UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

CENTRO DE CIÊNCIAS EXATAS E DA TERRA

PROGRAMA DE PÓS-GRADUAÇÃO EM MATEMÁTICA APLICADA E ESTATÍSTICA

JOSÉ CECÍLIO ROSA NETO

MODELAGEM DOS ALGORITMOS GENÉTICO SIMPLES E SIMULATED ANNEALING POR CADEIAS DE MARKOV

Natal, abril de 2010

Modelagem dos Algoritmos Genético Simples e Simulated Annealing por Cadeias de Markov

José Cecílio Rosa Neto

Dissertação apresentada ao Corpo Docente do Programa de Pós-Graduação em Matemática Aplicada e Estatística - CCET - UFRN, como requisito parcial para obtenção do título de Mestre em Matemática Aplicada e Estatística.

Área de Concentração: Probabilidade

Orientador: Prof. Dr. André Gustavo Campos Pereira

Natal, abril de 2010

Modelagem dos Algoritmos Genético Simples e Simulated Annealing por Cadeias de Markov

Este exemplar corresponde à redação final da dissertação devidamente corrigida, defendida por José Cecílio Rosa Neto e aprovada pela comissão julgadora.

Natal, abril de 2010

Banca examinadora:

- Prof. Dr. André Gustavo Campos Pereira
- Prof. Dr. Juan Rojas Cruz
- Profª Drª Michelli Karinne Barros da Silva

Resumo

Os Algoritmos Genético (AG) e o Simulated Annealing (SA) são algoritmos construídos para encontrar máximo ou mínimo de uma função que representa alguma característica do processo que está sendo modelado. Esses algoritmos possuem mecanismos que os fazem escapar de ótimos locais, entretanto, a evolução desses algoritmos no tempo se dá de forma completamente diferente. O SA no seu processo de busca trabalha com apenas um ponto, gerando a partir deste sempre um nova solução que é testada e que pode ser aceita ou não, já o AG trabalha com um conjunto de pontos, chamado população, da qual gera outra população que sempre é aceita. Em comum com esses dois algoritmos temos que a forma como o próximo ponto ou a próxima população é gerada obedece propriedades estocásticas. Nesse trabalho mostramos que a teoria matemática que descreve a evolução destes algoritmos é a teoria das cadeias de Markov. O AG é descrito por uma cadeia de Markov homogênea enquanto que o SA é descrito por uma cadeia de Markov não-homogênea, por fim serão feitos alguns exemplos computacionais comparando o desempenho desses dois algoritmos.
PALAVRAS CHAVE: Cadeias de Markov Homogêneas e Não-Homogêneas, Algoritmos Genético e Simulated Annealing.

Abstract

The Genetic Algorithms (GA) and the Simulated Annealing (SA) are algorithms used to find maximum (or minimum) value of a function which represents some special characteristic of the process that is being modelled. These algorithms have mechanisms that prevent them to get struck in local maximum (or minimum) values, however, the evolution of these algorithms in time works in a completely different way. The SA, on its search process, works with only one point, creating, from this point, a new solution which is tested and can be accepted or not, on the other hand the GA works with a set of points, called population, from which creates another population that is always accepted. The similarity between these two algorithms is the way the next point or next population is created obeying stochastic properties. On this work we show that the Markov chains theory describes the evolution of these algorithms. The GA is modelled by a stationary Markov's chain while the SA is modelled by a nonstationary one, in the end some computational examples are made comparing the performance of these two algorithms .

KEY WORDS: Stationary and nonstationary Markov chains, Genetic Algorithms and Simulates Annealing.

Agradecimentos

Agradeço a Deus por ter colocado no meu caminho pessoas tão especiais, muitas delas de formas diferentes, contribuíram para realização deste trabalho. A todas essas pessoas aqui o meu muito obrigado. No entanto, destaco:

A minha família, pois sem ela nada disso teria sido possível. Em especial minha mãe, que sempre acreditou em mim, meu tio Edgar, meu irmão Leandro, minha irmã Neidinha e minha sobrinha Paula, a vocês eu dedico essa vitória.

Ao meu orientador André Gustavo, pela sugestão do tema, pela orientação, cobrança, estímulo e dicas que levarei comigo por toda minha vida profissional.

A professora Dione, coordenadora do PPGMAE, pela atenção que sempre teve comigo e com os meus colegas alunos do mestrado.

As minhas colegas de graduação, hoje grandes amigas, Juliana e Lídia, que me deram a maior força para que eu fizesse esse curso.

Aos meus amigos Aline, Idevaldo, João Azevedo, Francilon e Tayana que mesmo estando longe se mostraram presentes me dando muita força.

Aos professores do mestrado, Damião, David, Jaques, Dione, Jeanete e Viviane pelos ensinamentos e ajuda.

Aos colegas do PPGMAE, Lenilson, Manassés, Carlos Gomes, Moisés, Aparecida, Tatiana, Enai, Rafael, Marcio André e João Batista. Patrícia pela companhia nos estudos aos sábados. Renata Rodrigues, Renata Fonseca, Allan e Daniel, que moramos juntos e hoje tenho como meus irmãos. Não posso deixar de falar da grande ajuda que meu amigo Daniel me deu na programação desses algoritmos.

Aos professores Juan e Michelli que participaram da banca examinadora.

Aos amigos que vieram para minha defesa e comemoraram junto comigo, Marcelo, Ana Hermínia, Thiago, Carlos Rafael, Aparecida, Moisés, Marconio, João Batista, André Gustavo, Dione, Pledson, Partícia, Daniel e Manu.

E aos funcionários do CCET.

Sumário

Capítulo 1

Introdução

Não é de hoje que o homem tenta emular os fenômenos naturais. Toda vez que alguém constrói um algoritmo que tenta imitar algum processo natural, utilizando suas propriedades com o objetivo de otimizar uma função (função custo) ou um processo, ele está desenvolvendo o que chamamos de heurística. Segundo Kirkpatrick (1983), existem duas estratégicas básicas para heurísticas: "dividir e conquistar" e melhoramentos iterativos. No primeiro, divide-se o problema em subproblemas de tamanho administrável, então resolve-se os subproblemas. As soluções dos subproblemas devem ser juntadas de alguma forma para obter a solução do problema original. Para este método produzir soluções muito boas, os subproblemas devem ser naturalmente disjuntos, e a divisão deve ser de forma apropriada a fim de que os erros cometidos na junção das soluções não prejudiquem os ganhos obtidos na aplicação de métodos mais poderosos à solução dos subproblemas. No melhoramento iterativo começamos com o sistema numa configuração conhecida. Uma operação de reorganização padrão é aplicada em todas as partes do sistema até que uma configuração reorganizada que melhore a função custo seja descoberta. A configuração reorganizada então se torna a nova configuração do sistema, e o processo continua até que nenhuma outra melhora seja encontrada. Melhoramento iterativo consiste de uma busca no espaço coordenado através de passos de reorganização das configurações, que nos leve para valores menores da função. Visto que esta busca usualmente fica presa em um mínimo local que não é

um mínimo global, é costume realizar várias vezes o processo, começando em diferentes configurações aleatoriamente geradas, e salvando o melhor resultado. Como uma outra alternativa, surgiram algoritmos que possuem mecanismos para impedir que essa busca fique presa em ótimos locais, dentre eles estão, o Algoritmo Genético (AG) e o Algoritmo *Simulated Annealing* (SA), os quais foram criados como heurísticas de processos naturais.

O AG introduzido por Holland (1975), é uma heurística baseada no processo de seleção natural de uma população. Neste processo a população passa por três fases: seleção, cruzamento e mutação. Após isso acontecer teremos uma nova população que também passará por estas três etapas até que uma condição de parada seja atingida.

Em Rudolph(1994) ele fala que o AG canônico não converge quase certamente para uma população que tenha o ponto de ótimo como um de seus pontos. Entretanto, considerando uma modificação neste algoritmo, a saber: o melhor ponto da população inicial é guardado em uma posição, e este ponto não participa como um ponto da população nas etapas de seleção, cruzamento e mutação. Além disso esse ponto é substituído se um ponto melhor que ele é encontrado nas próximas populações. O AG com essas modificações é chamado AG com elitismo. Neste mesmo artigo ele mostra que o AG com elitismo converge quase certamente para uma população que possui um ponto de ótimo dentro da população.

Existem muitos artigos que tratam do AG, cada um deles modificando a forma de gerar a próxima população, que alterando um critério de seleção, quer retornando uma das etapas e restringindo a forma de mutação definindo vizinhanças baseadas na quantidade de bits que permita mudar como em Dorea et. al.(2010). Com o elitismo ou sem elitismo como em Rudolph(1994). Modelando por cadeias de Markov de forma bem complicada, onde define-se operações com objetivo de estudar a convergência da cadeia, como em Nix and Vose (1992), Cerf(1996) and Cerf(1998), ou utilizando modelagem simples e intuitiva como em Rudolph(1994) e Dorea et. al.(2010) embora não entre no cálculo exato das condições suficientes sobre as etapas a fim de obter convergência.

O algoritmo SA segundo Tierney(1994) tem como base o algoritmo de Metropolis

apresentado no seu famoso artigo Metropolis(1953) o qual serve de referência para o desenvolvimento de vários outros algoritmos. Em Kirkpatrick(1983) ele explica que em Metropolis (1953) eles introduziram um algoritmo simples que pode ser usado para fornecer uma simulação eficiente de uma coleção de átomos em equilíbrio em uma dada temperatura. Em cada passo do algoritmo, é dado um deslocamento aleatório, ou seja, uma nova energia é gerada, e a mudança na energia do sistema ΔE é calculada. E essa nova energia é aceita como a nova configuração com uma probabilidade $P(\Delta E)$; caso contrário, a configuração original é usada para iniciar o próximo passo. Repetindo este passo muitas vezes, simulamos o movimento térmico dos átomos em contato com um banho de calor em temperatura T.

Usando a função objetivo no lugar da energia e definindo configurações como o conjunto de parâmetros $\{x_i\}$, temos com Metropolis o procedimento para gerar um população de configurações de um dado problema de otimização em alguma temperatura efetiva. Essa temperatura é simplesmente um parâmetro de controle. O processo de recozimento simulado consiste em primeiro "derreter" o sistema que está sendo otimizado numa temperatura alta, então diminuir a temperatura em estágios lentos até que o sistema "congele" e mais nenhuma mudança ocorra. Em cada temperatura, a simulação deve ocorrer tempo suficiente para que o sistema alcance o estado de equilíbrio. A sequência de temperaturas e o número de reorganizações $\{x_i\}$ utilizados para alcançar o equilíbrio em cada temperatura pode ser considerado a agenda de recozimento.

O recozimento, como implementado no procedimento de Metropolis, difere do melhoramento iterativo no que diz respeito a ficar preso em ótimos locais uma vez que transições para fora de ótimos locais são sempre possíveis em um temperatura não nula. Uma segunda e mais importante característica é que um tipo de "dividir e conquistar" ocorre. Características mais grosseiras dos estados do sistema aparecem em temperaturas mais altas; detalhes mais finos são desenvolvidos em temperaturas mais baixas.

Para o Algoritmo *Simulated Annealing*, Gibas (1985) fez o primeiro estudo deste algoritmo como uma cadeia de Markov não-homogênea, Anily and Federgruen (1987)

estudam este algoritmo para probabilidades de aceitação gerais, Cruz and Dorea (1998) apresentam condições simples para a convergência do algoritmo que generalizam os resultados de Anily and Federgruen (1987).

Este trabalho está estruturado em cinco capítulos. No capítulo dois apresentamos os conceitos e teoremas relacionados à teoria das cadeias de Markov homogêneas e não-homogêneas, os quais serão necessários para tratarmos da convergência dos algoritmos Genético e o *Simulated Annealing*. No capítulo três e quatro respectivamente, explicamos a dinâmica do Algoritmo Genético Simples, ilustrando as operações de seleção, cruzamento e mutação via exemplos, modelamos este AG por cadeias de Markov homogêneas, finalizamos demonstrando a convergência do Algoritmo Genético. Descrevemos o algoritmo *Simulated Annealing*, apresentando a modelagem por uma cadeia de Markov não-homogênea, e finalizamos dando condições que garantam a convergência. No capítulo cinco apresentamos as simulações feita usando o *software* livre R versão 2.9.1, com o intuito fazer comparações. Neste capítulo procuramos ilustrar a influência do tamanho da população, do domínio e do número de iterações no AG. Apresentamos o *Simulated Annealing* sob três formas de decrescimento de temperatura analisando a influência do número de pontos da discretização do domínio e do número de iterações, finalizamos apresentando nossas conclusões finais.

Capítulo 2

Cadeias de Markov

O objetivo deste capítulo é fazer uma breve revisão de algumas definições, conceitos e resultados da teoria das cadeias de Markov, os quais serão necessários para tratarmos da convergência dos Algoritmos Genético Simples e o *Simulated Annealing*. Como as cadeias de Markov homogêneas já são bem conhecidas nos limitaremos a apresentar os resultados que utilizaremos nos próximos capítulos. Já as cadeias de Markov não-homogêneas não são muito exploradas, por este motivo apresentaremos sua teoria com mais detalhes. Os leitores que desejarem se aprofundar nestes tópicos sugerimos como referência Isaacson and Madsen (1976).

Definição 2.0.1. *Um processo estocástico* $\{X_k\}_{k\in\mathbb{N}}$ *com espaço de estados* $S = \{1, 2, ...\}$ *satisfaz a propriedade de Markov se, para todo* $n \in \mathbb{N}$*, e todos estados* $i_1, i_2, \cdots$ *em S, temos:*

$$P(X_n = i_n | X_{n-1} = i_{n-1}, X_{n-2} = i_{n-2}, \cdots, X_1 = i_1) = P(X_n = i_n | X_{n-1} = i_{n-1})$$

Um processo a tempo discreto $\{X_k\}_{k\in\mathbb{N}}$*, com espaço de estados enumerável, que satisfaz a propriedade de Markov é chamado Cadeia de Markov.*

Este tipo de Processo Estocástico é também chamado de processo sem memória, uma vez que o passado é "esquecido"(desprezado).

As probabilidades condicionais $P(X_{n+1} = j | X_n = i)$ são denominadas **probabilidades de transição** e representam, portanto, a probabilidade do processo estar no

estado j no tempo $n+1$ dado que está no estado i no tempo n. Uma pergunta que surge naturalmente é se estas probabilidades dependem do tempo em que a transição do estado i para j acontece. Se essas probabilidades independem do tempo em que o passo está sendo dado dizemos que a cadeia de Markov é homogênea, caso contrário, temos uma cadeia de Markov não-homogênea.

2.1 Cadeias de Markov Homogênea

Definição 2.1.1. *Uma Cadeia de Markov é homogênea ou estacionária no tempo se a probabilidade de ir de um estado a outro é independente do tempo em que o passo é dado. Isto é, para todos os estados $i, j \in S$, temos:*

$$P(X_n = j | X_{n-1} = i) = P(X_{n+k} = j | X_{n+k-1} = i)$$

para $k = -(n-1), -(n-2), \cdots, -1, 0, 1, 2, \cdots$.Por outro lado, se a condição de estacionariedade falha, então a Cadeia é dita não homogênea ou não-estacionária.

Dada uma cadeia de Markov homogênea com espaço de estados $S = \{1, 2, ..., n\}$ podemos organizar as probabilidades p_{ij} numa matriz de $n \times n$ onde p_{ij} ocupa a i-ésima linha e a j-ésima coluna desta matriz. Esta matriz $P = (p_{ij})$ é chamada de **matriz de transição** de probabilidade da cadeia.

$$P = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1n} \\ p_{21} & p_{22} & \cdots & p_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{n1} & p_{n2} & \cdots & p_{nn} \end{bmatrix}$$

Esta matriz de transição tem as seguintes propriedades:

(i) Todas as entradas são não negativas.

(ii) A soma das entradas em cada linha é um.

A matriz de transição contém toda informação necessária para descrever o movimento da cadeia ao longo dos estados em S.

De maneira geral, se queremos as probabilidades de transição de n passos, essas probabilidades são dadas pela matriz P^n, onde os elementos de P^n são denotados por $p_{ij}^{(n)} = P(X_{n+k} = j|X_k = i)$.

Se estamos interessados em saber onde o processo está num tempo particular, devemos primeiro saber onde o processo começa.

Definição 2.1.2. *Um vetor $f^{(0)} = (f_1^{(0)}, f_2^{(0)}, ..., f_n^{(0)})$ dado por uma distribuição de probabilidade sobre os estados, é dito um vetor inicial se,*

$$f_i^{(0)} = P(X_0 = i) \geq 0, \text{ e } \sum_{i=1}^{n} f_i^{(0)} = 1.$$

Com a matriz de transição e o vetor inicial em mãos podemos calcular qualquer probabilidade sobre as variáveis do processo,

$$P(X_n = j) = (f^{(0)}P^n)_j$$

.

De maneira geral se queremos a distribuição de onde o processo está depois de n passos, dado que o vetor inicial foi $f^{(0)}$, simplesmente encontre,

$$f^{(n)} = f^{(0)}P^n$$

, onde o j-ésimo elemento de $f^{(n)}$ é dado por $f_j^{(n)} = P(X_n = j)$.

Uma ferramenta muito útil no estudo das cadeias de Markov é a equação de Chapman-Kolmogorov.

Teorema 2.1.1. *(Chapman-Kolmogorov) Para todos os inteiros não-negativos m e l temos:*

$$p_{ij}^{(m+l)} = \sum_{k \in S} p_{ik}^{(m)} p_{kj}^{(l)}. \tag{2.1}$$

2.1.1 Classificação de Estados

Definição 2.1.3. *Sejam i e $j \in S$. Um estado j diz-se acessível a partir do estado i ($i \to j$), se do estado i eu consigo atingir o estado j, ou seja, $\exists\, n \in \mathbb{N}$ tal que $p_{ij}^n > 0$.*

Definição 2.1.4. *Dizemos que i e j se comunicam se $i \to j$ e $j \to i$, ou seja, $\exists\, n \in \mathbb{N}$ tal que $p_{ij}^n > 0$ e $\exists\, k \in \mathbb{N}$ tal que $p_{ji}^k > 0$.*

Definição 2.1.5. *Quando todos os estados da cadeia se comunicam dizemos que a cadeia é **irredutível**.*

Definição 2.1.6. *Definimos o **período** de um estado como sendo,*

$$d(i) = mdc\{n \geq 1\,;\, p_{ii}^n > 0\}. \tag{2.2}$$

Se $d(i) = 1$, dizemos que i é aperiódico.

Proposição 2.1.1. *$i \leftrightarrow j \Leftrightarrow d(i) = d(j)$.*

Pela proposição (2.1.1), podemos concluir que se uma cadeia é irredutível ela é aperiódica ou todos os estados desta cadeia tem o mesmo período.

Definição 2.1.7. *Definimos o tempo da primeira visita do processo ao estado j como sendo,*

$$\tau_j = \inf\{n \geq 1\,;\, X_n = j\}. \tag{2.3}$$

Definição 2.1.8. *O número de visitas do processo ao estado j é dado por,*

$$\gamma(j) = \sum_{n=1}^{\infty} I_j(X_n), \tag{2.4}$$

onde $I_j(X_n) = 1$ se $X_n = j$, e zero caso contrário.

Definição 2.1.9. *Para quaisquer $i, j \in S$, a probabilidade de que partindo do estado i, a primeira visita do processo ao estado j se dê no passo n é dada por,*

$$f_{ij}^{(n)} = P(X_n = j, X_{n-1} \neq j, ..., X_1 \neq j | X_0 = i). \tag{2.5}$$

Definição 2.1.10. *Para quaisquer $i, j \in S$, a probabilidade de que partindo do estado i, a primeira visita do processo ao estado j se dê em um tempo finito é dada por,*

$$f_{ij} = P(\tau_j < \infty) = \sum_{n=1}^{\infty} f_{ij}^{(n)}. \tag{2.6}$$

Definição 2.1.11. *Dizemos que um estado j é:*

(i) Recorrente se $f_{jj} = 1$

(ii) Transiente se $f_{jj} < 1$.

Definição 2.1.12. *Se* $p_{jj}^1 = 1$, *chamamos* j *de estado absorvente.*

Teorema 2.1.2. *Seja* $j \in S$,

(i) j *é recorrente* $\Leftrightarrow \sum_{n=1}^{\infty} p_{jj}^n = +\infty$

(ii) j *é transiente* $\Leftrightarrow \sum_{n=1}^{\infty} p_{jj}^n < +\infty$.

Corolário 2.1.1. *Seja* $j \in S$,

(i) j *é recorrente* $\Leftrightarrow E_j(\gamma(j)) = +\infty$

(ii) j *é transiente* $\Leftrightarrow E_j(\gamma(j)) = \frac{f_{jj}}{1-f_{jj}}$.

Corolário 2.1.2. i *é recorrente e* $i \leftrightarrow j \Rightarrow j$ *é recorrente.*

Corolário 2.1.3. *Se* j *é transiente então* $P_j(\gamma(j) < +\infty) = 1$ *e* $E_i(\gamma(j)) = \frac{f_{ij}}{1-f_{jj}}$.

Observação 2.1.1. *Note que, se* j *é transiente,*

$$E_i(\gamma(j)) = \sum_{n=1}^{\infty} p_{ij}^n = \frac{f_{ij}}{1-f_{jj}} < \infty \Rightarrow \lim_{n\to\infty} p_{ij}^n = 0$$

, pois a série é convergente.

Se $\{X_n\}_{n\in\mathbb{N}}$ é uma Cadeia de Markov com espaço de estados finito, então podemos garantir que pelo menos um estado é recorrente?

Note que, se todas a entradas forem transientes,

$$\lim_{k\to\infty} \sum_{j=1}^{n} p_{ij}^k = 1 \neq \sum_{j=1}^{n} \lim_{k\to\infty} p_{ij}^k = 0$$

, que é uma contradição, portanto podemos garantir que pelo menos um estado é recorrente.

Definição 2.1.13. *Definimos o tempo médio de recorrência, ou seja, de quantos passos serão necessários em média para retornar a* j *como sendo,*

$$\mu_j = E(\tau_j) = \sum_{n=1}^{\infty} n f_{jj}^{(n)}. \tag{2.7}$$

Definição 2.1.14. *Seja j recorrente,*

(i) $\mu_j = E(\tau_j) < \infty \Rightarrow j$ *é recorrente positivo*

(ii) $\mu_j = E(\tau_j) = \infty \Rightarrow j$ *é recorrente nulo.*

Teorema 2.1.3. *Seja $j \in S$ recorrente, aperiódico e $i \in S$ tal que $i \leftrightarrow j$ então,* $\lim_{n\to\infty} p_{ij}^n = \frac{1}{\mu_j}$ *se* $\mu_j < \infty$ *e zero se* $\mu_j = \infty$.

Visto o Teorema 2.1.3 podemos concluir que, se temos uma cadeia de Markov, irredutível, aperiódica e recorrente positiva, então,

$$lim_{n\to\infty} p_{ij}^n = \frac{1}{\mu_j}$$

.

Corolário 2.1.4. *Toda cadeia irredutível e finita é recorrente positiva.*

2.1.2 Ergodicidade da Cadeia

Para encontrarmos o valor do $\lim_{n\to\infty} p_{ij}^n$ precisamos do valor $\mu_j = \sum_{n=1}^{\infty} n f_{jj}^{(n)}$, nem sempre isso é fácil ou possível de calcular.

Definição 2.1.15. *Seja P uma matriz de transição de uma cadeia de Markov. Se para todo $j \in S$ temos que* $\lim_{n\to\infty} p_{ij}^n = \pi_j$ *existe independentemente de $i \in S$ e* $\sum_{i\in S} \pi_i = 1$, *então dizemos que a cadeia é* ***ergódica****.*

Para n grande, a probabilidade $P(X_n = j)$ está próxima a um valor π_j, é como se a cadeia tivesse entrado em equilíbrio. Por isso $\{\pi_j; j \in S\}$ é chamada distribuição de equilíbrio ou distribuição limite de cadeia.

Definição 2.1.16. *Uma distribuição $\{\pi_j, j \geq 0\}$ é dita ser estacionária para a cadeia de Markov com matriz de transição $P = (p_{ij})_{i,j\in S}$, $S = \{1, 2, 3, ...\}$ se,*

$$\pi_j = \sum_{l\in S} \pi_l p_{lj}$$

para todo j. Em forma matricial,

$$\pi = \pi P = (\pi P)P = \pi P^2 = \pi P^3 = ...$$

Assim, $\pi = \pi P^n \ \forall n \in \mathbb{N}$.

Teorema 2.1.4. *Seja $\{X_n\}_{n\geq 0}$ uma cadeia de Markov, irredutível, aperiódica e recorrente positiva. Então $\{X_n\}_{n\geq 0}$ tem uma única distribuição estacionária π dada por $\pi_j = \frac{1}{\mu_j}$, $\forall j \in S$, onde os π_j's satisfazem a seguintes condições:*

$$\pi_j > 0, \quad \sum_{j\in S} \pi_j = 1 \quad \text{e} \quad \pi_j = \sum_{i\in S} \pi_i p_{ij}, \quad \forall \quad j \in S.$$

Corolário 2.1.5. *Toda cadeia de Markov homogênea irredutível, aperiódica com espaço de estados finito é convergente.*

Teorema 2.1.5. *Considere uma matriz estocástica P de ordem $n \times n$ com a decomposição $P^* = \begin{bmatrix} P_1 & 0 \\ Q & T \end{bmatrix}$ onde P_1 é matriz $n_1 \times n_1$ e T é matriz $n_2 \times n_2$. Suponha também que*

i) P_1 é irredutível e aperiódica.

ii) $\sum_{k=1}^{\infty} p_{ij}^k < \infty$ para todo $i \in C_2$, onde $C_2 = \{n_1 + 1, n_1 + 2, ..., n_1 + n_2\}$.

Então a matriz P é ergódica, mais ainda

$$\lim_{k\to\infty} p_{ij}^k = \begin{cases} \pi_j & se \quad j \in \{1, 2, ..., n_1\} \\ 0 & se \qquad j \in C_2 \end{cases}$$

O Lema 2.1.1 pode ser visto com detalhe em Pasturczak (2006).

Lema 2.1.1. *No teorema 2.1.5 podemos substituir a hipótese ii) pela condição*

$$ii^*) \ p_{ij} = \begin{cases} > 0 & se \quad i \in C_2 \, , \, j \in C_1 \\ 0 & se \quad i \in C_1 \, , \, j \in C_2 \end{cases}$$

onde $C_1 = \{1, 2, ..., n_1\}$ e $C_2 = \{n_1 + 1, ..., n_1 + n_2\}$.

2.2 Cadeias de Markov Não-Homogênea

2.2.1 Tipos de Comportamento Ergódico

Uma cadeia de Markov não-homogênea é descrita por um vetor inicial $f^{(0)}$ e uma sequência de matrizes de transição $\{P_k\}_{k=1}^{\infty}$.

Definição 2.2.1. *Seja $P_1, P_2, ...$ matrizes de transição de uma cadeia de Markov não-homogênea com vetor inicial $f^{(0)}$. Definimos:*

$$f^{(k)} = f^{(0)} * P_1 * P_2 * ... * P_k, \text{ e } f^{(m,k)} = f^{(0)} * P_{m+1} * P_{m+2} * ... * P_k$$

Definimos o j-ésimo elemento de $f^{(k)}$ por $f_j^{(k)} = P(X_k = j)$ e define-se o (i,j)-ésimo elemento de $P^{(m,k)} = P_{m+1} * P_{m+2} * ... * P_k$ por $p_{ij}^{(m,k)} = P(X_k = j|X_m = i)$. Analogamente para $g^{(k)}$, $g^{(m,k)}$ e $g_j^{(m,k)}$ quando usamos $g^{(0)}$ como o vetor inicial.

Estamos interessados em estudar o comportamento de $f^{(k)}$ e $f^{(m,k)}$, quando $k \to \infty$. É possível que o vetor $f^{(k)}$ convirja para o mesmo vetor q, independente do vetor inicial utilizado, isto é, o limite do vetor existe independentemente da escolha do vetor inicial. Quando isso ocorre, $f^{(k)}$, para k grande, nos dá pouca ou nenhuma informação sobre $f^{(0)}$, isto é, o efeito do vetor inicial é perdido com o tempo. Esta perda de informação sobre $f^{(0)}$, é conhecida como perda de memória. Quando $\lim_{k\to\infty} f^{(m,k)} = q$, para todo m, independentemente de $f^{(0)}$ temos convergência e perda de memória e diremos que a cadeia é fortemente ergódica. Por outro lado, a cadeia é dita fracamente ergódica, se tiver perda de memória em relação a distribuição inicial, ou seja, o efeito da distribuição inicial perde-se de modo que $f^{(k)}$ e $g^{(k)}$ está em algum sentido próximo, mas não necessariamente existe uma convergência de $f^{(k)}$ nem de $g^{(k)}$.

Como estamos trabalhando com convergência, precisamos de uma ferramenta que permita medir esta aproximação, para isso definimos as normas que serão utilizadas neste trabalho.

Definição 2.2.2. *Se $f = (f_1, f_2, ...)$ é um vetor, definimos a norma de f, por:*

$$\|f\| = \sum_{i=1}^{\infty} |f_i|$$

.

E se $A = (a_{ij})$ é uma matriz quadrada, definimos a norma de A como:

$$\|A\| = \sup_i \sum_{j=1}^{\infty} |a_{ij}|$$

.

Agora estamos aptos a formalizar os conceitos de convergência forte e fraca.

Definição 2.2.3. *uma cadeia de Markov não homogênea é chamada de* ***fracamente ergódica*** *se para todo m*

$$\lim_{k\to\infty} \sup_{f^{(0)},g^{(0)}} \left\| f^{(m,k)} - g^{(m,k)} \right\| = 0. \tag{2.8}$$

Definição 2.2.4. *A cadeia de Markov não-homogênea é chamada de* ***fortemente ergódica*** *se existir um vetor $q = (q_1, q_2, ...)$, onde $\|q\| = 1$ e $q \geq 0$, para $i = 1, 2, 3, ...$ tal que para todo m*

$$\lim_{k\to\infty} \sup_{f^{(0)}} \left\| f^{(m,k)} - q \right\| = 0. \tag{2.9}$$

Observação 2.2.1. *A norma utilizada é mais restritiva que a convergência coordenada a coordenada. Não é suficiente ter $\lim_{n\to\infty} \left| f_i^{(m,n)} - g_i^{(m,n)} \right| = 0$ ou $\lim_{n\to\infty} \left| f_i^{(m,n)} - q_i \right| = 0$ (isto é, convergência coordenada a coordenada). A fim de ter ergodicidade fraca ou forte, é necessário ter:*

$$\lim_{n\to\infty} \sum_{i=1}^{\infty} \left| f_i^{(m,n)} - g_i^{(m,n)} \right| = 0 \tag{2.10}$$

$$\lim_{n\to\infty} \sum_{i=1}^{\infty} \left| f_i^{(m,n)} - q_i \right| = 0. \tag{2.11}$$

Exemplo 1. Seja $f^{(n)} = (\underbrace{1/n, 1/n, ..., 1/n}_{n}, 0, 0, ...)$, onde as primeiras n coordenadas de $f^{(n)}$ são $1/n$, note que $\lim_{n\to\infty} f_i^{(n)} = 0$ para todo i, o limite que $f^{(n)}$ é o vetor nulo. Entretanto, $\left\| f^{(n)} - \varphi \right\| = 1$ para todo n, onde $\varphi = (0, 0, 0, ...)$. Assim $\{f^{(n)}\}$ não converge em norma para o vetor nulo. □

Geralmente é bastante difícil mostrar diretamente das definições que a cadeia é ergódica fraca ou forte. A principal dificuldade decorre de considerar o supremo sobre todos os possíveis vetores iniciais. Os seguintes exemplos foram criados para contornar esta dificuldade, e ilustram bastante esses dois tipos de comportamento ergódico.

Exemplo 2. Seja $\{X_n\}$ uma cadeia de Markov com matrizes de transição,

$$P_{2n-1} = \begin{bmatrix} 1-\frac{1}{2n-1} & \frac{1}{2n-1} \\ \\ 1-\frac{1}{2n-1} & \frac{1}{2n-1} \end{bmatrix}$$

$$P_{2n} = \begin{bmatrix} \frac{1}{2n} & 1-\frac{1}{2n} \\ \\ \frac{1}{2n} & 1-\frac{1}{2n} \end{bmatrix}$$

para $n = 1, 2, ...$, note que, para qualquer vetor inicial $f^{(0)}$, temos

$$f^{(m,k)} = (1-\tfrac{1}{k}, \tfrac{1}{k}), \text{ se } k \text{ é ímpar,}$$

$$= (\tfrac{1}{k}, 1-\tfrac{1}{k}) \text{ se k for par}$$

Assim a cadeia é fracamente ergódica mas não fortemente ergódica. □

Exemplo 3. Seja $\{X_n\}$ uma cadeia de Markov com matrizes de transição

$$P_n = \begin{bmatrix} \frac{1}{2}-\frac{1}{n+1} & \frac{1}{2}+\frac{1}{n+1} \\ \\ \frac{1}{2}-\frac{1}{n+1} & \frac{1}{2}+\frac{1}{n+1} \end{bmatrix}$$

para $n = 1, 2, ...$ para qualquer vetor inicial temos

$$f^{(m,k)} = \left(\tfrac{1}{2}-\tfrac{1}{1+k}, \tfrac{1}{2}+\tfrac{1}{k+1}\right) \to \left(\tfrac{1}{2}, \tfrac{1}{2}\right), \text{ para } k \to \infty.$$

pois a multiplicação de duas matrizes constante é uma matriz constante.

Note que,

$$\begin{aligned}\lim_{k\to\infty}\sup_{f^{(0)}}\left\|f^{(m,k)}-q\right\| &= \lim_{k\to\infty}\left\|\left(\frac{1}{2}-\frac{1}{1+k},\frac{1}{2}+\frac{1}{k+1}\right)-\left(\frac{1}{2},\frac{1}{2}\right)\right\| \\ &= \lim_{k\to\infty}\left\|\left(-\frac{1}{k+1},\frac{1}{k+1}\right)\right\| \\ &= \lim_{k\to\infty}\left(\left|-\frac{1}{k+1}\right|+\left|\frac{1}{k+1}\right|\right) \\ &= 0.\end{aligned}$$

Assim a cadeia é fortemente ergódica. □

2.2.2 Coeficiente Ergódico

A fim de estudar a ergodicidade fraca introduzimos o coeficiente ergódico.

Definição 2.2.5. *Seja P uma matriz estocástica. O coeficiente ergódico de P, denotado por $\alpha(P)$, é definido por,*

$$\alpha(P) = 1 - \sup_{i,k}\sum_{j=1}^{\infty}[p_{ij}-p_{kj}]^{+} \tag{2.12}$$

onde $[p_{ij}-p_{kj}]^{+} = \max(0, p_{ij}-p_{kj})$.

Lema 2.2.1. *O coeficiente ergódico satisfaz as seguintes condições:*

(i) $0 \le \alpha(P) \le 1$

(ii) $\alpha(P) = 1 - (\frac{1}{2})\sup_{i,k}\sum_{j=1}^{\infty}|p_{ij}-p_{kj}|$

Lema 2.2.2. *Seja P uma matriz estocástica. Então:*

$$\alpha(P) = \inf_{i,k}\sum_{j=1}^{\infty}\min(p_{ij}, p_{kj})$$

.

Na prática usamos o chamado coeficiente δ de Dobrushin . Dado por:

$$\delta(P) = 1 - \alpha(P)$$

Da Definição 2.2.5 e dos Lemas 2.2.1 e 2.2.2, temos que o coeficiente δ satisfaz

$$\begin{aligned}\delta(P) &= \sup_{i,k}\sum_{j=1}^{\infty}[p_{ij}-p_{kj}]^{+} \\ &= \left(\frac{1}{2}\right)\sup_{i,k}\sum_{j=1}^{\infty}|p_{ij}-p_{kj}| \\ &= 1-\inf_{i,k}\sum_{j=1}^{\infty}\min(p_{ij},p_{kj}).\end{aligned}$$

Na primeira igualdade acima vem que o coeficiente $\delta(P)$ mede a menor distância entre todas as linhas de uma matriz P. Se $\delta(P)=0$ significa que P possui todas as linhas iguais, ou seja, $p_{ij}=p_{kj}\,\forall\, i,k,j\in S$.

Veremos então que δ não é usado para medir o quão próximo a matriz está de zero, mas é o bastante para medir o quão próximo uma matriz não-negativa está de uma matriz constante. Veremos que $\{P_n\}$ é fracamente ergódica se e somente se $P_mP_{m+1}...P_n$ é próximo de uma matriz constante para n grande e δ será usado para medir com exatidão.

Lema 2.2.3. *Se P e Q são matrizes estocásticas, então*

$$\delta(QP)\leq\delta(Q)\delta(P)$$

Exemplo 4. Encontre $\delta(P)$, $\delta(Q)$, $\delta(PQ)$, $\delta(QP)$ e $\|Q-P\|$ onde

$$P=\begin{bmatrix}.1 & .3 & .6\\ 0 & .5 & .5\\ .4 & .6 & 0\end{bmatrix}\text{ e }Q=\begin{bmatrix}0 & .4 & .6\\ .5 & .5 & 0\\ .3 & .4 & .3\end{bmatrix}$$

$\delta(P)=1-\inf_{i,k}\sum_{j=1}^{3}\min(p_{ij},p_{kj})$

Note que,

para $i=1$ e $k=2$ temos: $\min(0.1,0)+\min(0.3,0.5)+\min(0.6,0.5)=0+0.3+0.5=0.8$

para $i = 1$ e $k = 3$ temos: 0.1+0.3+0=0.4

para $i = 2$ e $k = 3$ temos: 0+0.5+0=0.5

Assim,

$$\inf_{i,k} \sum_{j=1}^{\infty} \min(p_{ij}, p_{kj}) = \inf(0.8, 0.4, 0.5) = 0.4$$

Portanto,

$$\delta(P) = 1 - 0.4 = 0.6$$

Analogamente encontramos $\delta(Q) = 0.6$, $\delta(PQ) = 0.12$ e $\delta(QP) = 0.35$ onde

$$QP = \begin{bmatrix} .24 & .56 & .20 \\ .05 & .40 & .55 \\ .15 & .47 & .38 \end{bmatrix} \text{ e } PQ = \begin{bmatrix} .33 & .43 & .24 \\ .40 & .45 & .15 \\ .30 & .46 & .24 \end{bmatrix}$$

É importante observar que a desigualdade do Lema 2.2.3 é satisfeita.

Sendo $A = P - Q$ a matriz

$$A = \begin{bmatrix} -0.1 & 0.1 & 0 \\ 0.5 & 0 & -0.5 \\ -0.1 & -0.2 & 0.3 \end{bmatrix}$$

Temos,

$$\|P - Q\| = \|A\| = \sup_i \sum_{j=1}^{3} |a_{ij}| = 1$$

Lema 2.2.4. *Se P é uma matriz estocástica e se R é alguma matriz onde $\sum_{k=1}^{\infty} r_{ik} = 0$ para todo i e $\|R\| < \infty$, então*

$$\|RP\| \leq \|R\| \, \delta(P)$$

É bom observar que a ordenação das matrizes R e P do Lema 2.2.4 é crucial. O próximo exemplo mostra que se a matriz, R, com a soma das linhas iguais a zero não é à esquerda, então a desigualdade pode falhar.

Exemplo 5. Sejam as matrizes R e P definidas a seguir

$$P = \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} \; R = \begin{bmatrix} 1 & -1 \\ 0 & 0 \end{bmatrix}$$

$PR = \begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix}$, $\|R\| = 2$, $\delta(P) = 0$ e $\|PR\| = 2$. Dessa forma a desigualdade do lema anterior não é satisfeita.

Lema 2.2.5. *Para quaisquer que sejam as matrizes A e B, a seguinte inequação é válida*

$$\|AB\| \leq \|A\| \, \|B\|$$

2.2.3 Ergodicidade Fraca

Nesta seção veremos alguns teoremas em que o coeficiente ergódico e a norma são usados para garantir a ergodicidade fraca da cadeia. Em muitos casos a utilização de um desses teoremas será mais fácil que a definição original.

Teorema 2.2.1. *Uma cadeia de Markov não-homogênea é fracamente ergódica se, e somente se, para todo m, $\delta(P^{(m,k)}) \to 0$ quando $k \to \infty$.*

No uso do Teorema 2.2.1 para estabelecer ergodicidade fraca, por vezes é conveniente usar o Lema 2.2.3, isto é, desde que $\delta(P^{(m,k)}) \leq \prod_{j=m+1}^{k} \delta(P_j)$ é suficiente mostrar que esse produto vai para zero para todo m.

Exemplo 6. Seja $\{X_n\}$ uma cadeia de Markov não-homogênea com matrizes de transição definidas por

$$P_{2n-1} = \begin{bmatrix} .1 & .3 & .6 \\ 0 & .5 & .5 \\ .4 & .6 & 0 \end{bmatrix} \text{ e } P_{2n} = \begin{bmatrix} 0 & .4 & .6 \\ .5 & .5 & 0 \\ .3 & .4 & .3 \end{bmatrix} \text{ para todo } n = 1, 2, 3...$$

Note que $\delta(P_{2n-1}) = 0.6$ e $\delta(P_{2n}) = 0.6$. Assim,

$\delta(P^{(m,k)}) \leq \prod_{j=m+1}^{k} \delta(P_j) = \prod_{j=m+1}^{k} 0.6 = (0.6)^{k-m-1} \to 0$ quando $k \to \infty$, portanto $\delta(P^{(m,k)}) \to 0$ quando $k \to \infty$ e dizemos que a cadeia é fraca ergódica. □

Em algumas situações é preciso ter bastante cuidado na hora da utilização dessa simplificação, como pode ser observado no exemplo a seguir.

Exemplo 7. Seja $\{X_n\}$ uma cadeia de Markov não homogênea com matrizes de transição

$$P_{2n-1} = \begin{bmatrix} 0 & 1/2 & 1/2 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \text{ e } P_{2n} = \begin{bmatrix} 0 & 1 & 0 \\ 1/2 & 0 & 1/2 \\ 0 & 1 & 0 \end{bmatrix} \text{ para todo } n = 1, 2, 3...$$

Note que $\delta(P_{2n-1}) = 1$ e $\delta(P_{2n}) = 1$. Assim

$$\delta(P^{(m,k)}) \leq \prod_{j=m+1}^{k} \delta(P_j) = 1.1.1...1 = 1$$

Dessa forma não podemos dizer que a cadeia é fracamente ergódica. Por outro lado,

$$\begin{aligned} P_1P_2 = P_3P_4 = ... = P_{2n-1}P_{2n} &= \begin{bmatrix} 0 & 1/2 & 1/2 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ 1/2 & 0 & 1/2 \\ 0 & 1 & 0 \end{bmatrix} \\ &= \begin{bmatrix} 1/4 & 1/2 & 1/4 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} \end{aligned}$$

Assim $\delta(P_{2n-1}.P_{2n}) = 1/2$

$$\delta(p^{(m,k)}) \leq \delta(P_{m+1}P_{m+2})\delta(P_{m+3}P_{m+4})...\delta(P_{k-1}P_k) = (\frac{1}{2})^{\frac{(k-m)}{2}}$$

Logo $\delta(P^{(m,k)}) \to 0$ quando $k \to \infty$ e consequentemente a cadeia é fracamente ergódica embora $\prod_{j=m+1}^{k} \delta(P_j) = 1$ para todo m e k. □

Vimos no Exemplo 7 que o limite utilizado em $\delta(P^{(m,k)})$ pode às vezes ser obtido ao encontrar o coeficiente delta de blocos de produtos de matrizes de transição. Isso está descrito no seguinte

Teorema 2.2.2. *Seja $\{X_n\}$ uma cadeia de Markov não-homogênea com matrizes de transição $\{P_n\}_{n=1}^{\infty}$. A cadeia $\{X_n\}$ é fracamente ergódica, se e somente se, existe uma subdivisão de $P_1P_2P_3...$ em blocos de matrizes*

$$[P_1P_2...P_{n_1}][P_{n_1+1}P_{n_1+2}...P_{n_2}]...[P_{n_j+1}P_{n_j+2}...P_{n_{j+1}}]...$$

tal que

$$\sum_{j=0}^{\infty} \alpha(P^{(n_j,n_{j+1})}) = \infty$$

onde $n_0 = 0$

2.2.4 Ergodicidade Forte

Assim como no caso da ergodicidade fraca, geralmente não é fácil mostrar que uma cadeia de Markov não-homogênea é fortemente ergódica diretamente da definição. Nesta seção apresentaremos teoremas que dão condições suficientes para uma cadeia ser fortemente ergódica. Em muitos casos o uso desses teoremas é mais fácil que o uso da definição.

Teorema 2.2.3. *Uma cadeia de Markov com matrizes de transição $\{P_n\}$ é fortemente ergódica se e somente se, existir uma matriz constante, Q, tal que para todo m*

$$\lim_{k\to\infty} \left\| P^{(m,k)} - Q \right\| = 0$$

.

Teorema 2.2.4. *Seja $\{P_n\}$ uma sequência de matrizes de transição correspondentes a uma cadeia de Markov não-homogênea fracamente ergódica, e assuma que para todo n existe uma probabilidade $\pi_n = (\pi_n(j))_{j\in S}$ tal que $\pi_n = \pi_n P_n$ e que*

$$\sum_{j=1}^{\infty} \|\pi_i - \pi_{j+1}\| < \infty$$

. Então a cadeia é fortemente ergódica.

Corolário 2.2.1. *Seja P uma matriz de transição de uma cadeia de Markov homogênea tal que $\pi = \pi P$. Se a cadeia é fracamente ergódica, então é fortemente ergódica.*

Pelo Corolário 2.2.1 fica explicado por que não temos os conceitos de fraca e forte ergodicidade caso homogêneo. É impossível que uma cadeia homogênea seja ergódica fraca sem ser ergódica forte, isto é, é impossível para cadeia homogênea perder a memória sem convergir.

Os resultados a seguir serão usados no estudo da convergência do algoritmo *Simulated Annealing*.

Lema 2.2.6. *Se* $\{P_n\}_{n\geq 1}$ *é uma sequência de matrizes estocásticas tal que* $\|P_n - P\| \to 0$ *quando* $n \to \infty$*, então para cada inteiro positivo* k*,*

$$\left\|P_{n+1}P_{n+2}...P_{n+k} - P^k\right\| \to 0 \quad \textit{quando } n \to \infty.$$

Teorema 2.2.5. *Seja uma sequência de matrizes estocásticas* $\{P_n\}_{n\geq 1}$*, tais que* $\lim_{n\to\infty} P_n = P$*, com* P *ergódica, então*

$$\lim_{n\to\infty} P_1P_2...P_n = Q \quad \textit{onde} \quad Q = \lim_{n\to\infty} P^n$$

.

Capítulo 3

Modelagem do Algoritmo Genético por Cadeias de Markov

3.1 O Algoritmo Genético

A ideia dos Algoritmos Genéticos vem de uma analogia feita por Holand (1975) com processos naturais de seleção, nos quais, indivíduos com características genéticas melhores têm maiores chances de sobrevivência, enquanto indivíduos menos aptos tendem a desaparecer. Na natureza, há o processo de cruzamento, onde os indivíduos sobreviventes passarão para os seus descendentes suas características genéticas, que por sua vez terão chances de saírem também vencedores. Por outro lado, novos indivíduos poderão surgir através do processo de mutação, o qual a natureza insere material genético diferente. Se este ser que sofreu mutação estiver tão ou mais capacitado à sobrevivência quanto os atuais, terá grande chances de sobrevivência no futuro processo de seleção.

Os Algoritmos Genéticos tentam simular este processo natural da seguinte maneira: temos uma função $f : \Omega \to \mathbb{R}$ positiva (chamada função objetivo) da qual queremos encontrar seu(s) ponto(s) de ótimo (máximo). Ω é discretizado, ou seja, é criado uma malha de pontos de Ω que é chamado espaço de busca. Assim a solução será procurada neste espaço de busca. Quanto mais precisão se desejar, mais pontos a malha terá. Cada ponto desta malha (chamado de cromossomo) é um candidato ao ponto de ótimo. A

quantidade de pontos desta malha geralmente é uma potência de 2 possibilitando que na enumeração dos pontos da malha cada ponto seja escrito em forma binária. Cada bit (0 ou 1) é chamado de gene. Um cromossomo é mais apto que outro se sua imagem pela f for maior.

A ideia então é começar com um conjunto de pontos gerados aleatoriamente (chamada de população inicial) e utilizar a função objetivo f na etapa de seleção para "melhorar" a população inicial. Logo depois na etapa de cruzamento haverá troca de bits entre pares de cromossomos escolhidos segundo uma probabilidade (probabilidade de cruzamento). Finalmente temos a etapa de mutação onde bits poderão ser trocados segundo uma probabilidade (probabilidade de mutação). Depois destas três etapas teremos uma nova população de mesmo tamanho que a inicial que passará pelas mesmas etapas gerando novas populações.

O Algoritmo Genético que iremos tratar é o descrito por Goldberg (1989) conhecido como *Simple Genetic Algorithm*.

Resumindo, o Algoritmo Genético inicia sua busca com uma população, $(s_1^0, s_2^0, ..., s_n^0)$, geralmente aleatoriamente escolhida, a qual é chamada de população inicial. A seguir cria uma população $(s_1^{t+1}, s_2^{t+1}, ..., s_n^{t+1})$ no tempo $t+1$ a partir de uma população no tempo t. Para atingir esse objetivo, os indivíduos da população do tempo t passam pelas fases de seleção, cruzamento e mutação. Ao fim dessas operações é gerada a população no tempo $t+1$, a qual espera-se que seja "melhor", no sentido de ter pontos com a imagens maiores que as imagens da população anterior. Esse procedimento se repete até que um critério de parada seja satisfeito. O algoritmo a seguir mostra o mecanismo completo.

```
Algoritmo Genético Simples {
      Definindo {
         função objetivo
         formação do indivíduo e tamanho da população
         probabilidade dos operadores
         }
         Inicializar população aleatória
```

```
        Enquanto não alcançar critério de convergência faça{
              executar seleção
              executar cruzamento e mutação
               }
}
```

3.2 Modelagem do Algoritmo Genético por cadeias de Markov

Nesta seção modelaremos o algoritmo genético por cadeias de Markov. Vimos na seção anterior que a população seguinte é gerada levando em conta apenas a anterior. Vimos no capítulo anterior que a propriedade de Markov diz exatamente isso, ou seja, um processo em que o presente só depende do passado mais próximo. Desta forma seria natural considerar uma cadeia de Markov em que o espaço de estados sejam todas as possíveis populações.

3.2.1 Inicialização

Consideremos $f : \Omega \to \mathbb{R}$ uma função objetivo a qual queremos minimizar (ou maximizar), e consideremos Ω um subconjunto limitado de $\mathbb{R}^n$ da seguinte forma: $[a_1, b_1]\times[a_2, b_2]\times...\times[a_n, b_n]$. Dada a precisão desejada, os intervalos $[a_i, b_i], i = 1, 2, ..., n$ são particionados em 2^l partes iguais. Com isso temos que uma malha de 2^{nl} pontos é gerada.

Cada cromossomo (indivíduo) será formado por vetores nl dimensionais, cada entrada desses vetores é 0 ou 1, ou seja, um vetor de representação binária,

$$S = \{(x_1, x_2, ..., x_{nl}); x_i \in \{0, 1\}\}$$

Para ilustrar esta etapa, consideremos o problema de maximização da função $f : [-2, 4] \times [-2, 4] \to \mathbb{R}$ dada por:

$$f(x, y) = \frac{1}{0.3 + x^2 + y^2}.$$

Tomando $l = 4$, os intervalos ($[-2, 4]$) serão particionados em $2^4 = 16$ partes iguais, assim temos $r(x) = \frac{x_{max} - x_{min}}{2^l - 1}$ e $r(y) = \frac{y_{max} - y_{min}}{2^l - 1}$ as distâncias entre os pontos, neste caso $r(x) = r(y) = \frac{4-(-2)}{15} = 0.4$, e os intervalos serão particionados como ilustrado na Figura 3.1.

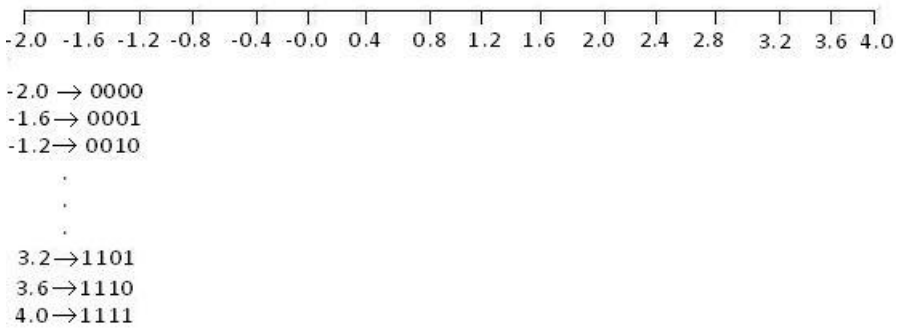

Figura 3.1: intervalo [-2,4] particionado

Assim uma malha de $2^{nl} = 2^8 = 256$ pontos é gerada, veja Figura 3.2. Por exemplo o ponto 00100100 representa o ponto $(-1.2(0010), -0.4(0100))$. Considere $S = \{(x_1, x_2, ..., x_8); x_i \in \{0, 1\}, i = 1, 2, ..., 8\}$, ou seja, o conjunto de todos os possíveis pontos da malha.

A população inicial bem como todas as outras são compostas por k indivíduos. O espaço de estados da cadeia de Markov é definido como,

$$S_1 = \{(y_1, y_2, ...y_k); y_i \in S, i = 1, ..., k\}. \tag{3.1}$$

Suponha a população inicial com 5 indivíduos:

C_1 10011000

C_2 01100101

C_3 00001101

C_4 01100110

C_5 01010101

O próximo passo será o cálculo da imagem pela função objetivo de cada ponto

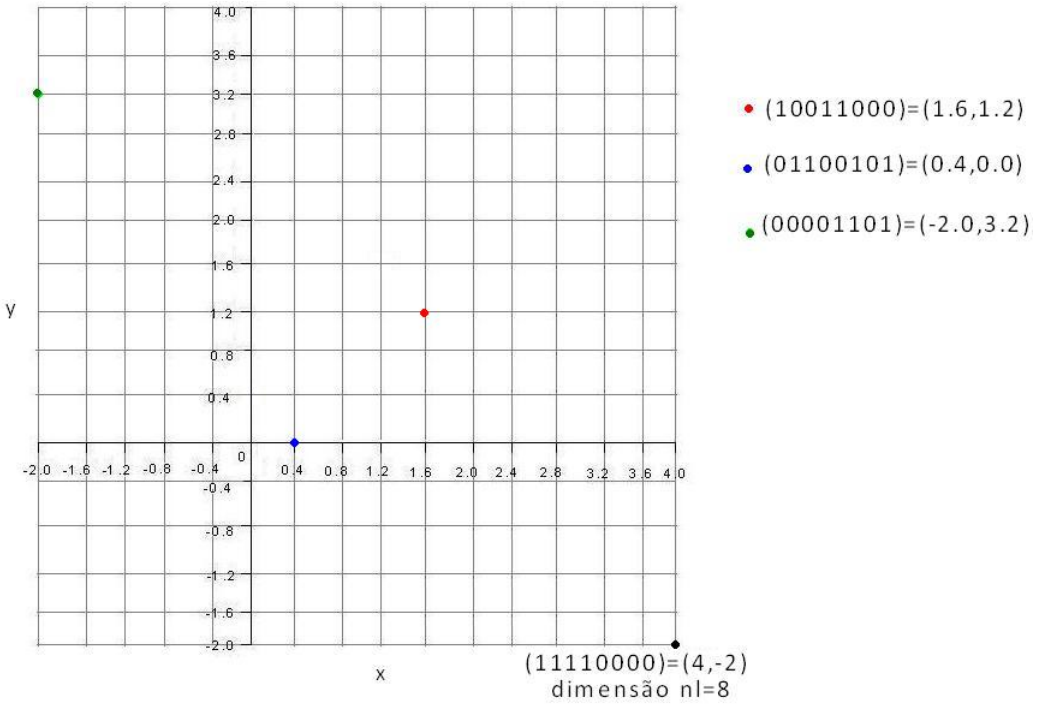

Figura 3.2: 256 pontos nl dimensionais

desses.

Para calcular o valor da função f nesses pontos, devemos converter primeiro estes vetores binários em seus valores reais, ou seja, cada vetor representa um par ordenado que será encontrado da seguinte maneira:

$$c_i = [b_{l-1}...b_2b_1b_0a_{l-1}...a_2a_1a_0] \quad a_i,b_i \in \{0,1\}$$

$$\overline{x} = \sum_{i=0}^{l-1} b_i 2^i \quad \text{e} \quad \overline{y} = \sum_{i=0}^{l-1} a_i 2^i.$$

Feito isso, deve-se calcular o valor de x e y reais dentro da regiões $[x_{min}, x_{max}]$ e $[y_{min}, y_{max}]$, através das seguintes equações,

$$x = x_{min} + \overline{x}.r(x) \quad \text{e} \quad y = y_{min} + \overline{y}.r(y).$$

Calculemos o valor da função para o cromossomo $C_1 = 10011000$ da população criada anteriormente.

Passando para base 10, temos

$\overline{x} = 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 9$

$\overline{y} = 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 8$

Os valores reais de $x, y \in [-2, 4]$, são dados por:

$x = -2 + 9 \times 0.4 = 1.6$

$y = -2 + 8 \times 0.4 = 1.2$

Daí temos,

$C_1 = (1.6, 1.2)$

E o valor da função de objetivo é

$f(1.6, 1.2) = 0.232$

Fazendo isso para cada cromossomo temos,

Tabela 3.1: Função de Objetivo

Indivíduo C_i		(x, y)	$f(x, y)$
C_1	10011000	$(1.6, 1.2)$	0.232
C_2	01100101	$(0.4, 0.0)$	2.173
C_3	00001101	$(-2.0, 3.2)$	0.068
C_4	01100110	$(0.4, 0.4)$	1.612
C_5	01010101	$(0.0, 0.0)$	3.333

3.2.2 Operação Seleção

Existem vários critérios de seleção dos indivíduos para compor a nova população. Dentre eles um dos mais usados é o da "roleta", o qual os indivíduos tem uma probabilidade de ser selecionado proporcional ao valor de sua imagem pela f, ou seja, o valor da função no ponto sobre o somatório do valor da função em todos os outros pontos da população $f(C_i)/\sum_{i=1}^{5} f(C_i)$.

Para visualizar este método, considere uma pizza dividida em k fatias (tamanho da população), onde cada fatia é proporcional ao valor da imagem do indivíduo pela f, veja Figura 3.3 . Depois de calculada o tamanho de todas as fatias a roleta é

girada (um valor r, uniformemente distribuído no intervalo $[0, 1]$ é gerado) tantas vezes quanto for o tamanho da população. A cada giro, um único indivíduo é escolhido, aquele correspondente a fatia em que a roleta parou.

Agora vamos aplicar a operação de seleção na população inicial, onde cada indivíduo tem uma probabilidade de ser selecionado proporcional a sua imagem pela função objetivo, como ilustrado na tabela 3.2

Tabela 3.2: Tabela de Seleção

Indivíduo C_i		(x, y)	$f(x, y)$	$f(C_i)/\sum_{i=1}^{5} f(C_i)$
C_1	10011000	$(1.6, 1.2)$	0.232	0.031
C_2	01100101	$(0.4, 0.0)$	2.173	0.292
C_3	00001101	$(-2.0, 3.2)$	0.068	0.009
C_4	01100110	$(0.4, 0.4)$	1.612	0.218
C_5	01010101	$(0.0, 0.0)$	3.333	0.450

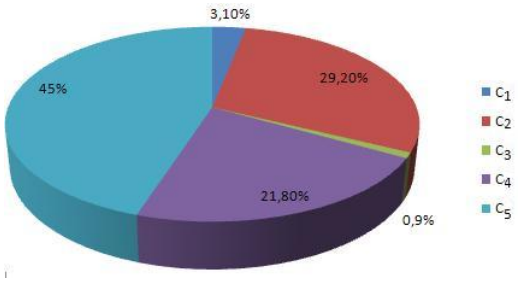

Figura 3.3: Roleta

Note que os indivíduos que possuem maior área terão maior probabilidade de serem escolhidos várias vezes para compor a nova população,

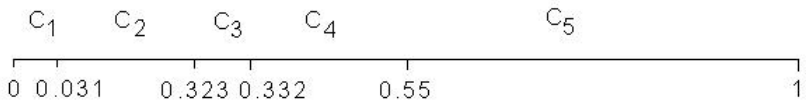

Figura 3.4: Indivíduos distribuídos proporcionalmente a sua função de adaptação

Para ilustrar isso, suponha que foram gerados os seguintes números aleatórios,

$r_1 = 0.61$, como $0.55 < r_1 < 1$, seleciona C_5

$r_2 = 0,008$, como $0.031 < r_2 < 0.323$, seleciona C_2

$r_3 = 0.47$, como $0.0332 < r_3 < 0.55$, seleciona C_3

$r_4 = 0.80$, como $0.55 < r_4 < 1$, seleciona C_5

$r_5 = 0.34$, como $0.332 < r_5 < 0.55$, seleciona C_4

Depois desta etapa, os cromossomos da nova população serão

C'_1 01010101

C'_2 01100101

C'_3 01100110

C'_4 01010101

C'_5 01100110

Dada uma população $Y = (y_1, y_2, ..., y_k)$ então, a probabilidade do algoritmo gerar a população $X = (x_1, x_2, ..., x_k)$ após a seleção é,

$$I_{\{y_1,y_2,...,y_k\}}(x_1)...I_{\{y_1,y_2,...,y_k\}}(x_k)\frac{a_1 f(x_1)}{\sum_{1\leq j\leq k} f(y_j)}...\frac{a_k f(x_k)}{\sum_{1\leq j\leq k} f(y_j)}$$

onde $I_{\{y_1,y_2,...,y_k\}}(x_i) = 1$ se x_i for um cromossomo da população anterior e zero caso contrário, e $a_i = \sum_{j=1}^{n} I_{y_i}(x_j)$ é a quantidade de vezes em que x_i aparece na população inicial.

3.2.3 Operação Cruzamento

O cruzamento permite a troca de material genético entre os indivíduos da população. A quantidade de cromossomos a ser submetida ao processo de cruzamento é definida através da probabilidade de cruzamento $p_c \in [0, 1]$, especificada pelo usuário. A escolha é feita gerando números aleatórios $r_i \in (0, 1)$, $i = 1, ..., k$, se $r_i < p_c$ o indivíduo i é escolhido para o cruzamento. Os indivíduos selecionados para o cruzamento são chamados de pais.

Existem várias maneiras de se proceder este cruzamento, neste trabalho estaremos usando o seguinte procedimento: o primeiro selecionado irá cruzar com o segundo, o terceiro com o quarto, e assim até que tenhamos $[s/2]$ pares, onde [.] é o menor inteiro

e s é a quantidade de indivíduos selecionados.

Seja $p_c = 0.30$ a probabilidade de cruzamento. E considere que foram gerados os seguintes números aleatórios,

$r_1 = 0.50 \Rightarrow r_1 > p_c$, não seleciona C'_1

$r_2 = 0.17 \Rightarrow r_2 < p_c$, seleciona C'_2

$r_3 = 0.40 \Rightarrow r_3 > p_c$, não seleciona C'_3

$r_4 = 0.20 \Rightarrow r_4 < p_c$, seleciona C'_4

$r_5 = 0.23 \Rightarrow r_5 < p_c$, seleciona C'_5

Para cada par de indivíduos escolhidos para o cruzamento, é gerado um ponto de corte, ou seja, um ponto que nos dará quantos bits serão trocados nesta etapa. Em um cromossomo de tamanho nl o ponto de corte será escolhido entre 1 e $nl - 1$. A partir desse ponto as informações genéticas dos pais serão trocadas, dando origem aos descendentes (filhos), como ilustrado na Figura 3.5.

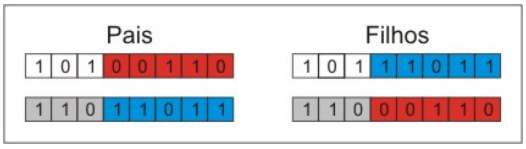

Figura 3.5: Cruzamento

Sendo que, foram selecionados para o cruzamento os cromossomos C'_2, C'_4 e C'_5. Temos que cruzarão apenas os indivíduos C'_2 e C'_4. Para gerar o ponto de corte do cruzamento, geramos um número aleatório t entre 1 e $nl - 1$. Os passos para fazer isso são os seguintes

$$t = floor(runif(1, 1, nl))$$

onde no R, floor é o maior inteiro e $runif(1, 1, nl)$ gera um número aleatória ente 1 e nl

Seja $t = 5$

Daí,

C'_2 01100|110

C'_4 01010|101

trocando os genes (bits), tem-se:

C''_2 01100101

C''_4 01010110

Assim, após a etapa de cruzamento, a população é dada por:

C'_1 01010101

C''_2 01100101

C'_3 01100110

C''_4 01010110

C'_5 01100110

O ponto de quebra sendo o mesmo para todos os pares, cada par de filhos tem uma probabilidade de ocorrência de

$$\frac{1}{(nl-1)}$$

.

3.2.4 Mutação

A mutação opera independentemente em cada bit do indivíduo da população através de uma probabilidade $p_m \in [0,1]$ chamada probabilidade de mutação .

Para aplicar o operador de mutação, é necessário gerar $k \times nl = 40$ números aleatórios $r \in [0,1]$. Seja $p_m = 0.01$ a probabilidade de mutação, se r_i for menor que p_m, será feita a mutação no bit correspondente.

Considere que foram gerados 40 números aleatórios $r \in [0,1]$ e que dois tiveram probabilidades menores que p_m. Foram os seguintes:

$r_{13} = 0.009 < p_m$

$r_{39} = 0.0025 < p_m$

Considerando a população atual e pensando na organização da população em um vetor da forma $C'_1, C''_2, C'_3, C'_4,' C'_5$ onde o primeiro bit será o primeiro bit de C'_1 no caso 0. O oitavo elemento será o oitavo elemento de C'_1 no caso 1, o nono bit será o primeiro elemento de C''_2 no caso 0, e assim sucessivamente, temos:

C'_1 01010101

C''_2 0110$\underbrace{0}_{bit13}$101

C'_3 01100110

C''_4 01010110

C'_5 011001$\underbrace{1}_{bit39}$0

Submetendo os bits 13 e 39 ao processo de mutação, têm-se:

C'_1 01010101

C''_2 0110$\underline{1}$101

C'_3 01100110

C''_4 01010110

C'_5 0110010

Concluindo a geração do algoritmo,

Tabela 3.3: Função de Adaptação

Indivíduo C_i		(x, y)	$f(x, y)$
C'_1	01010101	$(0.0, 0.0)$	3.333
C''_2	01101101	$(0.4, 3.2)$	0.093
C'_3	01100110	$(0.4, 0.4)$	1.612
C''_4	01010110	$(0.0, 0.4)$	2.173
C'_5	01100100	$(0.4, -0.4)$	1.612

Assim, a probabilidade de sair de uma população $Y = (y_1, y_2, ..., y_k)$ para uma população $X = (x_1, x_2, ..., x_k)$ é dado por,

$$p_m^{H(X,Y)}(1 - p_m)^{nl-H(X,Y)}$$

$H(X,Y)$ é a distância de Hamimng entre as populações X e Y a qual representa o número de bits que devem ser alterados pela mutação para transformar Y em X.

Depois desta etapa completamos uma geração do algoritmo.

Consideremos agora uma cadeia de Markov $\{X_n\}_{n\in\mathbb{N}}$ com espaço de estados S_1 como definida em (3.1) que é o conjunto de todas as populações formadas por k cromossomos. Gostaríamos de saber quem é a matriz de transição desta cadeia e a resposta está demonstrada no seguinte teorema.

Teorema 3.2.1. *A evolução do algoritmo Genético é modelada pela matriz de transição P que é decomposta por um produto de matrizes estocásticas $P = SCM$, onde S, C e M modelam as transições realizadas nos passos de seleção, cruzamento e mutação respectivamente.*

Prova) Denotemos por X_s, X_c e X_m as etapas intermediárias entre o passo n e $n+1$, ou seja, X_s representa a etapa em que o ponto inicial é a população atual e o resultado é a população resultante da seleção. Vemos que esta etapa é ela própria uma cadeia de Markov, pois o resultado da seleção só depende da etapa anterior. Assim podemos construir sua matriz de transição por $S_{(A,B)} = P(X_s = B|X_n = A)$. Da mesma forma as matrizes de transição do cruzamento e da mutação podem ser construídas da seguinte forma: $C_{(A,B)} = P(X_c = B|X_s = A)$ e $M_{(A,B)} = P(X_m = B|X_c = A)$. Note que X_m será o resultado da mutação, ou seja, a próxima população, assim $M_{(A,B)} = P(X_m = B|X_c = A) = P(X_{n+1} = B|X_c = A)$. Com isso temos

$$
\begin{aligned}
p_{AB} &= P(X_{n+1} = B | X_n = A) \\
&= \frac{P(X_{n+1} = B, X_n = A)}{P(X_n = A)} \\
&= \sum_{C \in S_1} \frac{P(X_{n+1} = B, X_s = C, X_n = A)}{P(X_n = A)} \\
&= \sum_{D \in S_1} \sum_{C \in S_1} \frac{P(X_{n+1} = B, X_c = D, X_s = C, X_n = A)}{P(X_n = A)} \\
&= \sum_{D \in S_1} \sum_{C \in S_1} P(X_{n+1} = B | X_c = D, X_s = C, X_n = A) P(X_c = D | X_s = C, X_n = A) \\
&\quad P(X_s = C | X_n = A) \\
&= \sum_{D \in S_1} \sum_{C \in S_1} P(X_{n+1} = B | X_c = D) P(X_c = D | X_s = C) P(X_s = C | X_n = A) \\
&= \sum_{D \in S_1} \sum_{C \in S_1} S_{(A,C)} C_{(C,D)} M_{(D,B)} = SCM_{(A,B)}
\end{aligned}
$$

Como esta igualdade vale qualquer que seja $A, B \in S_1$, temos que a matriz de transição $P = SCM$. □

3.3 Convergência do Algoritmo Genético

Nesta seção trataremos de dar condições sobre as matrizes de transição de modo que o algoritmo genético convirja. Para tanto relembraremos algumas definições e finalizaremos a seção provando o teorema que garante a convergência.

Definição 3.3.1. *Dizemos que uma matriz é positiva quando todos os seus elementos são positivos.*

Definição 3.3.2. *Dizemos que uma matriz quadrada é estocástica quando todos os elementos são não negativos e a soma dos elementos de cada linha é igual a* 1

Teorema 3.3.1. *A matriz P obtida no teorema anterior é positiva*

Prova) Na seção anterior mostramos que as entradas da matriz de transição da mutação M é dada por $M(X, Y) = p_m^{H(X,Y)}(1 - p_m)^{nl - H(X,Y)}$. Como $p_m > 0$ temos que

todas as entradas desta matriz são positivas, portanto M é positiva. Como o produto de matrizes estocásticas é estocástica temos que SC é estocástica, logo podemos concluir que $P = SCM$ é positiva. □

Teorema 3.3.2. *A cadeia de Markov representada pela matriz $P = SCM$ acima é convergente.*

Prova) Como P é positiva, temos que todas as suas entradas são positivas, isso implica que $P(A, B) > 0$ e $P(B, A) > 0$ para todos as populações $A, B \in S_1$. Isso implica pela Definição 2.1.5 que a cadeia é irredutível. Já pela Proposição 2.1.1 temos que todos os estados possuem o mesmo período,e como $P(A, A) > 0$ pelo fato da P ser positiva, temos que o período da cadeia é 1 e pela Definição 2.1.6 a cadeia é aperiódica. Como a quantidade de pontos do espaço de estados é finito temos que a cadeia em questão é finita, irredutível e aperiódica, logo pelo Corolário 2.1.5 temos que a cadeia é convergente. □

Capítulo 4

Modelagem do Simulated Annealing por Cadeias de Markov

4.1 O Algoritmo Simulated Annealing

O algoritmo *Simulated Annealing* (SA) é uma técnica de busca baseada no fenômeno físico proposta originalmente por Kirkpatrick (1983), o qual consiste em resfriar lentamente um metal, originalmente em alta temperatura e em estado líquido. O resfriamento terminará quando o metal tenha atingido seu ponto de solidificação. O ideal é que estando no estado sólido o metal tenha atingido o estado mínimo de energia. Se o resfriamento ocorrer de maneira muito rápida, o estado sólido será atingido sem que o estado mínimo de energia seja também atingido, o que implicará num metal com pontos de energia acumulada (imperfeições). Estes pontos apresentarão menor resistência, comprometendo o metal como um todo.

O algoritmo *Simulated Annealing*, tenta emular o procedimento descrito acima para encontrar o ponto de mínimo de uma dada função. Para isso a função objetivo representa a energia do metal e o algoritmo simula um resfriamento tal maneira que, à medida em que a temperatura diminui o algoritmo encontra novos pontos, o que se espera é que convirja para a solução ótima.

O procedimento de busca pela solução ótima é feito estabelecendo uma temperatura inicial T_0 (geralmente alta) e gerando inicialmente um ponto aleatório s do domínio e

a partir daí é gerado um outro ponto s' a partir da posição s em que o algoritmo se encontra de acordo com uma certa distribuição $q_1(s'|s)$. O ponto gerado pode ser aceito ou não de acordo com uma probabilidade de aceitação $a_1(s'|s)$. Se aceito, o próximo ponto em que o algoritmo se encontra será s', caso contrário o próximo ponto será o próprio s. Uma redução na temperatura $T_0 \rightarrow T_1$ é efetuada e outro ponto s'' é gerado de acordo com uma certa distribuição $q_2(s''|s')$, caso s' seja o ponto em que o algoritmo se encontra, e este ponto é aceito ou não segundo a probabilidade de aceitação $a_2(s''|s')$, e assim sucessivamente.

Veremos que T é o parâmetro que regula a probabilidade de aceitar soluções piores. Quanto maior for o valor de T, maior a probabilidade de aceitar soluções piores.

Para exemplificar o funcionamento do SA, tem-se na Figura 4.1 a representação de uma função objetivo de uma variável. O ótimo global desta função é representado pelo ponto P_{OG}. Partindo-se de um ponto P_0, se fosse uma busca local, em que só se aceitasse pontos melhores, o próximo ponto escolhido seria o ponto P_1 e a solução convergiria para um ótimo local, representado pelo ponto P_{OL}, mas no SA o ponto P_2 pode ser aceito com uma certa probabilidade, $a_n(.|.)$, e a partir de então pode atingir o ótimo global, P_{OG}. Dependendo da sequência de distribuições $q_n(.|.)$ utilizada, a geração de pontos distantes do ponto atual pode ser possível. Com este procedimento de aceitação de pontos piores, o SA consegue escapar de ótimos locais.

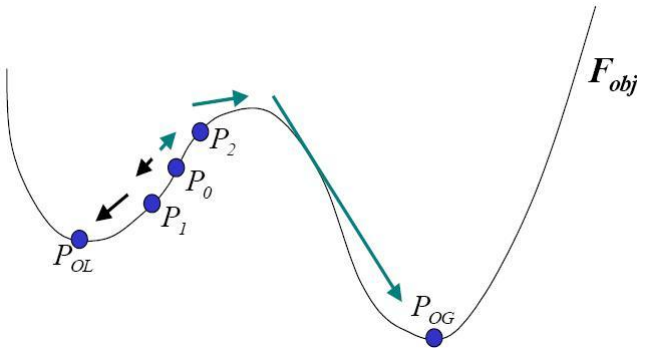

Figura 4.1: Comportamento Gráfico do SA

No SA a probabilidade de aceitar ponto piores diminui gradativamente à medida

que a temperatura vai baixando. O processo de decréscimo da temperatura chamamos de agendamento de resfriamento.

Neste trabalho utilizaremos vários agendamento de resfriamento, um dos mais utilizados é aquela em que a temperatura é gradativamente diminuída por uma razão α, tal que $T_i = \alpha T_{i-1}$, sendo $0 < \alpha < 1$.

Esquematicamente podemos representar o SA da seguinte maneira:

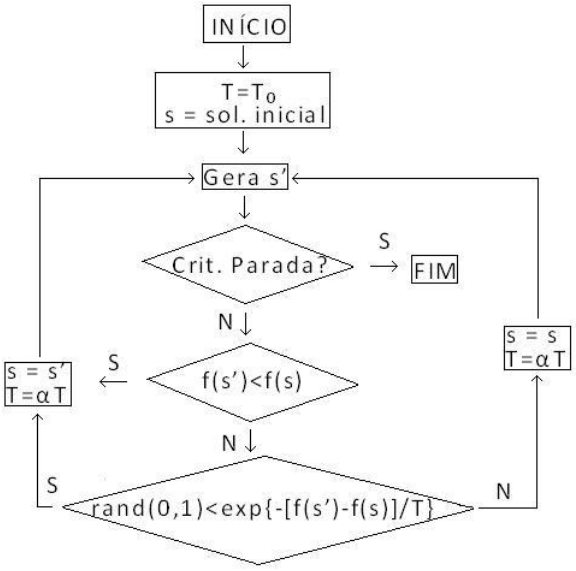

Figura 4.2: Flluxograma do Algoritmo SA

4.2 Modelagem do Algoritmo Simulated Annealing por Cadeias de Markov

Seja $f : S \to \mathbb{R}$ uma função a qual queremos encontrar $f_{\min} = \min_{i \in S} f(i)$. Vimos na seção anterior que precisamos de uma sequência de distribuição de probabilidades para gerar os próximos pontos a partir do ponto em que o algoritmo se encontra, de uma sequência de probabilidades de aceitação que nos dará a probabilidade de aceitar ou não o ponto gerado, de uma temperatura inicial T_0 e de uma agenda de resfriamento.

Existem várias tipos de sequências de distribuição de probabilidades, por exemplo,

se $S = \{1, 2, ..., n\}$ podemos utilizar uma sequência $Q_n = Q$ expressa pela matriz

$$Q = \begin{bmatrix} 1/n & 1/n & \cdots & 1/n \\ 1/n & 1/n & \cdots & 1/n \\ \vdots & \vdots & \ddots & \vdots \\ 1/n & 1/n & \cdots & 1/n \end{bmatrix}$$

que significa que todos os pontos de $S = \{1, 2, ..., n\}$ possuem a mesma chance de serem escolhidos dado que algoritmo se encontra em qualquer ponto, ou seja, $Q(i, j) = q(i|j) = \frac{1}{n}, \forall i, j \in S$.

A probabilidade de aceitação é dada por

$$a_n(x_j/x_i) = \min\left\{1, \exp\left[-\frac{f(x_j) - f(x_i)}{T_n}\right]\right\}, \tag{4.1}$$

onde T_n é a temperatura no passo n do algoritmo ($T_n \downarrow 0$ quando $n \to \infty$). Vemos então que se $f(x_j) < f(x_i)$, ou seja, o ponto gerado é melhor que o ponto atual, $a_n(x_j/x_i) = 1$. E caso seja pior, ou seja, $f(x_j) > f(x_i)$ ainda temos a chance de aceitar x_j com probabilidade $\exp\left[-\frac{f(x_j)-f(x_i)}{T_n}\right]$.

A dinâmica do algoritmo é a seguinte:

- Escolha uma solução aleatória $x_1 \in S$.

- No passo $n + 1$ um novo candidato Y_{n+1} é gerado com probabilidade $Q(i, j) = P(Y_{n+1} = x_j | X_n = x_i)$ onde $X_n = x_i$ é a solução atual, e a troca de X_n por Y_{n+1} é realizada com probabilidade de aceitação $a_n(Y_{n+1}|X_n)$, caso contrário, $X_{n+1} = X_n$, ou seja,

$$X_{n+1} = \begin{cases} Y_{n+1}, & \text{com probabilidade} \quad a_n(Y_{n+1}|X_n) \\ X_n, & \text{com probabilidade} \quad 1 - a_n(Y_{n+1}|X_n) \end{cases}$$

Assim temos uma sequência de matrizes estocásticas $\{P(n)\}_{n\geq 1}$, onde o (i,j)-ésimo elemento da matriz $P(n)$ é dado por,

$$P_{ij}(n) = P(X_{n+1} = j | X_n = i) = \begin{cases} Q(i,j)a_n(j|i), & \text{se} \quad i \neq j \\ 1 - \sum_{l \neq i} Q(i,l)a_n(l|i), & \text{se} \quad i = j \end{cases} \tag{4.2}$$

Com isso vemos que o algoritmo *Simulated Annealing* é modelado por uma cadeia de Markov não-homogênea. Estudaremos na próxima seção condições para as quais garantimos a convergência deste algoritmo.

4.3 Convergência do Algoritmo Simulated Annealing

Nesta seção daremos condições que serão suficientes para a convergência do algoritmo *Simulated Annealing*.

Vimos na seção anterior que o algoritmo *Simulated Annealing* é modelado por uma cadeia de Markov não-homogênea. Agora queremos obter condições sobre as quais garantimos que a cadeia converge para o ponto de mínimo, ou seja, se considerarmos $S_{\min} = \{X \in S / X \text{ ponto de mínimo de} f\}$, queremos saber

$$P^{(m,n)}(i,j) \to 0 \quad \text{se} \, j \notin S_{min}, \forall i \in S \quad \text{quado} \quad n \to \infty \tag{4.3}$$

Consideremos que a sequência de matrizes utilizadas para gerar os pontos seja fixa, ou seja $Q_n = Q$ e que

i) $Q = (Q_{(i,j)})$ tal que $Q_{(i,j)} > 0$ para todo $i,j \in S$

Pediremos que as probabilidades de aceitação satisfaçam

ii) $\lim_{k \to \infty} a_k(j/i) = \begin{cases} 1 & \text{se} \quad f(j) \leq f(i) \\ 0 & \text{se} \quad f(j) > f(i) \end{cases}$

Teorema 4.3.1. *Sob hipóteses i) e ii) acima, a cadeia não-homogênea que modela o Simulated Annealing satisfaz (4.3).*

Demonstração Por (4.2) e por i), a matriz P_k, é dada por

$$P_k = \begin{bmatrix} 1-\sum_{l\neq 1} q_{1l}a_k(l|1) & q_{12}a_k(2|1) & \cdots & q_{1n}a_k(n|1) \\ q_{21}a_k(1|2) & 1-\sum_{l\neq 2} q_{2l}a_k(l|2) & \cdots & q_{2n}a_k(n|2) \\ \vdots & \vdots & \ddots & \vdots \\ q_{n1}a_k(1|n) & q_{n2}a_k(2|n) & \cdots & 1-\sum_{l\neq n} q_{nl}a_k(l|n) \end{bmatrix}$$

Por ii) P_k converge para uma matriz P, ou seja

$$\lim_{k\to\infty} P_k = P$$

onde $P = (p_{ij})$, é uma matriz onde o (i,j)-ésimo elemento é dado por

$$p_{ij} = \begin{cases} q_{ij} & \text{se} \quad f(j) \leq f(i) \quad \text{para} \quad i \neq j \\ 1-\sum_{l\neq i} q_{il} & \text{se} \quad i = j \\ 0 & \text{se} \quad f(j) > f(i). \end{cases}$$

Considere uma reordenação dos estados de S da seguinte forma; $S_{\min} = \{1, 2, 3..., n_1\}$ e $S - S_{\min} = \{n_1 + 1, ..., n_1 + n_2\}$, onde $S_{\min} = \{l \in S \,/\, f(l) = f_{\min}\}$.

P pode ser escrita como

$$P = \begin{bmatrix} P_1 & 0 \\ R & T \end{bmatrix}$$

Sendo P_1 uma matriz $n_1 \times n_1$, temos que P_1 é irredutível, aperiódica e

$$p_{ij} = \begin{cases} 0 & \text{se} \quad i \in S_{\min} \quad \text{e} \quad j \notin S_{\min} \\ q_{ij} > 0 & \text{se} \quad j \in S_{\min} \quad \text{e} \quad i \notin S_{\min} \\ 1-\sum_{l\neq i} q_{il} & \text{se} \quad i \in S_{\min} \quad \text{e} \quad i = j \end{cases}$$

Pelo Teorema 2.1.5 juntamente com o Lema 2.1.1, temos que $P = (p_{ij})$ é ergódica e:

$$\lim_{n\to\infty} p_{ij}^n = \begin{cases} \pi_j & \text{se} \quad j \in S_{min} \\ 0 & \text{se} \quad j \notin S_{min} \end{cases}$$

E pelo Teorema 2.2.5 temos que $P_1P_2...P_n \to Q$, onde $Q = lim_{n\to\infty}P^n$, ou seja,

$$\lim_{n\to\infty} (P_1P_2...P_n)_{ij} = \begin{cases} \pi_j & \text{se} \quad j \in S_{min} \\ 0 & \text{se} \quad j \notin S_{min} \end{cases}$$

Finalmente pelo Lema 2.2.6 temos que

$$\|P_m P_{m+1} ... P_n - Q\| \leq \|P_m P_{m+1} ... P_n - P^{n-m+1}\| + \|P^{n-m+1} - Q\|$$

$$\lim_{n\to\infty} \|P_m P_{m+1} ... P_n - Q\| \leq \lim_{n\to\infty} \|P_m P_{m+1} ... P_n - P^{n-m+1}\| + \lim_{n\to\infty} \|P^{n-m+1} - Q\|$$

$$\lim_{n\to\infty} \|P_m P_{m+1} ... P_n - Q\| = 0$$

Portanto temos que (4.3) vale. □

O teorema acima mostra que sob as condições i) e ii) o algoritmo *Simulated Annealing* converge para o conjunto de pontos de ótimo.

No próximo capítulo apresentamos várias simulações do Algoritmo Genético e do Algoritmo *Simulated Annealing* e finalizamos com nossas conclusões.

Capítulo 5

Conclusões

Com o intuito de ajudar em nossas conclusões, apresentamos a seguir vários exemplos. As tabelas apresentam quantos sucessos foram obtidos ao realizar 30 repetições (conforme Soares(1997)) do mesmo algoritmo sob as mesmas condições.

Utilizamos três funções, cujos pontos de ótimo são previamente conhecidos, e foi verificado o desempenho desses algoritmos na busca dos pontos ótimos dessas funções.

Em cada tabela que se encontra nessa seção é apresentado o resultado de 30 realizações do mesmo algoritmo sob as mesmas condições. Os números apresentados representam a quantidade de vezes, em que o ponto de ótimo pertencia a população ao final do número de iterações realizadas.

Nas tabelas as quais encontram-se os resultado do Algoritmo *Simulated Annealing* (SA) modificamos o número de pontos no domínio (2^6, 2^8, 2^{10}, 2^{12}, 2^{14}, 2^{16}, 2^{18}, 2^{20}) e comparamos três tipos de agendamento de resfriamento (T_1, T_2 e T_3) que será explicado com detalhes na seção 5.2.

Já nas tabelas as quais encontram-se os resultado do Algoritmo Genético (AG), modificamos o número de pontos no domínio (2^6, 2^8, 2^{10}, 2^{12}, 2^{14}, 2^{16}, 2^{18}, 2^{20}) e comparamos os resultado com populações de diferentes tamanhos 2, 3, 5, 10, 15, 20 indivíduos.

Cada tabela foi obtida para uma determinada quantidade de iterações e realizamos os procedimentos usando 500, 1000, 5000 e 10000 iterações.

Para tanto, o ambiente de simulação utilizado será o R, versão 2.9.1.

5.1 Funções Teste

As funções escolhidas possuem as seguintes características: a primeira possui um único ponto de ótimo, a segunda vários ótimos locais, e a ultima 5 ótimos locais.

A primeira função é definida por $f : [-2, 4] \times [-2, 4] \rightarrow \mathbb{R}$, definida por

$$f(x, y) = \left(\frac{1}{0.3 + x^2 + y^2} \right), \tag{5.1}$$

cujo o valor de máximo é $f(x, y) = 3.333$ e acontece no ponto $(0, 0)$. A Figura 5.1 representa o gráfico da função f.

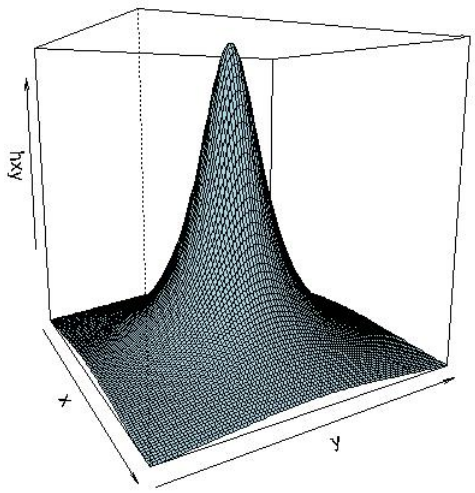

Figura 5.1: Gráfico da primeira função (Função 1)

Quando utilizamos o SA para calcular o máximo dessa função aplicamos na verdade o SA em $-f$, já que este algoritmo é utilizado para encontrar o ponto de mínimo.

A segunda função é definida por $f : [-2, 5] \times [-2, 5] \rightarrow \mathbb{R}$, definida por

$$f(x, y) = 6 + x^2 - 3\cos(2\pi x) + y^2 - 3cos(2\pi y), \tag{5.2}$$

cujo o valor de máximo é $f(x, y) = 53.1$ e acontece no ponto $(4.5, 4.6)$. A Figura

5.2 representa o gráfico da função f.

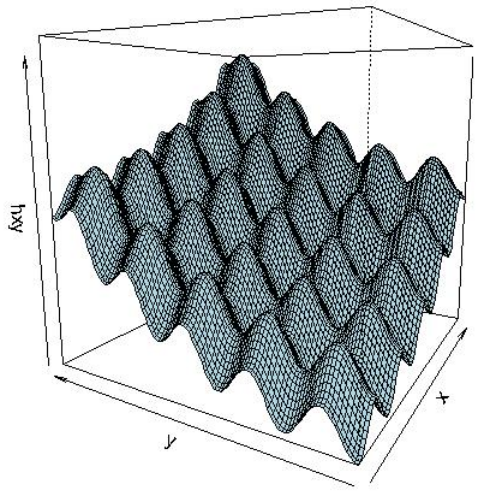

Figura 5.2: Gráfico da segunda função (Função 2)

A terceira função é definida por $f : [-2, 4] \times [-2, 4] \to \mathbb{R}$, definida por

$$f(x, y) = 3(1 - x)^2 e^{-x^2-(y+1)^2} + \left| 10(\frac{x}{5} - x^3 - y^5)e^{-x^2-y^2} + \frac{1}{3}e^{-(x-1)^2-y^2} \right|, \qquad (5.3)$$

cujo o valor de máximo é $f(x, y) = 10.91$ e acontece no ponto $(-0.25, -1.47)$. A Figura 5.3 representa o gráfico da função f.

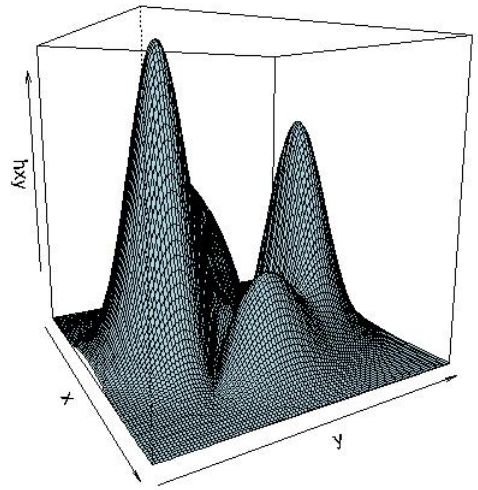

Figura 5.3: Gráfico da terceira função (Função 3)

5.2 Testando o Algoritmo Simulated Annealing

O Algoritmo *Simulated Annealing* (SA) no seu processo de busca trabalha com apenas um ponto, gerando a partir desse ponto uma nova solução a qual é testada e é aceita ou não. O fato de aceitar ou não o próximo ponto está associado ao parâmetro de controle (temperatura), o qual no início de processo assume um valor relativamente alto e a cada iteração decresce gradualmente.

O critério de mudança da temperatura (agendamento de resfriamento) é um parâmetro crucial para o desempenho do Algoritmo SA. Sendo assim, três tipos de agendamento de resfriamento foram utilizados para testarmos o desempenho deste algoritmo: O primeiro utiliza a fórmula logarítmica (5.4), que prevê um resfriamento rápido no início do processo e um mais lento à medida em que o número de iterações aumenta.

$$T_i = \frac{(i-1)T_{i-1} + T_0}{i(1 - log(i))}, \tag{5.4}$$

onde T_i é a temperatura na interação i, i é a iteração atual e T_0 a temperatura inicial.

Enquanto outro tipo utiliza uma taxa constante $\alpha \in (0, 1)$, sendo a temperatura seguinte multiplicada por esta taxa, resultando na expressão (5.5).

$$T_i = \alpha T_{i-1}. \tag{5.5}$$

Outra alternativa é fazer o decaimento da temperatura por meio da expressão (5.6), a qual utiliza também uma constante $\alpha \in (0, 1)$, que prevê um resfriamento muito lento quando os valores de α estão próximos de zero.

$$T_i = \frac{T_{i-1}}{1 + \alpha\sqrt{T_{i-1}}}. \tag{5.6}$$

Nas tabelas abaixo representamos os tipos de agendamento de resfriamento (5.4), (5.5) e (5.6) como sendo T_1, T_2 e T_3 respectivamente.

Observe que em cada tabela estão representados os resultados que o algoritmo apresentou ao final de cada "rodada", ou seja, ao obedecer o critério de parada, a qual

adotamos número de iterações.

As Tabelas 5.1, 5.2, 5.3 e 5.4, apresentam os resultados obtidos para a Função 1 com 500, 1000, 5000 e 10000 iterações.

Tabela 5.1: Função 1 com 500 iterações

	2^6	2^8	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
T_1	18	13	5	2	2	1	1	3
T_2	0	0	0	0	0	0	0	0
T_3	29	24	12	4	3	2	1	5

Tabela 5.2: Função 1 com 1000 iterações

	2^6	2^8	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
T_1	30	28	12	4	3	4	3	4
T_2	30	24	6	4	2	0	1	1
T_3	30	29	18	11	4	4	2	4

Tabela 5.3: Função 1 com 5000 iterações

	2^6	2^8	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
T_1	30	30	26	24	17	19	12	17
T_2	30	30	29	25	19	24	21	24
T_3	30	30	29	26	20	25	25	24

Tabela 5.4: Função 1 com 10000 iterações

	2^6	2^8	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
T_1	30	30	30	25	23	24	28	24
T_2	30	30	30	29	26	28	28	29
T_3	30	30	30	29	27	27	28	29

Observe que quanto maior o número de iterações, o algoritmo apresenta melhores resultados independentemente do tipo de agendamento de resfriamento utilizado.

Veja nas Tabelas 5.5, 5.6, 5.7 e 5.8 os resultados obtidos para a Função 2 com 500, 1000, 5000 e 10000 iterações.

Tabela 5.5: Função 2 com 500 iterações

	2^6	2^8	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
T_1	30	23	10	2	3	1	1	5
T_2	9	6	1	0	0	0	0	0
T_3	30	26	14	2	3	1	2	6

Tabela 5.6: Função 2 com 1000 iterações

	2^6	2^8	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
T_1	30	29	18	6	5	3	0	1
T_2	30	25	11	3	2	1	0	1
T_3	30	30	18	8	6	4	1	1

Tabela 5.7: Função 2 com 5000 iterações

	2^6	2^8	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
T_1	30	30	30	25	23	7	12	16
T_2	30	30	30	22	21	17	11	19
T_3	30	30	30	21	24	10	13	17

Tabela 5.8: Função 2 com 10000 iterações

	2^6	2^8	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
T_1	30	30	30	24	28	20	25	21
T_2	30	30	30	23	27	18	26	22
T_3	30	30	30	25	28	27	25	25

Nas Tabelas 5.9, 5.10, 5.11 e 5.12 encontram-se os resultados obtidos para a Função 3 com 500, 1000, 5000 e 10000 iterações.

Tabela 5.9: Função 3 com 500 iterações

	2^6	2^8	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
T_1	30	7	9	3	3	7	8	5
T_2	0	0	1	0	0	0	0	0
T_3	30	24	13	6	6	6	8	10

Tabela 5.10: Função 3 com 1000 iterações

	2^6	2^8	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
T_1	30	20	15	13	6	9	9	9
T_2	30	19	8	10	4	8	6	6
T_3	30	30	18	17	15	12	16	13

Tabela 5.11: Função 3 com 5000 iterações

	2^6	2^8	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
T_1	30	30	30	28	28	27	28	28
T_2	30	30	30	30	27	29	30	29
T_3	30	30	30	30	28	29	30	30

Tabela 5.12: Função 3 com 10000 iterações

	2^6	2^8	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
T_1	30	30	30	30	30	30	30	30
T_2	30	30	30	30	30	30	30	30
T_3	30	30	30	30	30	30	30	30

Constatamos que ao escolhermos mais iterações, maior a chance de o ótimo ser o ponto onde o algoritmo se encontra.

5.3 Testando o Algoritmo Genético

O Algoritmo Genético (AG) trabalha com um conjunto de pontos, chamado população, a qual passa pelos processos de seleção cruzamento e mutação, gerando sempre outra população que substitui a anterior. A quantidade de indivíduos para compor a população é um parâmetro crucial para o desempenho do AG.

Para os parâmetros, probabilidade de cruzamento e probabilidade de mutação, usaremos os valores apresentados em De Jong (1975), no qual foi sugerido o cruzamento e a mutação com probabilidades 0.001 e 0.6 respectivamente.

Nas Tabelas 5.13, 5.14, 5.15 e 5.16 encontram-se os resultados obtidos para a Função 1 com 500, 1000, 5000 e 10000 iterações.

Tabela 5.13: Função 1 com 500 iterações

	2^6	2^8	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
2	6	2	0	0	0	0	0	0
3	5	3	2	0	0	0	0	0
5	16	9	2	0	0	0	0	0
10	26	10	5	2	2	0	0	2
15	28	15	3	1	1	2	2	0
20	28	24	5	3	3	2	2	2

Tabela 5.14: Função 1 com 1000 iterações

	2^6	2^8	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
2	6	1	1	0	0	1	0	0
3	14	2	1	1	0	0	0	0
5	17	7	2	1	2	1	0	0
10	26	20	4	2	1	0	1	1
15	20	26	3	2	1	2	3	0
20	29	25	9	5	2	2	4	3

Tabela 5.15: Função 1 com 5000 iterações

	2^6	2^8	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
2	9	6	0	0	1	0	0	0
3	18	16	2	1	0	0	0	0
5	24	25	6	3	2	2	1	2
10	28	20	2	4	3	4	3	3
15	28	26	9	2	3	3	3	3
20	30	27	13	8	4	8	3	3

Tabela 5.16: Função 1 com 10000 iterações

	2^6	2^8	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
2	18	6	1	2	1	0	1	0
3	22	15	3	1	0	1	0	0
5	24	24	8	5	2	3	0	1
10	29	26	2	2	3	6	1	3
15	29	28	9	2	3	5	3	3
20	30	30	13	12	7	8	13	5

Observe que quanto maior o número de indivíduos para compor a população, maior a chance da população final conter o ponto de ótimo.

Para a Função 2, veja as Tabelas 5.17, 5.18, 5.19 e 5.20.

Tabela 5.17: Função 2 com 500 iterações

	2^6	2^8	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
2	1	1	0	1	1	0	0	0
3	6	3	1	0	0	1	0	0
5	8	4	1	1	0	1	0	0
10	17	9	1	1	2	1	1	0
15	29	14	4	1	2	1	1	1
20	28	16	4	2	2	3	2	1

Tabela 5.18: Função 2 com 1000 iterações

	2^6	2^8	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
2	3	1	0	1	0	0	1	0
3	7	1	0	0	1	0	0	0
5	14	5	1	1	0	0	1	0
10	25	8	4	2	1	1	0	1
15	29	17	7	2	4	1	1	1
20	30	16	3	3	2	2	2	2

Tabela 5.19: Função 2 com 5000 iterações

	2^6	2^8	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
2	9	1	1	0	1	0	0	0
3	11	5	2	1	0	0	0	0
5	17	6	2	1	1	0	1	0
10	28	12	2	2	1	1	2	1
15	30	13	11	1	5	3	2	1
20	30	19	12	5	2	2	1	2

Tabela 5.20: Função 2 com 10000 iterações

	2^6	2^8	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
2	4	3	1	2	0	1	0	0
3	7	3	2	0	1	0	0	0
5	17	8	3	1	2	0	0	1
10	28	13	6	3	3	1	2	1
15	30	25	7	1	2	3	2	2
20	30	20	13	5	2	4	2	2

Para a Função 3 veja as Tabelas 5.21, 5.22, 5.23 e 5.24.

Tabela 5.21: Função 3 com 500 iterações

	2^6	2^8	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
2	4	1	0	0	1	0	0	1
3	6	2	0	1	1	0	0	0
5	9	2	2	2	0	1	0	0
10	18	5	3	1	2	1	1	0
15	13	11	4	2	5	3	2	0
20	21	11	7	3	3	3	1	1

Tabela 5.22: Função 3 com 1000 iterações

	2^6	2^8	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
2	2	2	1	1	0	0	1	0
3	7	4	2	0	1	0	0	0
5	7	3	3	2	1	1	0	1
10	20	5	3	1	1	1	2	0
15	16	4	3	2	2	2	1	1
20	27	9	6	3	3	2	2	1

Tabela 5.23: Função 3 com 5000 iterações

	2^6	2^8	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
2	5	1	1	0	0	0	0	0
3	7	3	1	1	0	1	0	0
5	19	5	2	1	0	2	3	0
10	21	6	6	1	1	2	2	1
15	23	12	10	2	7	1	1	0
20	25	12	8	3	2	2	2	1

Tabela 5.24: Função 3 com 10000 iterações

	2^6	2^8	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
2	6	3	1	0	1	1	0	0
3	8	3	1	1	2	0	1	0
5	16	5	3	2	0	1	0	2
10	24	10	6	1	2	4	1	2
15	24	11	7	2	2	2	2	3
20	25	9	9	2	1	3	4	2

5.4 Conclusões Finais

Verificamos nos testes que fizemos que: para as funções testes utilizadas, para o algoritmo programado como no apêndice, para as partições do domínio utilizadas neste capítulo, que o SA apresentou melhores resultados que o AG.

Entretanto devemos ressaltar que utilizamos a forma mais simples de AG. Existem outras formas, como por exemplo, o AG com o elitismo no qual o melhor ponto da população é guardado e levado para a população seguinte, só sendo trocado se um elemento melhor aparecer na população, existem outras formas de cruzamento e mutação que talvez apresentassem resultados melhores, infelizmente por falta de tempo

não podemos nos deter a estes outros tipos de AG. O que pretendemos agora é realizar novas comparações utilizando outras versões do AG.

Apêndice A

Programas

Neste apêndice apresentamos os programas que utilizamos para obtermos os resultados. Para tanto foi usado a versão 2.9.1 do *software* R.

Segue o programa para o Algoritmo *Simulated Annealing* ,

```
annealing = function(l,a,b,c,d,temp,xold,yold,fxy, totiter)
{
caminho = matrix(0,totiter,3)
fxyold = fxy(xold,yold)
otimo = hxyold
xotimo = xold
yotimo = yold
X  =  seq(a,b,(b-a)/(2^l-1))
Y  =  seq(c,d,(d-c)/(2^l-1))
iter = 1

    while(iter<totiter){

                    iter = iter+1
                    temp = temp*0.99
                    xnew = sample(X,1)
                    ynew = sample(Y,1)

                    fxynew = fxy(xnew,ynew)
                    prob = exp(-(fxynew-fxyold)/temp)
```

```
                    if(runif(1)< prob)

                                          {
                                             xold = xnew
                                             yold = ynew
                                             fxyold = fxynew
                                                              }
  caminho[iter,1] = xold
  caminho[iter,2] = yold
  caminho[iter,3] = fxyold

      xotimo = caminho[iter,1]
      yotimo = caminho[iter,2]
      otimo = caminho[iter,3]


}

return(otimo,xotimo,yotimo,temp)


   }
```

Segue o programa para o Algoritmo Genético,

```
genetic = function(l,a,b,c,d,pop,pc,pm,int,fxy) # pc - probabilidade de cruzamento.
{
A = matrix(0,pop, 2*l + 3)
k = 2*l+3
for(i in 1:pop)
{
for(j in 1:k)
{
if(j < 2*l + 1)
```

```
{
A[i,j] = sample(c(1,0),1, replace = T)

}

if(j == 2*l + 1)
{
seq1=rep(2,l)
seq2=c((l-1):0)
seq3=seq1^seq2
seq4=seq3*A[i,1:l]
Xbar=sum(seq4)
A[i,j] = a + Xbar*(b-a)/(2^l - 1)

}

if(j == 2*l + 2)
{
seq1=rep(2,l)
seq2=c((l-1):0)
seq3=seq1^seq2
seq4=seq3*A[i,((l+1):(2*l))]
Ybar=sum(seq4)
A[i,j] = c + Ybar*(d-c)/(2^l - 1)
}

if(j == 2*l + 3)
{
A[i,j] = fxy(A[i,2*l + 1],A[i,2*l + 2])
}

}
}
number = 0
repeat{
number = number + 1
```

```
# GERANDO A MATRIZ PARA SELEÇÂO #
MS1 = A[,(2*l + 3)]/(sum(A[,(2*l + 3)]))
MS2 = as.vector(NULL)
for(i in 1:pop)
{
MS2[i] = sum(MS1[1:i])
}
MS = cbind(MS1,MS2)

B1 = matrix(0,pop,k)
for(i in 1:pop)
{
R = runif(1)
DA = (MS[,2] > R)
Number = 0
for(j in 1:pop)
{
if(DA[j] == "FALSE") {Number = Number + 1} else stop
}
B1[i,] = A[Number + 1,] # B1 Matriz resultante da seleção.
}


                    ###############
                    # CRUZAMENTO #
                    ###############

          U = runif(pop)

   PC = matrix(pc,pop,1) # probabilidade de seleção de individos para cruzamento.

           SelCr = (U < PC)

           Elem = 0
           for(i in 1:pop)
{
```

```
if(SelCr[i] == "TRUE") {Elem = c(Elem,i)}
}

Elem = Elem[-1]  # Posição dos elementos a serem selecionados.
                 z=length(Elem)
                 Pontoc = floor(runif(1, 1, (2*l - 0.1))) # Ponto de Corte
                 B1new = matrix(0,pop,(2*l))
                 B1new = as.matrix(B1new)

                if( z < 2){B1new=B1[,1:(2*l)]}

                if( z > 1)

                    {
                       control=1
                       while(z>=2)
                                  {

                                Aux= matrix(0,2,2*l)
                                Aux= as.matrix(Aux)

                             for(p in 1:Pontoc)
                                  {
                                     Aux[1,p]= B1[Elem[(2*control-1)],p]
                                     Aux[2,p]= B1[Elem[(2*control)],p]
                                  }


                            for(p in ((Pontoc+1):(2*l)))
                                 {

                                    Aux[1,p]= B1[Elem[(2*control)],p]
                                    Aux[2,p]= B1[Elem[(2*control-1)],p]
                                 }

        B1[Elem[2*control-1],(1:(2*l))]=Aux[1,]
```

```
       B1[Elem[2*control],(1:(2*l))]=Aux[2,]

   control=control+1
   z=z-2

  B1new=B1[,(1:(2*l))]

}

} else stop

B1new

  ######################
#      Mutação     #
######################
N_rand = runif(pop*2*l,0,1) # Números aleatórios utilizados para a mutação.
M_mutacao = matrix(N_rand,pop,2*l) # Matriz de números aleatórios utilizados para p
M_pm = matrix(pm,pop,2*l) # Matrix dos pm's
M = (M_mutacao < M_pm)

B2New = matrix(0,pop,2*l)
for(i in (1:pop))
{
for(j in (1:(2*l)))
{
if(B1new[i,j] == 0) w = 1
if(B1new[i,j] == 1) w = -1
if(M[i,j] == "TRUE"){B2New[i,j] = B1new[i,j] + w}
if(M[i,j] == "FALSE"){B2New[i,j] = B1new[i,j]}
}
}

comp = matrix(0,pop,3)
A = cbind(B2New,comp)
for(i in 1:pop)
```

```
{
for(j in 1:k)
{
if(j == 2*l + 1)
{
seq1=rep(2,l)
seq2=c((l-1):0)
seq3=seq1^seq2
seq4=seq3*A[i,1:l]
Xbar=sum(seq4)
A[i,j] = a + Xbar*(b-a)/(2^l - 1)

}

if(j == 2*l + 2)
{
seq1=rep(2,l)
seq2=c((l-1):0)
seq3=seq1^seq2
seq4=seq3*A[i,((l+1):(2*l))]
Ybar=sum(seq4)
A[i,j] = c + Ybar*(d-c)/(2^l - 1)
}

if(j == 2*l + 3)
{
A[i,j] = fxy(A[i,2*l + 1],A[i,2*l + 2])
}
}
}

if(number == int) break
}
return(A)
}
```

Referências

[1] ANILY, S. and FEDERGRUEN, A.(1987) *Simulated Annealing methods with general acceptance probabilities* , J. Appl. Probab.,24, 657-667.

[2] CAO, Y.J. and WU, Q.H. *Theache Genetic Algorithm Using Matlab*, J. Elect. Enging. Educ, 36, 139-153.

[3] CERF, R.(1996) *A new genetic algorithm*, Ann. Appl. Probab., Volume 6, Number 3, 778-817.

[4] CERF, R.(1998) *Asymptotic Convergence of Genetic Algorithms*, Adv. Appl. Prob., 30, 521-550.

[5] CRUZ, J.A R. and DOREA, C.C.Y.(1998) *Simple Conditions for the Convergence of Simulated Annealing Type Algorithms*, Appl. Prob.,v.30, n.4, 885-892.

[6] DE JONG, K.A.(1975) *Analysis of the Behavior of a Class Genetic Adaptive Systems*, PhD. thesis, Dep Computer and Comunication Siences, Univ. Michigan, Ann Abor.

[7] DOREA, C.C.Y., JUNIOR, J.A.G., MORGADO, R. , PEREIRA, A.G.C.(2010) *Multistage Markov chain modelling of the Genetic Algorithm and convergence results*, Numerical Functional Analysis and Optimization, a aparecer.

[8] GOLDBERG, D.E.(1989) *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading MA: Addison Wesley.

[9] GIBAS, B.(1985) *Nonstationary Markov Chains and the Convergence of the Annealing Algorithm*, J. Statist. Phys., 39, 73-131.

[10] HOLLAND, J.H.(1975) *Adaptation in Natural and Artificial Systems*. Univ. Michigan Press, Ann Arbor.

[11] ISAACSON, D.L and MADSEN, R.W.(1976) *Markov Chains: Theory and applications*. Wiley, New York.

[12] KIRKPATRICK, S. GELATT, C.D. and VECCHI, M.P.JR.(1983) *Optimization by Simulated Annealing*. Science, 220:671-680.

[13] METROPOLIS, N., ROSENBLUTH, A,W.,ROSENBLUTH,M.N., TELLER A.H e TELLER, E.(1953) *Equations of state calculations by fast computing machines*. J. Chem.Phys. 21 1087-1091.

[14] NIX A. E. and VOSE M. D.(1992)*Modeling genetic algorithms with Markov chains*. Annals of Mathematics and Artificial Inteligence, 5:79-88.

[15] PASTURCZAK, R. P.(2006) Otimização Aleatória Global: O Algoritmo Simulated Annealing, Monografia, Departamento de Matemática, Universidade de Brasília.

[16] R development Core Team (2009), *R: A language and environment for statistical computing*. Vienna, Austria: R Foundation for Statistical Computing. (http://www.R-project.org). ISBN 3-900051-07-0.

[17] RUDOLPH, G.(1994) Convegence analysis of canonical genetic algorithms. IEEE Transactions on Neural Networks, special issue on Evolutionary Computation, 5, 96-101.

[18] SOARES, J.L.(1997) Algoritmos Genéticos: Estudo, Novas Técnicas e Aplicações, Dissertação de Mestrado, Centro de Pesquisa e Desenvolvimento em Engenharia Elétrica, UFMG.

[19] TIERNEY, L.(1994) *Markov Chains for exploring posterior distributions*. The annals of Statistics, 22 No. 4, 1701-1762.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo