subseteq \bigcup_{z \in x} V(z),$$

não havendo restrições quanto ao número de filhos produzidos.

Definição 2.4.4 *Uma função seleção* $f_S \colon C \times \mathbb{P}(S) \to \mathbb{P}(S)$*, seleciona os indivíduos de uma população estendida que sobrevivem para a próxima geração, ou seja*

$$\forall\, \gamma \in C, \forall\, x \in \mathbb{P}(S) \Rightarrow f_S(\gamma, x) \subseteq x.$$

Com base nestas definições, apresentamos o algoritmo:

`Construir população inicial` $x \in \mathbb{P}(S)$.
`Enquanto “critério de parada” não for satisfeito`
    `Gerar` α, β, γ `de` A, B, C `respectivamente`.
    $q = f_E(\alpha, x)$
    $y = \bigcup_{z \in q} f_P(\beta, z)$
    $x' = x \cup y$
    $x = f_S(\gamma, x')$
`Exibir o melhor elemento da população`.

Em seguida apresentamos as idéias de como obter o *simulated annealing* como caso particular do algoritmo genético abstrato. Seja

C o intervalo $(0,1] \subseteq \mathbb{R}$.

$P = \{\{i\} | i \in S\}$.

$f_E(\alpha, \{i\}) = \{\{i\}\} \quad \forall\, \alpha \in A$.

$f_P(\beta, \{i\}) = \{j(\beta)\} \quad \forall\, \beta \in B$ tal que $j(\beta) \in V(i)$.

$$f_S(\gamma, \{i,j\}) = \begin{cases} \{j\} & \text{se } \gamma < \exp\left(\dfrac{f(i) - f(j)}{T}\right) \\ \{i\} & \text{caso contrário} \end{cases}$$

Antes do teorema de convergência, apresentamos as seguintes definições:

Definição 2.4.5 *Uma estrutura de vizinhanças é conexa se é possível, a partir de qualquer indivíduo, alcançar qualquer outro indivíduo através dos vizinhos. Ou seja,*

$$\forall\, i \in S, \quad \forall\, j \in S, \quad \exists\, p \geq 1 \ e\ \exists\, l_0, l_1, \ldots, l_p \in S$$

$$com\ l_0 = i \ e\ l_p = j \ tal\ que\ l_{k+1} \in V(l_k)$$

$$\forall\, k = 0, 1, \ldots, p-1.$$

Definição 2.4.6 *Uma função escolha é generosa se, cada subconjunto unitário de S é um conjunto–pai, além disso a probabilidade de cada um desses subconjuntos ser escolhido é positiva. Ou seja,*

$$\{i\} \in P \quad \forall\, i \in S \ e$$

$$\forall\, x \in \mathbb{P}(S), \quad \forall\, i \in x, \quad \forall\, \alpha \in A \Rightarrow \mathbf{P}\{\{i\} \in f_E(\alpha, x)\} > 0$$

Definição 2.4.7 *Uma função produção é generosa se, na produção de um conjunto–pai $\{i\}$, existe uma probabilidade positiva de j ser um dos filhos, $\forall\, j \in V(i)$. Ou seja,*

$$\forall\, i \in S, \quad \forall\, j \in V(i), \quad \forall\, \beta \in B \Rightarrow \mathbf{P}\{j \in f_P(\beta, \{i\})\} > 0$$

Definição 2.4.8 *Uma função seleção é generosa se, para cada indivíduo na população estendida, existe uma probabilidade positiva deste ser selecionado. Ou seja,*

$$\forall\, x \in \mathbb{P}(S), \quad \forall\, i \in x, \quad \forall\, \gamma \in C \Rightarrow \mathbf{P}\{i \in f_S(\gamma, x)\} > 0$$

Definição 2.4.9 *Uma função seleção é conservativa se, a próxima geração contém o melhor elemento da população anterior, ou pelo menos um deles, caso exista mais de um melhor elemento. Ou seja,*

$$\forall\, x \in \mathbb{P}(S), \quad \forall\, \gamma \in C$$
$$\Rightarrow M_x \cap f_S(\gamma, x) \neq \emptyset$$
$$\textit{onde } M_x = \{i \in x | \forall\, j \in x : f(i) \leq f(j)\}$$

Seja $(\Omega, \mathcal{A}, \mathbf{P})$ um espaço de probabilidade, onde Ω é um conjunto, $\mathcal{A} \subseteq \mathbb{P}(\Omega)$ é uma σ–álgebra em Ω e $\mathbf{P} : \mathcal{A} \to \mathbb{R}$ é uma medida de probabilidade em $\mathcal{A}$, não negativa e σ–aditiva, com $\mathbf{P}\{\Omega\} = 1$. Seja $Z_n \colon \Omega \to A \times B \times C$ a seqüência de variáveis aleatórias independentes que definem os parâmetros α, β, γ para cada geração.

Teorema 2.4.10 *Se A, B, C são finitos, a estrutura de vizinhanças é conexa, Z_n são identicamente distribuídos, as funções escolha, produção e seleção são generosas, a função seleção é conservativa. Então independente da população inicial, o algoritmo genético generalizado encontra um minimizador global com probabilidade* 1*, isto é*

$$\lim_{k \to +\infty} \mathbf{P}\left\{X_k \cap S_{\mathtt{opt}}\right\} = 1.$$

A demonstração deste teorema pode ser vista em [18].

O Teorema 2.4.10 mostra a importância da mutação em qualquer algoritmo genético.

Note que o *simulated annealing* como caso particular do algoritmo genético generalizado, não possui função seleção conservativa. Podemos modificar levemente o algoritmo ao introduzir um novo elemento na população, o melhor elemento encontrado até o momento, assim a função seleção para o *simulated annealing* se torna conservativa.

Capítulo 3

Algoritmos para problemas canalizados

A proposta central deste capítulo é apresentar um algoritmo para otimização de problemas canalizados que incorpora um processo heurístico e um solver para obtenção de ótimos locais. Um algoritmo nesta linha foi proposto por Ma, Tian e Zhang em 2000 [26], com a fase local realizada através de um processo de busca local sem uso de derivadas.

Neste trabalhos optamos por formular algoritmos em que a heurística associada é *simulated annealing* ou algoritmo genético, e para a obtenção do ótimo local escolhemos um algoritmo robusto com propriedade de convergência global.

3.1 DBMS

Em 2000, Ma, Tian e Zhang apresentam o método DBMS (*Darwin and Boltzmann Mixed Strategy*) [26]. Este método busca pela solução global de problemas não lineares com restrições em caixas, sem utilizar informações da derivada. A idéia é discretizar o espaço de busca e aplicar o *simulated annealing*, com um critério de parada, decréscimo de temperatura e escolha de vizinhos específicos, além disso, a cada troca do valor da temperatura T, a solução é refinada com uma busca local particular.

Considere o problema

$$\begin{aligned} &\min_x f(x) \\ &\text{s.a.} \quad l \leq x \leq u \end{aligned}$$

Onde $l, x, u \in \mathbb{R}^n$, e $f\colon \mathbb{R}^n \to \mathbb{R}$.

Dados: a precisão $\delta > 0$, o parâmetro $\alpha \in [0.8, 0.99]$ para o decréscimo da temperatura, a temperatura inicial T_0, a temperatura final $T_f < T_0$, $T_f \in [0.001, 0.01]$, o tamanho

da k–ésima cadeia de Markov L_k e um ponto inicial x. Definimos o algoritmo DBMS abaixo, sendo que função DAS(x) (*Discrete Argument Sampling*) escolhe um elemento na vizinhança de x e DANS(x) (*Discrete Argument Neighbourhood Searching*) busca um minimizador em uma vizinhança, a partir de x.

$k := 0$
REPETIR
    PARA $l = 1$ ATÉ L_k
        $y = \mathrm{DAS}(x)$
        SE $f(y) \leq f(x)$ ENTÃO $x := y$
        SENÃO SE $\texttt{random}[0,1) < \exp\left(\dfrac{f(x) - f(y)}{T_k}\right)$
            ENTÃO $x := y$
    $k := k + 1$
    $T_k = \alpha T_{k-1}$
    $x = \mathrm{DANS}(x)$
ATÉ $T_k < T_f$

As funções DAS(x) e DANS(x) estão descritas abaixo:
função $y =$DAS(x)
$y := x$
Escolha com igual probabilidade $r \in \{1, 2, \ldots, n\}$
Escolha com igual probabilidade $s_r \in \{0, 1, \ldots, [(u_r - l_r)/2]\}$
Onde $[z]$ é o maior inteiro menor ou igual a z.
$\Delta x_r := \delta s_r$.
SE $x_r + \Delta x_r > u_r$
    ENTÃO $y_r = x_r$ ou $y_r = x_r - \Delta x_r$ com igual probabilidade.
SENÃO SE $x_r - \Delta x_r < l_r$
    ENTÃO $y_r = x_r$ ou $y_r = x_r + \Delta x_r$ com igual probabilidade.
SENÃO $y_r = x_r$ com probabilidade $1/2$, $y_r = x_r - \Delta x_r$ com probabilidade $1/4$ e $y_r = x_r + \Delta x_r$ com probabilidade $1/4$.
FIM.

função $y = \mathrm{DANS}(x)$
Dadas constantes $J_r \leq \frac{1}{2}(u_r - l_r) \quad \forall\, r = 1, \ldots, n$.
$y := x$
PARA $r = 1$ ATÉ n
    $j_r := 0$
    REPETIR
        $j_r := j_r + \delta$.
        SE $y_r + j_r \leq u_r$

ENTÃO $z = (y_1, \ldots, y_{r-1}, y_r + j_r, y_{r+1}, \ldots, y_n)$.
SE $f(z) < f(y)$ ENTÃO $y := z$
SE $y_r - j_r \geq l_r$
ENTÃO $z = (y_1, \ldots, y_{r-1}, y_r - j_r, y_{r+1}, \ldots, y_n)$.
SE $f(z) < f(y)$ ENTÃO $y := z$
ATÉ $j_r \geq J_r$.
FIM.

Seja S a discretização com precisão δ do espaço de busca $l \leq x \leq u$, seja $\mathbb{K} = \{k_1, \ldots, k_m\}$ uma ordenação de S e seja G a matriz de geração, onde G_{ij} é a probabilidade de k_j ser igual a DAS(k_i).

Teorema 3.1.1 *A probabilidade do algoritmo DBMS convergir para um minimizador global de f em S, tende a 1 conforme $T_f \to 0$ e $L_k \to +\infty, \quad \forall\, k$.*

A demonstração deste teorema pode ser encontrada em [26].

3.2 SA–GENCAN

No algoritmo que propomos para minimização em caixas com função objetivo diferenciável, o *simulated annealing* é aplicado, mas aqui o espaço de busca não é discretizado como em DBMS, sendo assim, um novo elemento deve ser escolhido da vizinhança do ponto atual, sendo esta vizinhança um subconjunto de $\mathbb{R}^n$ com interior em geral não vazio.

Fixado $\varepsilon > 0$, definimos a vizinhança V de cada ponto $\bar{x}$ da caixa, como o conjunto dos pontos da caixa que distam no máximo ε de $\bar{x}$, na norma infinito. Um novo ponto é obtido gerando uma variável aleatória uniforme na vizinhança, o que pode ser feito por componentes já que cada vizinhança também é uma caixa, devido à utilização da norma infinito.

Em DBMS, há uma busca local após cada troca de temperatura. Em nossa proposta empregamos na fase local o solver GENCAN para problemas diferenciáveis, não lineares com restrições de caixa, proposto e implementado por Birgin e Martínez em 2002, [12]. Trata-se de um método de restrições ativas com gradiente espectral projetado, onde a cada iteração uma aproximação quadrática do problema é resolvido em uma região de confiança. Em [12] os autores demonstram a convergência global do método.

Em SA–GENCAN, a solução encontrada pela busca local não é empregada como ponto inicial para o próximo valor de temperatura como em DBMS, ao invés disto, construímos uma lista de soluções encontradas por GENCAN, sendo que as iterações seguem com o ponto que possuíamos antes de aplicarmos GENCAN. Acreditamos que deste modo estamos sendo mais coerentes com a teoria de *simulated annealing*, pois evitamos mudanças

bruscas no valor da função objetivo ao longo da cadeia de Markov gerada.

Executamos GENCAN com os parâmetros padrões, e acrescentamos um novo critério de parada: retornamos o ponto atual do GENCAN como solução se ele estiver à uma distância euclidiana menor do que 10^{-2} de algum dos pontos da lista. Se este critério não for acionado, acrescentamos este novo ponto à lista (mesmo que o GENCAN não tenha declarado convergência para um minimizador local). O objetivo da lista, além de armazenar a melhor solução, é evitar que GENCAN encontre uma solução já encontrada anteriormente.

O esquema de resfriamento implementado para o *simulated annealing* foi o descrito em [35], apresentado na Seção 2.3.4 com os valores típicos para os parâmetros, juntamente com o critério de parada descrito em [2, 14], apresentado na seção 2.3.4. Além disto limitamos em 100 o número máximo de trocas de temperatura.

Segue abaixo um esboço do algoritmo:

```
INICIALIZAR x, T_0, L_0
k := 0
REPETIR
    PARA l = 1 ATÉ L_k
        GERAR y de V(x)
        SE f(y) ≤ f(x) ENTÃO x := y
        SENÃO SE random[0,1) < exp((f(x) − f(y))/T_k)
                ENTÃO x := y
    k := k + 1
    CALCULAR L_k e T_k
    EXECUTAR GENCAN A PARTIR de x
ATÉ "critério de parada"
```

3.3 AG–GENCAN

Na tentativa de mesclar heurísticas para otimização global com solvers de otimização local, introduzimos um algoritmo genético como anteriormente, mas com a modificação de que, sempre que o menor elemento da população se mantém o mesmo por um certo número de iterações (neste caso dizemos que o algoritmo genético está estagnado), executamos uma iteração do solver local GENCAN, a partir deste melhor elemento e substituimos a solução encontrada pelo pior elemento da população.

3.4 Experimentos Numéricos: Canalizados

Nesta seção apresentamos os resultados obtidos pelos diversos solvers apresentados. A implementação foi feita em Fortran 77 em uma máquina Pentium 4, 3.5 GHz, 2Gb Ram. Na Seção 3.4.1 detalhamos o desempenho computacional de algoritmo genético, AG–GENCAN, esquemas evolutivos, simulated annealing, SA–GENCAN e DBMS aplicados ao problema OVO. Na Seção 3.4.2 apresentamos o desempenho computacional de SA–GENCAN, GENCAN, *simulated annealing* e DBMS na resolução dos 28 problemas descritos no Apêndice A.

3.4.1 O problema OVO

Descrição do problema

Utilizamos o problema OVO (*Order–Value Optimization*), de acordo com [7, 8, 9, 10]. A seguir definimos o problema.

Dadas m funções $f_1, \ldots, f_m$ definidas no conjunto $\Omega \subseteq \mathbb{R}^n$ e um inteiro positivo $p \leq m$, definimos as m funções índices $i_1, \ldots, i_m : \Omega \to \{1, 2, \ldots, m\}$ de modo que $\{i_1(x), i_2(x), \ldots, i_m(x)\}$ é uma permutação de $\{1, 2, \ldots, m\}$ e

$$f_{i_1(x)}(x) \leq f_{i_2(x)}(x) \leq \cdots \leq f_{i_p(x)}(x) \leq \cdots \leq f_{i_m(x)}(x) \quad \forall\, x \in \Omega.$$

Assim, podemos definir o problema OVO:

$$\begin{aligned} \min_x \ & \sum_{j=1}^{p} f_{i_j(x)}(x) \\ \text{s.a.} \quad & x \in \Omega \end{aligned}$$

Em particular, queremos ajustar o modelo $y(t, x) = x_1 + x_2 t + x_3 t^2 + x_4 t^3$ em um conjunto de dados (t_i, y_i), $i = 1, \ldots, m$, desprezando os $m - p$ piores ajustes. As funções f_i são as funções erro $f_i(x) = (y(t_i, x) - y_i)^2$.

Sendo $\Omega = [-3, 3]^4$ e dada a solução $x^* = (x_1^*, x_2^*, x_3^*, x_4^*)$ escolhida aleatoriamente em Ω, $m = 101$ e $p = 80$ definimos $t_i = -1 + 0.02(i - 1)$ e $w_i = y(t_i, x^*) \quad \forall\, i = 1, 2, \ldots\, m$.

Escolhemos aleatoriamente $m - p = 21$ elementos de $\{1, 2 \ldots, 101\}$, a saber $p_1, p_2, \ldots, p_{21}$ e definimos

$$y_i = \begin{cases} 10 & \text{se} \quad i \in \{p_1, p_2, \ldots, p_{21}\} \\ w_i & \text{c.c.} \end{cases}$$

A solução deste problema é x^* com valor nulo para a função objetivo.

Algoritmo Genético

A implementação foi feita conforme descrito anteriormente, com a diferença de que adotamos um critério elitista, onde o melhor elemento sempre se mantém na população. A discretização utilizada é $h_i = 10^{-6}$ para cada componente, deste modo cada cromossomo é um vetor binário de tamanho $m = 92$. O tamanho da população é $n_{pop} = 20$ e as probabilidades de cruzamento e mutação são respectivamente $p_c = 0.25$ e $p_m = 0.01$.

Na Figura 3.1 apresentamos o valor médio da função objetivo da população e na Figura 3.2 o valor da função objetivo do melhor elemento da população em cada iteração/geração.

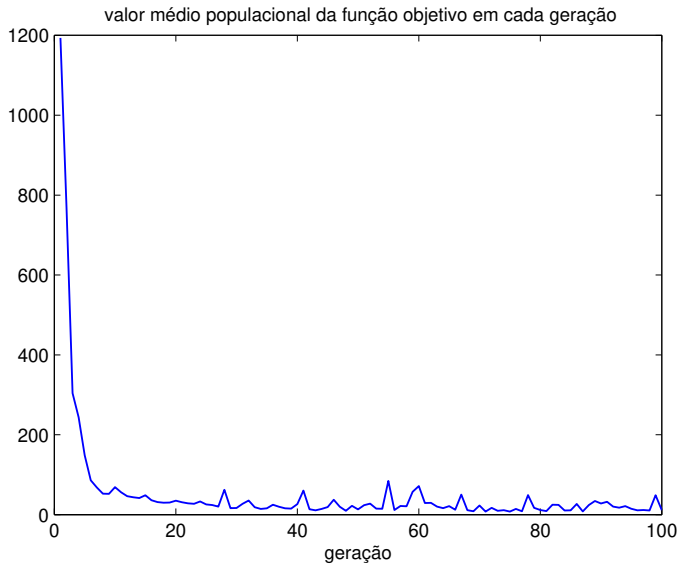

Figura 3.1: Desempenho médio da população (algoritmo genético).

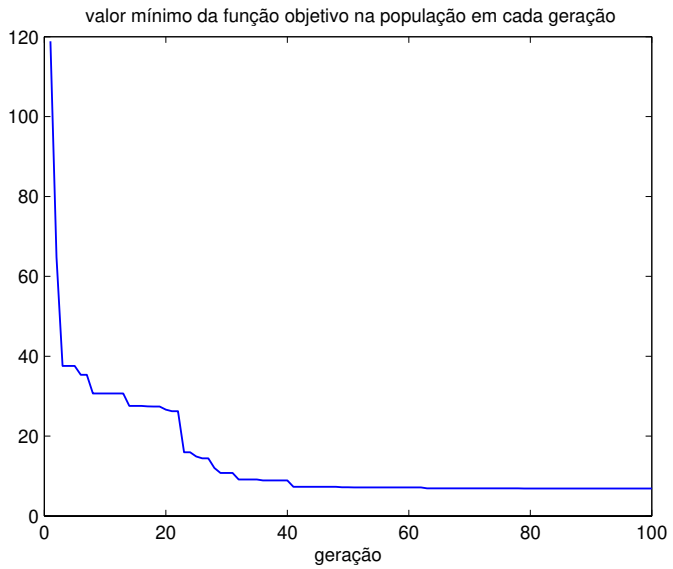

Figura 3.2: Valor da função objetivo do melhor elemento da população (algoritmo genético).

AG–GENCAN

Consideramos que o algoritmo genético está estagnado quando o melhor elemento se repete por cinco iterações.

Na Figura 3.3, marcamos com pequenos círculos as posições das chamadas do GENCAN. Comparando as Figuras 3.2 e 3.3 podemos perceber que uma única chamada do solver local fez com que o método escapasse da estagnação, encontrando a solução global do problema.

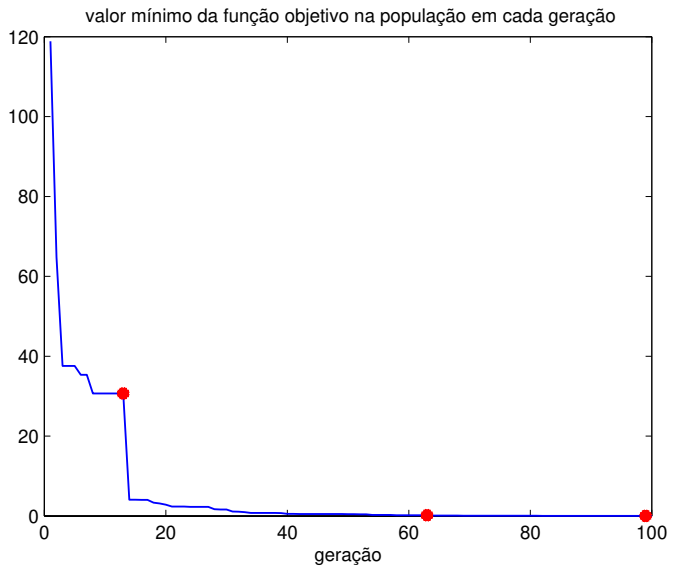

Figura 3.3: Algoritmo genético com acelerador local (AG–GENCAN).

Esquemas Evolutivos

Utilizamos uma população de tamanho $n_{pop} = 20$ e geramos $\mu = 5$ novos elementos por iteração através do cruzamento discreto. $\Delta\sigma = 1$ e a população é inicializada aleatoriamente em Ω, sendo $\sigma = 1$ o desvio padrão inicial de cada elemento da população.

Na Figura 3.4 apresentamos o valor médio da função objetivo da população e na Figura 3.5 o valor da função objetivo do melhor elemento da população por iteração/geração.

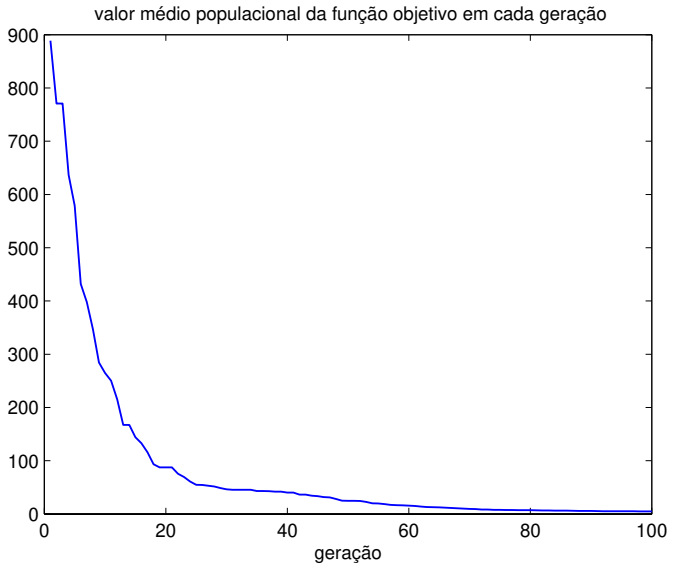

Figura 3.4: Desempenho médio da população (esquemas evolutivos).

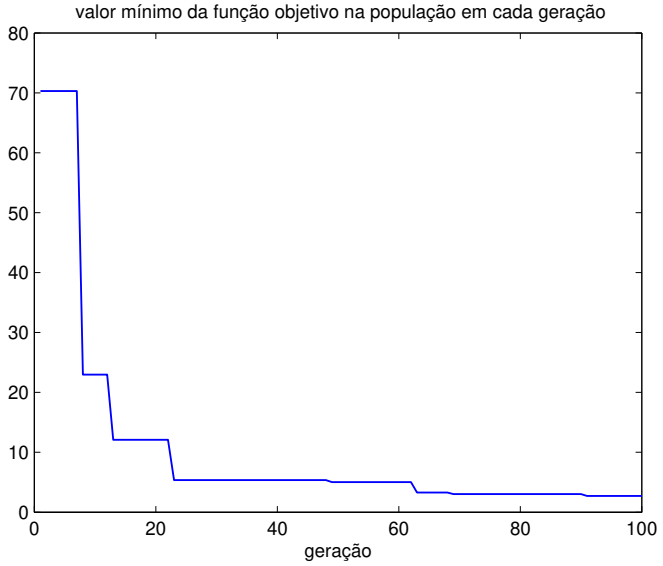

Figura 3.5: Valor da função objetivo do melhor elemento da população (esquemas evolutivos).

Os resultados obtidos são semelhantes aos obtidos com algoritmo genético.

Assim como fazemos com o algoritmo genético, poderíamos utilizar um solver local para evitar a estagnação.

Simulated Annealing

Nesta implementação do *simulated annealing* o esquema de decréscimo da temperatura é dado por

$$T_{k+1} = T_k \frac{\log(100)}{\log(100+k)}, \quad T_0 = 100,$$

Na Figura 3.6 apresentamos o gráfico de T em função de k. A justificativa para escolha deste esquema para este problema é que o comportamento da probabilidade média de aceitação de pontos que pioram o valor da função objetivo tem um decréscimo adequado ao longo das iterações, como pode ser visto na Figura 3.7.

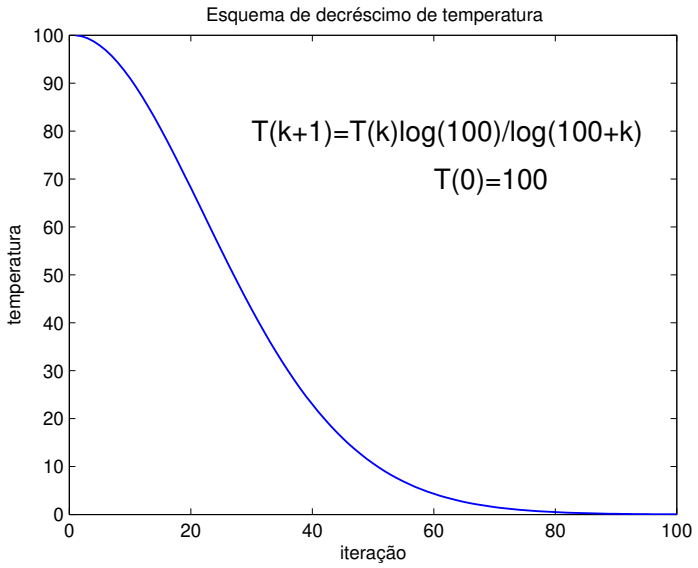

Figura 3.6: Decréscimo da temperatura.

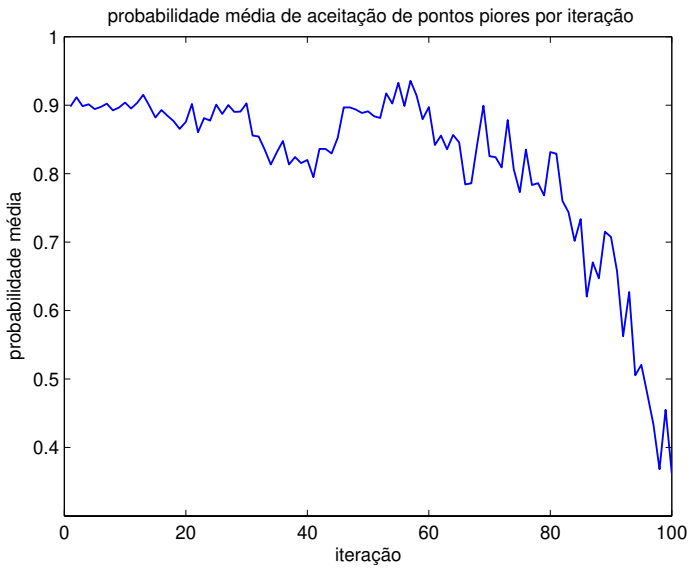

Figura 3.7: Probabilidade média de aceitação de pontos que pioram o valor da função objetivo.

Pela Figura 3.6 vemos que a temperatura está próxima de zero, como prevê a teoria, conforme nos aproximamos do critério de parada, dado pela execução de 100 iterações. Mais importante do que isto, é o fato mostrado pela Figura 3.7, de que a probabilidade média de aceitação de pontos piores é inicialmente próxima de um, e ao nos aproximarmos do critério de parada, esta probabilidade está próxima de zero.

A função vizinhança associa a cada $i \in \Omega$ o conjunto de pontos de Ω que distam menos de uma unidade de i na norma infinito.

O critério de parada é baseado no número de iterações, limitado em 100. A cada iteração geramos $L_k = 100$ vizinhos.

Na Figura 3.8 temos o valor da função objetivo de cada elemento da cadeia de Markov heterogênea gerada, isto é, o valor da função objetivo de cada um dos $L_k = 100$ pontos gerados entre cada troca de temperatura.

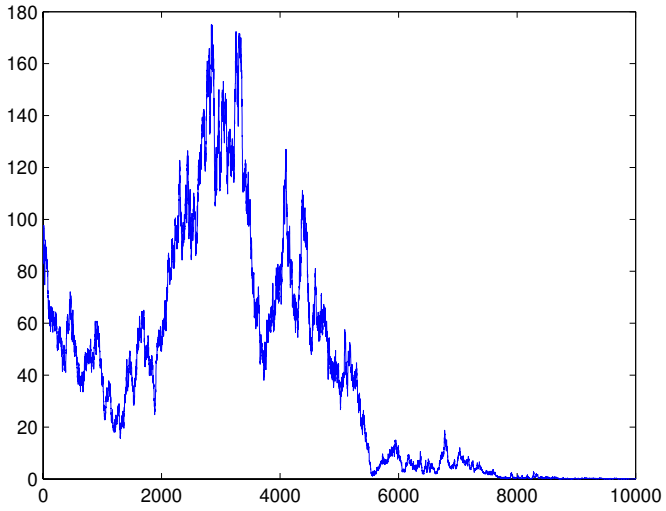

Figura 3.8: Valor da função objetivo de cada elemento da cadeia de Markov gerada.

Pela Figura 3.8 percebemos que de fato, nas iterações iniciais o método oscila entre pontos bons e ruins, enquanto que nas iterações finais, pontos com valores piores da função objetivo não são aceitos.

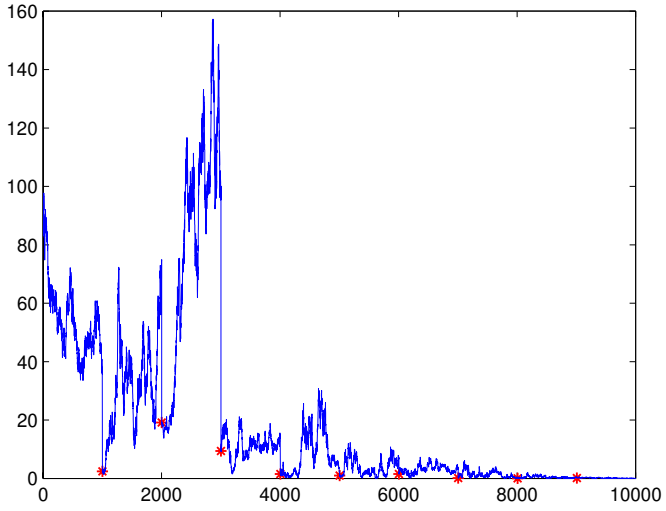

Figura 3.9: Valor da função objetivo da cadeia de Markov gerada por SA–GENCAN

Na Figura 3.9 temos a cadeia de Markov gerada como no exemplo anterior, com

a diferença de que antes de cada troca de temperatura melhoramos o valor da função objetivo através da execução de GENCAN, cada execução está sinalizada na figura com o símbolo *. Notamos neste caso uma oscilação pequena no valor da função objetivo após gerar 3000 pontos, o que demora mais para ocorrer no caso em que apenas *simulated annealing* é executado.

DBMS

Utilizamos os mesmos parâmetros utilizados em [26], isto é, discretização de 10^{-2}, temperatura inicial $T_0 = 100$ sendo que a temperatura decresce de acordo com $T_{k+1} = 0.8T_k$, e o critério de parada dado pela temperatura final $T_f = 0.01$. A quantidade de vizinhos gerados antes da busca local é $L_k = 10$. Na Figura 3.10 apresentamos os resultados.

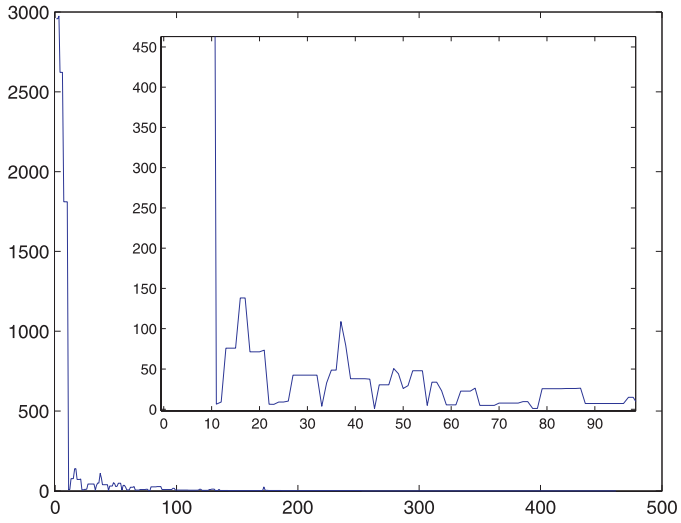

Figura 3.10: valores gerados por DBMS, e ampliação das primeiras iterações.

Como podemos ver na Figura 3.10, o método consegue reduzir significativamente o valor da função objetivo nas dez primeiras iterações, mas apresenta grande variação entre as iterações 10 e 100, antes de conseguir atingir a solução.

3.4.2 Outros problemas

Nesta seção apresentamos uma comparação dos resultados obtidos por SA–GENCAN, GENCAN, *simulated annealing* e DBMS na resolução dos 28 problemas apresentados no Apêndice A, são problemas clássicos com solução conhecida extraídos de [20, 21, 26, 29,

35]. A aproximação inicial é escolhida aleatoriamente no espaço de busca, e cada problema é resolvido 20 vezes, variando a aproximação inicial. Na Tabela 3.1 temos os resultados em quantidade de problemas resolvidos pelo SA–GENCAN, sendo que um problema é considerado resolvido se o desvio relativo entre a solução encontrada e a solução global conhecida é menor do que 0.01. Além disso, a tabela apresenta a dimensão n do problema, o número médio de avaliações de função e gradiente, o número médio de soluções encontradas por GENCAN (minimizadores locais ou não), o número médio de minimizadores locais encontrados e o número médio de chamadas do GENCAN.

Tabela 3.1: SA–GENCAN

	n	problemas resolvidos	avaliações de função	avaliações do gradiente	minimizadores/ soluções	chamadas de GENCAN
1	2	20	5251.1	325.5	24.1/25.1	45.3
2	4	20	3624.7	254.2	20.2/21.2	29.9
3	4	20	3858.9	276.8	22.1/23.2	31.6
4	4	20	4149.0	304.8	26.7/27.9	34.2
5	2	20	11768.1	1474.6	63.5/64.5	100.0
6	2	20	9471.7	520.3	58.2/59.2	86.3
7	2	20	8092.2	362.0	41.5/58.8	58.0
8	2	20	7403.4	482.2	49.6/50.6	65.4
9	2	20	4876.0	132.3	45.2/46.2	46.0
10	1	20	10995.9	622.7	43.7/44.7	100.0
11	1	20	11645.5	1534.7	0.9/2.0	100.0
12	1	20	8894.6	370.4	2.9/3.9	82.3
13	1	20	10796.0	496.2	21.5/22.5	99.6
14	2	20	10984.2	654.8	41.0/42.0	100.0
15	2	20	10009.4	575.7	91.8/92.8	91.9
16	1	20	9995.7	504.0	39.3/40.3	91.1
17	2	20	10524.6	543.8	36.5/37.5	94.7
18	1	20	14319.6	250.5	13.6/85.0	86.9
19	4	20	19226.7	6108.8	41.9/42.9	100.0
20	4	20	41538.7	27345.4	83.8/87.8	93.8
21	10	18	74506.0	26685.2	0.0/3.5	100.0
22	10	20	556249.8	512239.7	0.0/52.2	100.0
23	4	20	21914.8	11721.0	65.2/75.8	91.6
24	10	20	110846.0	96871.5	21.2/88.9	88.3
25	100	20	6911.6	471.9	0.1/2.3	60.8
26	100	20	21776.2	7785.7	8.1/11.8	64.7
27	100	20	12225.7	894.8	1.0/2.0	95.6
28	9	16	870524.2	51501.6	4.6/19.0	18.2

A Tabela 3.2 mostra a freqüência com que cada critério de parada foi acionado em SA–GENCAN.

Tabela 3.2: Critérios de parada acionados em SA–GENCAN

Critérios de Parada	Porcentagem
Variação pequena da função objetivo	35.96%
Número máximo de iterações	47.31%
Número máximo de avaliações de função	3.28%
Temperatura muito pequena	3.85%
Nenhuma melhora nas últimas iterações	9.61%

Na Tabela 3.3 mostramos a quantidade de problemas resolvidos em 20 chamadas de GENCAN, juntamente com a dimensão n do problema e o número médio de avaliações de função e gradiente. Na Tabela 3.4 mostramos o desempenho de *simulated annealing*, ao ser usado para obter uma boa aproximação inicial para GENCAN e na Tabela 3.5 temos o desempenho de DBMS. Os resultados são referentes à uma média de 20 execuções.

Podemos observar a dificuldade de DBMS na resolução de problemas com dimensões 10 e 100.

Na Figura 3.11 apresentamos o desempenho de SA–GENCAN, GENCAN e *simulated annealing* em quantidade de problemas resolvidos, onde podemos ver o desempenho superior de SA–GENCAN.

Tabela 3.3: GENCAN

	n	problemas resolvidos	avaliações de função	avaliações do gradiente
1	2	13	115.8	7.9
2	4	9	118.6	8.9
3	4	7	118.2	8.8
4	4	5	117.4	8.6
5	2	4	120.7	16.4
6	2	20	112.7	7.6
7	2	3	141.3	6.7
8	2	8	114.5	8.0
9	2	3	105.1	2.5
10	1	5	112.8	8.4
11	1	20	123.7	21.2
12	1	18	111.8	7.4
13	1	20	112.8	7.4
14	2	8	112.3	7.6
15	2	15	108.8	6.4
16	1	12	111.4	6.9
17	2	16	113.2	6.8
18	1	0	154.7	3.0
19	4	20	198.6	66.4
20	4	11	284.6	146.0
21	10	11	958.0	771.8
22	10	16	6115.4	5541.0
23	4	16	277.7	161.6
24	10	3	272.2	125.8
25	100	20	149.9	11.6
26	100	17	390.4	178.3
27	100	20	132.6	10.5
28	9	1	71479.6	4061.6

Tabela 3.4: *simulated annealing*

	n	problemas resolvidos	avaliações de função	avaliações do gradiente
1	2	19	4984.4	4.6
2	4	4	10116.4	8.3
3	4	5	10117.2	8.7
4	4	9	9915.8	9.4
5	2	12	10113.0	9.8
6	2	20	8971.6	4.2
7	2	16	6338.6	4.2
8	2	20	6857.3	5.8
9	2	7	5292.5	3.5
10	1	11	10111.3	6.8
11	1	20	10119.0	16.1
12	1	20	8812.0	4.7
13	1	20	10109.0	6.0
14	2	9	10109.8	6.2
15	2	20	9451.0	4.6
16	1	20	9496.2	5.1
17	2	16	8528.5	4.9
18	1	1	8971.0	2.3
19	4	20	10211.4	71.7
20	4	16	10647.7	536.9
21	10	11	11948.0	769.6
22	10	12	16765.3	6314.7
23	4	13	10160.3	73.3
24	10	5	11216.2	1094.0
25	100	20	9877.2	10.1
26	100	17	10370.2	175.4
27	100	20	10126.8	9.6
28	9	5	51813.6	2562.1

Tabela 3.5: *DBMS*

	n	problemas resolvidos	avaliações de função
1	2	14	135211.8
2	4	16	287391.8
3	4	7	284423.3
4	4	6	277611.4
5	2	10	56105.8
6	2	20	176492.5
7	2	20	256696.3
8	2	13	53916.2
9	2	20	124340.6
10	1	20	62432.0
11	1	20	5879.6
12	1	20	73798.3
13	1	20	74871.8
14	2	16	157117.1
15	2	20	121975.8
16	1	20	76851.0
17	2	20	30517.5
18	1	20	47753.2
19	4	0	197627.3
20	4	0	184358.5
21	10	0	1230135.8
22	10	0	1267201.9
23	4	0	51366.8
24	10	0	145010.8
25	100	0	17435784.0
26	100	0	5004338.5
27	100	0	5003387.0
28	9	10	1153736.9

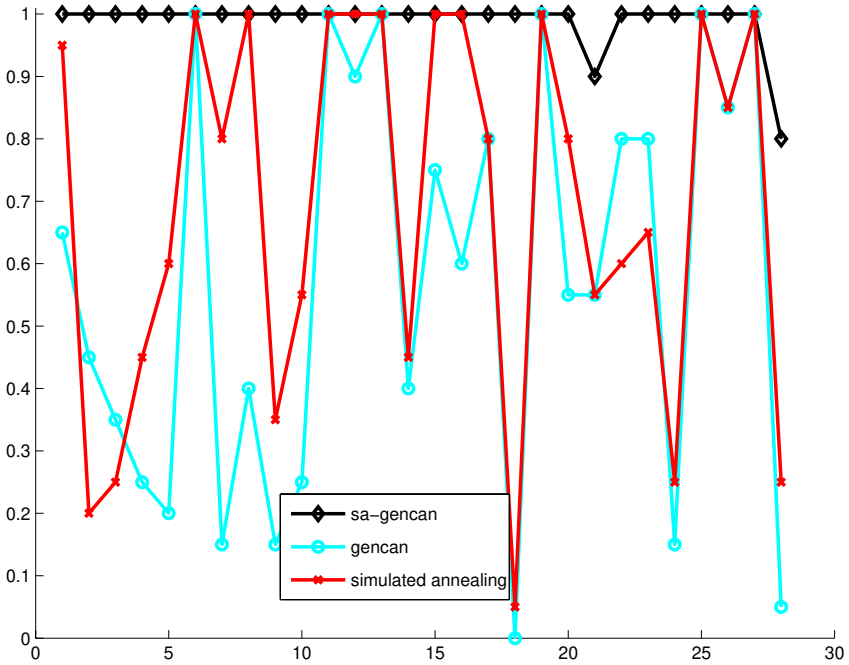

Figura 3.11: Comparação: GENCAN, SA–GENCAN, *simulated annealing*

Observamos também que apenas GENCAN não é eficiente neste caso, uma vez que o número de problemas resolvidos é bem reduzido, o que era de se esperar pois GENCAN é um método local. O que pode ser feito para melhorar sua performance é executar GENCAN mais de uma vez com aproximações iniciais escolhidas aleatoriamente no espaço de busca, neste caso os resultados são semelhantes aos obtidos por SA–GENCAN nos problemas de dimensão pequena e se torna ineficiente para problemas de dimensões maiores. Além disto, em SA-GENCAN o procedimento realizado pelo *simulated annealing* escolhe aproximações iniciais mais promissoras o que resulta numa vantagem para este algoritmo.

Comparando o desempenho de SA–GENCAN na Tabela 3.1 com o desempenho de DBMS na Tabela 3.5 percebemos que ao utilizar um solver local robusto como GENCAN, ao invés de uma busca local sem derivadas, temos uma forte melhora de performance.

Analisando os resultados temos que SA–GENCAN é o que consegue resolver o maior número de problemas, embora o custo computacional seja substancialmente maior, como era de se esperar já que em SA–GENCAN são feitas chamadas de GENCAN e também há o mesmo esforço presente em *simulated annealing*.

Capítulo 4

Algoritmo para problemas restritos

4.1 SA–ALGENCAN

Considere o seguinte problema geral de programação não linear:

$$\begin{aligned} &\min_x f(x) \\ &\text{s.a.} \\ &g(x) \leq 0 \\ &h(x) = 0 \\ &l \leq x \leq u \end{aligned}$$

Onde $l, x, u \in \mathbb{R}^n$, $f : \mathbb{R}^n \to \mathbb{R}$, $g : \mathbb{R}^n \to \mathbb{R}^p$, e $g : \mathbb{R}^n \to \mathbb{R}^{m-p}$. As funções f, g, e h são suaves.

Na literatura não há uma proposta única sobre como utilizar *simulated annealing* neste tipo de problema, devido a questões sobre como lidar com as restrições. Nos trabalho de Cardoso, Salcedo, Azevedo em [14] e Salcedo, Gonçalves, Azevedo em [32], os pontos gerados que não satisfazem as restrições são rejeitados. Esta abordagem pode ser desgastante, dependendo do tipo de região formada pelas restrições.

Neste trabalho utilizamos um procedimento como SA–GENCAN, com a diferença que o solver local utilizado é o ALGENCAN em Fortran, um solver para problemas gerais de programação não linear do tipo Lagrangiano Aumentado que utiliza GENCAN, descrito em [5, 6].

Em SA–ALGENCAN, nossa proposta para a função objetivo utilizada pelo *simulated annealing* é denotada por $f_{\texttt{SA}}$ definida por:

$$f_{\mathtt{SA}}(x) = f(x) + \rho_{\mathtt{SA}} \left(||h(x)||^2 + ||g^+(x)||^2 \right)$$

Onde $||\cdot||$ é a norma Euclidiana, $g^+(x) = \max\{0, g(x)\}$ e $\rho_{\mathtt{SA}} \geq 0$ é incrementado com um fator multiplicativo a cada troca de temperatura.

4.2 Experimentos Numéricos

Os experimentos numéricos a seguir foram implementados em Fortran 77, num Pentium 4 3.5 GHz, 2Gb Ram.

Antes de executar ALGENCAN é preciso definir uma aproximação inicial para os multiplicadores de Lagrange. Tomamos esta aproximação sempre nula na primeira vez que ALGENCAN é executado. Porém, como executamos o ALGENCAN várias vezes para o mesmo problema, é possível usar como aproximação inicial para os multiplicadores de Lagrange, os multiplicadores finais da execução anterior ao invés da aproximação nula. Com esta estratégia corremos o risco de viciarmos a próxima execução para o minimizador encontrado na última execução, mas talvez usar estes multiplicadores inicialmente seja melhor do que utilizar uma aproximação nula. Implementamos ambas as estratégias para os 12 problemas restritos no Apêndice B, que são problemas clássicos e de solução conhecida.

Nas tabelas apresentadas, SA–ALGENCAN foi executado 20 vezes para cada problema. Um problema é dito resolvido se o desvio relativo do ponto encontrado para a solução correta é menor do que 0.01. Nas tabelas temos a dimensão n do problema, o número de restrições m, o numero médio de avaliações de função e gradiente, o número médio de soluções encontradas por ALGENCAN (minimizadores locais ou não) e o número médio de minimizadores locais encontrados.

Na Tabela 4.1 apresentamos os resultados obtidos com SA–ALGENCAN sem reinicializar os multiplicadores de Lagrange, isto é, a aproximação inicial para os multiplicadores utilizados é sempre o que foi retornado na última execução. Na Tabela 4.2 os multiplicadores são reinicializados em zero antes de cada execução de ALGENCAN. Nesta execução $\rho_{\mathtt{SA}}$ foi inicializado como 1 e após cada execução de ALGENCAN seu valor é multiplicado por 10.

Tabela 4.1: SA–ALGENCAN: $\rho_{\texttt{SA}} = 1$, sem reinicializar os multiplicadores

	(n, m)	problemas resolvidos	avaliações de função	avaliações do gradiente	número de soluções	número de minimizadores
1	$(6, 6)$	12	12884.8	62.3	2.5	2.5
2	$(3, 2)$	12	12116.7	88.2	2.7	2.7
3	$(3, 1)$	20	10940.8	66.5	1.0	1.0
4	$(7, 6)$	20	55306.2	23708.9	1.0	1.0
5	$(32, 19)$	20	512286.9	67355.0	2.2	0.0
6	$(5, 3)$	19	14521.2	246.1	2.2	2.2
7	$(5, 1)$	3	12178.1	18.6	3.0	3.0
8	$(20, 10)$	3	15756.8	102.3	1.6	1.6
9	$(5, 6)$	20	13533.2	44.7	1.0	1.0
10	$(2, 2)$	19	12085.3	125.9	1.4	1.4
11	$(13, 7)$	8	10244.1	135.4	2.7	2.7
12	$(16, 13)$	20	237402.3	18585.8	1.9	1.8

Tabela 4.2: SA–ALGENCAN: $\rho_{\texttt{SA}} = 1$, reinicializando os multiplicadores

	(n, m)	problemas resolvidos	avaliações de função	avaliações do gradiente	número de soluções	número de minimizadores
1	$(6, 6)$	12	12887.2	66.0	2.6	2.6
2	$(3, 2)$	13	12108.3	82.6	2.7	2.7
3	$(3, 1)$	20	10952.9	75.3	1.0	1.0
4	$(7, 6)$	20	55765.3	26772.8	1.0	1.0
5	$(32, 19)$	20	893670.5	90617.0	2.2	0.1
6	$(5, 3)$	10	14415.2	243.6	1.5	1.5
7	$(5, 1)$	3	12178.5	18.5	3.0	3.0
8	$(20, 10)$	2	15739.2	92.7	1.5	1.5
9	$(5, 6)$	20	13541.5	47.1	1.0	1.0
10	$(2, 2)$	19	12061.5	123.0	1.4	1.4
11	$(3, 7)$	9	10179.1	79.6	2.8	2.8
12	$(16, 13)$	20	60636.8	15945.0	1.4	1.2

Conforme diminuimos o valor de $\rho_{\texttt{SA}}$ inicial percebemos uma melhora no desempenho, o que nos fez concluir que a melhor escolha para $\rho_{\texttt{SA}}$ é 0, o que equivale a considerar para o *simulated annealing* apenas a função objetivo do problema original. Nas Tabelas 4.3 e 4.4 temos o desempenho de SA–ALGENCAN com $\rho_{\texttt{SA}} = 0$ sem reinicializar os multiplicadores de Lagrange a cada execução de ALGENCAN e respectivamente reinicializando

os multiplicadores em zero.

Tabela 4.3: SA–ALGENCAN: $\rho_{\texttt{SA}} = 0$, sem reinicializar os multiplicadores

	(n, m)	problemas resolvidos	avaliações de função	avaliações do gradiente	número de soluções	número de minimizadores
1	$(6, 6)$	20	20286.6	964.5	12.1	12.1
2	$(3, 2)$	20	15574.9	979.3	5.6	5.6
3	$(3, 1)$	20	13858.5	802.3	1.0	1.0
4	$(7, 6)$	20	1191726.2	477466.8	1.6	1.0
5	$(32, 19)$	20	2432632.0	408042.1	30.1	0.1
6	$(5, 3)$	20	23137.1	3025.9	3.5	3.5
7	$(5, 1)$	9	14738.0	76.6	9.6	9.6
8	$(20, 10)$	19	28994.2	3280.6	12.6	12.6
9	$(5, 6)$	20	20178.2	800.8	1.0	1.0
10	$(2, 2)$	19	19848.2	775.0	2.3	2.3
11	$(13, 7)$	19	22982.9	1855.6	38.2	38.2
12	$(16, 13)$	19	167376.5	43647.0	2.0	1.9

Tabela 4.4: SA–ALGENCAN: $\rho_{\texttt{SA}} = 0$, reinicializando os multiplicadores

	(n, m)	problemas resolvidos	avaliações de função	avaliações do gradiente	número de soluções	número de minimizadores
1	$(6, 6)$	20	20278.6	813.0	13.4	13.4
2	$(3, 2)$	20	15954.0	1056.0	5.6	5.6
3	$(3, 1)$	20	13543.0	497.4	1.0	1.0
4	$(7, 6)$	20	921847.2	600550.6	1.0	1.0
5	$(32, 19)$	20	31001168.0	3419800.5	75.6	1.2
6	$(5, 3)$	20	23247.8	2824.3	2.0	2.0
7	$(5, 1)$	9	14741.5	78.3	9.6	9.6
8	$(20, 10)$	19	28426.5	3027.3	12.8	12.8
9	$(5, 6)$	20	20981.2	982.0	1.0	1.0
10	$(2, 2)$	20	19739.2	785.7	2.8	2.8
11	$(13, 7)$	20	23548.8	2340.4	25.2	25.2
12	$(16, 13)$	20	624123.9	424496.9	2.0	1.5

Na Tabela 4.5 temos o desempenho médio de 20 execuções de ALGENCAN para cada problema.

Tabela 4.5: ALGENCAN

	(n, m)	problemas resolvidos	avaliações de função	avaliações do gradiente
1	$(6, 6)$	1	12435.8	22.4
2	$(3, 2)$	8	10039.2	45.7
3	$(3, 1)$	20	9601.5	9.2
4	$(7, 6)$	20	28420.2	10564.8
5	$(32, 19)$	19	450581.6	40589.5
6	$(5, 3)$	9	13304.8	66.4
7	$(5, 1)$	0	10751.8	4.8
8	$(20, 10)$	2	12580.6	75.6
9	$(5, 6)$	20	13337.0	31.5
10	$(2, 2)$	17	9461.0	31.5
11	$(13, 7)$	4	9016.6	26.6
12	$(16, 13)$	15	41905.4	7386.2

Analisando os resultados vemos que SA–ALGENCAN com $\rho_{\mathtt{SA}} = 0$ e reinicializando os multiplicadores de Lagrange em zero antes de cada execução de ALGENCAN é o que se destaca entre as alternativas propostas. Na comparação com ALGENCAN, temos que SA–ALGENCAN resolve mais problemas, embora ocorra um número elevado de avaliações de função e gradiente.

Capítulo 5

Conclusões

Neste trabalho estudamos a teoria de *simulated annealing* e a generalização do algoritmo genético de Aarts, Eiben e van Hee. Propusemos um algoritmo, SA-GENCAN, do tipo duas fases para problemas canalizados no qual utilizamos *simulated annealing* e o solver local GENCAN. Comparamos os resultados do novo método com *simulated annealing*, GENCAN e DBMS. Observamos que SA–GENCAN é mais robusto pois obtém a solução global um maior número de vezes que os demais métodos, embora o custo computacional seja, em geral, maior. Para problemas restritos apresentamos um algoritmo semelhante, utilizando *simulated annealing* e o solver local ALGENCAN e avaliamos propostas de como trabalhar com a função objetivo e as restrições na fase heurística. Os resultados computacionais mostram um bom desempenho do algoritmo proposto. Observamos que ainda há muita pesquisa a ser feita na resolução de problemas restritos, principalmente na maneira com que as heurísticas tratam as restrições.

Apêndice A

Problemas Teste: Canalizados

A lista abaixo apresenta uma descrição dos problemas canalizados utilizados nos experimentos numéricos realizados. Os problemas foram extraídos de [20, 21, 26, 29, 35].

Os problemas são todos do tipo

$$\begin{aligned} &\min_x f(x) \\ &\text{s.a.} \quad l \leq x \leq u \end{aligned}$$

Onde $l, x, u \in \mathbb{R}^n$, e $f\colon \mathbb{R}^n \to \mathbb{R}$.

Os problemas são descritos ao explicitarmos a dimensão n, a função $f(\cdot)$, os vetores $l, u \in \mathbb{R}^n$ e o valor $f_{\min}$ que minimiza a função objetivo. Ao atribuirmos um escalar a l ou u, significa que cada componente é igual ao escalar.

1. *Shekel SQRN(3)*

   - $n = 2$,
   - $m = 3$,
   - $f(x) = -\sum_{j=1}^{m} \frac{1}{(x - A_j)^{\mathrm{T}}(x - A_j)^{\mathrm{T}} + c_j}$,
   - $l = 0, u = 10$.
   - $f_{\min} = -10.0860$

   Onde A_j é o vetor formado pelos n primeiros elementos da j–ésima linha da matriz A definida abaixo:

$$A = \begin{pmatrix} 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 \\ 8 & 8 & 8 & 8 \\ 6 & 6 & 6 & 6 \\ 3 & 7 & 3 & 7 \\ 2 & 9 & 2 & 9 \\ 5 & 5 & 3 & 3 \\ 8 & 1 & 8 & 1 \\ 6 & 2 & 6 & 2 \\ 7 & 3.6 & 7 & 3.6 \end{pmatrix}, \qquad c = \begin{pmatrix} 0.1 \\ 0.2 \\ 0.2 \\ 0.4 \\ 0.6 \\ 0.6 \\ 0.3 \\ 0.7 \\ 0.5 \\ 0.5 \end{pmatrix}.$$

Os três próximos problemas são como este, mas com $m = 5$, 7 e 10 respectivamente, e $n = 4$.

2. *Shekel SQRN(5)*

   - $f_{opt} = -10.1527$

3. *Shekel SQRN(7)*

   - $f_{\min} = -10.4023$

4. *Shekel SQRN(10)*

   - $f_{\min} = -10.5358$

5. *Goldstein–Price*

   - $n = 2$,
   - $f(x) = (1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2))(30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2))$,
   - $l = -2, u = 2$.
   - $f_{\min} = 3$

6. *Branin RCOS*

   - $n = 2$,
   - $f(x) = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right) + 10$,
   - $l^{\mathtt{T}} = (\ -5 \quad 0\ ), u^{\mathtt{T}} = (\ 10 \quad 15\ )$.
   - $f_{\min} = 0.397888$

7. *2-D Shubert*

- $n = 2,$
- $f(x) = \sum_{i=1}^{5} i\cos((i+1)x_1 + i) \sum_{i=1}^{5} i\cos((i+1)x_2 + i),$
- $l = -10, u = 10.$
- $f_{\min} = -186.730639$

8. *2-D Six-hump Camel Back*
   - $n = 2,$
   - $f(x) = \left(4 - 2.1x_1^2 + \frac{x_1^4}{3}\right) x_1 + x_1x_2 + 4(x_2^2 - 1)x_2^2,$
   - $l^{\mathsf{T}} = (\ -3 \quad -2\ ), u^{\mathsf{T}} = (\ 3 \quad 2\ ).$
   - $f_{\min} = -39.65$
9. *Easom*
   - $n = 2,$
   - $f(x) = -\cos(x_1)\cos(x_2)\exp(-(x_1 - \pi)^2 - (x_2 - \pi)^2),$
   - $l = -5, u = 5.$
   - $f_{\min} = -0.999999$
10. *Wingo (1985)*
    - $n = 1,$
    - $f(x) = x^6 - \frac{52}{25}x^5 + \frac{39}{80}x^4 + \frac{71}{10}x^3 - \frac{79}{20}x^2 - x + \frac{1}{10},$
    - $l = -2, u = 11.$
    - $f_{\min} = -7.58731$
11. *Moore (1979)*
    - $n = 1,$
    - $f(x) = \sum_{i=1}^{50} a_i x^i,$
    - $l = 1, u = 2.$
    - $f_{\min} = -663.5$

Onde $a = (-500.0, 2.5, 1.666666666, 1.25, 1.0, 0.8333333, 0.714285714,$
$0.625, 0.555555555, 1.0, -43.6363636, 0.41666666, 0.384615384,$
$0.357142857, 0.3333333, 0.3125, 0.294117647, 0.277777777, 0.263157894,$
$0.25, 0.238095238, 0.227272727, 0.217391304, 0.208333333, 0.2,$
$0.192307692, 0.185185185, 0.178571428, 0.344827586, 0.6666666,$
$-15.48387097, 0.15625, 0.1515151, 0.14705882, 0.14285712, 0.138888888,$
$0.135135135, 0.131578947, 0.128205128, 0.125, 0.121951219, 0.119047619,$
$0.116279069, 0.113636363, 0.1111111, 0.108695652, 0.106382978,$
$0.208333333, 0.408163265, 0.8)$.

12. *Wilkinson (1963)*

    - $n = 1$,
    - $f(x) = 0.000089248x - 0.0218343x^2 + 0.998266x^3 - 1.6995x^4 + 0.2x^5$,
    - $l = 0, u = 10$.
    - $f_{\min} = -443.67$

13. *Dixon e Szegö (1975)*

    - $n = 1$,
    - $f(x) = 4x^2 - 4x^3 + x^4$,
    - $l = -5, u = 5$.
    - $f_{\min} = 0$

14. *2-D Three-hump Camel Back*

    - $n = 2$,
    - $f(x) = 2x_1^2 - 1.05x_1^4 + \dfrac{x_1^6}{6} - x_1x_2 + x_2^2$,
    - $l = -5, u = 5$.
    - $f_{\min} = 0$

15. *Goldstein e Price (1971)*

    - $n = 1$,
    - $f(x) = x^6 - 15x^4 + 27x^2 + 250$,
    - $l = -5, u = 5$.
    - $f_{\min} = 7$

16. *Dixon (1990)*

- $n = 1$,
- $f(x) = x^4 - 3x^3 - 1.5x^2 + 10x$
- $l = -5, u = 5$.
- $f_{\min} = -7.5$

17. *Adjiman et al. (1998)*

    - $n = 2$,
    - $f(x) = \cos(x_1)\text{sen}(x_2) - \dfrac{x_1}{x_2^2 + 1}$,
    - $l = -1, u = (\ 2 \quad 1\ )$
    - $f_{\min} = -2.02181$

18. *Pseudo–etano*

    O objetivo deste exemplo é determinar a estrutura molecular do pseudo–etano, um etano onde todos os átomos de hidrogênio foram trocados por átomos de carbono, nitrogênio ou oxigênio.

    - $n = 1$,
    - $$f(x) = \frac{588600}{\left(3r_0^2 - 4\cos\theta r_0^2 - 2\left(\text{sen}^2\theta\cos\left(x - \frac{2\pi}{3}\right) - \cos^2\theta\right) r_0^2\right)^6} - \frac{1079.1}{\left(3r_0^2 - 4\cos\theta r_0^2 - 2\left(\text{sen}^2\theta\cos\left(x - \frac{2\pi}{3}\right) - \cos^2\theta\right) r_0^2\right)^3} + \frac{600800}{\left(3r_0^2 - 4\cos\theta r_0^2 - 2\left(\text{sen}^2\theta\cos x - \cos^2\theta\right) r_0^2\right)^6} - \frac{1071.5}{\left(3r_0^2 - 4\cos\theta r_0^2 - 2\left(\text{sen}^2\theta\cos x - \cos^2\theta\right) r_0^2\right)^3} + \frac{481300}{\left(3r_0^2 - 4\cos\theta r_0^2 - 2\left(\text{sen}^2\theta\cos\left(x + \frac{2\pi}{3}\right) - \cos^2\theta\right) r_0^2\right)^6} - \frac{1064.6}{\left(3r_0^2 - 4\cos\theta r_0^2 - 2\left(\text{sen}^2\theta\cos\left(x + \frac{2\pi}{3}\right) - \cos^2\theta\right) r_0^2\right)^3},$$
    - $l = 0, u = 2\pi$.
    - $f_{\min} = -1.0711$

    onde $r_0 = 1.54$ e $\theta = 1.9111$.

19. PERM(4,50)

    - $n = 4$, $\beta = 50$,

- $f(x) = \sum_{k=1}^{n} \left(\sum_{i=1}^{n} \left(i^k + \beta \right) \left(\left(\frac{x_i}{i} \right)^k - 1 \right) \right)^2$,
- $l = -n$, $u = n$.
- $f_{\min} = 0$

Os próximo problemas são como este mas com $(n, \beta) = (4, 0.5)$, $(10, 10^9)$ e $(10, 10^7)$, respectivamente.

20. PERM(4,0.5)
    - $f_{\min} = 0$
21. PERM(10,10^9)
    - $f_{\min} = 0$
22. PERM(10,10^7)
    - $f_{\min} = 0$
23. PERM0(4,10)
    - $n = 4$, $\beta = 10$,
    - $f(x) = \sum_{k=1}^{n} \left(\sum_{i=1}^{n} (i + \beta) \left(x_i^k - \left(\frac{1}{i} \right)^k \right) \right)^2$,
    - $l = -1$, $u = 1$.
    - $f_{\min} = 0$

    O próximo problema é como este mas com $n = 10$ e $\beta = 100$.

24. PERM0(10,100)
    - $f_{\min} = 0$
25. TRID
    - $n = 100$,
    - $f(x) = \sum_{i=1}^{n} (x_i - 1)^2 - \sum_{i=2}^{n} x_i x_{i-1}$
    - $l = -n^2$, $u = n^2$.
    - $f_{\min} = \frac{-n(n+4)(n-1)}{6}$
26. Rosenbrock

- $n = 100$,
- $f(x) = \sum_{j=1}^{n-1} 100 \left(x_j^2 - x_{j+1}\right)^2 + (x_j - 1)^2$,
- $l = -5$, $u = 10$.
- $f_{\min} = 0$

27. Zakharov

- $n = 100$,
- $f(x) = \left(\sum_{j=1}^{n} x_j^2\right) + \left(\sum_{j=1}^{n} \frac{j x_j}{2}\right)^2 + \left(\sum_{j=1}^{n} \frac{j x_j}{2}\right)^4$,
- $l = -5$, $u = 10$.
- $f_{\min} = 0$

28. Lennard-Jones

- $n = 9$, $m = 3$,
- $f(x) = \sum_{j=2}^{m} \sum_{i=1}^{j-1} r_{ij}$,
- $r_{ij} = (x_i - x_j)^2 + (x_{i+m} - x_{j+m})^2 + (x_{i+2m} - x_{j+2m})^2$,
- $l = -3$, $u = 3$.
- $f_{\min} = -3$

Apêndice B

Problemas Teste: Com restrições

A lista abaixo apresenta uma descrição dos problemas utilizados nos experimentos numéricos realizados. Os problemas foram extraídos de [20, 21].

Os problemas são todos do tipo

$$\begin{aligned} &\min_x f(x) \\ &\text{s.a.} \\ &g(x) \leq 0 \\ &h(x) = 0 \\ &l \leq x \leq u \end{aligned}$$

Onde $l, x, u \in \mathbb{R}^n$, $f\colon \mathbb{R}^n \to \mathbb{R}$, $g\colon \mathbb{R}^n \to \mathbb{R}^p$, e $g\colon \mathbb{R}^n \to \mathbb{R}^{m-p}$.
Os problemas foram extraídos de [14, 20, 21, 32].

1. *Hesse(1973)*

   - $n = 6$
   - $m = 6$
   - $f(x) = -25(x_1 - 2)^2 - (x_2 - 2)^2 - (x_3 - 1)^2 - (x_4 - 4)^2 - (x_5 - 1)^2 - (x_6 - 4)^2$
   - $l = (\ 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0\ )$
   - $u = (\ 10 \quad 10 \quad 5 \quad 6 \quad 5 \quad 10\ )$
   - $f_{\min} = -310$

   Restrições:

   $(x_3 - 3)^2 + x_4 \geq 4$
   $(x_5 - 3)^2 + x_6 \geq 4$

$x_1 - 3x_2 \leq 2$
$-x_1 + x_2 \leq 2$
$x_1 + x_2 \leq 6$
$x_1 + x_2 \geq 2$

2. *Luus(1974)*

   - $n = 3$
   - $m = 2$
   - $f(x) = -x_1^2 - x_2^2 - x_3^2$
   - $l = 2.3$
   - $u = 2.7$
   - $f_{\min} = -11.67664$

   Restrições:

   $4(x_1 - 0.5)^2 + 2(x_2 - 0.2)^2 + x_3^2 + 0.1x_1x_2 + 0.2x_2x_3 \leq 16$
   $2x_1^2 + x_2^2 - 2x_3^2 \geq 2$

3. *Luus and Jaakola(1973)*

   - $n = 3$
   - $m = 1$
   - $f(x) = (1.4609 + 0.15186x_1 + 0.00145x_1^2)x_2 + (0.8008 + 0.2031(50 - x_1) + 0.000916(50 - x_1)^2)x_3$
   - $l = (\ 25 \quad 0 \quad 0\ )$
   - $u = (\ 30 \quad 1 \quad 1\ )$
   - $f_{\min} = 3.0521$

   Restrições:

   $(1 - x_2)(1.5742 + 0.1631x_1 + 0.001358x_1^2) + (1 - x_3)(0.7266 + 0.2256(50 - x_1) + 0.000778(50 - x_1)^2) \leq 10$

4. *Ben-Tal(1994)*

   - $n = 7$
   - $m = 6$

- $f(x) = -(9 - 6x_1 - 16x_2 - 15x_3)x_4 - (15 - 6x_1 - 16x_2 - 15x_3)x_5 + x_6 - 5x_7$
- $l = 0$
- $u = (\ 1 \quad 1 \quad 1 \quad 1 \quad 200 \quad 100 \quad 200\ )$
- $f_{\min} = -450$

Restrições:

$x_3x_4 + x_3x_5 \leq 50$
$x_4 + x_6 \leq 100$
$x_5 + x_7 \leq 200$
$(3x_1 + x_2 + x_3 - 2.5)x_4 - 0.5x_6 \leq 0$
$(3x_1 + x_2 + x_3 - 1.5)x_5 + 0.5x_7 \leq 0$
$x_1 + x_2 + x_3 = 1$

5. *Ben-Tal(1994)*

- $n = 32$
- $m = 19$
- $f(x) = (-18 + 6x_1 + 16x_4 + 15x_7 + 12x_{10})x_{13} +$
  $(-18 + 6x_2 + 16x_5 + 15x_8 + 12x_{11})x_{14} +$
  $(-18 + 6x_3 + 16x_6 + 15x_9 + 12x_{12})x_{15} +$
  $(-15 + 6x_1 + 16x_4 + 15x_7 + 12x_{10})x_{16} +$
  $(-15 + 6x_2 + 16x_5 + 15x_8 + 12x_{11})x_{17} +$
  $(-15 + 6x_3 + 16x_6 + 15x_9 + 12x_{12})x_{18} +$
  $(-19 + 6x_1 + 16x_4 + 15x_7 + 12x_{10})x_{19} +$
  $(-19 + 6x_2 + 16x_5 + 15x_8 + 12x_{11})x_{20} +$
  $(-19 + 6x_3 + 16x_6 + 15x_9 + 12x_{12})x_{21} +$
  $(-16 + 6x_1 + 16x_4 + 15x_7 + 12x_{10})x_{22} +$
  $(-16 + 6x_2 + 16x_5 + 15x_8 + 12x_{11})x_{23} +$
  $(-16 + 6x_3 + 16x_6 + 15x_9 + 12x_{12})x_{24} +$
  $(-14 + 6x_1 + 16x_4 + 15x_7 + 12x_{10})x_{25} +$
  $(-14 + 6x_2 + 16x_5 + 15x_8 + 12x_{11})x_{26} +$
  $(-14 + 6x_3 + 16x_6 + 15x_9 + 12x_{12})x_{27}$
  $- 8x_{28} - 5x_{29} - 9x_{30} - 6x_{31} - 4x_{32}$
- $l = 0$
- $u = (1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 100\ 100\ 100\ 200\ 200\ 200$
  $100\ 100\ 100\ 100\ 100\ 100\ 100\ 200\ 100\ 100\ 100)$
- $f_{\min} = -3500$

Restrições:

$x_7(x_{13}+x_{16}+x_{19}+x_{22}+x_{25})+x_8(x_{14}+x_{17}+x_{20}+x_{23}+x_{26})+x_9(x_{15}+x_{18}+x_{21}+x_{24}+x_{27}) \leq 50$

$x_{13}+x_{14}+x_{15}+x_{28} \leq 100$
$x_{16}+x_{17}+x_{18}+x_{29} \leq 200$
$x_{19}+x_{20}+x_{21}+x_{30} \leq 100$
$x_{22}+x_{23}+x_{24}+x_{31} \leq 100$
$x_{25}+x_{26}+x_{27}+x_{32} \leq 100$

$(3x_1+x_4+x_7+1.5x_{10}-2.5)x_{13}+$
$(3x_2+x_5+x_8+1.5x_{11}-2.5)x_{14}+$
$(3x_3+x_6+x_9+1.5x_{12}-2.5)x_{15}-0.5x_{28} \leq 0$

$(x_1+3x_4+2.5x_7+2.5x_{10}-2)x_{13}+$
$(x_2+3x_5+2.5x_8+2.5x_{11}-2)x_{14}+$
$(x_3+3x_6+2.5x_9+2.5x_{12}-2)x_{15}+0.5x_{28} \leq 0$

$(3x_1+x_4+x_7+1.5x_{10}-1.5)x_{16}+$
$(3x_2+x_5+x_8+1.5x_{11}-1.5)x_{17}+$
$(3x_3+x_6+x_9+1.5x_{12}-1.5)x_{18}+0.5x_{29} \leq 0$

$(x_1+3x_4+2.5x_7+2.5x_{10}-2.5)x_{16}+$
$(x_2+3x_5+2.5x_8+2.5x_{11}-2.5)x_{17}+$
$(x_3+3x_6+2.5x_9+2.5x_{12}-2.5)x_{18} \leq 0$

$(3x_1+x_4+x_7+1.5x_{10}-2)x_{19}+$
$(3x_2+x_5+x_8+1.5x_{11}-2)x_{20}+$
$(3x_3+x_6+x_9+1.5x_{12}-2)x_{21} \leq 0$

$(x_1+3x_4+2.5x_7+2.5x_{10}-2.6)x_{19}+$
$(x_2+3x_5+2.5x_8+2.5x_{11}-2.6)x_{20}+$
$(x_3+3x_6+2.5x_9+2.5x_{12}-2.6)x_{21}-0.1x_{30} \leq 0$

$(3x_1+x_4+x_7+1.5x_{10}-2)x_{22}+$
$(3x_2+x_5+x_8+1.5x_{11}-2)x_{23}+$
$(3x_3+x_6+x_9+1.5x_{12}-2)x_{24} \leq 0$

$(x_1 + 3x_4 + 2.5x_7 + 2.5x_{10} - 2)x_{22} +$
$(x_2 + 3x_5 + 2.5x_8 + 2.5x_{11} - 2)x_{23} +$
$(x_3 + 3x_6 + 2.5x_9 + 2.5x_{12} - 2)x_{24} + 0.5x_{31} \leq 0$
$5\ (3x_1 + x_4 + x_7 + 1.5x_{10} - 2)x_{25} +$
$(3x_2 + x_5 + x_8 + 1.5x_{11} - 2)x_{26} +$
$(3x_3 + x_6 + x_9 + 1.5x_{12} - 2)x_{27} \leq 0$

$(x_1 + 3x_4 + 2.5x_7 + 2.5x_{10} - 2)x_{25} +$
$(x_2 + 3x_5 + 2.5x_8 + 2.5x_{11} - 2)x_{26} +$
$(x_3 + 3x_6 + 2.5x_9 + 2.5x_{12} - 2)x_{27} + 0.5x_{32} \leq 0$

$x_1 + x_4 + x_7 + x_{10} = 1$
$x_2 + x_5 + x_8 + x_{11} = 1$
$x_3 + x_6 + x_9 + x_{12} = 1$

6. *Murtagh and Saunders(1993)*

   - $n = 5$
   - $m = 3$
   - $f(x) = (x_1 - 1)^2 + (x_1 - x_2)^2 + (x_2 - x_3)^3 + (x_3 - x_4)^4 + (x_4 - x_5)^4$
   - $l = -5$
   - $u = 5$
   - $f_{\min} = 0.0293$

   Restrições:

   $x_1 + x_2^2 + x_3^3 = 3\sqrt{2} + 2$
   $x_2 - x_3^2 + x_4 = 2\sqrt{2} - 2$
   $x_1 x_5 = 2$

7. *Função objetivo quadrática 1*

   - $n = 5$
   - $m = 1$
   - $f(x) = 42x_1 + 44x_2 + 45x_3 + 47x_4 + 47.5x_5 - 50(x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2)$
   - $l = 0$
   - $u = 1$
   - $f_{\min} = -17$

Restrições:

$20x_1 + 12x_2 + 11x_3 + 7x_4 + 4x_5 \leq 40$

8. *Função objetivo quadrática 2*

   - $n = 20$
   - $m = 10$
   - $f(x) = -0.5 \sum_{i=1}^{20} (x_i - 2)^2$
   - $l = 0$
   - $u = 30$
   - $f_{\min} = -394.7506$

   Restrições:

   $Ax \geq b$

   Onde

$$b^{\mathrm{T}} = (-5, 2, -1, -3, 5, 4, -1, 0, 9, 40)$$

$$A^{\mathrm{T}} = \begin{pmatrix} -3 & 7 & 0 & -5 & 1 & 1 & 0 & 2 & -1 & 1 \\ 7 & 0 & -5 & 1 & 1 & 0 & 2 & -1 & -1 & 1 \\ 0 & -5 & 1 & 1 & 0 & 2 & -1 & -1 & -9 & 1 \\ -5 & 1 & 1 & 0 & 2 & -1 & -1 & -9 & 3 & 1 \\ 1 & 1 & 0 & 2 & -1 & -1 & -9 & 3 & 5 & 1 \\ 1 & 0 & 2 & -1 & -1 & -9 & 3 & 5 & 0 & 1 \\ 0 & 2 & -1 & -1 & -9 & 3 & 5 & 0 & 0 & 1 \\ 2 & -1 & -1 & -9 & 3 & 5 & 0 & 0 & 1 & 1 \\ -1 & -1 & -9 & 3 & 5 & 0 & 0 & 1 & 7 & 1 \\ -1 & -9 & 3 & 5 & 0 & 0 & 1 & 7 & -7 & 1 \\ -9 & 3 & 5 & 0 & 0 & 1 & 7 & -7 & -4 & 1 \\ 3 & 5 & 0 & 0 & 1 & 7 & -7 & -4 & -6 & 1 \\ 5 & 0 & 0 & 1 & 7 & -7 & -4 & -6 & -3 & 1 \\ 0 & 0 & 1 & 7 & -7 & -4 & -6 & -3 & 7 & 1 \\ 0 & 1 & 7 & -7 & -4 & -6 & -3 & 7 & 0 & 1 \\ 1 & 7 & -7 & -4 & -6 & -3 & 7 & 0 & -5 & 1 \\ 7 & -7 & -4 & -6 & -3 & 7 & 0 & -5 & 1 & 1 \\ -7 & -4 & -6 & -3 & 7 & 0 & -5 & 1 & 1 & 1 \\ -4 & -6 & -3 & 7 & 0 & -5 & 1 & 1 & 0 & 1 \\ -6 & -3 & 7 & 0 & -5 & 1 & 1 & 0 & 2 & 1 \end{pmatrix}$$

9. *Restrições quadráticas*

   - $n = 5$
   - $m = 6$
   - $f(x) = 37.293239x_1 + 0.8356891x_1x_5 + 5.3578547x_3^2 2 - 40792.141$
   - $l = \begin{pmatrix} 78 & 33 & 27 & 27 & 27 \end{pmatrix}$
   - $u = \begin{pmatrix} 102 & 45 & 45 & 45 & 45 \end{pmatrix}$
   - $f_{\min} = -30665.5387$

   Restrições:

   $-0.0022053x_3x_5 + 0.0056858x_2x_5 + 0.0006262x_1x_4 - 6.665593 \leq 0$
   $0.0022053x_3x_5 - 0.0056858x_2x_5 - 0.0006262x_1x_4 - 85.334407 \leq 0$
   $0.0071317x_2x_5 + 0.0021813x_3^2 + 0.0029955x_1x_2 - 29.48751 \leq 0$
   $-0.0071317x_2x_5 - 0.0021813x_3^2 - 0.0029955x_1x_2 + 9.48751 \leq 0$
   $0.0047026x_3x_5 + 0.0019085x_3x_4 + 0.0012547x_1x_3 - 15.699039 \leq 0$
   $-0.0047026x_3x_5 - 0.0019085x_3x_4 - 0.0012547x_1x_3 + 10.699039 \leq 0$

10.
   - $n = 2$
   - $m = 2$
   - $f(x) = -x_1 - x_2$
   - $l = 0$
   - $u = \begin{pmatrix} 3 & 4 \end{pmatrix}$
   - $f_{\min} = -5.5079$

   Restrições:

   $x_2 \leq 2x_1^4 - 8x_1^3 + 8x_1^2 + 2$
   $x_2 \leq 4x_1^4 - 32x_1^3 + 88x_1^2 - 96x_1 + 36$

11. *Encontrar o maior quadrado inscrito em um cubo*

   - $n = 13$
   - $m = 7$
   - $f(x) = -x_{13}$
   - $l = 0$
   - $u = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 3 \end{pmatrix}$
   - $f_{\min} = -1.125$

Restrições:

$(x_1 - x_4)^2 + (x_2 - x_5)^2 + (x_3 - x_6)^2 = x_{13}$
$(x_4 - x_7)^2 + (x_5 - x_8)^2 + (x_6 - x_9)^2 = x_{13}$
$(x_7 - x_{10})^2 + (x_8 - x_{11})^2 + (x_9 - x_{12})^2 = x_{13}$
$(x_1 - x_4)(x_4 - x_7) + (x_2 - x_5)(x_5 - x_8) + (x_3 - x_6)(x_6 - x_9) = 0$
$(x_1 - x_4)(x_1 - x_{10}) + (x_2 - x_5)(x_2 - x_{11}) + (x_3 - x_6)(x_3 - x_{12}) = 0$
$(x_{10} - x_7)(x_7 - x_4) + (x_{11} - x_8)(x_8 - x_5) + (x_{12} - x_9)(x_9 - x_6) = 0$
$(x_{10} - x_7)(x_{10} - x_1) + (x_{11} - x_8)(x_{11} - x_2) + (x_{12} - x_9)(x_{12} - x_3) = 0$

12. *Modelo de troca de calor 1*

- $n = 16$
- $m = 13$
- $f(x) = 1300 \left(\dfrac{1000}{0.05 \left(\frac{2}{3}\sqrt{x_1 x_2} + \frac{1}{6}(x_1 + x_2) \right)} \right)^{0.6} +$
  $1300 \left(\dfrac{600}{0.05 \left(\frac{2}{3}\sqrt{x_3 x_4} + \frac{1}{6}(x_3 + x_4) \right)} \right)^{0.6}$
- $l = \begin{pmatrix} 10 & 10 & 10 & 10 & 0 & 0 & 0 & 0 & 0 & 0 & 2.941 & 3.158 & 150 & 250 & 150 & 250 \end{pmatrix}$
- $u = \begin{pmatrix} 250 & 100 & 140 & 50 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 240 & 490 & 190 & 340 \end{pmatrix}$
- $f_{\min} = 56825$

Restrições:

$x_5 + x_6 \leq 10$
$x_5 + x_7 - x_{11} \leq 0$
$x_6 + x_8 - x_{12} \leq 0$
$x_9 + x_8 - x_{11} \leq 0$
$x_{10} + x_7 - x_{12} \leq 0$
$150x_5 + x_{16}x_7 - x_{13}x_{11} \leq 0$
$150x_6 + x_{14}x_8 - x_{15}x_{12} \leq 0$
$x_{11}(x_{14} - x_{13}) \leq 1000$
$x_{12}(x_{16} - x_{15}) \leq 600$
$x_1 + x_{14} \leq 500$
$x_2 + x_{13} \leq 250$
$x_3 + x_{16} \leq 350$
$x_4 + x_{15} \leq 200$

Referências Bibliográficas

[1] E.H.L. Aarts, A.E. Eiben e K.M. van Hee. A general theory of genetic algorithms. Technical report, Eindhoven University of Technology, Department of Mathematics and Computer Science, Holanda, 1989.

[2] E.H.L. Aarts e J.H.M. Korst. *Simulated Annealing and Boltzmann Machines.* John Wiley & Sons, 1989.

[3] E.H.L. Aarts, J.H.M. Korst e P.J.M. van Laarhoven. Simulated annealing. In E.H.L. Aarts e J.K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 91–120. John Wiley & Sons, 1997.

[4] E.H.L. Aarts e P.J.M. van Laarhoven. Statistical cooling: a general approach to combinatorial optimization problems. *Philips Journal of Research*, 40:193–226, 1985.

[5] R. Andreani, E.G. Birgin, J.M. Martínez e M.L. Schuverdt. Augmented lagrangian methods under the constant positive linear dependence constraint qualification. Technical Report MCDO-040806, Departamento de Matemática Aplicada, UNICAMP, Brasil, 2004.

[6] R. Andreani, E.G. Birgin, J.M. Martínez e M.L. Schuverdt. On augmented lagrangian methods with general lower-level constraints. Technical Report MCDO-050304, Departamento de Matemática Aplicada, UNICAMP, Brasil, 2005.

[7] R. Andreani, C. Dunder e J.M. Martínez. Order-Value Optimization: formulation and solution by means of a primal Cauchy method. *Mathematical Methods of Operations Research*, 58:387–399, 2003.

[8] R. Andreani, C. Dunder e J.M. Martínez. Nonlinear-programming reformulation of the Order-Value Optimization problem. *Mathematical Methods of Operations Research*, 61:365–384, 2005.

[9] R. Andreani, J.M. Martínez, M. Salvatierra e F. Yano. Global rder–Value Optimization by means of a multistart harmonic oscillator tunneling strategy. *Global Optimization: from Theory to Implementation.*

[10] R. Andreani, J.M. Martínez, M. Salvatierra e F. Yano. Quasi-Newton methods for Order–Value Optimization and Value-at-Risk calculations. *Pacific Journal of Optimization*, 2:11–33, 2006.

[11] T. Bäch, F. Hoffmeister e H.-P. Schwefel. A survey of evolution strategies. In R. Belew and L. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, San Mateo, CA, 1991. Morgan Kaufmann Publishers.

[12] E.G. Birgin e J.M. Martínez. Large–scale active–set box–constrained optimization method with spectral projected gradients. *Computational Optimization and Applications*, 23:101–125, 2002.

[13] M. F. Cardoso, R. L. Salcedo, S. Feyo de Azevedo e D. Barbosa. A simulated annealing to the solution of MINLP problems. *Computers and Chemical Engineering*, 1997.

[14] M. F. Cardoso, R. L. Salcedo e S. Feyo de Azevedo. The simplex–simulated annealing approach to continuous non–linear optimization. *Computers Chem. Engng.*, 20(9):1065–1080, 1995.

[15] V. Černy. Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45:41–51, 1985.

[16] R. Chelouah e P. Siarry. Tabu search applied to global optimization. *European Journal of Operational Research*, 2000.

[17] A. Corana, M. Marchesi, C. Martini e S. Ridella. Minimizing multimodal functions of continuous variable with the simulated annealing algorithm. *ACM Transactions on Mathematical Software*, 13(3):262–280, 1987.

[18] A.E. Eiben, E.H.L. Aarts e K.M. van Hee. Global convergence of genetic algorithms: a Markov chain analysis. In H.-P. Schwefel e R. Männer, editors, *Parallel Problem Solving from Nature*, volume 496 of Lecture Notes in Computer Science. Springer, Berlim, 1991.

[19] C. A. Floudas, A. Aggarwal e A. R. Ciric. Global optimum search for nonconvex NLP and MINLP problems. *Computers and Chemical Engineering*, 1989.

[20] C.A. Floudas e P.M. Pardalos. A collection of test problems for constrained global optimization algorithms. In G. Goos e J. Hartmanis, editors, *Lecture Notes in Computer Science*, volume 455. Springer–Verlag, 1990.

[21] C.A. Floudas e P.M. Pardalos. *Handbook of Test Problems in Local and Global Optimization*. Kluwer Academic Publishers, 1999.

[22] F. Glover. Tabu search: Part I. *ORSA Journal on Computing*, 1989.

[23] F. Glover. Tabu search: Part II. *ORSA Journal on Computing*, 1990.

[24] S. Kirkpatrick, C. D. Gelatt e M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[25] P.J.M. van Laarhoven e E.H.L. Aarts. *Simulated Annealing: Theory and Applications.* Kluwer Academic Publishers, 1987.

[26] J. Ma, P. Tian e D.-M. Zhang. Global optimization by Darwin and Boltzmann mixed strategy. *Computers & Operations Research*, 27:143–159, 2000.

[27] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller e E. Teller. Equations of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[28] Z. Michalewicz. *Genetic Algorithms + Data Structure = Evolution Programs.* Springer–Verlag, 1996.

[29] A. Neumaier. Some hard global optimization problems. *http://www.mat.univie.ac.at/∼neum/glopt/my_problems.html.*

[30] P. M. Pardalos, H. E. Romeijn e H. Tuy. Recent developments and trends in global optimization. *Journal of Computational and Applied Mathematics*, 124:209–228, 2000.

[31] H. S Ryoo e N. V. Sahinidis. Global optimization of nonconvex NLPs and MINLPs with applications in process design. *Computers Chemical Engineering*, 1995.

[32] R. L. Salcedo, M. J. Gonçalves e S. Feyo de Azevedo. And improved random-search algorithm for non–linear optimization. *Computers Chem. Engng.*, 14(10):1111–1126, 1990.

[33] H.-P. Schwefel. *Evolution & Optimum Seeking.* John Wiley & Sons, Cichester, UK, 1995.

[34] P. Siarry, G. Berthiau, F. Durbin e J. Haussi. Enhanced simulated annealing for globally minimizing functions for many-continuous variables. *ACM Transactions on Mathematical Software*, 1997.

[35] P. Siarry, G. Berthiau, F. Durbin e J. Haussy. Enhanced simulated annealing for globally minimizing functions of many continuous variables. *ACM Transactions on Mathematical Software*, 23(2):209–228, 1997.

[36] M. W. Trosset. What is simulated annealing? *Optimization and Engineering*, 2(2):201–213, 2001.