Relatório Científico

03/11695-8

Título: Algoritmo duas fases em otimização global.
Bolsista: Gabriel Haeser
Orientadora: Márcia Ap. Gomes–Ruggiero

Disciplinas Cursadas

Primeiro semestre de 2004

Análise Aplicada

Ementa: Espaços normados. Transformações lineares contínuas. Compacidade. Espaços de Banach. Espaços de Hilbert. Teorema do ponto fixo de Banach. Aplicações. Cálculo em Espaços de Banach. Aplicações.
Conceito: A

Matrizes

Ementa: Álgebra de matrizes. Métricas, sensibilidade. Eliminação Gaussiana. Ortogonalização. Sistemas lineares especiais. Autovalores. Métodos iterativos.
Conceito: A

Métodos computacionais de Álgebra linear

Ementa: Linguagens de programação. Análise numérica de sistemas algébricos. Análise de erro. Suporte computacional para o curso Matrizes.
Conceito: A

Segundo semestre de 2004

Análise Numérica I

Ementa: Teoria da aproximação em Espaços de Banach e de Hilbert. Melhor aproximação em Subespaços de Dimensão Finita. Interpolação Polinomial por partes. Construção de Espaços de Elementos Finitos. Método dos

Quadrados Mínimos. Diferenciação e Integração Numérica. Aplicações.
Conceito: A

Métodos Computacionais de Otimização

Ementa: Sistemas não lineares e minimizaçao sem restrições. Métodos numéricos. Convexidade e dualidade. Otimalidade em programação não linear. Métodos para minimização com restrições.
Conceito: A

Seminários de Otimização

Ementa: Apresentação dos trabalhos desenvolvidos pelo grupo de otimização do departamento.
Conceito: Disciplina não atribui conceito.

Tópicos em Programação Não-Linear

Ementa: Otimização em dois níveis.
Conceito: Disciplina não atribui conceito.

Verão de 2005

Medida e Integração

Ementa: Teoremas de extensão de medidas e integrais. Teoremas básicos de convergência. Medidas com sinal. Teorema de decomposição de Hahn-Jordan. Medidas absolutamente contínuas. Teorema de decomposição de Lebesgue. Teorema de Radón-Nikodym. Espaços Lp: propriedades básicas, dualidade. Espaços produto. Teorema de Fubini-Tonelli. Teorema de representação de Riesz-Markov. Convergência em medida. Relação entre diferenciação e integração: Teorema de Vitali; Teorema de diferenciação de Lebesgue.
Conceito: A+
Obs: Esta disciplina foi cursada no IMPA.

Primeiro semestre de 2005

Seminários de Otimização

Ementa: Apresentação dos trabalhos desenvolvidos pelo grupo de otimização do departamento, onde o bolsista apresentou um seminário sobre algoritmo genético e *simulated annealing*.
Conceito: Disciplina não atribui conceito.

Resumo do plano inicial

Neste período, agosto de 2004 a julho de 2005, além de cursar as disciplinas relativas ao curso de Mestrado era previsto o desenvolvimento do trabalho de tese: "Algoritmos duas fases em otimização global". Um resumo deste tema, segue abaixo: Em pesquisas na literatura especializada sobre aplicações de técnicas em otimização, principalmente na área de Engenharia Química, é possível constatar que um grande número de problemas resulta em modelos não lineares com funções não convexas e 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 que os algoritmos usuais em otimização dependem fortemente do ponto inicial e a sequência de aproximações converge a um ponto estacionário, ótimo local, que pode ou não ser o ótimo global.

A proposta nesta pesquisa é elaborar um algoritmo de otimização global do tipo duas–fases, isto é, ciclos que envolvem uma etapa onde são acionadas as heurísticas e uma etapa local onde é acionado um otimizador local.

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 de `GENCAN`, desenvolvido no grupo de Otimização.

Resumo do que foi realizado no período

Apresentamos a seguir um resumo das heurísticas estudadas (Algoritmo Genético, Esquemas Evolutivos e *Simulated Annealing*), seguido do novo algoritmo proposto (SA-GENCAN) e dos trabalhos atuais e perspectivas de trabalho futuro.

1 Algoritmo Genético

Algoritmo genético é utilizado para resolver problemas do tipo:

$$\begin{aligned} &\max_x f(x) \\ &\text{suj.} \quad l \leq x \leq u \end{aligned}$$

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

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.`

1.1 Representação

Escolhemos h_i, o tamanho da discretização do espaço de busca na i-ésima coordenada, para $i = 1, \ldots, n$. Em sequida 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.

1.2 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 será 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.

1.3 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:

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

1.4 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_{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.

1.5 Seleção

Calculamos o valor funcional f_k de cada cromossomo v_k, para $k = 1, \ldots, n_{\mathsf{pop}}$, 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:

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

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_{pop} vezes de modo que obtemos uma nova população de tamanho n_{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 [16].

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.

Com os esquemas evolutivos, é possível trabalhar com restrições de desigualdade gerais.

2.1 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.

2.2 Inicialização

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

2.3 Cruzamento

Fixamos o inteiro positivo $\mu \leq n_{\mathsf{pop}}/2$ 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.

2.4 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.

2.5 Seleção

Dos μ novos indivíduos produzidos, descartamos os que não satisfazem as restrições. Dentre os restantes e a população anterior escolhemos os n_{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 [16, 9, 17].

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 [13] e independentemente por Černy em 1985 [11], utilizando o algoritmo de Metropolis de 1953 [15].

Esta técnica é utilizada em problemas de otimização combinatória:

$$\begin{aligned} &\min_{x} f(x) \\ &\text{suj.} \quad x \in S \end{aligned}$$

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

O desenvolvimento a seguir é baseado em [2, 18, 19].

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 $V\colon S \to \mathbb{P}(S), i \mapsto V(i)$. 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 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 $T \to 0$

Para desenvolvermos 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.

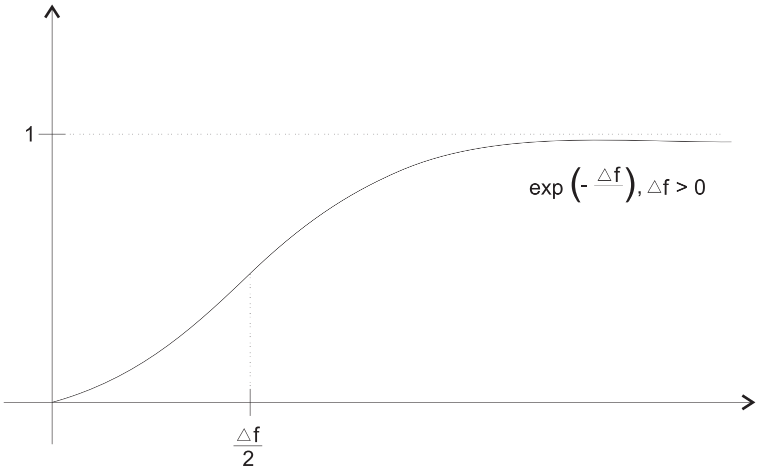

Figura 1: Gráfico de $\exp(-\frac{\Delta f}{T}), \Delta f > 0$ para T positivo. $\frac{\Delta f}{2}$ é o ponto que anula a segunda derivada.

3.1 Cadeias de Markov

Definição 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 $a^{(k)} = (a_1^{(k)}, a_2^{(k)}, \dots)^{\mathtt{T}}$ 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 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 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,

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

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

Definição 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 3.5 abaixo.

Teorema 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 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 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 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 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 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.

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 $\alpha_{ij} = \min \left\{ \dfrac{q_j}{q_i}, 1 \right\}$, 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 $\dfrac{q_j}{q_i}$.

Lema 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 3.5 $P_{ii}^{m} \geq P_{ij}^{k} P_{ji}^{l} > 0$, além disso, aplicando duas vezes o Teorema 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 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*

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

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 entre si e diferentes de i e j. Usando o Teorema 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 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 3.12 garantem as hipóteses do Teorema 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.

3.3 Distribuição de Boltzmann

Dada $f : S \to \mathbb{R}$, 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:

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

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

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

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

onde $\mathbb{I}(i \in A) = 1$ 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 $q_i > 0, \quad \forall\, i \in S$. 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,

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

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. Veja detalhes de esquemas de resfriamento em [2, 19, 3, 4].

4 Algoritmo Genético Generalizado

Em 1989, Eiben, Aarts e van Hee [1, 12] 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 resolve problemas de otimização combinatória:

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

Onde $f\colon S \to \mathbb{R}$, 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 4.1 *Uma função vizinhança $V : S \rightarrow \mathbb{P}(S)$, 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

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

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 4.2 *Uma função escolha $f_E : A \times \mathbb{P}(S) \rightarrow \mathbb{P}(P)$, 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 4.3 *Uma função produção $f_P : B \times P \rightarrow \mathbb{P}(S)$, 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 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 $\alpha, \beta, \gamma$ 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 algoritmo *simulated annealing* como caso particular do algoritmo genético abstrato. Seja

S arbitrário.

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 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,*

$$\forall\, i \in S, \quad \forall\, j \in S, \quad \exists\, p \geq 1 \text{ e } \exists\, l_0, l_1, \ldots, l_p \in S$$
$$\text{com } l_0 = i \text{ e } l_p = j \text{ tal que } l_{k+1} \in V(l_k)$$

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

Definição 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,*

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

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

Definição 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 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 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, $\mathcal{A}$ é uma σ–álgebra em Ω e $\mathbf{P}$ 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 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*

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

A demonstração deste teorema pode ser vista em [12].

O Teorema 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.

5 Algoritmo SA-GENCAN

Em 2000, Ma, Tian e Zhang apresentam o algoritmo DBMS (Darwin and Boltzmann Mixed Strategy). Este algoritmo resolve problemas de minimização não linear com restrições em caixas. A idéia é discretizar o espaço de busca e aplicar o *simulated annealing*, com um critério específico para a escolha dos vizinhos, além disso, a cada troca do valor da temperatura T, a solução é refinada com uma busca local particular, ver detalhes em [14].

No algoritmo que propomos para minimização em caixas, o *simulated annealing* é aplicado, com a diferença que 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 não vazio.

Fixado $\varepsilon > 0$, a vizinhança, ou ε–vizinhança de cada ponto x da caixa é definida como o conjunto dos pontos da caixa que distam no máximo ε de x, na norma infinito. A geração do vizinho é feita 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.

Como em DBMS, há uma busca local após cada troca de temperatura, para isso utilizamos o solver GENCAN em Fortran para problemas não lineares com restrições de caixa, um método de restrições ativas com gradiente espectral projetado, [10].

6 Testes Numéricos

6.1 Descrição do problema

Escolhemos o problema OVO (*Order-Value Optimization*), de acordo com [8, 7] para os testes numéricos. Resolvemos o problema utilizando as técnicas descritas acima. A seguir definimos o problema OVO.

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

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

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), $i = 1, \dots, m$, 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 $w_i = y(t_i, x^*) \quad \forall\, i = 1, 2, \dots m$.

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

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

A solução deste problema é x^* com valor funcional nulo.

6.2 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 2 vemos o valor funcional médio da população e na Figura 3 temos o valor funcional do melhor elemento da população em cada iteração/geração.

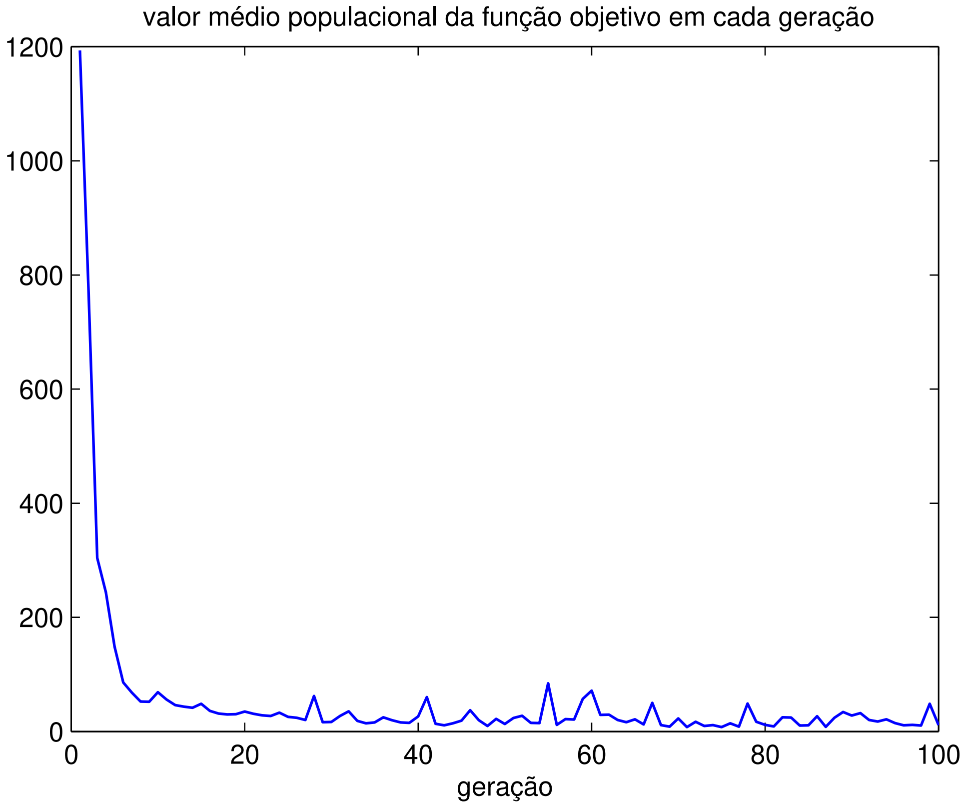

Figura 2: Desempenho médio da população (algoritmo genético).

Na tentativa de mesclar heurísticas para otimização global com solvers de otimização local, implementamos um algoritmo genético como acima, mas com a modificação de que, sempre que o menor elemento da população se mantém o mesmo por 5% do número total de iterações, executamos uma iteração do solver local GENCAN, a partir deste melhor elemento e substituimos a solução encontrada pelo pior elemento da população. Nomeamos este método de GA-GENCAN. Na Figura 4, marcamos com pequenos círculos as posições das chamadas do GENCAN.

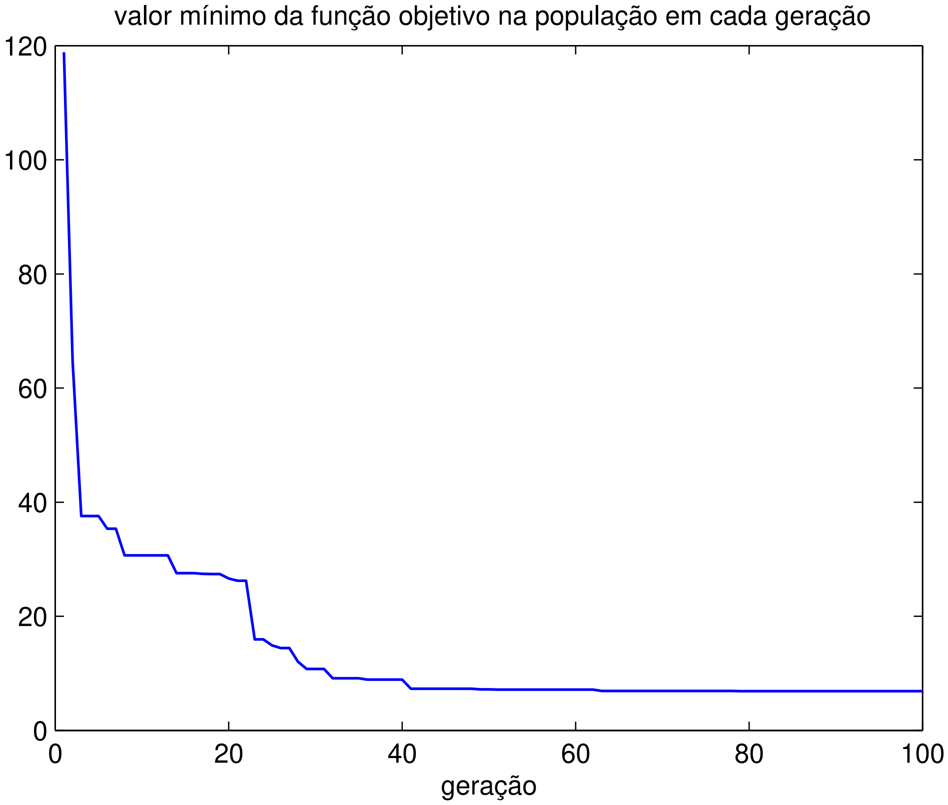

Figura 3: Valor funcional do melhor elemento da população (algoritmo genético).

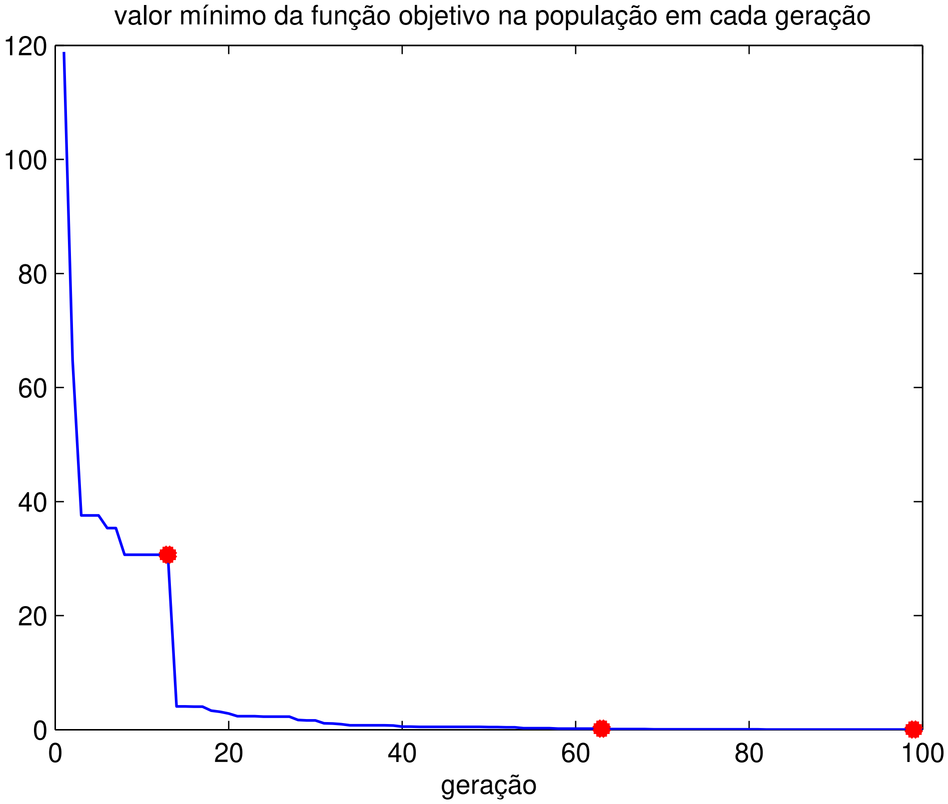

Figura 4: Algoritmo genético com acelerador local (GA-GENCAN).

Comparando as Figuras 3 e 4 podemos perceber que uma única chamada do solver local fez com que o método escapasse da estagnação, encontrando a solução global do problema.

6.3 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 5 vemos o valor funcional médio da população e na Figura 6 temos o valor funcional do melhor elemento da população por iteração/geração.

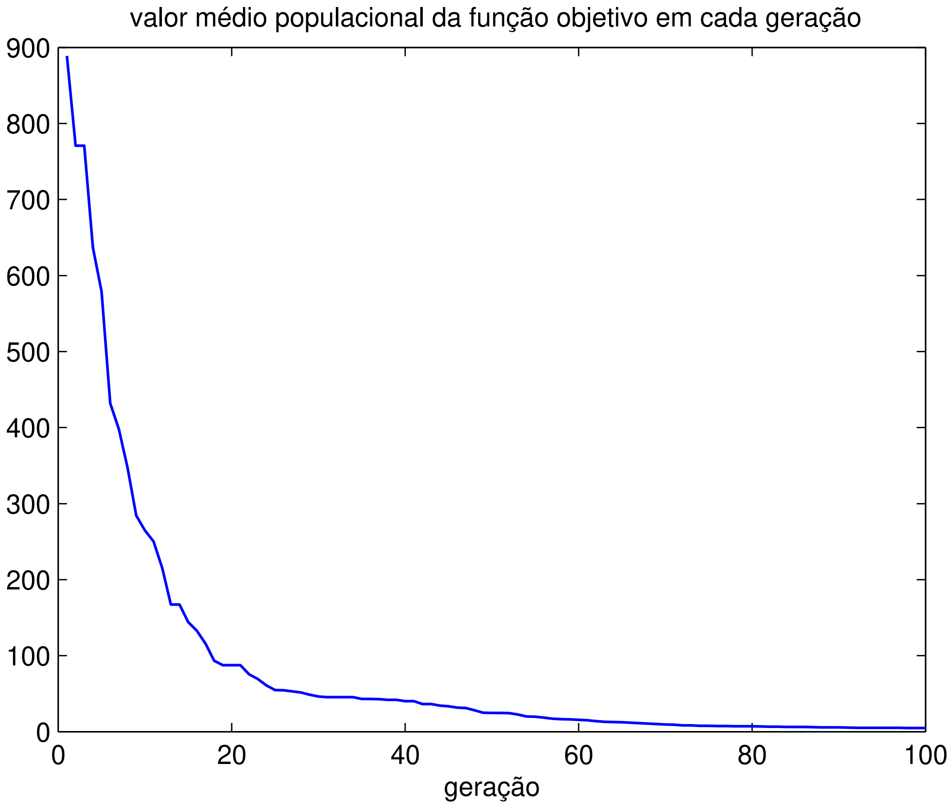

Figura 5: Desempenho médio da população (esquemas evolutivos).

Os resultados obtidos são semelhantes aos obtidos com algoritmo genético.

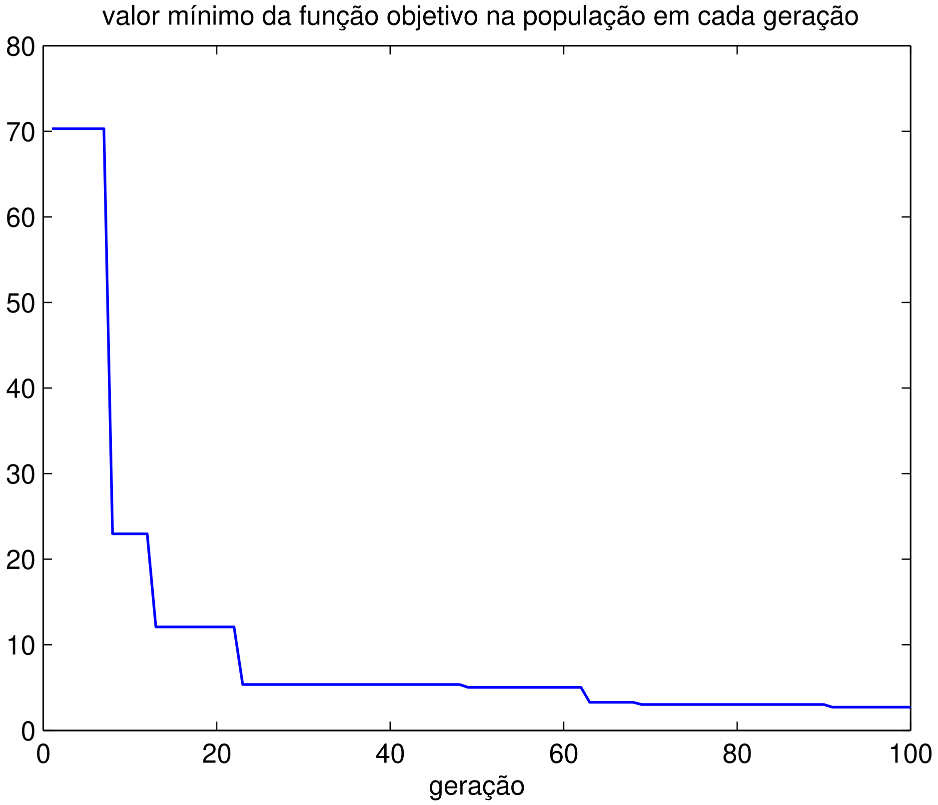

Figura 6: Valor funcional do melhor elemento da população (esquemas evolutivos).

Assim como em algoritmo genético, poderíamos utilizar um solver local para evitar estagnação nas iterações.

6.4 Simulated Annealing

Nesta implementação do *simulated annealing* usamos a temperatura inicial $T_0 = 100$ e a temperatura decresce de acordo com $T_{k+1} = T_k \frac{\log(100)}{\log(100+k)}$, conforme mostra a Figura 7.

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.

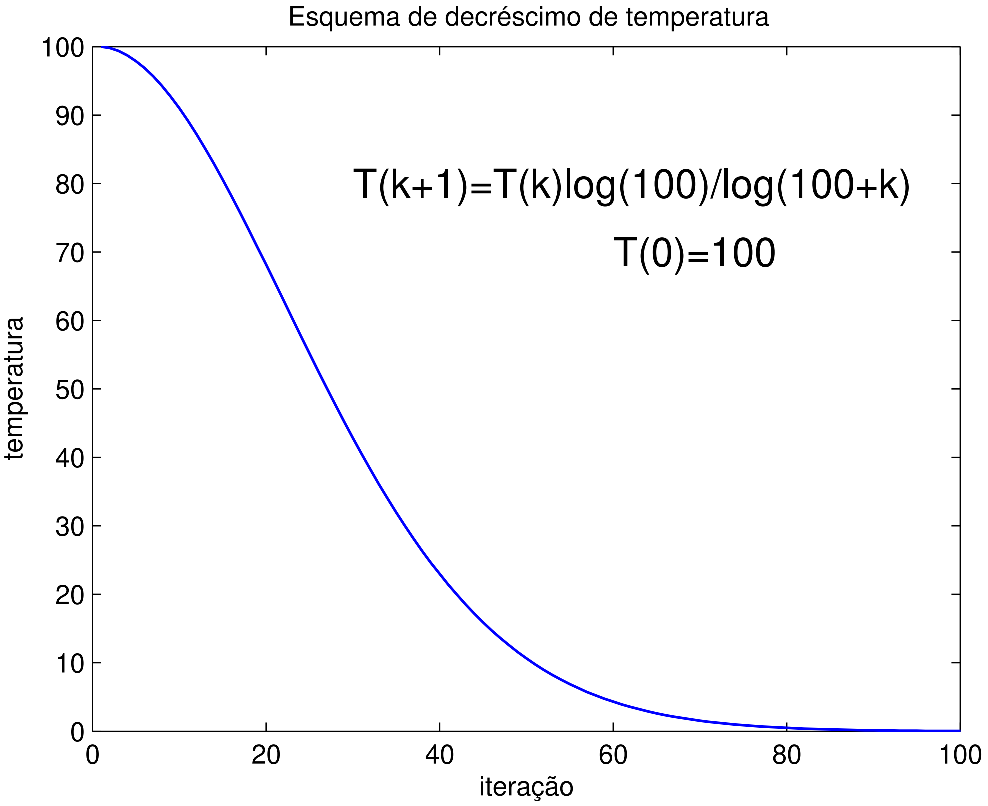

Figura 7: Decréscimo da temperatura.

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

Na Figura 8 temos a probabilidade média de aceitação de pontos que pioram o valor da função objetivo, e na Figura 9 temos o valor funcional de cada elemento da cadeia de Markov heterogênea gerada.

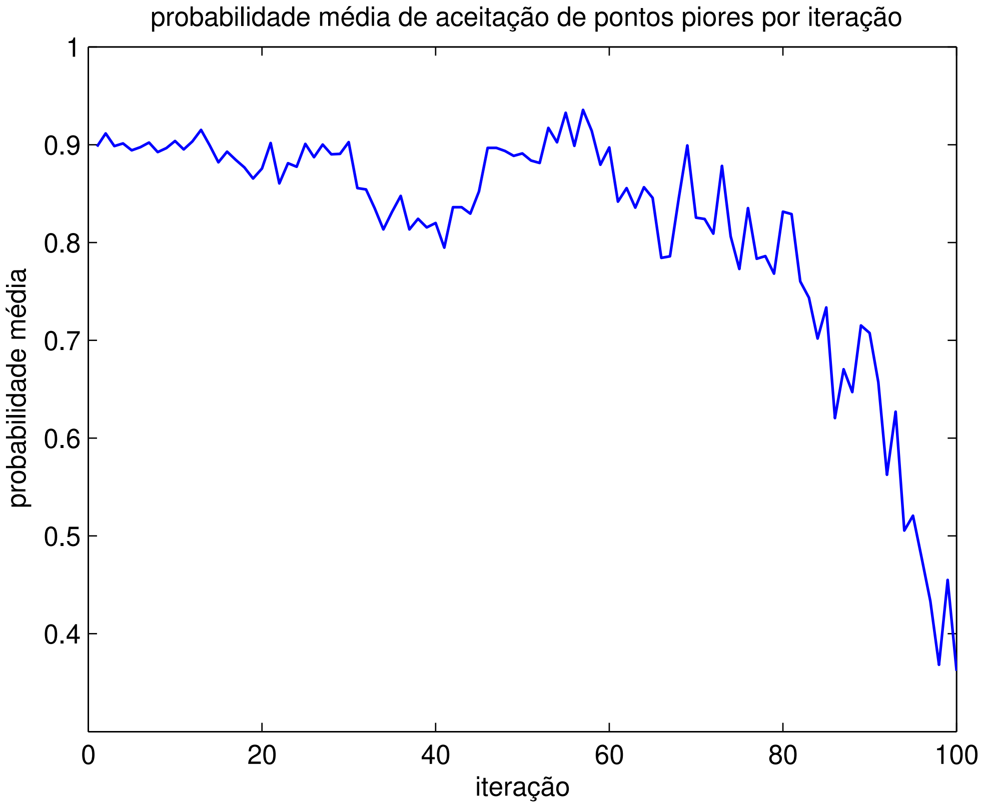

Figura 8: Probabilidade média de aceitação de pontos que pioram o valor da função objetivo.

Pela Figura 7 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 8, de que a probabilidade de aceitação de pontos piores é inicialmente próxima de um, e ao nos aproximarmos do critério de parada, esta probabilidade está próxima de zero.

Pela Figura 9 perebemos que de fato, nas iterações iniciais o método oscila entre pontos com melhores e piores valores funcionais, enquanto que nas iterações finais, pontos com valores funcionais piores não são aceitos.

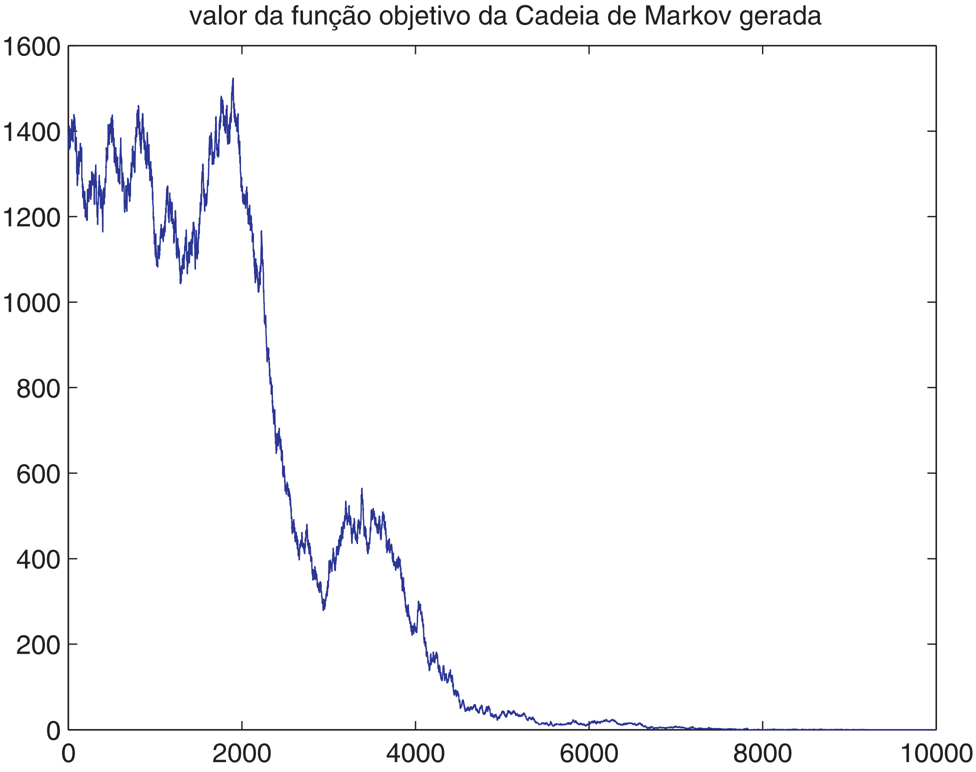

Figura 9: Valor funcional de cada elemento da cadeia de Markov gerada.

6.5 DBMS

Utilizamos os mesmos parâmetros utilizados em [14], 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 10 apresentamos os resultados.

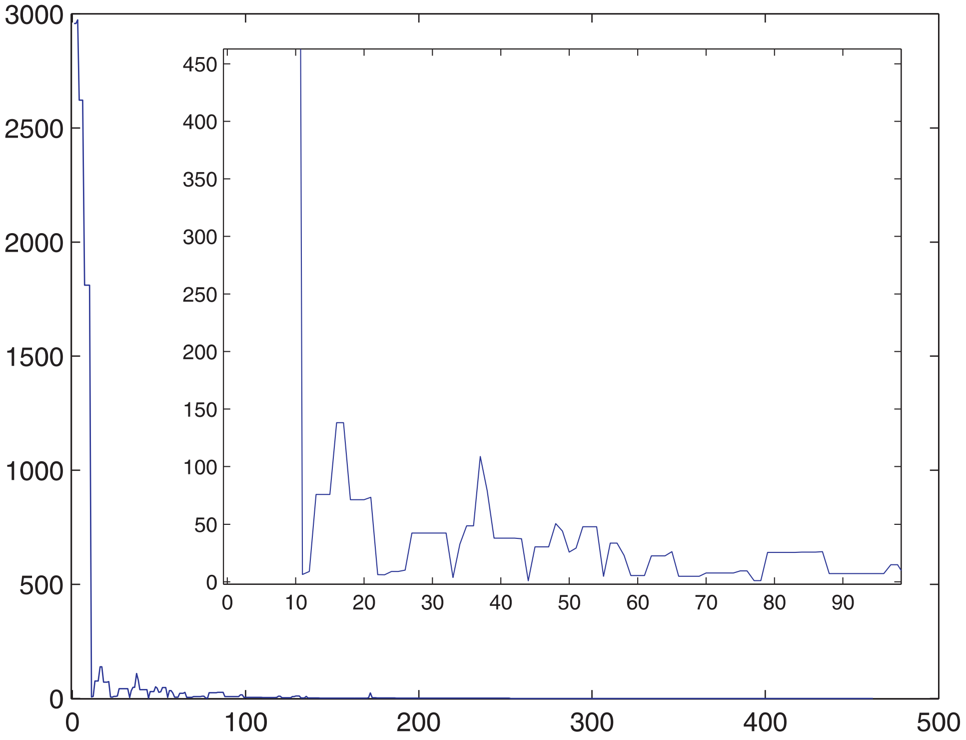

Figura 10: valores gerados por DBMS, e ampliação das primeiras iterações.

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

Detalhamento dos progressos realizados, resultados parciais obtidos

Os créditos para o a conclusão do curso já foram cumpridos em sua totalidade.

Atualmente estamos implementando SA-GENCAN em Fortran e testando sua performance em problemas clássicos da literatura, para isso estamos buscando um esquema de resfriamento adequado para o *simulated annealing*.

Em seguida pretendemos generalizar SA-GENCAN para que este resolva problemas gerais de programação não linear, com o auxílio do solver ALGENCAN [5, 6].

Pretendemos também utilizar GENCAN e/ou ALGENCAN juntamente com algoritmo genético, na procura de um método competitivo para otimização global.

Parte deste relatório será incluída na redação final da tese.

Plano de Trabalho e cronograma para a próxima etapa

Agosto:

- Realizar exame de qualificação.
- Finalizar testes dos algoritmos com problemas irrestritos.

Setembro-Outubro:

- Elaborar e implementar algoritmos do tipo SA-GENCAN e GA-GENCAN para problemas restritos.
- Realizar testes computacionais.
- Redigir o trabalho realizado nestes dois meses.

Novembro-Dezembro

- Finalizar teste computacionais.
- Finalizar redaçãoda tese.

Janeiro de 2006

- Concluir a tese.

Fevereiro de 2006

- Preparar material para defesa de tese.
- Defesa da tese em 28 de fevereiro.

Referências

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

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

[4] E.H.L. Aarts and 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, and 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, and 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, and 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, and J.M. Martínez. Nonlinear-programming reformulation of the Order-Value Optimization problem. *Mathematical Methods of Operations Research*, 61:365–384, 2005.

[9] T. Bäch, F. Hoffmeister, and 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.

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

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

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

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

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

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

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

[17] H.-P. Schwefel. *Evolution and Optimum Seeking.* John Wiley, Cichester, UK, 1995.

[18] M. W. Trosset. What is Simulated Annealing? *Optimization and Engineering*, 2(2):201–213, 2001.

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

Gabriel Haeser

Márcia Ap. Gomes–Ruggiero