UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

Uma Proposta de Especificação Formal e Fundamentação Teórica para Simulated Annealing

por

VANECI BRUSCH IZQUIERDO

Dissertação submetida à avaliação,
como requisito parcial para obtenção do grau de Mestre
em Ciência da Computação

Profa. Dra. LAIRA VIEIRA TOSCANI
Orientadora

Porto Alegre, julho de 2000.

CIP - Catalogação na Publicação

Izquierdo, Vaneci Brusch

Uma Proposta de Especificação Formal e Fundamentação Teórica para Simulated Annealing / por Vaneci Brusch Izquierdo. – Porto Alegre: PPGC da UFRGS, 2000.

150p.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR-RS, 2000. Orientadora: Toscani, Laira Vieira.

1. Simulated Annealing. 2. Otimização combinatória. 3. Metaheurística. 4. Desenvolvimento de algoritmos. 5. Métodos Monte Carlo. I. Toscani, Laira Vieira. III. Título.

Agradecimentos

Agradeço:

- À orientadora professora Dra. Laira Vieira Toscani, excelente amiga e mestra, pelas dedicação, disponibilidade, paciência, pelo auxílio sempre que necessário, por tudo que me ensinou não só nos aspectos técnicos, mas também como exemplo de vida;
- Aos professores, colegas e pesquisadores, que, na grande maioria sem tomarem conhecimento, colaboraram comigo e permitiram o acesso a informações que foram essenciais para o meu trabalho, em especial, aos que me dispensaram o seu precioso tempo;
- Aos colaboradores da professora Laira Vieira Toscani que conviveram e colaboraram comigo, pela atenção e incentivo;
- Aos funcionários do Instituto de Informática pela disponibilidade dos recursos e atenção, em especial, aos funcionários da Biblioteca e laboratórios;
- Aos amigos, colegas antigos de trabalho, que me incentivaram para iniciar o curso;
- À minha família, pelo carinho com que acompanharam e pela compreensão pelo tempo que dediquei ao curso e que roubei deles;
- Ao CNPq, pelo auxílio em forma de bolsa.

Dedico este trabalho aos

queridos Juarez, Jorge e Joel.

Sumário

Lista de Símbolos

φ	Asserção de entrada
ψ	Asserção de saída
$\|C\|$	Cardinalidade do conjunto C
L	Comprimento da cadeia de Markov
$\rightarrow$	Conetivo lógico condicional
$\wedge$	Conetivo lógico E
$\neg$	Conetivo lógico Não
$\vee$	Conetivo lógico OU
S_{opt}	Conjunto de soluções ótimas
S_i	Conjunto de soluções vizinhas (vizinhança)
$\aleph$	Conjunto dos números naturais
$\Re$	Conjunto dos números reais
k_B	Constante de Boltzman
f_E	Função energia
f_{opt}	Custo ótimo
σ	Desvio padrão
$\neq$	diferente
q	Distribuição estacionária
q^*	Distribuição estacionária
E	Energia
S	Entropia
S	Espaço de soluções (conjunto de soluções)
$\subset$	Está contido
$\subseteq$	Está contido e igual

N	Estrutura de vizinhança
α	Fator para decréscimo de temperatura
$\chi_{(A)}(a)$	Função característica
f	Função objetivo
d	Função perturbação
$\Rightarrow$	Implicação
∞	Infinito
$\cap$	Interseção
m	Movimento ou transição
$\leftarrow$	Operador de atribuição
δ	Parâmetro de distância
ε_s	Parâmetro de término
t	Parâmetro temperatura
$\geq$	Precedência maior e igual
$\leq$	Precedência menor e igual
A_{ij}	Probabilidade de aceitação
G_{ij}	Probabilidade de geração
$\prod$	Produtório
Π	Programa
$\exists$	Quantificador existencial
$\forall$	Quantificador universal
$\notin$	Relação de não-pertinência
$\in$	Relação de pertinência
$\Leftrightarrow$	Se, e somente se
$\sum$	Somatório
Θ	Tamanho da vizinhança
χ	Taxa de aceitação

T	Temperatura
$\cup$	União
$\langle x \rangle$	Valor esperado de x
σ^2	Variância
$N(s)$	Vizinhança da solução s

Lista de Figuras

Lista de Tabelas

Resumo

Os algoritmos baseados no paradigma Simulated Annealing e suas variações são atualmente usados de forma ampla na resolução de problemas de otimização de larga escala. Esta popularidade é resultado da estrutura extremamente simples e aparentemente universal dos algoritmos, da aplicabilidade geral e da habilidade de fornecer soluções bastante próximas da ótima. No início da década de 80, Kirkpatrick e outros apresentaram uma proposta de utilização dos conceitos de annealing (resfriamento lento e controlado de sólidos) em otimização combinatória. Esta proposta considera a forte analogia entre o processo físico de annealing e a resolução de problemas grandes de otimização combinatória. Simulated Annealing (SA) é um denominação genérica para os algoritmos desenvolvidos com base nesta proposta. Estes algoritmos combinam técnicas de busca local e de randomização. O objetivo do presente trabalho é proporcionar um entendimento das características do Simulated Annealing e facilitar o desenvolvimento de algoritmos com estas características. Assim, é apresentado como Simulated Annealing e suas variações estão sendo utilizados na resolução de problemas de otimização combinatória, proposta uma formalização através de um método de desenvolvimento de algoritmos e analisados aspectos de complexidade. O método de desenvolvimento especifica um programa abstrato para um algoritmo Simulated Annealing seqüencial, identifica funções e predicados que constituem os procedimentos deste programa abstrato e estabelece axiomas que permitem a visualização das propriedades que estes procedimentos devem satisfazer. A complexidade do Simulated Annealing é analisada a partir do programa abstrato desenvolvido e de seus principais procedimentos, permitindo o estabelecimento de uma equação genérica para a complexidade. Esta equação genérica é aplicável aos algoritmos desenvolvidos com base no método proposto. Uma prova de correção é apresentada para o programa abstrato e um código exemplo é analisado com relação aos axiomas estabelecidos. O estabelecimento de axiomas tem como propósito definir uma semântica para o algoritmo, o que permite a um desenvolvedor analisar a correção do código especificado para um algoritmo levando em consideração estes axiomas. O trabalho foi realizado a partir de um estudo introdutório de otimização combinatória, de técnicas de resolução de problemas, de um levantamento histórico do uso do Simulated Annealing, das variações em torno do modelo e de embasamentos matemáticos documentados. Isto permitiu identificar as características essenciais dos algoritmos baseados no paradigma, analisar os aspectos relacionados com estas características, como as diferentes formas de realizar uma prescrição de resfriamento e percorrer um espaço de soluções, e construir a fundamentação teórica genérica proposta.

Palavras-chaves: Simulated Annealing, Otimização Combinatória, Metaheurística, Desenvolvimento de Algoritmos, Métodos Monte Carlo.

TITLE: “A formal specification and theoretical basement proposal for Simulated Annealing”.

Abstract

The algorithms based on Simulated Annealing paradigm and their variations have been widely used in large-scale optimization problems. Reasons of their popularity are the extremely simple and apparently universal structure of the algorithms, the general applicability and the ability to provide extremely near optimum solutions. In the early 1980’s Kirkpatrick and others introduced the concepts of annealing in combinatorial optimization. These concepts are based on a strong analogy between the physical annealing process of solids and the problem of solving large combinatorial optimization problems. Simulated Annealing (SA) is a generic denomination for algorithms based on these concepts. These algorithms combine local search and randomization techniques. The objective of the present work is to provide an understanding of Simulated Annealing characteristics and to allow the development of algorithms of such kind. So, it is presented how Simulated Annealing and their variations have been used to solve combinatorial optimization problems. Formalization by an algorithm development method is proposed and the complexity aspects are analyzed. The development method specifies an abstract program for a sequential Simulated Annealing algorithm, identifies functions and predicates that are the abstract program procedures and establishes axioms that allows the visualization the properties these procedures must satisfy. The complexity of the Simulated Annealing is analyzed from the abstract program developed and of its main procedures, allowing the establishment of a generic equation for the complexity. This generic equation is applicable to the developed algorithms on the basis of the considered method. The correction prove is presented to the abstract program and a code example is analyzed and relate to the established axioms. The axioms, used to define a semantic for the algorithm, allow a developer to analyze the correction of the code specified for an algorithm taking into account these axioms. The work was developed following an introductory study about combinatorial optimization, problems solving techniques, a survey about Simulated Annealing and variants and about documented mathematics foundation. That allowed the identification of the algorithms essentials characteristics based on the paradigm, analyzing the aspects related to these characteristics, such different ways to achieve a cooling schedule and search through a solution space, and develop the generic theoretical basement proposal.

Keywords: Simulated Annealing, Combinatorial Optimization, Metaheuristic, Algorithms Development, Monte Carlo Methods.

1 Introdução

1.1 Tema

Simulated Annealing (SA) é uma técnica difundida a partir de trabalhos de Kirkpatrick [KIR 83] e outros na década de 80. Esta técnica, originada em conceitos de mecânica estatística, estabelece a aplicação do algoritmo de Metropolis [MET 53] para resolução de problemas de otimização.

Annealing é o processo físico que consiste do aquecimento de um sólido em banho térmico até tornar-se líquido, seguido do resfriamento lento e controlado até cristalização formando-se uma estrutura molecular estável. Durante este processo a energia livre do sólido é minimizada. Em otimização combinatória, é possível definir um processo similar. Este processo pode ser formulado como o problema de encontrar – entre o grande número de soluções em potencial - uma solução com custo mínimo. Assim, estabelecendo-se uma correspondência entre a função custo e a energia livre, e entre as soluções e os estados físicos, é definido um método de resolução para problemas de otimização combinatória baseado no processo físico de Annealing. Os algoritmos desenvolvidos usando estes conceitos recebem o nome genérico de "Simulated Annealing" (Recozimento Simulado).

Recozimento (annealing), segundo a norma ABNT - NBR 8653 de Novembro de 1984, é um termo genérico que indica um tratamento térmico composto de aquecimento controlado até uma determinada temperatura, permanência nessa temperatura durante um certo intervalo de tempo e resfriamento regulado para a finalidade em vista.

A aplicabilidade geral e a habilidade de fornecer soluções de qualidade bastante próximas da ótima são apontadas como as principais vantagens do Simulated Annealing [AAR 89].

O Simulated Annealing é utilizado em muitos tipos de aplicações como: (a) na resolução dos problemas clássicos de otimização combinatória, como o problema do caixeiro-viajante e problemas de grafos em geral; (b) na resolução dos problemas de projeto de circuitos; (c) em análise de dados; (d) no processamento de imagens; (e) em redes neurais, como na "máquina Boltzmann"; (f) em biologia molecular, como para análise da estrutura de cadeias complexas de proteínas; (g) em física, como no modelo Ising do magnetismo; (h) em geofísica, como para modelar o comportamento de ondas sísmicas; (i) em finanças, na resolução de problemas de previsão de investimentos; e, (j) na área militar, em problemas de defesa [AAR 89] [ING 93].

1.2 Objetivos

Os objetivos estabelecidos para o trabalho são:

- Identificar as principais características do algoritmo SA seqüencial;
- Ajudar no entendimento destas características;
- Prover uma melhor compreensão do porque estas características são importantes, como elas interagem entre si e o efeito das mesmas sobre a performance do SA;
- Identificar os principais procedimentos componentes do algoritmo e o relacionamento entre os mesmos;
- Construir uma fundamentação teórica do algoritmo;
- Representar o conhecimento sobre o algoritmo através de um formalismo;
- Estabelecer, usando este formalismo, os requisitos que os procedimentos que compõe o algoritmo devem satisfazer.

1.3 Revisão de Literatura

Esta seção relaciona os principais autores e suas obras por assunto. No entanto, toda a literatura referenciada e/ou mencionada na seção Bibliografia foi utilizada como fonte de consulta e pesquisa.

Algoritmos e Complexidade

- Garey e Johnson [GAR 79] – livro básico para o estudo da intratabilidade de problemas.
- Horowitz e Sahni [HOR 78] – fundamentos de algoritmos, estruturas de dados e técnicas de resolução.
- Toscani e Szwarcfiter [TOS 86] – uma introdução a problemas de otimização e algoritmos aproximativos.
- Toscani [TOS 88] – estabelece um método para desenvolvimento de algoritmos.

Heurísticas e Metaheurísticas

- Osman e Kelly [OSM 96]: uma introdução sobre metaheurísticas e capítulos sobre algoritmos genéticos, simulated annealing e busca tabu.
- Campello [CAM 94]: estudo básico dos problemas caixeiro-viajante, mochila, coloração de grafos e outros e técnicas heurísticas aplicáveis aos mesmos.
- Muller [MUL 96]: texto sobre heurísticas e metaheurísticas em geral.

Métodos Monte Carlo / Geradores de Números Aleatórios / Processos Estocásticos / Cadeias de Markov

- Kalos e Whitlock [KAL 86] – uma boa introdução aos métodos Monte Carlo de forma geral e, em especial, algoritmo de Metropolis. Tipos e testes de geradores de números aleatórios.
- Press e outros [PRE 88] – capítulo sobre tipos de geradores de números aleatórios, métodos de rejeição e integração Monte Carlo.
- Kovacs [KOV 96] – um livro básico sobre teoria da probalidade e processos estocásticos.
- Knuth [KNU 81] – teoria sobre o que é uma seqüência aleatória, geradores de números aleatórios e testes estatísticos de verificação.
- Rosenthal e outros [ROB 97] [ROS 99] – diversos documentos recentes sobre cadeias de markov e convergência de algoritmos MCMC (Monte Carlo Markov Chain).

Otimização

- Crescenzi e Kann [CRE 97] – um compêndio dos problemas de otimização matemática, especificando cada problema e a sua situação atual relativa à aproximabilidade.

Simulated Annealing

- Aarts e Korst [AAR 89] – livro básico sobre Simulated Annealing, contém os fundamentos básicos teóricos do algoritmo e prova de convergência baseada na teoria de cadeias de Markov.
- Azencott [AZE 92] – introdução sobre Simulated Annealing seqüencial e técnicas de paralelização.
- Johnson, Aragon, McGeoch, Schevon [JOH 89] [JOH 91] – artigos que avaliam experimentalmente o Simulated Annealing.

- Kirkpatrick, Gellat e Vecchi [KIR 83] – um dos primeiros artigos publicados sobre Simulated Annealing, introduzindo a aplicação dos conceitos de mecânica estatística à resolução dos problemas de otimização, utilizando o algoritmo de Metropolis.
- Torreão [TOR 92] – apresenta de forma clara uma introdução aos conceitos de mecânica estatística e Simulated Annealing.

Simulated Annealing e TSP

- Johnson e outros [JOH 97] – bem elaborado e detalhado capítulo sobre o problema do caixeiro-viajante e técnicas de busca local.

1.4 Organização do Texto

O presente trabalho é constituído dos seguintes capítulos: o capítulo 1 (presente) é uma introdução geral apresentando o tema do trabalho, objetivos e revisão bibliográfica; o capítulo 2 introduz uma fundamentação teórica da otimização; o capítulo 3 enfoca o Simulated Annealing, suas origens, características e componentes principais e desenvolve uma representação formal através de um algoritmo abstrato; o capítulo 4 apresenta como o algoritmo vem sendo utilizado através de suas variações e aplicações documentadas na literatura; e, finalmente, o capítulo 5 apresenta conclusões sobre os assuntos abordados e resultados alcançados. No anexo A1 é apresentada a verificação da correção da especificação para o programa abstrato proposto. No anexo A2 são analisados, com maior profundidade, alguns aspectos técnicos de convergência e complexidade do algoritmo.

2 Otimização

A otimização envolve a procura da melhor forma de realizar determinadas tarefas e sua aplicação por engenheiros, administradores, economistas, nutricionistas e outros profissionais nas mais diversas áreas de atuação, é um fator muito importante para a tomada de decisões.

A teoria da otimização é conhecida dos matemáticos a séculos, mas a tediosa e volumosa computação dificultou muitas aplicações práticas. O desenvolvimento de computadores cada vez mais rápidos, não só tornou atrativos os velhos métodos como encorajou novas pesquisas.

A maioria dos problemas encontrados na prática e que têm um número finito ou infinito contável de soluções alternativas podem ser formulados como problemas de otimização combinatória.

Otimização Combinatória pode ser definida como a parte do estudo matemático que busca encontrar um ótimo arranjo, agrupamento, ordenação ou seleção de objetos discretos usualmente de número finito [OSM 96].

Nesta seção serão apresentados conceitos relativos a problemas em geral, e, em especial, a problemas de otimização e a técnicas heurísticas de busca local.

2.1 Problemas de Otimização

Um problema [GAR 79] é uma questão geral a ser respondida, usualmente possuindo vários parâmetros, ou seja, variáveis livres, cujos valores não são estabelecidos. Um problema é especificado por uma descrição genérica de todos os seus parâmetros e o estabelecimento das propriedades que a resposta, ou solução, deve satisfazer.

Uma instância de um problema é obtida pela atribuição de valores particulares para todos os seus parâmetros.

Um problema pode ser de decisão, de localização ou de otimização [SZW 84]. Um problema de decisão é qualquer problema cuja solução é "*sim*" ou "*não*". Em um problema de localização procura-se localizar uma certa estrutura que satisfaça um conjunto de propriedades dadas. Em um problema de otimização procura-se a melhor estrutura que satisfaça um determinado conjunto de propriedades.

2.1.1 Definição de Problema de Decisão

Um problema de decisão fica bem caracterizado pela definição dada por Bovet [BOV 94]:

Definição 2.1: Um problema de decisão é uma tripla $\langle X, sol, \pi \rangle$ tal que

X é um conjunto de instâncias x do problema.

Dado uma instância x de X, $sol(x)$ denota o conjunto de soluções possíveis de x.

π é um predicado tal que, para qualquer instância x e para qualquer possível solução $s \in sol(x)$, $\pi(x, s)$ é verdadeiro se e somente se s é uma solução viável de x.

Resolver um problema de decisão $\langle X, sol, \pi \rangle$ consiste em decidir, para uma dada instância $x \in X$, se o conjunto $\{s : s \in sol(x) \wedge \pi(x, s)\}$ não é um conjunto vazio.

Os problemas de decisão são de grande importância teórica pois a Teoria da Complexidade usa os mesmos como base para classificação de problemas [GAR 79] [BOV 94]. A classe de complexidade P inclui todos os problemas de decisão que podem ser resolvidos em tempo polinomial. A classe de complexidade NP inclui todos os problemas de decisão que podem ser resolvidos em tempo polinomial por um algoritmo não determinístico. Claramente, $P \subseteq NP$. A conjetura $P \neq NP$ continua não resolvida após muitos anos de pesquisa e sua resolução em um futuro próximo parece remota.

2.1.2 Definição de Problema de Otimização

Um problema de otimização pode ser visto como constituído de três elementos básicos:

Uma função objetivo a qual se deseja minimizar ou maximizar. Por exemplo, em um processo de manufatura, o objetivo pode ser maximizar o lucro ou minimizar o custo.

Um conjunto de variáveis ou desconhecidos que afetam o valor da função objetivo. No processo de manufatura estas variáveis incluem quantidades de diferentes recursos usados ou o tempo gasto em cada atividade.

Um conjunto de restrições que definem que as variáveis podem assumir determinados valores e excluem outros. No processo de manufatura, as variáveis associadas a gasto de tempo em atividades devem ser restritas a valores não negativos.

Resumindo: resolver um problema de otimização consiste em encontrar valores para as variáveis que minimizem ou maximizem a função objetivo mantendo satisfeitas as restrições.

Na resolução de um problema de otimização são considerados os seguintes aspectos principais [NIE 95]: (a) espaço de soluções, (b) estrutura de vizinhança, (c) meta ou critério de busca, (d) algoritmo de busca, (e) estruturas de dados.

- Espaço de soluções ou espaço de busca é um conjunto apropriado de soluções que correspondem a uma dada instância de um problema. Nos problemas de otimização este conjunto é uma estrutura rica chamada de "espaço". Projetar um espaço de soluções efetivo é o primeiro e principal passo para obter-se bons resultados. É importante lembrar que a construção de um espaço de soluções é uma oportunidade de conhecer os aspectos específicos do problema e possibilitar o uso eficiente de uma determinada técnica de busca. Uma solução para o problema é uma estrutura ou configuração candidata que obedeça as regras estabelecidas pelas variáveis e restrições.

- Estrutura de vizinhança é uma relação, usualmente incluída no espaço de soluções, que expressa restrições na forma de percorrer o espaço de estados, ou seja, expressa como ir para uma solução adjacente a partir de uma solução atual. Esta solução adjacente é uma vizinha da solução atual. Assim, o espaço de soluções é freqüentemente modelado como um grafo com arestas direcionadas ou não. Estas arestas conectam as soluções vizinhas.

- A meta ou critério de busca pode variar muito com relação a quantidade ou qualidade: de uma solução simples desejada a uma enumeração exaustiva de todas as soluções do espaço. A meta faz parte do problema, no entanto, muitas vezes é realizada uma relaxação desta meta possibilitando encontrar um objetivo parcial mais rapidamente. Em geral, é atribuída uma medida a cada solução do espaço e esta medida é usada como forma de avaliar as soluções estabelecendo um critério para alcançar a meta, seja de forma exata ou aproximada. Nos problemas de otimização a meta é maximizar ou minimizar uma função objetivo, sendo esta função objetivo a forma de avaliar a qualidade de cada solução.

- O algoritmo de busca especifica a forma como explorar o espaço de soluções. A estratégia a utilizar na resolução de problemas de otimização combinatória depende de uma série de fatores como finalidade teórica ou prática, tipo e abrangência de aplicação, freqüência de uso da aplicação. Neste trabalho são focalizados os algoritmos baseados no paradigma Simulated Annealing.

As estruturas de dados são usadas para suportar a busca. Salvo quando essenciais para o algoritmo não são detalhadas neste trabalho, ou seja, são tratadas estruturas abstratas de dados.

Um problema de otimização fica bem caracterizado pela definição a seguir, baseada em Bovet [BOV 94] e Crescenzi [CRE 97]. Esta definição considera uma classe de problemas de otimização para os quais os problemas de decisão subjacente estão em NP:

Definição 2.2: Um problema de otimização Π é uma tupla $\langle X, sol, \pi, f_{obj}, meta \rangle$ tal que

(a) X é um conjunto de instâncias onde cada instância x é reconhecível em tempo polinomial.

(b) Dado uma instância x de X, $sol(x)$ denota o conjunto de soluções possíveis de x. Estas soluções são pequenas, isto é, existe um polinômio p tal que, para qualquer $i \in sol(x)$, $|i| \leq p(|x|)$. Além disso, é possível decidir em tempo polinomial quando, para qualquer x e para qualquer i tal que $|i| \leq p(|x|)$, se $i \in sol(x)$.

(c) π é um predicado tal que, para qualquer instância x e para qualquer possível solução $i \in sol(x)$, $\pi(x, i)$ é verdadeiro se e somente se i é uma solução viável de x; é assumido que, para qualquer instância x, existe pelo menos uma solução de x e é possível identificar em tempo polinomial se $\pi(x, i)$ é verdadeiro.

(d) Dado uma instância x e uma solução viável i de x, $f_{obj}(x, i)$ denota um valor positivo inteiro que é uma medida de i. A função f_{obj}, chamada de *função objetivo,* é computável em tempo polinomial.

(e) $meta \in \{max,min\}$.

Resolver um problema de otimização (X, sol, π, f_{obj}, $meta$) consiste em encontrar uma *solução ótima* para uma instância x, isto é, uma solução viável tal que

$$f_{obj}(x, i_{opt}) = meta \{ f_{obj}(x, i) : i \in sol(x) \wedge \pi(x, i) \}.$$

Ainda opt denotará uma função que associa a medida de uma solução ótima com qualquer instância $x \in X$.

O conjunto de todos os problemas de otimização, acima definidos, constituem a classe NPO.

O subconjunto de soluções $sol_v(x) \subseteq sol(x)$ para as quais o predicado π é verdadeiro é chamado conjunto de soluções viáveis.

O subconjunto de soluções $sol_{opt}(x) \subseteq sol_v(x)$ é o conjunto constituído das soluções de valor ótimo.

Neste trabalho uma instância $x \in X$ de um problema de otimização será denotada por uma dupla (S, f) onde S é o conjunto de soluções e f a função objetivo.

2.1.3 Exemplos de Problemas de Otimização

Uma lista de problemas de otimização da classe NPO pode ser encontrada no compêndio desenvolvido por Crescenzi e Kann [CRE 97]. Esta lista define cada problema, identifica a situação do mesmo com referência a aproximalidade e relaciona com a lista de problemas publicada por Garey e Johnson [GAR 79].

A seguir são relacionados os problemas de otimização utilizados como exemplos nas próximas seções:

Problema PCV:

Caixeiro-viajante (minimum travelling salesperson – problema ND29 em [CRE 97] – problema ND22 em [GAR 79]):

Instância: conjunto C de m cidades, distâncias $d(c_i,c_j) \in N$ para cada par de cidades $c_i,c_j \in C$.
Solução: uma rota de C, isto é, uma permutação $\pi : [1..m] \rightarrow [1..m]$.
Função objetivo: o comprimento da rota, isto é,

$$d\left(\left\{c_{\pi(m)}, c_{\pi(1)}\right\}\right) + \sum d\left(\left\{c_{\pi(i)}, c_{\pi(i+1)}\right\}\right).$$

Meta: encontrar rota de custo mínimo passando uma única vez em cada cidade.
Obs.: um estudo detalhado do problema pode ser encontrado em [CAM 94] e [JOH 97].

Problema PPG:
Particionamento de grafos ([JOH 89] :
Instância: grafo $G = (V, A)$, onde V é um conjunto de vértices (com $|V|$ par) e A é um conjunto de arestas.
Solução: partição de V em 2 subconjuntos de mesmo tamanho.
Função objetivo: número de arestas em A com vértices em ambos os subconjuntos.
Meta: minimizar o número de arestas com vértices em ambos os subconjuntos.

Problema PCG:
Coloração de grafos (minimum graph coloring, minimum chromatic number [JOH 91] CAM 94] – problema GT5 em [CRE 97]) – problema GT4 em [GAR 79]):
Instância: grafo $G = (V, A)$, onde V é um conjunto de vértices e A é um conjunto de arestas.
Solução: partição de V em subconjuntos disjuntos V_1, V_2, ...,V_k tal que cada V_i é um conjunto independente para G (uma coloração de G).
Função objetivo: cardinalidade da coloração, isto é, o número de conjuntos disjuntos independentes.
Meta: mínima coloração.
Obs.: um estudo básico do problema pode ser encontrado em [CAM 94].

Problema PPN:
Particionamento de Números (number partitioning [JOH 91):
Instância: conjunto $A = \{a_1, a_2,..., a_n\}$, de números no intervalo [0,1].
Solução: partição de A em 2 conjuntos disjuntos A_1, A_2.
Função objetivo:

$$\left| \sum_{a_i \in A_1} a_i - \sum_{a_i \in A} a_i \right|$$

Meta: minimizar.

2.2 Espaço de Soluções

A terminologia "o espaço de soluções do problema", utilizada com freqüência, não é correta. O espaço de soluções não é um atributo do problema, mas é projetado de forma a modelar adequadamente a instância do problema. Uma instância pode ter muitos espaços de soluções plausíveis com grandes diferenças em estrutura, tamanho e facilidade de entendimento. Em muitos casos, projetar um espaço de soluções, entender sua estrutura e encontrar uma representação adequada é um dos mais importantes ingredientes para a obtenção de um resultado eficiente [NIE 95].

Quando o problema de otimização é combinatório pode-se, informalmente, definir estrutura de uma instância como uma lista das soluções onde cada solução $s \in S$ é associada ao valor desta solução dado pela função objetivo f e associada ao predicado π que identifica se a solução é viável ou não. Esta estrutura corresponde a uma enumeração explícita das soluções que constituem o espaço de soluções a ser percorrido. É claro que esta enumeração explícita nem sempre é possível por desconhecimento de algum elemento ou simplesmente devido a explosão combinatória.

Na especificação de um espaço de soluções adequado é importante estabelecer se serão incluídas soluções não viáveis e o impacto desta inclusão nos outros aspectos a serem considerados, como na avaliação da função objetivo.

O segundo passo é escolher a representação da solução. A representação da solução é dependente do problema. No entanto, estudos tem sido realizados no sentido de encontrar uma modelagem adequada que permita ao usuário especificar um problema usando alguma notação matemática a qual pudesse ser processada diretamente por um algoritmo de propósito geral [ABR 97]. Isso tem sido conseguido, dentro de certos limites, para problemas que seguem determinados padrões, ou seja, por exemplo, para problemas que possam ser especificados usando variáveis com valores restritos a 0 e1 e função objetivo e restrições lineares. Abramson [ABR 97] propõe uma formulação genérica para alguns tipos de problemas que podem ser especificados usando variáveis inteiras. Outras propostas têm como objetivo uma formalização mais ampla, por exemplo, usando algum tipo de especificação algébrica [HEL 88].

Uma formalização para o espaço de soluções deve ter propriedades que permitam:

- caracterizar a instância do problema como uma entrada a ser tratada por um algoritmo de propósito geral;
- representar a solução;
- representar os elementos que compõe cada solução (aspectos combinatórios);
- caracterizar os movimentos de uma solução para outra, os quais permitem que um determinado algoritmo explore um espaço de soluções, permitindo representar como uma solução é perturbada (modificada) para obter-se uma nova solução;

- caracterizar a função objetivo como uma medida de qualidade da solução, gerando uma ordenação para o conjunto de soluções.

2.3 Vizinhança

Para cada espaço de soluções especificado para um determinado problema de otimização é possível estabelecer uma ou mais estruturas de vizinhanças. A definição formal de problema de otimização dada em (2.2) não inclui a vizinhança, pois, mesmo sendo um aspecto dependente do problema, ele está relacionado com a forma de resolvê-lo, ou seja, a técnica de resolução pode ou não utilizar uma estrutura de vizinhança. As técnicas que se baseiam na utilização de uma estrutura de vizinhança para percorrer o espaço de soluçoes são conhecidas como técnicas de busca local e serão introduzidas na seção 2.4.

Estrutura de Vizinhança

Definição 2.3 [AAR 89]: Seja S o espaço de soluções de uma instância de um problema de otimização combinatória. Uma estrutura de vizinhança é definida por um mapeamento

$$N : S \rightarrow 2^S,$$

o qual define para cada solução $i \in S$. um conjunto $S_i \subset S$ de soluções que são "próximas" de i de alguma forma. O conjunto S_i é chamado de vizinhança da solução i, e cada $j \in S_i$ é chamado uma solução vizinha de i.

A estrutura de vizinhança N pode ser representada por um grafo direto $G = (V,A)$ onde $V = S$ e $(i, j) \in A \Leftrightarrow j \in S_i$. G é chamado de grafo de vizinhança.

Uma vizinhança é conectada fracamente se for possível encontrar uma solução ótima a partir de qualquer solução e conectada fortemente se for possível encontrar cada solução a partir de qualquer outra solução.

As vizinhas de uma solução não são obtidas explicitamente, mas podem ser construídas por uma função computável em tempo polinomial. Esta função é denotada como função perturbação. Muitas vizinhanças são construídas baseadas na substituição de alguns poucos elementos que constituem a solução por outros.

Métricas

Para melhor caracterizar uma vizinhança é importante estabelecer o que é "proximidade", ou seja, determinar o significado de distância entre duas soluções. Esta noção de proximidade é influenciada pelo forma como é representada a solução e, pode ser melhor entendida se for estabelecida uma métrica adequada para o espaço de soluções.

Portanto, inicialmente serão lembradas as definições de métrica e vizinhança (topologia geral):

Definição 2.4 [STV 94]: Seja $X \neq \varnothing$ um conjunto. Uma métrica em X é uma função $\rho : X \times X \rightarrow \Re$ onde para cada $x, y, z \in X$,

(a) $\rho(x , y) \geq 0$; $\rho(x , y) = 0 \Leftrightarrow x = y$;

(b) $\rho(x, y) = \rho(y, x)$;

(c) $\rho(x , y) + \rho(x , z) \geq \rho(y, z)$.

O valor de ρ é chamado distância. O par (X, ρ) é chamado um espaço métrico. Cada métrica estabelecida em X gera uma topologia via uma base topológica de "bolas"ou "esferas" abertas:

$$B(x, r) = \{ y \in X : \rho(x, y) < r\}.$$

A identificação de uma métrica permite definir vizinhança de uma forma mais precisa, onde o significado de "proximidade" é estabelecido através desta métrica, conforme a definição abaixo:

Definição 2.5 [SHR 91]: Seja uma esfera com centro x_0 e raio r. X é um conjunto de pontos x_i no espaço S, tal que que $\rho(x_0, x_i) \leq r$. Uma vizinhança N_r de $x_0 \in S$ é qualquer esfera com centro x_0 e raio $\leq r$.

Na otimização, tradicionalmente, a vizinhança inclui a fronteira. No caso dos problemas de otimização combinatória, onde cada solução é um elemento discreto do conjunto de soluções, a vizinhança pode ser vista, então, como uma esfera fechada.

No caso de problemas onde o espaço de soluções é contínuo é usualmente executada alguma forma de discretização, pois computacionalmente todos os espaços são discretos.

Quando for possível estabelecer uma métrica para o espaço de soluções e uma estrutura de vizinhança baseadas nas definições (2.4) e (2.5) acima, então as seguintes propriedades devem ser satisfeitas:

(a) propriedade 1 - não existe vizinhança vazia, pois o centro é elemento da vizinhança;

(b) propriedade 2 (simetria) - uma solução y é vizinha de x então a solução x é vizinha de y;

(c) propriedade 3 (desigualdade triangular) - a menor distância entre dois elementos é o caminho direto, se este caminho existir.

A escolha de uma métrica permite verificar se o tamanho da vizinhança é de ordem polinomial e se a complexidade da procura (perturbação e avaliação da nova solução) também é polinomial.

A seleção de uma métrica é dependente do problema e reflete a estrutura do mesmo, mas deve obedecer as propriedades da definição (2.4).

Entre estas métricas podem ser mencionadas [SHR 91]:

- Métrica em um espaço de permutações

  Seja M o conjunto de objetos combinatórios, então o conjunto de todas as permutações possíveis S contém $|M|!$ elementos. Seja s_i, s_j e s_k elementos do conjunto de permutações. Uma distância $\rho(s_i , s_j)$ entre estes elementos s_i e s_j é definida como o número mínimo de permutações elementares que transformam s_i em s_j, onde uma permutação elementar é a troca de posições entre dois elementos do conjunto ordenado M. Esta distância assim definida satisfaz os axiomas de uma métrica de espaços, ou seja:

  (a) $\rho(s_i , s_j) = 0$ se e somente se $s_i = s_j$;

  (b) $\rho(s_i , s_j) = \rho(s_j, s_i)$;

  (c) $\rho(s_i , s_j) + \rho(s_i , s_k) \geq \rho(s_i , s_k)$, porque s_i pode ser transformada em s_k em $\rho(s_i , s_j) + \rho(s_i , s_k)$ permutações elementares (mas, este número não é necessariamente o mínimo).

  Assim, uma esfera de raio 1 inclui os pontos possíveis de serem encontrados a partir de um ponto inicial através de uma permutação e uma esfera de raio 2 inclui os pontos possíveis de serem encontrados a partir de um ponto inicial através de duas permutação, etc.

- Métrica em um espaço de amostras

  Seja M um conjunto de objetos a partir do qual amostras são selecionadas. Seja B o conjunto de todos as amostras. Seja x, y e z amostras diferentes deste conjunto B. Seja a métrica definida como $\rho(x , y) = |(x \cup y) \setminus (x \cap y)|$. Esta distância assim definida satisfaz os axiomas de uma métrica de espaços, ou seja:

  a) $\rho(x , y) = 0$ se e somente se $x = y$;

  b) $\rho(x, y) = \rho(y, x)$;

  c) $\rho(x , y) + \rho(x , z) \geq \rho(y, z)$, pois

$$\begin{aligned}\rho(x , y) + \rho(x , z) &= |(x \cup y) \setminus (x \cap y)| + |(x \cup z) \setminus (x \cap z)| \\ &\geq |[(x \cup y) \setminus (x \cap y)] \cup [(x \cup z) \setminus (x \cap z)] \\ &= |(x \cup y \cup z) \setminus (x \cap y \cap z)| \\ &\geq |(x \cup z) \setminus (x \cap z)|\end{aligned}$$

$= \rho(x, z)$.

este resultado é baseado nas seguintes condições $(x \cap y \cap z) \neq \varnothing$ e a inclusão $(x \cup y \cup z) \supset (y \cup z) \setminus (y \cap z)$.

Uma métrica no espaço de permutações é adequada para problemas tipo do caixeiro-viajante (PCV) onde cada solução(rota) corresponde a uma permutação de objetos combinatórios e cada objeto um segmento desta rota. Uma métrica no espaço de amostras é adequada para problemas tipo coloração de grafos (PCG).

Ótimo Local

Estabelecido um conceito de vizinhança é possível definir solução localmente ótima ou ótimo local como uma solução melhor ou igual a qualquer outra quando comparada com as soluções vizinhas levando-se em consideração a função objetivo.

Definição2.6: Seja (S, f) uma instância de um problema de otimização e N uma estrutura de vizinhança. Uma *solução localmente ótima ou ótimo local* $i_{optl} \in S$ é uma solução melhor ou igual a qualquer outra quando comparada com as soluções vizinhas levando-se em consideração a função objetivo. Pode ser um mínimo ou um máximo local, ou seja, para minimização ($f(i_{optl}) \leq f(j)$), para todo j vizinho de i_{optl} ou maximização ($f(i_{optl}) \geq f(j)$), para todo j vizinho de i_{optl}, respectivamente.

Uma solução é um *mínimo local forte* se $f(i_{optl}) < f(j)$ e um *mínimo local fraco* se $f(i_{optl}) \leq f(j)$ para todo j vizinho de i_{optl} (minimização).

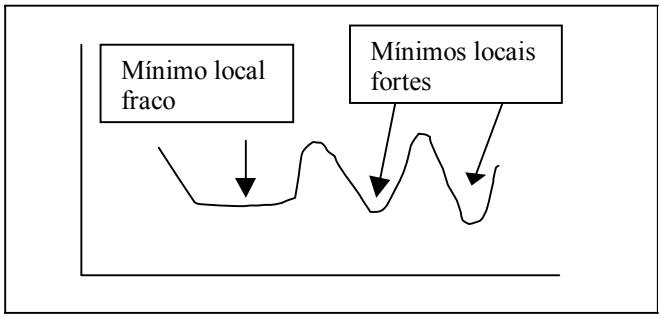

FIGURA 2.1 – Mínimos Locais

Quando numa vizinhança N cada $i_{optl} \in S$ que é um ótimo local com referência a N é também um ótimo global a vizinhança N é chamada exata.

2.4 Algoritmos

Neste trabalho, algoritmo é utilizado no sentido abrangente como um procedimento genérico, especificado passo a passo, utilizado para resolver problemas. Diz-se que um algoritmo A resolve um problema Π qualquer se o algoritmo A pode ser aplicado para qualquer instância x de Π e garante sempre fornecer uma solução para esta instância [GAR 79].

Resolver, ou seja, encontrar uma solução ótima para muitos problemas de otimização, ainda hoje é inviável, mesmo com auxílio dos computadores com grande capacidade de processamento disponíveis, pois o tempo de computação dos algoritmos cresce de forma exponencial relativamente ao tamanho da instância do problema. Estes problemas são vistos como *Intratáveis* pela Teoria da Complexidade. Mais precisamente, os resultados dos estudos da NP-Completude, já na década de 70, mostraram que, a menos que P = NP , a grande maioria dos problemas de otimização combinatória originados nas diversas áreas das ciências, engenharias e comércio são intratáveis, isto é, não existem métodos de resolução que evitem a explosão combinatória [KAR 86].

2.4.1 Algoritmos Aproximativos

Para resolver problemas deste tipo, isto é, para produzir um algoritmo de complexidade polinomial baixa, é necessário então relaxar o significado de resolver. Horowitz e Sahni [HOR 78] propõem dois tipos de relaxação para o significado de resolver. Na primeira o requerimento de que o algoritmo para o problema Π gere sempre uma solução ótima para o problema é substituído por gerar sempre uma solução viável com valor próximo ao valor da solução ótima. Esta solução viável com valor próximo ao valor da solução ótima é chamada uma solução aproximada. Assim, um algoritmo aproximativo para o problema Π gera soluções aproximadas para Π. Uma solução aproximada pode ser qualquer uma obtida em um tempo razoável de computação. Na segunda relaxação o requerimento é de que o algoritmo para o problema Π gere quase sempre uma solução ótima. Algoritmos com esta característica são chamados algoritmos probabilisticamente bons.

Outros autores, como Alimonti [ALI 94], definem (de uma forma mais livre) que um algoritmo A é um algoritmo aproximativo para um dado problema de otimização $\Pi = \langle X, S, \pi, f, meta \rangle$ se para qualquer instância $x \in X$ ele fornece uma solução viável $A(x) \in S(x)$ [ALI 94]. Esta definição não estabelece nenhuma caracterização adicional à solução viável fornecida pelo algoritmo.

Problemas Aproximáveis

Os algoritmos aproximativos podem ser agrupados em classes considerando a qualidade da solução aproximada [HOR 78] [CRE 97]. Os problemas de otimização para os quais é conhecido pelo menos um algoritmo aproximativo que forneça solução aproximada garantida com uma determinada qualidade é chamado de problema aproximável.

2.4.2 Heurísticas e Metaheurísticas

Quando o tempo de computação torna inviável o uso de algoritmo exato, ou mesmo, o uso de um algoritmo aproximativo de qualidade garantida, tem-se buscado outras alternativas. Usualmente, estas alternativas são as chamadas técnicas heurísticas (ou simplesmente heurísticas) com as quais se procura obter soluções de boa qualidade (isto é, próximas da ótima) a um custo computacional razoável sem garantia de otimalidade. Não sendo possível estabelecer nem mesmo o quanto uma solução específica obtida está próxima da ótima [RAY 96].

Entre as técnicas heurísticas mais utilizadas, na prática, para computar aproximações em tempo polinomial para problemas de otimização, estão as chamadas técnicas de busca local. Existem variações a partir de um conceito básico comum, sendo bastante conhecidas busca tabu e simulated annealing, entre outras. Estas técnicas mais aprimoradas utilizam procedimentos de busca local de uma forma mais inteligente, permitindo obter soluções de melhor qualidade e contornando a principal desvantagem das técnicas de busca local mais simples que é a chegada prematura a um ótimo local. Estas são chamadas de técnicas inteligentes de busca e de metaheurísticas.

Uma metaheurística é um processo mestre iterativo que guia e modifica as operações de heurísticas subordinadas com o objetivo de produzir eficientemente soluções de alta qualidade, utilizando procedimentos de busca que evitam a parada prematura em ótimos locais que sejam distantes do ótimo global. São metaheurísticas bastante conhecidas: busca tabu [MUL 93] [ROG 99], algoritmo genético [AGU 98], GRASP [SIL 99], simulated annealing.

2.4.3 Fundamentos da Busca Local

Busca Local é um procedimento iterativo que vai de uma solução para outra no espaço de soluções S, pesquisando sistematicamente em S, com movimentos que podem ser restritos de alguma forma. Estes movimentos são realizados considerando uma estrutura de vizinhança, ou seja, o subconjunto de soluções que podem ser encontrados em um passo a partir de uma dada solução i . O conjunto S_i é chamado de vizinhança de i.

Passos Básicos

Os passos básicos de um procedimento de busca local são:

- Iniciar com uma solução corrente $i \in S$;
- Percorrer a vizinhança S_i, seguindo uma estratégia determinada, até um critério de término;
- Retornar uma nova solução corrente $j \in S$, melhor que i com relação a determinado critério, eventualmente pode acontecer de $j = i$.

Este procedimento termina com alguma solução que é um ótimo local com relação a sua vizinhança e pode diferir de forma considerável do ótimo global.

O critério de término pode ser, entre outros:

- Uma busca exaustiva, quando toda a vizinhança é percorrida em busca da melhor solução;
- Uma busca pela melhor solução associada a outro critério como número de iterações total ou número de iterações sem melhoria;

A estratégia para percorrer a vizinhança é, em geral, um procedimento iterativo constituído por duas etapas:

- Um mecanismo de perturbação ou de geração de uma vizinha da solução corrente i;
- Um critério de avaliação desta solução candidata, a qual será avaliada com relação a i;

O mecanismo de perturbação pode ser determinístico ou aleatório ("Simulated Annealing"). O critério de avaliação pode incluir fatores probabilísticos ("Simulated Annealing") ou não ("hill-climbing", busca tabu).

Para ser possível utilizar métodos determinísticos para percorrer uma vizinhança é necessário que possa ser estabelecida uma estruturação adequada para a mesma.

Um movimento ou salto proposto pelo mecanismo de perturbação pode ser aceito ou rejeitado pelo critério de avaliação. Quando um movimento é aceito ocorre uma transição, ou seja, a passagem de solução para outra. Um caminho no espaço de soluções consiste numa seqüência de soluções, onde duas soluções consecutivas quaisquer são vizinhas [RIB 98].

Perturbação

Poucas definições formais para perturbação são encontradas na literatura, pois é um termo muitas vezes usado de forma imprecisa significando a modificação do valor de algum parâmetro ou função e a verificação do efeito da modificação na solução obtida [GRE 99]. Uma definição no âmbito da otimização combinatória, mais formal, é apresentada por Nurmela [NUR 93]:

Definição 2.7: Uma perturbação d é uma função de um subconjunto $S(d)$ de S (o domínio da perturbação) para S:

$$d : S(d) \rightarrow S.$$

O conjunto de todas as perturbações é D. A união dos domínios de todas as perturbações em D é o conjunto de soluções, isto é,

$$\bigcup_{d \in D} S(d) = S,$$

Assim, não há soluções em que uma perturbação não seja aplicável.

A aplicação de uma perturbação d a uma solução $s \in S$ é denotado por $d(s)$. Uma perturbação pode potencialmente degenerar se

$$s = d(s)$$

para algum $s \in s(d)$.

Algumas propriedades podem ser estabelecidas para a função perturbação, como localidade de ação, invertabilidade, condições de consistência, terminalidade e complexidade.

- Localidade de ação – estabelece se a solução gerada é restrita a uma vizinhança ou não.
- Invertabilidade – O inverso de uma perturbação, se existir, é denotada por d^{-1} e deve satisfazer

  $$s = d^{-1}(d(s))$$

  para qualquer $s \in S$.
- Consistência – a solução gerada deve obedecer as características estabelecidas para o espaço de soluções.
- Terminalidade – a aplicação sucessiva deve levar a uma solução aceitável como saída do algoritmo.
- Complexidade – deve ser em tempo polinomial.

Vantagens e Desvantagens da Busca Local

As principais vantagens dos procedimentos de busca local são a simplicidade e a generalidade do enfoque. No entanto, a qualidade do ótimo local fornecido é dependente do tempo de processamento, da estratégia de exploração do espaço de soluções e, muitas vezes, da solução inicial. A garantia da qualidade da solução é, em geral, dada através da análise dos resultados obtidos em testes exaustivos e comparativos com outras heurísticas e com algoritmos exatos, quando estes existirem.

2.4.4 Algoritmos Aleatórios

De uma maneira informal, diz-se que um algoritmo é aleatório quando inclui procedimentos que simulam o conhecido tirar "cara ou coroa". São algoritmos determinísticos que fazem escolhas aleatórias ao longo de sua execução. O estudo sistemático deste tipo de algoritmo iniciou na década de 70 com a proposição por Solovay e Strassen de um algoritmo aleatório para verificar se um número é primo e de outro com o mesmo objetivo por Rabin. Estudos posteriores de Rabin e outros criaram os fundamentos de uma teoria geral para algoritmos aleatórios [KAR 86].

Um algoritmo aleatório inclui um procedimento que pode ser visto como uma "caixa preta" que recebe dados como entrada e uma cadeia aleatória de bits que permitam ao algoritmo fazer escolhas aleatórias. Os algoritmos aleatórios podem ser do tipo Las Vegas e Monte Carlo. Um algoritmo Las Vegas termina fornecendo uma solução correta ou termina fornecendo uma resposta do tipo "não sei". Portanto, um algoritmo Las Vegas é um algoritmo confiável. Um algoritmo Monte Carlo termina fornecendo uma solução possivelmente correta ou termina fornecendo uma resposta do tipo "não sei". Um algoritmo Las Vegas é um Monte Carlo com probabilidade de erro zero. Um algoritmo Las Vegas é eficiente se para cada entrada seu pior tempo de execução é limitado por uma função polinomial do tamanho da entrada. Um algoritmo Monte Carlo é eficiente se para cada entrada seu pior tempo de execução é limitado por uma função polinomial do tamanho da entrada [AND 98] [BOV 94].

3 Simulated Annealing

Simulated Annealing (SA) é um método iterativo que combina técnicas de busca local e randomização que, quando aplicado a um problema de otimização, procura evitar a parada prematura em um ótimo local. O enfoque original do SA é baseado em conceitos de mecânica estatística, explorando similaridades entre o comportamento de sistemas de muitos graus de liberdade, isolado em equilíbrio térmico a uma dada temperatura, e o problema de encontrar o mínimo de uma função dependente de muitos parâmetros [MET 53] [KIR 83]. Kirkpatrick e outros sugeriram um método de resolução de problemas de otimização através da simulação do processo de recozimento (annealing) de sólidos.

A seção 3.1 deste capítulo tem como objetivo colocar o paradigma Simulated Annealing em um contexto mais abrangente. Assim, será feita uma introdução ao estudo do algoritmo de Metropolis, incluindo os métodos Monte Carlo e mecânica estatística, procurando estabelecer informalmente alguns conceitos básicos. A seção 3.2 descreve o Simulated Annealing tradicional baseado nas idéias de Kirkpatrick. A seção 3.3 desenvolve a idéia de Simulated Annealing genérico, visualizado através de componentes ou procedimentos independentes que se relacionam. Na seção 3.4 é desenvolvida uma especificação formal que captura a idéia central do paradigma, abstraindo-se de particularidades não essenciais. A seção 3.5 apresenta um método de desenvolvimento de algoritmo SA. A seção 3.6 apresenta um estudo de caso.

3.1 Algoritmo de Metropolis (M(RT)^2)

3.1.1 Métodos Monte Carlo

Esta é uma pequena introdução aos métodos Monte Carlo, baseado em Kalos e Withlock [KAL 86] e Jerrum e Sinclair [JER 97].

O nome Monte Carlo foi aplicado inicialmente pelos cientistas que trabalhavam no desenvolvimento da bomba atômica na década de 40 em Los Alamos, USA, aplicado a uma classe de métodos matemáticos. A essência destes métodos é a invenção de esquemas de transformação que permitam estudar o comportamento e os resultados obtidos em um dado fenômeno de interesse.

Considere os seguintes exemplos:

- Seja um círculo e seu quadrado circunscrito. A razão entre a área do círculo e a área do quadrado é $\pi/4$. É razoável supor que se forem colocados pontos aleatoriamente no quadrado, uma fração $\pi/4$ destes pontos caia dentro do círculo. É também razoável supor que o valor $\pi/4$ possa ser estimado pela razão entre os volumes de água obtidos pela coleta de chuva num recipiente constituído por uma forma de bolo redonda de diâmetro L colocada dentro de uma forma quadrada de lado L. Isso poderia ser simulado em um computador por um programa que gerasse pontos representados por pares aleatórios de coordenadas cartesianas em um quadrado e contar a fração que cai dentro do círculo. Se este experimento fosse repetido um grande número de vezes (1.000.000 de experimentos, por exemplo) poderia ser usado para estimar o valor de $\pi/4$. Este exemplo ilustra como uma amostragem aleatória pode ser usada para resolver um problema matemático, neste caso, a avaliação de uma integral definitiva,

$$I = \int_0^1 \int_0^{\sqrt{1-x^2}} dxdy.$$

  As respostas obtidas são de natureza estatísticas e sujeitas às leis de mudança, o que é uma desvantagem dos métodos Monte Carlo, mas, muitas vezes, é possível determinar a precisão da resposta e, se necessário, realizar mais experimentos para obter respostas mais precisas. Algumas vezes, apesar do caráter aleatório da resposta, esta resposta é a mais precisa que se pode obter considerando um determinado tempo de computação. A determinação do valor π, é claro, pode ser realizada mais rapidamente e com mais precisão por métodos não-Monte Carlo. Em outros casos, no entanto, métodos Monte Carlo são a única forma efetiva de avaliar integrais.

- Considere o jogo de cartas Paciência. Qual a possibilidade de vencer, assumindo que as cartas do baralho estejam perfeitamente embaralhadas antes de iniciar a tentativa de colocar as cartas em ordem? Uma vez escolhida a estratégia de empilhamento das cartas, o problema é elementar para a teoria da probabilidade. Não é difícil definir um algoritmo para simular o jogo, utilizando listas aleatórias que representam as 52 cartas e listas representando as diferentes pilhas de cartas. É, então, possível estimar a possibilidade de sucesso pela observação de um grande número de jogadas. Este jogo no computador é uma simulação fiel de um processo aleatório real representado pelo embaralhamento das cartas.

Procedimentos como os descritos acima quando simulados em computador fazem uso de números aleatórios. A utilização de seqüências aleatórias de números é freqüente em computadores, por exemplo, em jogos e na geração de dados sintéticos para testes. No entanto, alguns destes processos, como no caso o jogo de Paciência, não são considerados do tipo Monte Carlo, uma vez que não produzem resultados numéricos.

Diz-se que um método é um Monte Carlo quando faz uso deliberado de números aleatórios em um cálculo que segue a estrutura de um processo estocástico. Um processo estocástico

significa uma seqüência de estados cuja evolução é determinada por eventos aleatórios, os quais, em um computador, são gerados pela utilização de números aleatórios.

Uma distinção pode ser feita entre simulação e Monte Carlo. Simulação é uma transcrição direta para termos computacionais de um processo naturalmente aleatório. É o caso do jogo de Paciência. Monte Carlo é a busca de uma solução usando um método probabilístico de um problema não probabilístico. É o caso exemplificado do cálculo de π. No entanto, esta distinção nem sempre é possível de ser mantida. A emissão de radiação de átomos e sua interação com a matéria é um exemplo de um processo naturalmente aleatório uma vez que cada evento é em algum grau imprevisível. Entretanto, o comportamento médio desta radiação pode ser descrita por equações matemáticas cuja solução numérica pode ser obtida usando um método Monte Carlo. Assim, um mesmo algoritmo pode ser visto como um processo de simulação ou como uma solução utilizando amostragem aleatória.

A utilização de amostragem aleatória na resolução de certos tipos de integrais é antiga. No entanto, foi nas décadas de 1940 e 1950 que o trabalho conjunto de cientistas como Von-Neumann, Fermi, Ulam e Metropolis e o início da utilização dos computadores digitais modernos resultaram em um grande ímpeto para o desenvolvimento dos métodos Monte Carlo, aplicados a resolução de problemas em diversas áreas como mecânica estatística, transporte de radiação e modelagem de sistemas econômicos. Entre os inventos importantes, que resultaram em influência decisiva para o desenvolvimento dos métodos Monte Carlo, foi o algoritmo $M(RT)^2$, que é mais comumente conhecido como algoritmo de Metropolis [MET 53] [KAL 86].

Na última década estudos têm sido realizados no sentido de obter-se um melhor embasamento teórico relacionado com a qualidade dos algoritmos. Assim, uma classe de métodos chamados de Monte Carlo Markov Chain (MCMC), para os quais é possível uma análise matemática mais rigorosa, vêm sendo utilizados para uma grande variedade de problemas e, com freqüência, mostrando que constituem técnicas eficientes para solução em tempo polinomial deste problemas.

Um método Monte Carlo Markov Chain [JER 97] fornece um algoritmo para resolver uma tarefa que pode ser descrita genericamente como: Seja *S* um conjunto muito grande(porém finito) de estruturas combinatórias(como o conjunto de estados possíveis de um sistema físico, ou o conjunto de soluções para um problema de otimização combinatória) e seja *q* uma distribuição de probabilidade em *S*. A tarefa é simplesmente encontrar um elemento de *S* aleatoriamente de acordo com a distribuição *q*.

Problemas de amostragem combinatória deste tipo tem muitas aplicações computacionais. Entre as mais importantes estão as seguintes:

(a) Contagem aproximada: ou seja, estimar a cardinalidade de *S*. Uma generalização natural é a integração discreta, que consiste em estimar uma soma ponderada na forma $\sum_{x \in S} w(x)$, onde *w* é uma função positiva definida sobre *S*.

(b) Física estatística: onde *S* é um conjunto de estados possíveis de um sistema em mecânica estatística e seja *q* uma distribuição natural de probabilidade em *S* (tal como a distribuição de Gibbs), na qual a probalidade de um estado está relacionado com a sua energia. O objetivo é amostrar estados de acordo com *q*, para examinar uma

configuração típica e examinar a esperança de certas variáveis aleatórias naturais (como a energia média). Computações deste tipo são conhecidos como "experimentos Monte Carlo".

(c) Otimização combinatória: onde S é um conjunto de soluções de um problema de otimização e q uma distribuição que associa de alguma forma um maior peso a soluções com melhores valores de função objetivo. Amostragem usando q favorece as melhores soluções. Um exemplo desta enfoque é o Simulated Annealing.

Nas aplicações acima procedimentos estatísticos são utilizados para inferir a informação desejada a partir de uma seqüência de amostras aleatórias independentes. Em algoritmos deste tipo a amostragem representa o maior desafio. A resolução passa pela definição das características de uma cadeia de Markov que representa adequadamente o processo. A observação empírica do comportamento dos algoritmos e a intuição baseada em propriedades físicas ou combinatórias continua sendo uma forma de analisar a performance dos mesmos, e a construção de fundamentos de bases sólidas é ainda um desafio considerável para a Ciência da Computação Teórica [JER 97].

3.1.2 Algoritmo de Metropolis

Metropolis e outros [MET 53] propuseram o seu algoritmo como um método genérico para investigar as propriedades das equações de estados de substâncias que são consideradas como constituídas pela interação de um número elevado de partículas, baseado em estatística clássica e adotando as suposições usuais na teoria de líquidos.

Uma idéia simples da funcionalidade do algoritmo pode ser deduzida pelo exemplo dado por Kalos e Whitlock [KAL 86]. Seja um modelo altamente simplificado de partículas movendo-se em uma superfície qualquer. A representação do modelo é dada por um quadrado de lado L que contém um grande número de discos. Cada disco de diâmetro efetivo a é localizado pelas coordenadas de seu centro. Considerando que há M discos, então se tem $2M$ coordenadas cartesianas, e um estado do sistema assim constituído pode ser indicado pelo vetor

$$X=(x_1, y_1, x_2, y_2, \ldots\ldots, x_m, y_m).$$

A função de distribuição de probabilidades que descreve o sistema, $f(X)$, é uma constante com exceção quando $x_k < 0$ ou $x_k > L$ ou $(x_l - x_k)^2 + (y_l - y_k)^2 < a^2$ para quaisquer l e k. Se qualquer uma destas condições acontece $f(X) = 0$. Estas restrições previnem a movimentação dos discos para fora do quadrado e sobreposição de discos.

Dado o sistema como o representado pela Figura 3.1, qual a probabilidade de que dois discos estejam a uma distância *r* um do outro?

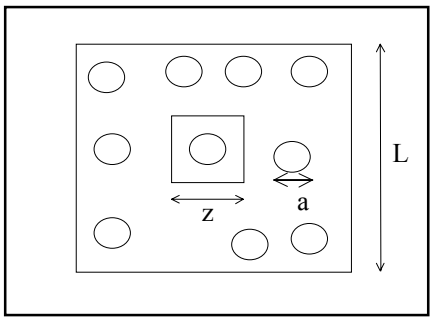

FIGURA 3.1 – Modelo Simplificado em Superfície Plana

A resposta pode ser encontrada por um algoritmo de Metropolis que gere muitas estados (configurações) *X* e meça a freqüência de ocorrência da distância *r*. Um estado inicial pode ser qualquer disposição dos discos sem sobreposição. Um novo estado é obtido pela movimentação de um disco. Este disco é escolhido aleatoriamente e a movimentação é delimitada por um quadrado pequeno de lado *z* conforme Fig. 3.1. A probabilidade de geração de um estado para outro é dado por

$$P_T(X \mid Y) = \sum f_l \, t\,(x_l, y_l \mid x_l^{'}, y_l^{''}),$$

onde $f_l = 1 / M$ e $t\,(x_l, y_l \mid x_l^{'}, y_l^{'})$ é densidade da transição para mover uniformemente no quadrado pequeno e corresponde a uma constante $1/z^2$, onde *z* é o lado do quadrado. A probabilidade de aceitação do novo estado é estabelecida como: (i) se a movimentação acarretar uma sobreposição ou movimentação para fora do quadrado grande, o novo estado é sempre rejeitado; (ii) ao contrário, o novo estado é aceito. O processo é repetido muitas vezes e uma função de freqüência é tabulada para *r*. Os movimentos não acarretam nenhuma alteração em propriedade físicas porque a cinética estabelecida para o exemplo é totalmente artificial não havendo nenhuma forma de interação entre os discos.

A estimativa obtida pelo modelo é determinada pelo tamanho do quadrado pequeno. Se este quadrado é muito grande, haverá maiores problemas de movimentação devido a maior probalidade de ocorrência de sobreposição, ou seja, uma maior dificuldade para encontrar um espaço livre. Uma grande quantidade de movimentos serão rejeitados e uma configuração será repetida muitas vezes. Isto resultará em um tempo de computação desperdiçado. No outro extremo, se este quadrado for muito pequeno, o novo estado quase sempre aceito e as configurações vão se modificando lenta e vagarosamente resultando também em desperdício de tempo. A regra prática é escolher um tamanho adequado para o quadrado pequeno. A escolha deste tamanho pode ser por tentativas baseado numa probabilidade de aceitação de novos estados, por exemplo, em 50%.

O algoritmo de Metropolis é baseado nos conceitos de mecânica estatística, que é a disciplina principal do estudo da física da matéria condensada e inclui os métodos que analisam as propriedades agregadas do grande número de átomos que podem ser encontrados em uma amostra de líquido ou sólido. Como o número de átomos é da ordem de 10^{23} por cm^3, o comportamento de um sistema em equilíbrio térmico a uma dada temperatura é observado através de experimentos. Este comportamento é caracterizado pela média e pequenas flutuações em torno da média, portanto, pode ser estudado usando o conceito de "*ensemble*" introduzido por Gibbs [KIR 83] [TOR 92]. Lembrando que pela hipótese de ergodicidade [AAR 89] um sistema físico de muitas partículas é compatível com um *ensemble* estatístico e que as correspondentes médias do *ensemble* determina as médias das quantidades macroscópicas observáveis do sistema físico. Um *ensemble* estatístico é a realização mental, ou virtual, de um grande número de cópias de um sistema em consideração, todas elas submetidas às condições pertinentes a uma determinada situação em estudo[TOR 92].

Neste "*ensemble*" cada configuração, definida pelo conjunto de posições atômicas, $\{s_i\}$, de um sistema é ponderado por um fator, $exp(-E(\{s_i\})/k_BT)$, onde $E(\{s_i\}$ denota a energia da configuração, T denota a temperatura e k_B uma constante conhecida como constante de Boltzmann.

Para que o algoritmo de Metropolis possa ser utilizado como uma simulação de uma coleção de partículas (em um banho térmico) em equilíbrio a uma dada temperatura, será considerado que, no exemplo anterior, cada disco representa uma partícula de uma substância e estas partículas interagem entre si. Nesta caso, cada movimento aleatório de uma partícula (uma pequena perturbação ou distorção) corresponde a uma modificação de energia, ΔE. A cada passo do algoritmo esta diferença de energia é calculada e,

(a) se for menor ou igual a zero, a nova configuração é aceita sempre,

(b) se for maior que zero, a nova configuração é aceita com certa probabilidade dada por $exp(\Delta E / k_BT)$,

(c) se a nova configuração proposta não for aceita permanece a atual.

A etapa (b) resulta na parte aleatória do algoritmo. Uma forma conveniente de implementar esta parte aleatória em computador é utilizando números aleatórios distribuídos uniformente no intervalo (0,1).

Neste modelo definido por Metropolis é verificado que:

- o método é ergódico [TOR 92], ou seja, cada partícula tem possibilidade de alterar o seu estado para qualquer um dos seus estados disponíveis e, como isso é verdade para todas as partículas do sistema, é possível representar todos os estados do sistema;

- a escolha deste fator de probabilidade tem como conseqüência, após a execução de inúmeros passos a uma dada temperatura, a evolução do sistema para uma distribuição de Boltzmann (Gibbs) caracterizando um equilíbrio térmico[TOR 92]. A distribuição de Boltzmann e a distribuição de Gibbs são similares, sendo a segunda uma generalização da primeira. A distribuição de Boltzmann dá a

probabilidade da substância estar no estado i com energia E_i a uma dada temperatura T:

$$\mathrm{P}_T\{\mathrm{X}=i\}=\frac{1}{\mathrm{Z}(T)}\exp\left(\frac{-\mathrm{E}_i}{k_B T}\right) \qquad (3.01)$$

onde X é uma variável estocástica denotando estado corrente e

$$\mathrm{Z}(T)=\sum_j \exp\left(\frac{-\mathrm{E}_j}{k_B T}\right) \qquad (3.02)$$

uma função de partição onde o somatório inclui todos os possíveis estados.

A função de partição definida pela equação(3.02) leva em consideração todos os estados de um sistema e, portanto, sua determinação requer o conhecimento de todos os seus estados e de suas respectivas energias. Em conseqüência, para sistemas de muitos corpos, esta determinação é, muitas vezes, impossível.

Uma outra questão fundamental em mecânica estatística está relacionada com o que acontece com o sistema no limite de temperatura inferior: se a substância permanece fluída ou solidifica, e ,caso solidificando, a solidificação resulta numa forma cristalina perfeita ou não. O estado destes estados fundamentais requer atenção porque a distribuição de Boltzmann sucumbe a baixas temperaturas. Outro aspecto é que, na prática, uma baixa temperatura não é uma condição suficiente para determinar um estado fundamental. Os experimentos mostram que velocidade de resfriamento pode determinar as características deste estado fundamental. Um estado cristalino sem defeitos é, usualmente, obtido por um aquecimento até a fusão, seguido de um resfriamento lento e cuidadoso, com gasto de tempo considerável quando se aproxima da temperatura de cristalização ou congelamento. Em caso contrário, se obtém a matéria em um estado meta-estável, que corresponde a uma estrutura cristalina com defeitos ou um vidro.

3.2 Descrição do Simulated Annealing Clássico

Encontrar um estado fundamental para um sistema quando uma prescrição para o cálculo de energia é conhecida, usando os conceitos de mecânica estatística, é um problema de otimização com aspectos comuns aos de otimização combinatória. Esta idéia foi explorada por Kirkpatrick, Gelatt e Vecchi [KIR83] para o problema do caixeiro-viajante e o projeto físico de circuitos. Entretanto, como o conceito de temperatura de um sistema físico não tem um equivalente óbvio em otimização é introduzido um parâmetro que faz o papel de temperatura. Os procedimentos de melhoramento iterativo de busca local usados na resolução de problemas de otimização são similares aos processos de mecânica estatística. As pequenas modificações aplicadas a uma solução para obter uma outra solução lembram os rearranjos microscópicos de partículas de uma substância e a avaliação da função objetivo pode ser descrita como uma avaliação de energia.

O algoritmo de Metropolis pode ser generalizado para estes procedimentos de melhoria iterativa, permitindo aceitação controlada de solução que aumenta o valor da função objetivo para tentar escapar de uma parada prematura em um mínimo local, assim como, na simulação de um processo físico de recozimento se procura fugir de uma cristalização defeituosa. A redução da temperatura lenta e controlada é também similar nos dois processos para que uma distribuição estacionária seja alcançada. A tabela 3.1 resume estas analogias:

TABELA. 3.1 – Analogias Recozimento e Problema Combinatório

Annealing de um Sólido (recozimento)	**Problema de Otimização Combinatória**
• Estado físico	• Solução
• Energia do estado	• Custo da solução (valor da função objetivo)
• Temperatura	• Parâmetro de controle faz o papel de temperatura
• Prescrição de resfriamento (cooling schedule)	• Regras que regem a inicialização e decréscimo do parâmetro de controle
• Estado fundamental do sólido (congelamento)	• Ótimo global
• Resfriamento rápido (quenching)	• Procedimento de busca local guloso
• Resfriamento lento e controlado	• Simulated Annnealing

Assim, especificados uma instância de um problema de otimização combinatória, uma estrutura de vizinhança e uma solução inicial aleatória o seguinte processo iterativo é realizado:

- uma nova solução j é escolhida aleatoriamente na vizinhança S_i, a partir da solução corrente i utilizando um mecanismo de geração (perturbação),
- a nova solução j é aceita ou não pelo uso de um critério de aceitação, aplicando probabilidades através de uma fórmula como

$$\min\{1, exp(-(f(j)-f(i))/t_k)\},$$

onde $f(j)-f(i)) = \Delta E$, representa a modificação no nível de energia, t_k é uma seqüência de valores positivos (parâmetros de controle) com $\lim_{k\to\infty} t = 0$, onde t representa k_BT da equação de Boltzmann. O parâmetro t fica conhecido como temperatura e k é um tempo virtual que representa a evolução do algoritmo. O parâmetro t deve ter inicialmente um valor elevado e sofre decréscimo no início de cada nova iteração, isto é, a cada tempo k de acordo com uma determinada regra de decréscimo. O critério de aceitação é, assim, função da temperatura t corrente e da diferença de custo(energia) ΔE entre as soluções.

A probabilidade de aceitação é definida pela função matemática ($exp(-(f(j)-f(i))/t_k$). No início do processo, a temperaturas altas, esta função gera valores próximos de 1 e valores menores em sucessivas iterações, resultado do decréscimos de temperatura (resfriamento). Assim, no início, a maioria das soluções propostas são aceitas e, no decorrer do processo, com maior probabilidade, somente as soluções que produzem melhores resultados são aceitas[RAB 95].

Os passos básicos de um algoritmo Simulated Annealing (para minimização) são:

1. $t \leftarrow t_0$;
2. ciclo_externo: enquanto **não critério_congelamento** faça
3. $l_k \leftarrow 0$;
4. ciclo_interno: enquanto **não critério_equilíbrio** faça
5. gera s_j vizinha de s_i;
6. $\Delta E \leftarrow E(s_j) - E(s_i)$;
7. se $\Delta E \geq 0$ então $s_i \leftarrow s_j$
8. senão $s_i \leftarrow s_j$ com probabilidade $exp(-\Delta E/t)$;
9. $l_k \leftarrow l_k + 1$;
10. fim_enquanto;
11. $t \leftarrow \alpha t$;
12. fim_enquanto;

onde t_0 é uma temperatura inicial , $0 > \alpha < 1$ e α próximo 0,9.

3.3 Componentes do Simulated Annealing Genérico

O Simulated Annealing pode ser visto mais como um enfoque de resolução do que como um único algoritmo devido a inúmeras variações encontradas. Esta seção trata o Simulated Annealing como um algoritmo genérico, constituído de vários procedimentos componentes. Não há preocupação em detalhar cada procedimento, mas analisar aspectos que são importantes para o entendimento de cada procedimento isolado e, também, da forma como eles interagem entre si. Assim, inicialmente, o algoritmo SA será visto de uma forma geral, como um processo iterativo que é repetido até que determinadas condições sejam satisfeitas. A terminologia utilizada segue a analogia com a termodinâmica aplicada à resolução de problemas de otimização.

Na prática, os algoritmos que são variações em torno do paradigma SA podem ser classificados em três tipos básicos levando em conta a forma como é realizado o resfriamento, ou seja, a forma como é tratado o parâmetro temperatura:

- Simulated Annealing (SA) – resfriamento lento e controlado, com probabilidade maior de convergência. Teoricamente, é o modelo ideal.
- Simulated Quenching (SQ) – resfriamento mais rápido com o objetivo de obter um tempo computacional de execução menor.
- Simulated Tempering (ST) ou reannealing – o resfriamento não é monótono, ocorrendo aquecimentos intermediários.

Cabe ressaltar que estas denominações nem sempre estão bem caracterizadas nas implementações documentadas na literatura e a denominação Simulated Annealing tem sido usada genericamente, e assim, será usada neste trabalho.

Inicialmente serão estabelecidas aspectos derivados de conceitos da mecânica estatística, a seguir, analisadas as entradas e saídas para este algoritmo e, posteriormente, discutidos os aspectos relativos aos procedimentos que constituem o algoritmo propriamente dito e suas variações.

3.3.1 Algoritmo Genérico

O algoritmo SA, representado na Figura 3.2 sem detalhamentos, é um procedimento seqüencial e genérico, que recebe como entrada dados relativos a aspectos dependentes do problema, aspectos genéricos e aspectos de ajuste. Como saída fornece uma solução possivelmente de boa qualidade. O algoritmo SA será tratado como um núcleo genérico e teórico que é um padrão a ser implementado para qualquer tipo de problema, procurando isolar o que é dependente do problema e da instância (aspectos dependentes do problema) e o que é dependente de uma implementação específica (aspectos genéricos e de ajuste).

Dados aqui pode ser um dado elementar, uma estrutura de dados ou, ainda, uma função computacional.

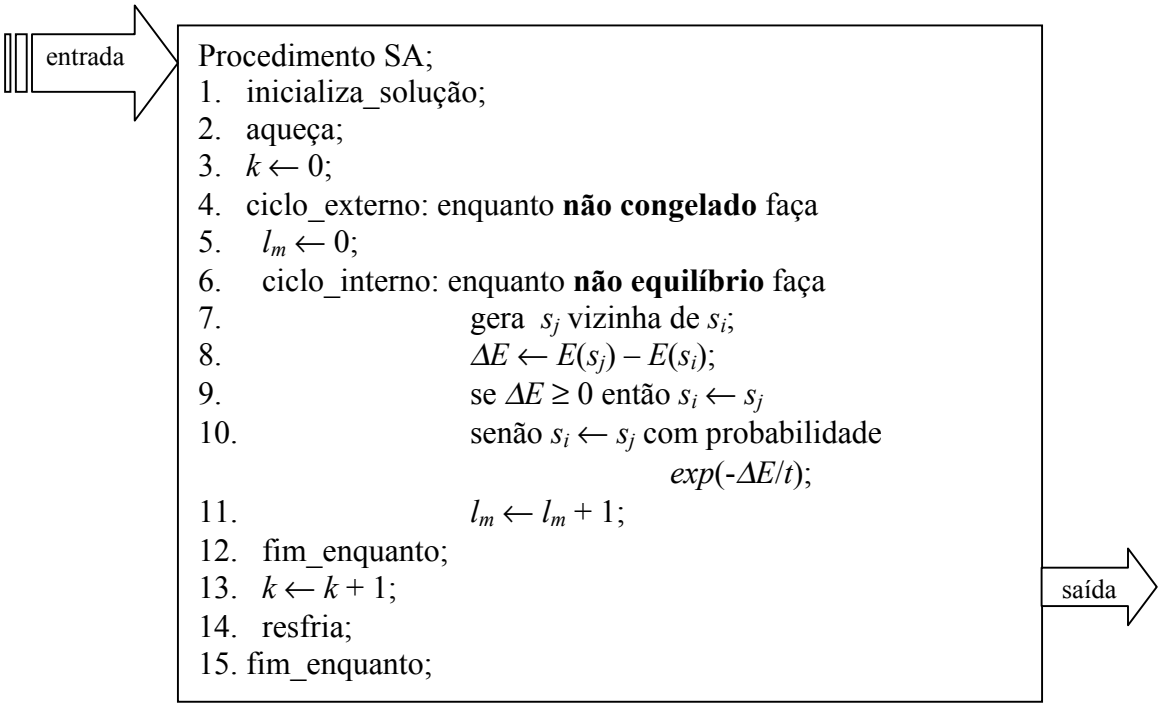

FIGURA 3.2 – Algoritmo SA Genérico

Este procedimento SA pode ser visto como uma série de k Cadeias de Markov, onde o comprimento de cada cadeia l_m corresponde ao número de iterações do ciclo_interno até que a condição de equilíbrio seja satisfeita. Na capítulo 4 Análise de Convergência será feito uma introdução à teoria de Cadeias de Markov e analisado os aspectos de convergência relativo ao algoritmo Simulated Annealing modelado segundo esta teoria.

Justificativa do Enfoque

Este enfoque tem como finalidade propiciar um entendimento do algoritmo e ser uma base teórica para implementações e está levando em consideração as diversas formas como o algoritmo têm sido implementado, como, por exemplo:

- Frameworks: Andreatta e outros [AND 99] apresentam uma arquitetura base para o desenvolvimento das metaheurísticas baseadas em busca local usando os conceitos de orientação a objetos, aplicável à resolução de problemas de otimização combinatória formulados como programas lineares inteiros 0-1 e apontam com uma tendência atual o desenvolvimento deste tipo de framework. O entendimento adequado dos componentes e do relacionamento entre os mesmos é o primeiro e essencial passo para este tipo de desenvolvimento.

- ASA (Adaptative Simulated Annealing) [ING 99]: constituído de um núcleo central genérico que procura determinar uma prescrição de resfriamento adequada para um problema em particular e uma interface para módulos de software desenvolvidos pelo usuário para o problema em particular a ser resolvido como funções para cálculo da função objetivo e mecanismo de transição entre soluções vizinhas.
- GPSIMAN e INTSA [ABR 97]: implementações do algoritmo Simulated Annealing de propósito geral, que recebem como entrada uma especificação algébrica do problema em particular. Esta especificação algébrica é baseada numa formulação linear inteira-0,1 (GPSIMAN) e linear inteira (INTSA) para o problema.

O estabelecimento de um modelo para o algoritmo SA onde os procedimentos essenciais são determinados de acordo com suas funcionalidades foi baseado Johnson [JOH 89], que caracteriza os aspectos dependentes do problema, os genéricos e os de ajuste. São aspectos dependentes do problema:

- A definição do que é uma solução e, em conseqüência, de que consiste o espaço de soluções,
- Estabelecimento da forma como avaliar o custo da solução, ou seja, qual será a função objetivo considerada.
- Identificação de uma estrutura de vizinhança adequada, ou seja, qual o significado de uma solução ser vizinha da outra,
- Determinação de uma solução inicial.

Estes aspectos serão analisados como representação da instância do problema.

Os aspectos genéricos [JOH 89], consistindo o que é denominado prescrição de resfriamento (cooling schedule) são relacionados com a simulação do parâmetro de controle temperatura:

- Determinação de uma temperatura inicial.
- Estabelecimento das regras que vão determinar a velocidade de resfriamento.
- Definição do que é equilíbrio térmico.
- Determinação do significado de congelamento.

Os aspectos de ajuste são aspectos relacionados com a implementação, como o gerador de números aleatórios utilizado e limites de tempo computacional gasto em determinadas fases do algoritmo.

Para este algoritmo genérico serão consideradas as seguintes definições baseadas em Aarts e Korst [AAR 89] e considerando problemas de minimização:

Seja uma instância de um problema de otimização (S, f) e $i \in S$, $j \in S$ duas soluções com custo $f(i)$ e $f(j)$, respectivamente. Sendo j construída a partir i por um mecanismo de geração ou perturbação.

Uma transição ou movimento é uma ação combinada que resulta na transformação de uma solução corrente *i* na solução subsequente *j* e consiste na aplicação da perturbação ou mecanismo de geração seguido da aplicação do critério de aceitação. Pode ainda ser chamado salto.

Um critério de aceitação é determinado pela equação que estabelece que uma solução *j* é aceita a partir de *i* pela aplicação da seguinte probabilidade de aceitação:

$$P_t\{aceitar\ j\} = \begin{cases} 1 & se\ f(j) \le f(i) \\ \exp((f(i) - f(j))/t) & se\ f(j) > f(i) \end{cases}, \qquad (3.03)$$

onde $t \subset \Re^+$ é um parâmetro de controle que representa a temperatura.

A taxa de aceitação χ_t a uma determinada temperatura *t* é definida como:

$$\chi_t = (m_a / m_p)_t, \qquad (3.04)$$

onde m_a é o número de movimentos aceitos e m_p é o número de transições propostas naquela temperatura.

As equações (3.03) e (3.04) são as mais simples e usuais para critério de aceitação e taxa de aceitação respectivamente. No entanto, outras são utilizadas dependendo do tipo de prescrição de resfriamento usado (seção 3.4).

3.3.2 Representação da Instância do Problema

Nesta seção serão analisados os aspectos que caracterizam a instância do problema. Uma estrutura que represente a instância é uma entrada para o algoritmo genérico. Para especificar esta estrutura de entrada é necessário projetar:

(a) a representação da instância;

(b) a representação da solução;

(c) a constituição do espaço de soluções;

(d) a função energia;

(e) uma topologia para o espaço de solução, pela especificação de uma ou mais estruturas de vizinhança, o que resulta na especificação de uma mais funções de perturbação.

Esses aspectos estão relacionados entre si e o projeto deve ser analisado como um todo.

Considere o problema do caixeiro-viajante (PCV):

(a) a instância pode ser representada por uma matriz *n* x *n* [d_{pq}], onde os elementos denotam a distância entre as cidades p e q – por exemplo, sejam 6 cidades, as distâncias entre as mesmas poderiam se representadas pela matriz

0	5	4	6	3	9
5	0	7	9	4	5
4	7	0	5	2	7
6	9	5	0	7	3
3	4	2	7	0	3
9	5	7	3	3	0

FIGURA 3.3 – Representação de uma Instância

(b) a solução é uma rota, um caminho fechado passando exatamente uma vez em cada cidade representada por uma lista das *n* cidades – são rotas (1,2,3,4,5,6), (1,6,5,4,3,2), ...;

(c) o espaço de soluções é constituído pelas permutações destas *n* cidades, onde , fixando a cidade 1, se obtém (n-1)! soluções;

(d) a função energia correspondente a cada soma solução é o comprimento total da rota;

(e) vários tipos de perturbações poderiam ser definidas correspondendo a diferentes estruturas de vizinhanças, como:

- deslocamento – uma cidade é escolhida aleatoriamente e deslocada para uma posição aleatória, cidade 3 escolhida e posição 5 :(1,2,3,4,5,6) → (1,2,4,5,3,6);
- intercâmbio – duas cidades escolhidas aleatoriamente trocam de posição, cidades 2 e 4 escolhidas : (1,2,3,4,5,6) → (1,4,3,2,5,6);
- inversão – duas cidades são escolhidas aleatoriamente e o segmento da rota entre elas é invertido, cidades 3 e 4 escolhidas: (1,2,3,4,5,6) → (1,2,4,3,5,6);
- 2-mudança – dois segmentos da rota são eliminados e substituídos por outros dois, (1,2,3,4,5,6) → (1,2,5,4,3,6);
- k-mudança – é uma generalização da 2-mudança;

Configuração do Espaço de Soluções

No SA tradicional, cada ciclo interno do algoritmo é caracterizado por uma solução corrente que pertence ao conjunto de soluções a serem exploradas pelo mesmo. Este conjunto de soluções é o espaço de soluções. Como cada ciclo interno pode ser visto como um estado a ser explorado então cada estado corresponde a uma solução corrente. Desta forma, o conjunto de estados explorados corresponde a um espaço de estados.

A definição de um espaço adequado de soluções a ser explorado influencia na velocidade de convergência e na qualidade da solução obtida. Os dois principais aspectos são;

- a definição do tipo de solução, e
- a topologia imposta pelo tipo de estrutura de vizinhança.

Dependendo do tipo de problema, pode-se optar por um espaço de soluções constituído somente de soluções viáveis ou por um espaço de soluções constituído por soluções viáveis e não viáveis. Em alguns problemas é mais simples construir uma vizinhança quando são incluídas as soluções não viáveis, no entanto, nestes casos, a solução obtida no final do algoritmo pode ser não viável, tornando necessária a utilização de uma heurística para converter esta solução não viável numa solução viável de qualidade equivalente.

Por exemplo, no PPG (particionamento de grafos), uma solução viável é uma partição V = $V_1 \cup V_2$ onde V_1 e V_2 são do mesmo tamanho. Neste caso, o espaço de estados corresponde ao conjunto de soluções viáveis, ou seja, inclui aquelas soluções que atendem à restrição mesmo tamanho. No entanto, este problema é, mais comumente resolvido usando uma nova definição de estado, onde V_1 e V_2 não são necessariamente de mesmo tamanho [JOH 89].

Seleção da Estrutura de Vizinhança e Perturbação

Nos algoritmos que seguem o paradigma SA, a vizinhança é estabelecida implicitamente através de um procedimento perturbação. A cada iteração do ciclo interno do algoritmo é gerada aleatoriamente uma solução vizinha pela perturbação da solução corrente. Uma perturbação respeita determinadas regras e limites de espaço que são dependentes do problema e do tipo de algoritmo.

Como estudo de caso, será analisado o problema PPG, exemplificando duas escolhas de vizinhança e correspondente procedimento de perturbação:

- o espaço de estados constituído de soluções viáveis e não viáveis e sendo $s_i = (V_1 \cup V_2)$, uma partição de V, onde V_1 e V_2 não são necessariamente do mesmo tamanho, e $v \in V_1$ um vértice qualquer, escolhido aleatoriamente; uma perturbação pode ser definida como a movimentação deste vértice v de V_1 para V_2. Esta perturbação aplicada a partição $s_i = (V_1 \cup V_2)$ gera uma nova $s_j = ((V_1 - \{v\}) \cup (V_2 \cup \{v\}))$, onde s_i e s_j são vizinhas.

- o espaço de soluções constituído apenas por soluções viáveis, um outro tipo de perturbação poderia ser definida como uma troca de vértices entre os dois conjuntos, ou seja, sendo $s_i = (V_1 \cup V_2)$ viável, $v \in V_1$ um vértice escolhido aleatoriamente e $u \in V_2$, também escolhido aleatoriamente; uma perturbação aplicada a s_i gera uma nova solução viável $s_j = (((V_1 - \{v\}) \cup \{u\}) \cup ((V_2 - \{u\}) \cup \{v\}))$.

Uma propriedade geral da perturbação em um algoritmo SA é a aleatoriedade, ou seja, para qualquer estrutura de vizinhança escolhida, a vizinha que constituirá uma nova solução proposta deve ser gerada por mecanismos que inclui aleatoriedade. Além disso, a proposta de movimento é "cega", não sendo baseada em nenhum critério de avaliação com relação aos resultados anteriores.

As características de volta a uma solução prévia e terminalidade serão analisadas no anexo A2 sobre convergência do algoritmo.

Vizinhança Dinâmica

A utilização de mais de um tipo de perturbação em um mesmo algoritmo, ou seja, mais de uma estrutura de vizinhança, é uma alternativa documentada por alguns autores [PRE 88]. Isto constitui uma generalização do algoritmo de Metropolis, com a inclusão de um cardápio de possibilidades de perturbações. A escolha do tipo de perturbação a ser adotada em uma determinada iteração pode ser estabelecida de acordo com alguma probabilidade definida. As condições de convergência do algoritmo (capítulo 4) podem ser alteradas pela utilização deste tipo de estratégia.

Seleção de uma Função Energia

Seja S o espaço de soluções de uma instância de um problema de otimização. A função $E: S \rightarrow \Re$, é uma função arbitrária que recebe o nome de função energia. Corresponde a função objetivo definida de forma geral para os problemas de otimização. A energia pode ser vista como uma propriedade que pode ser avaliada para cada solução de um problema de otimização.

A função energia é a forma de avaliar a qualidade de uma solução, ou seja, deve ser capaz de representar uma ordenação no espaço de estados permitindo estabelecer quando uma solução é melhor do que outra. Deve ser definida levando em consideração o projeto do espaço de estados, que pode excluir soluções não viáveis ou incluir soluções não viáveis aplicando penalidades para a inviabilidade. No caso, aplicar penalidade têm como finalidade tornar as soluções não viáveis menos atrativas.

A função energia pode ser analisada como constituída por várias parcelas de energia [OLI 97].

A energia da solução proposta é computada a cada iteração do algoritmo para comparação com o valor da solução corrente, como já foi visto. Assim, sempre que for adequado e possível pode ser computada diretamente a diferença de energia. Em alguns casos, esta computação direta da diferença pode levar a uma deterioração do valor da energia da solução, requerendo a computação direta a cada *n* iterações (*n* arbitrário).

A função energia será denotada nas seções seguintes por *f*.

Dependência entre espaço de soluções, vizinhança e função energia

Será apresentado a seguir um estudo comparativo de três enfoques para o SA para o problema PCG (coloração de grafos), realizado por Jonhson e outros [JOH 91], sem incluir a análise dos resultados obtidos, com a finalidade de exemplificar alternativas diferenciadas para os projetos do espaço de estados, vizinhança e função energia:

- Enfoque penalidade: envolve soluções não viáveis e função penalidade, no entanto, a estrutura de vizinhança é simplificada. Uma solução é qualquer partição de V em conjuntos disjuntos não vazios $C_1, C_2, ..., C_k$, com $1 \le k \le |V|$, onde C_i pode ser ou não uma classe de coloração legal. Duas soluções são vizinhas quando podem ser transformadas uma na outra pela movimentação de uma vértice de uma classe de coloração para outra. Para gerar uma vizinha aleatória, são escolhidos aleatoriamente uma classe não vazia C_{old}, um vértice $v \in C_{old}$ e um inteiro i, onde $1 \le i \le k+1$ e k o número atual de classes de coloração. A vizinha é obtida pela movimentação de v para a classe C_i. Se $i = k+1$ significa que C_i é uma nova classe anteriormente vazia. Se v já pertencia a C_i é feita nova tentativa. A função objetivo é constituída de dois componentes, um favorece classes grandes e o outro conjuntos independentes. Seja $s = (C_1, C_2, ..., C_k)$ uma solução, e A_i, $1 \le i \le k$ o conjunto de arestas de A com ambos os pontos finais em C_i, isto é, o conjunto de arestas "ruins" em C_i, a função objetivo é definida como

$$f(s) = -\sum_{i=1}^{k} |C_i|^2 + \sum_{i=1}^{k} 2|C_i| \times |A_i|.$$

- Enfoque cadeias de Kempe: envolve soluções viáveis e a função objetivo é definida como

$$f(s) = -\sum |C_i|^2.$$

  A dificuldade deste enfoque é definir a perturbação que gera a solução vizinha mantendo a viabilidade. O mecanismo escolhido em [JOH 91] é baseado em conceito chamado cadeias de Kempe e é mais dispendioso do que o mecanismo utilizado no enfoque utilizando função penalidade. Suponha que C e D são conjuntos disjuntos independentes em um grafo G. Uma cadeia Kempe para C e D é qualquer componente

conectado em um subgrafo de G induzido por $C \cup D$. Faça $X \Delta Y$ denotar a diferença simétrica $(X - Y) \cup (Y - X)$ entre dois conjuntos X e Y. A observação chave é que se H é uma cadeia de Kempe para os conjuntos disjuntos independentes C e D, então $C \Delta H$ e $D \Delta H$ são também conjuntos disjuntos independentes cuja a união é $C \cup D$. A seguinte perturbação pode ser usada: escolher aleatoriamente uma classe de coloração C não vazia e um vértice $v \in C$. Então escolher aleatoriamente uma classe de coloração D diferente de C, e faça H a cadeia de Kempe para C e D que contém v. Repetir esse procedimento até obter C, v, D e H tal que $H \neq C \cup D$ (isto é, tal que H não seja "cheio"), e neste caso a próxima solução é obtida substituindo C por $C \Delta$ H e D por $D \Delta H$.

- Enfoque k-fixo: neste enfoque o espaço de estados é representado por soluções não viáveis e a primeira solução viável encontrada, de custo zero, caracteriza a condição de término. Dado um grafo $G = (V, A)$ e um número arbitrário de cores k, cada solução corresponde a uma partição de V em k conjuntos (conjuntos vazios são aceitos), e o custo de uma solução é simplesmente o número total de arestas "ruins". Uma solução s_i é vizinha de outra s_2 se as duas diferem somente de um vértice v que é ponto final de uma aresta "ruim". Para escolher aleatoriamente uma vizinha da solução s_i, é primeiro escolhido um vértice "ruim" (ou seja, ponto final de uma aresta "ruim") v aleatório e então escolhida aleatoriamente uma nova classe de coloração para o vértice v entre as k-1 classes que não contém v. A relação de vizinha neste caso não é simétrica, e , em particular, uma solução viável não tem vizinhas porque não tem arestas "ruins". A dificuldade está em estabelecer um valor adequado para k.

Pode-se definir estado para um algoritmo SA como:

Seja S o espaço de soluções de uma instância de um problema de otimização e A um algoritmo SA que resolve este problema de otimização. Um estado explorado ou visitado por este algoritmo A é uma tupla $(s, \pi, E(s), t, l_m)$, onde $s \in S$ é uma solução, π é um predicado que identifica se a solução é viável, $E(s)$ a energia do estado avaliada pela função objetivo correspondente a solução s, t uma temperatura e l_m o número da iteração nesta temperatura t.

Panorama de energia

O espaço de soluções pode ser melhor entendido se visualizado como uma área geográfica constituída de montanhas e vales, delimitada de alguma forma em função de dois eixos que determinam as direções norte-sul e leste-oeste. A altura das montanhas e vales é determinada em relação ao nível do mar. O objetivo é encontrar o vale mais baixo. A altura em relação ao nível do mar é, portanto, uma medida da qualidade de um determinado ponto na área geográfica, é uma propriedade que pode ser avaliada. Esta idéia pode ser generalizada e, portanto, um espaço de soluções pode ser visto como um panorama de energia.

Tecnicamente, um panorama de energia é definido como grafo (grande, porém finito) onde cada vértice representa uma solução e onde as arestas representam as conecções entre soluções vizinhas. Os valores da função energia podem ser visualizados como atribuindo "altitude" ao panorama [GEH 99].

Nos últimos anos, alguns pesquisadores têm analisado o comportamento destes panoramas de energia, considerando aspectos estruturais e estatísticos para problemas nas áreas de física e química e para os problemas de otimização combinatória. O uso de técnicas de amostragem por caminhos aleatórios não inclinados, com base no algoritmo de Metropolis, mostraram-se adequadas para estas pesquisas. Stadler e Schnabel [STA 92] apresentaram os resultados do estudo de panoramas de energia para instâncias aleatórias do problema do caixeiro-viajante. Um destes resultados comprova o fato que já era conhecido pela utilização prática do SA para instâncias do problema do caixeiro-viajante simétrico: o SA é mais eficiente usando o procedimento 2-mudança do que transposição. Isso gera uma expectativa com relação ao estudo de alternativas de procedimentos de perturbação para outros problemas e um melhor entendimento teórico do comportamento dos algoritmos de resolução com diferentes estruturas de vizinhanças.

Escolha da Solução Inicial

O resultado de um algoritmo SA deve ser independente da solução inicial. Portanto, a solução inicial é escolhida arbitrariamente, não havendo neste sentido nenhum preprocessamento definido. No entanto, deve ser observado a coerência com o tipo de soluções a serem incluídas no espaço de estados.

Alguns pesquisadores sugerem a utilização de uma solução inicial construída por alguma outra boa e, razoavelmente bem sucedida heurística, como uma forma de economizar tempo de execução na fase de resfriamento. Estas variantes do SA usualmente recebem a denominação de Simulated Annealing em Duas Fases ("Two-stage Simulated Annealing") [ROS 93] [VAR 93].

3.3.3 Annealing (Recozimento)

Nesta seção são analisados os aspectos genéricos que estão relacionados com o processo de recozimento. Considerando a analogia com a termodinâmica o recozimento é constituído de um procedimento de aquecimento e um procedimento de resfriamento, ambos relacionados com o parâmetro temperatura.

O procedimento aquecimento tem como objetivo a escolha de um valor suficientemente alto para a temperatura. Este valor será utilizado como valor inicial no procedimento de resfriamento.

O resfriamento é controlado por uma prescrição de resfriamento (cooling schedule). Esta prescrição de resfriamento pode ser definida de uma forma geral como:

Prescrição de resfriamento *Pr* é uma seqüência arbitrária de números $t_n > 0$ tal que

$$t_n \geq t_{n+1} \; para\, todo\, n \geq 0,$$
$$\lim_{n\to\infty} t_n = 0,$$

onde t_n denota o parâmetro chamado temperatura e n é um tempo virtual correspondendo ao número da iteração do algoritmo [AZE 92].

Numa prescrição de resfriamento o valor da "temperatura inicial" t_0 deve ser "alto". O valor elevado tem como significado físico um valor superior ao da temperatura de fusão de um sólido em banho térmico, o que determina um estado onde as partículas estão arranjadas de forma totalmente aleatória. O valor da "temperatura final" t_n, na teoria deveria ser igual a zero, ou melhor, uma temperatura que corresponda a um estado de energia mínima, ou seja , a um estado fundamental de um sólido.

A prescrição de resfriamento deve ser o mais independente possível da instância do problema, pois o que se busca é a robustez do algoritmo. Esta independência do algoritmo com relação à instância pode ser conseguida pela análise das características estruturais e estatísticas do panorama de energia, através do uso do algoritmo de Metropolis como uma técnica de amostragem. Alguns estudos permitem considerar esta possibilidade [STA 92] [MCA 97].

Temperatura Inicial

A analogia do espaço de soluções com uma área geográfica constituída de montanhas e vales, pode ser usada para o entendimento do parâmetro temperatura. Visualize uma bola quicando entre estas montanhas e vales. De alguma forma esta bola adquiriu alta temperatura, que vai decrescendo a cada salto. No início a bola está numa temperatura suficiente para passar por cima das mais altas montanhas e, portanto, realizar qualquer movimento na área geográfica. Fatores aleatórios relacionados com a temperatura influenciam na permanência em determinado vale ou não. Posteriormente, a medida que a bola vai esfriando vai perdendo a capacidade de saltos maiores e menor número de movimentos são aceitos, até que, após um determinado momento, quase nenhum movimento é aceito e a bola permanece estacionária. Se a sorte(fatores aleatórios) ajudar no vale mais baixo, ou, pelo menos, em um vale suficientemente baixo para as necessidades estabelecidas [ING 99].

Em um limite superior de temperatura, qualquer tentativa de movimento seria aceita, independente do aumento ou decréscimo de energia. Assim, a temperaturas muito elevadas, o sistema se movimenta aleatoriamente entre seus estados, e a energia média observada aproxima-se da energia média de todos os possíveis estados do sistema [WHI 84].

Na resolução de um problema de otimização combinatória o processo de aquecimento físico corresponde a um procedimento de "aquecimento"("warm up") que deve ser realizado com primeira etapa do algoritmo. Este procedimento de "aquecimento" que tem como objetivo a busca de um valor adequado para temperatura inicial tem sido motivo de estudos variados e pode ser estabelecida de diversas formas, como:

- temperatura inicial igual a uma constante, usado nas implementações mais simples para resolução de um problema específico e/ou quando o conhecimento da instância do problema permitir esta escolha, e , por tentativas, em execuções sucessivas, é feito o ajuste do parâmetro [KIR 83];
- temperatura inicial estabelecida por um procedimento que utiliza uma taxa de aceitação inicial como critério para definir uma temperatura adequada para início do processo de annealing [AAR 89];
- temperatura inicial estabelecida por um procedimento que utiliza o conceito de entropia como critério para definir uma temperatura adequada para início do processo de annealing [ROD 93];
- temperatura inicial baseada no comportamento da curva energia média vs. temperatura [WHI 84].

A escolha de "um valor elevado" para a temperatura para um problema de otimização deveria ser o mais independente possível do problema e de sua instância [WHI 84].

A qualidade do resultado obtido com o algoritmo SA baseado no modelo de cadeias de Markov, ou seja, a convergência assintótica a um ótimo global, foi teoricamente demonstrada por Aarts[AAR 89]. Estudos mais recentes tratam os aspectos de convergência para os algoritmos MCMC (Markov chain Monte Carlo) de uma forma geral. Simulated Annealing é um subtipo deste tipo de algoritmo. No anexo A2 serão analisados, com maiores detalhes, estes aspectos.

Nesta seção a prescrição de resfriamento será analisada como um parâmetro de entrada para um algoritmo seqüencial genérico. Assim, será vista como uma prescrição de resfriamento de tempo finito e baseada nos casos práticos apresentados na literatura.

Decréscimo de temperatura

O comportamento genérico da parâmetro temperatura pode ser classificado como:

- Monotônico - a temperatura é reduzida de acordo com uma determinada velocidade de resfriamento até que termina quando uma determinada temperatura é encontrada ou outro critério de término é satisfeito.
- Têmpera - a temperatura é reduzida, de acordo com uma determinada velocidade de resfriamento, até que certo valor seja encontrado e, então, aumentada para um valor alto e novamente reduzida de forma gradativa. Isso é repetido de forma periódica.

Dois aspectos devem ser analisados com relação ao decréscimo de temperatura:

(a) a regra de decréscimo da temperatura (forma da curva); e

(b) quanto tempo deve permanecer numa mesma temperatura, ou seja, quando é encontrado o equilíbrio térmico, ou ainda, quantas iterações são executadas em mesma temperatura. Uma regra prática é o número de iterações proporcional ao tamanho médio da vizinhança.

Estes aspectos determinam a qualidade e a velocidade da convergência do algoritmo para um ótimo global (anexo A2).

Considerando a regra de decréscimo uma prescrição de resfriamento pode ser:

- **Exponencial ou Clássica**

  Descrito inicialmente por Kirkpatrick [KIR 83] e outros na década de 80. Um grande número de aplicações documentadas usam alguma variação deste tipo de prescrição de resfriamento.
  A regra de decréscimo de temperatura segue a seguinte forma geral:

$$t_{k-1} = t_{k-1} . \alpha_k$$

  onde $0 < \alpha_k < 1$ e α arbitrário usualmente com valores entre 0,90 e 0,99.

- **Logarítmica**

  A regra de decréscimo de temperatura segue a seguinte forma geral:

$$t_k \sim t_{k-1} \cdot (\log \Gamma_{k-1})^{-1},$$

  onde Γ_{k-1} é alguma função baseada em dados estatísticos armazenados na cadeia de Markov *k-1*.

  Este tipo de prescrição foi proposta inicialmente para o problema PPG [AAR 85].

  Mais genericamente tem-se:

$$t_k \sim R \cdot (\log k)^{-1},$$

  onde *R* é uma constante adequada [AZE 92].
  As propriedades de convergência de uma prescrição de resfriamento logarítmica foram estudadas por vários pesquisadores, entre eles Aarts e Korst [AAR 85].

- **Prescrição Dinâmica**

  Quando um ou mais parâmetros são reajustados durante o processo de resfriamento.

  Rodrigues e Anjo[ROD 93] sugerem uma prescrição de resfriamento dinâmica baseada na utilização do conceito de entropia. Basicamente é uma prescrição exponencial com o comprimento da cadeia de Markov variável e o "equilíbrio" térmico caracterizado pela observação de variações insignificantes na entropia.

Temperatura final

A temperatura final corresponde a um congelamento, a um estado de mínima energia ou de mínima entropia. Para os problemas de otimização combinatória é estabelecido um critério de parada. Este critério de parada usualmente é implementado como:

(a) modificação mínima do valor da solução encontrada nas últimas temperaturas;

(b) verificação de que o número de iterações máximo foi ultrapassado;

(c) verificação de que o número movimentações aceitas não ultrapassa um determinado percentual (pequeno) das movimentações propostas.

Estes critérios podem usados isolados ser ou combinados entre si ou, ainda, outros critérios mais sofisticados podem ser utilizados, alguns dependentes do tipo de problema.

3.3.4 Aspectos de Ajuste

O aspecto de ajuste mais importante, possivelmente, é o relacionado com a aleatoriedade. Assim, inicialmente são apresentados alguns aspectos relacionados com os geradores de números aleatórios.

Geradores de Números Aleatórios

Como foi visto nas seções anteriores, um algoritmo SA usa escolhas aleatórias quando:

(a) gera uma solução inicial;

(b) propõe uma nova solução pela perturbação da solução corrente;

(c) aplica o critério de aceitação.

Portanto, o resultado obtido por um algoritmo SA pode ser fortemente influenciado pelo tipo de escolhas aleatórias utilizadas e o desempenho pode ser altamente dependente do tempo de geração de seqüências aleatórias. Cabe, então, fazer algumas considerações sobre aleatoriedade no contexto de seqüências geradas por um procedimento em computador.

Aleatoriedade é uma propriedade negativa, é ausência de qualquer ordem. Aleatoriedade é propriedade de uma seqüência infinita de números.

Considerando a propriedade de aleatoriedade uma seqüência pode ser:

(a) realmente aleatória – quando exibe uma verdadeira aleatoriedade, como a seqüência de tempos entre os ``tics'' de um contador Geiger exposto a um elemento radioativo.

(b) quase-aleatória – São permutações pseudo-aleatórias de um conjunto de números. Seja o conjunto de inteiros [0, 9]. As seqüências dos números de 0 até 9, em qualquer ordem ((0,1,2...9),(9,8,7,...0),(5,7,9,...1),...), são quase-aleatórias, isto é, a ordem é "aleatória", mas a quantidade de números é mantida.

(c) pseudo-aleatória – tem a aparência de aleatoriedade, contudo exibe um padrão específico repetitivo.

Uma definição prática [PRE 88] de aleatoriedade no contexto de seqüências geradas em computador é que o procedimento que produz a seqüência aleatória deve ser diferente e, em todos os aspectos possíveis de serem mensuráveis, estatisticamente não correlacionado com o procedimento usuário da seqüência gerada. Adicionalmente, se forem utilizados dois geradores diferentes acoplados a um mesmo procedimento usuário os resultados produzidos por ambos devem ser similares estatisticamente. Em caso contrário, um ou os dois geradores não são adequados.

Assim, embora a fundamentação teórica do SA é baseada em geradores de números "realmente" aleatórios, na prática, são utilizados os geradores de números pseudo-aleatórios, pois os computadores atualmente constituídos são determinísticos e, portanto a única forma possível de aleatoriedade é buscar alguma informação "externa" (como, por exemplo, o tempo em microsegundos mantido pelo relógio do computador) para inicialização ou alteração dos valores gerados pelo procedimento. Números pseudo-aleatórios são seqüências de números em um intervalo (tipicamente 0,1) que são facilmente gerados por um procedimento em computador e que satisfazem alguns testes estatísticos relativos a aleatoriedade [KAL 86]. Uma vez que estas seqüências são geradas por um algoritmo determinístico, elas não são realmente aleatórias e, portanto, podem não satisfazer um ou mais testes estatísticos de aleatoriedade. O critério de escolha de um gerador de números pseudo-aleatórios para um determinado uso é que ele satisfaça testes que estejam relacionados com o caso particular.

As propriedades desejáveis de gerador de números pseudo-aleatórios são [CEP 98] [ALL 96]:

(a) Determinismo – um gerador de números aleatórios perfeito deve parecer aleatório a menos que se conheça o algoritmo e seus estados. Isso é necessário para fins de

depuração de código. Cada número aleatório r_k gerado é, usualmente, obtido por uma função $f(r_{k-1})$.

(b) Período ou ciclo longo – suficientemente longo para a aplicação em vista.

(c) Uniformidade - significa que no caso de que todo um ciclo ocorra, todos os números possíveis de serem gerados devem ocorrer uma vez.

(d) Correlação - significa que cada subseqüência de números aleatórios não deve ser correlacionada com qualquer outra subseqüência de números aleatórios, ou seja, as seqüências de números gerados devem ser não correlacionadas serialmente.

(e) Eficiência – o gerador deve ser eficiente (observar um consumo de tempo menor que 1% do tempo da aplicação utilizada).

Outros Ajustes

Número máximo de iterações:

O número máximo de iterações normalmente é estabelecido considerando exclusivamente aspectos práticos, como uma forma de garantir o término do algoritmo em tempo aceitável.

O número máximo de iterações no procedimento de aquecimento também é estabelecido empiricamente de forma a garantir um tempo de computação proporcional a cada fase do algoritmo.

Tabelas de funções:

A função exponencial é geralmente grande consumidora de recursos computacionais. Na prática, pode ser importante manter uma tabela de valores em função da diferença de energia e temperatura calculada a cada temperatura.

3.4 Prescrições de Resfriamento

Nesta seção são descritos algumas prescrições de resfriamento documentadas na literatura.

Estas prescrições de resfriamento usualmente são implementadas através dos seguintes procedimentos:

- Aquecimento,
- Resfriamento,
- Identificação do equilíbrio térmico (critério de equilíbrio), e
- Identificação do estado de congelamento (critério de parada).

Para controlar a execução destes procedimentos devem ser atribuídos valores a determinados parâmetros de entrada, que são definidos para cada tipo de prescrição.

3.4.1 Prescrição de Resfriamento de Kirkpatrick

Este tipo de prescrição, baseado na definição inicial de Kirkpatrick [KIR 83] [AAR 89], tem sido utilizado em muitas aplicações de SA e é baseada em regras empíricas conceitualmente simples. Os parâmetros de entrada para este tipo de prescrição são:

(a) Taxa de aceitação inicial χ_0,

(b) Parâmetro α para atualização da temperatura, e

(c) Parâmetros de ajuste.

Aquecimento

A regra geral para arbitrar o valor da temperatura inicial t_0 é buscar uma temperatura onde virtualmente todas as transições propostas são aceitas, assim a taxa de aceitação $\chi_0 = \chi(t_0)$ deve ser próximo de 1.

Uma forma simples de simular o processo de aquecimento é iniciar com um valor baixo inteiro positivo para t_0, multiplicar este valor por uma constante positiva maior que 1 até obter uma taxa de aceitação próxima de 1, calculada a partir das transições geradas.

Resfriamento

A regra de decréscimo de temperatura segue a seguinte forma geral:

$$t_{k+1} = t_k . \alpha$$

onde $0 < \alpha < 1$ e α usualmente com valores entre 0,80 e 0,99.

Congelamento

O algoritmo termina quando o valor da energia obtido na última iteração a uma dada temperatura(cadeia de Markov) permanece inalterado por um certo número de iterações(cadeias de Markov) consecutivas.

Equilíbrio térmico

O equilíbrio térmico é baseado no requerimento que a cada valor de t_k um quase-quilíbrio deve ser restabelecido. O número de transições necessários para chegar a isto é calculado usando o argumento empírico de que um quase-equilíbrio será restabelecido após um número fixo de transições aceitas. No entanto, como as transições são aceitas com probabilidade decrescente, poderia ocorrer que $l_k \rightarrow \infty$ quando $t_k \downarrow 0$. Em conseqüência, na prática, deve ser estabelecido um limite l_{lim} para evitar um número de iterações muito grande a cada temperatura.

3.4.2 Prescrição de Resfriamento de Aarts

Aarts e Laarhoven [AAR 89] sugerem uma prescrição de resfriamento que resulta numa execução em tempo polinomial para a execução do algoritmo SA, mas para a qual não há garantia do desvio entre a solução obtida e o custo ótimo. Os parâmetros de entrada (detalhados nos parágrafos seguintes) para este tipo de prescrição são:

(a) Taxa de aceitação inicial χ_0,

(b) Parâmetro de distância δ para atualização da temperatura,

(c) Parâmetro de parada ε_s usado no critério de parada, e

(d) Parâmetros de ajuste.

Estes parâmetros não são dependentes da instância do problema.

Os aspectos descritos a seguir são baseados em Aarts e Laarhoven e sumarizam as idéias dos autores. Maiores informações e provas podem ser encontradas em [AAR 89].

Aquecimento

O valor inicial t_0 é obtido considerando que virtualmente todas as transições propostas devem ser aceitas, assim a taxa de aceitação deve ser aproximada pela seguindo a expressão:

$$\chi \approx \frac{m_1 + m_2\left(\exp\left(\frac{-\overline{\Delta f^{(+)}}}{t}\right)\right)}{m_1 + m_2} \qquad (3.04)$$

onde

- m_1 denota o número de transições propostas de uma solução i para para uma solução j onde $f(j) \leq f(i)$, isto é, transições de custo não crescente,
- m_2 denota o número de transições propostas de i para j onde $f(j) > f(i)$, isto é, transições de custo crescente, e
- a média da diferença de custos entre as transições com custo crescente m_2 é representado por

$$\overline{\Delta f^{(+)}}$$

de onde se obtém

$$t = \frac{\overline{\Delta f^{(+)}}}{\ln \frac{m_2}{m_2\chi - m_1(1-\chi)}}. \qquad (3.05)$$

Um valor inicial para t_0 é calculado usando a (3.05) da seguinte forma: inicializa-se t_0 com zero; gera-se uma seqüência m_0 de experimentos; a cada experimento um novo valor é calculado para t_0 usando a equação (3.05) e $\chi = \chi_0$, até obter um convergência para um valor final para t_0.

Resfriamento

Um novo valor para temperatura é calculado, baseada em dados estatísticos armazenados nas iterações realizadas na temperatura corrente, utilizando a seguinte equação

$$t_{k+1} = \frac{t_k}{1 + \frac{t_k . \ln(1+\delta)}{3\sigma_{t_k}}}, \quad k = 0,1,..... \tag{3.06}$$

onde o parâmetro de distância arbitrário δ é um valor positivo (tipicamente 0,10) e σ_k é desvio médio do custo calculado a partir dos valores obtidos na temperatura t_K, usando as

$$\bar{f}(t_k) = \frac{1}{L}\sum_{i=1}^{L} f_i(t_k) \tag{3.07}$$

seguintes equações:

e

$$\sigma(t_k) = \left(\frac{1}{L}\sum_{i=1}^{L} \left(f_i(t_k) - \bar{f}(t_k) \right)^2 \right)^{\frac{1}{2}} \tag{3.08}$$

onde a média é obtida dos valores da função custo $f_i(t_k)$, com i = 1,2,3,..., L das L soluções geradas a um dado valor da temperatura t_k..

Um valor maior para o parâmetro de distância δ significa um decréscimo maior de temperatura e um valor menor, um decréscimo menor de temperatura.

Congelamento

O valor tempetura final é dado critério de parada definido pela seguinte equação

$$\frac{t_k}{\langle f \rangle_\infty} \left. \frac{\partial \langle f \rangle_t}{\partial t} \right|_{t=t_k} < \varepsilon_{s,} \tag{3.09}$$

onde ε_S é um parâmetro de parada arbitrário (tipicamente $\varepsilon_S = 10^{-5}$).

Equilíbrio Térmico

O critério de equilíbrio é formulado considerando que a cada temperatura um quase-equilíbrio deva ser restabelecido. Para que isso possa ser obtido deve ser realizado um número de transições a cada valor de temperatura que permita uma alta probabilidade do algoritmo visitar pelo menos a maior parte da vizinhança da uma dada solução. É considerado um valor adequado para o número de iterações L_k um valor igual ao tamanho da vizinhança. Sendo o tamanho da vizinhança constante então

$$L_k = L = \Theta, \quad k = 0,1,..... \tag{3.10}$$

Estimativa do tempo de computação

A estimativa de um tempo de computação é dada por

$$0\left(\tau.L.\ln|S|\right)$$

onde L denota o comprimento de cada cadeia de Markov, $\ln|S|$ denota um limite superior para para o número de cadeias de Markov, $|S|$ denota a cardinalidade do espaço de soluções e τ denota o tempo para cada transição individual.

3.4.3 Prescrição de Resfriamento de Rodrigues

Rodrigues e Batel [ROD 93] propõem uma prescrição de resfriamento adaptativa baseada no conceito de entropia considerando os pontos de vista da Teoria da Informação e da Termodinâmica. Uma estimativa empírica da entropia termodinâmica $\bar{S}$ é obtida com:

$$\bar{S} = \frac{m_a}{m_a + m_r} \tag{3.11}$$

onde m_a é número de movimentos aceitos e m_r é número de movimentos rejeitados a cada temperatura.

Aquecimento

O valor da temperatura inicial é calculado, usando a equação (3.11), pelo seguinte procedimento: escolher um valor desejável z_0 para entropia inicial, usualmente 0,85; escolher um valor arbitrário t_0 para a temperatura; gerar uma seqüência m_0 de experimentos; a cada experimento verificar o valor da entropia usando a (3.11); enquanto a entropia for menor que z_0 multiplicar a temperatura por 2.

Resfriamento

A temperatura é atualizada usando a equação

$$t_{k+1} \leftarrow 0{,}85\ t_k.$$

Equilíbrio

O número de iterações a cada temperatura (comprimento da cadeia de Markov) é variável e verificado pela modificação insignificante no valor da entropia, calculada pela equação (3.11). O significado de modificação insignificante é arbitrário e dependente do problema.

Congelamento

Temperatura final é uma constante $t_f \leftarrow 0{,}1$.

3.5 Especificação Formal de um Método de Desenvolvimento de Algoritmo SA

Nesta seção é construída um especificação formal de um método de desenvolvimento de algoritmos. Esta especificação formal consiste de um diagrama sintático que define os domínios das funções e predicados, um programa abstrato cuja estrutura segue o paradigma do Simulated Annealing e um conjunto de axiomas que estabelece as características das funções e predicados e o relacionamento entre os mesmos. Um estudo de caso no qual um programa exemplo é analisado com relação a esta especificação é apresentado na seção 3.6. Uma verificação de correção da especificação do programa abstrato é dada no Anexo A1. Esta verificação é importante como fator de avaliação da qualidade da especificação, no entanto, por sua leitura ser mais tediosa, está em um anexo.

3.5.1 Introdução

Um problema [VEL 84] é uma terna $p = (D, R, q)$, onde D é o domínio dos dados, R o domínio dos resultados e q uma relação de D em R. Esta relação q define o problema. A solução de um problema é uma função $\alpha : D \rightarrow R$ tal que para todo $d \in D$, tem-se o par $(d, \alpha(d)) \in D$. Um algoritmo é um método abstrato [KNU 83] que computa α.

Um método de desenvolvimento de algoritmos [TOS 88] é um par (*Prog*, *Axio*), onde *Prog* é um programa abstrato especificado a partir de funções sintaticamente bem definidas e *Axio* é um conjunto de axiomas que definem a semântica das funções. Se m = (*Prog*, *Axio*) é um método de desenvolvimento de algoritmos, uma instância de *Prog* que satisfaz *Axio* é um algoritmo desenvolvido por m. Se p é um problema e α é uma solução de p, o conjunto de todos os algoritmos desenvolvidos por m que computam α é denotado por $i(m, p, \alpha)$. O domínio de m, $D(m)$, é o conjunto de todos os pares (p, α) tal que p é um problema, α é a solução de p e $i(m, p, \alpha) \neq \varnothing$.

Uma especificação formal de um método de desenvolvimento de algoritmos [AGU 98] consiste de um diagrama sintático que define o domínio das funções e predicados, um programa abstrato que dá a estrutura algorítmica do método, uma axiomatização que estabelece o relacionamento entre funções e predicados e uma verificação de correção do programa abstrato. Um algoritmo gerado por um método de desenvolvimento de algoritmos é uma instância do programa abstrato obtido pela instanciação das funções e predicados que satisfazem os axiomas especificados.

Assim, um programa abstrato é especificado para identificar as características essenciais de um algoritmo SA e suas propriedades e complementado através de axiomas definidos para os principais procedimentos e predicados especificados para este programa abstrato.

O algoritmo SA será formalizado como constituído de duas etapas básicas:

- Inicialização – Esta etapa tem como objetivos básicos gerar uma solução inicial e atribuir um valor inicial a temperatura. A solução inicial é construída por um procedimento usando os elementos que definem a instância do problema.

- Procedimento iterativo de melhoramento – Nesta etapa a solução inicial é aprimorada até que determinadas condições sejam satisfeitas. Em cada iteração é construída uma solução a partir da solução corrente (perturbação da solução corrente). Esta nova solução deve ser vizinha da solução corrente e é aceita ou não de acordo com o critério especificado, em geral, o conhecido como critério de Metropolis (equação 3.03). Este critério de aceitação avalia e compara as soluções considerando uma função Energia, definida a partir da solução objetivo original do problema, e um fator aleatório. Um parâmetro, a temperatura, é utilizado para controlar o processo, e usado na expressão de comparação das soluções. Usualmente, um número L de iterações é realizado a um dado valor de temperatura até que determinadas condições sejam satisfeitas. O valor da temperatura é decrescido usando uma função especificada, e o processo, a um valor constante para temperatura, repetido. Este procedimento a uma temperatura constante será denotado por metropolis. O número L pode ser constante ou variável durante o melhoramento, eventualmente igual a 1. Maiores detalhes deste processo já foram vistos nas seções anteriores deste capítulo. Para um estudo teórico mais aprofundado, este processo pode ser modelado por uma seqüência de k_m cadeias de Markov, onde cada uma dessas cadeias é constituída de L experimentos realizados a uma temperatura constante, onde cada experimento consiste da perturbação seguido da aplicação do critério de aceitação. A utilização do modelo baseado em cadeias de Markov é justificado pelo fato de que o modelo tem sido usado por vários autores em análise de convergência do algoritmo SA (anexo A2).

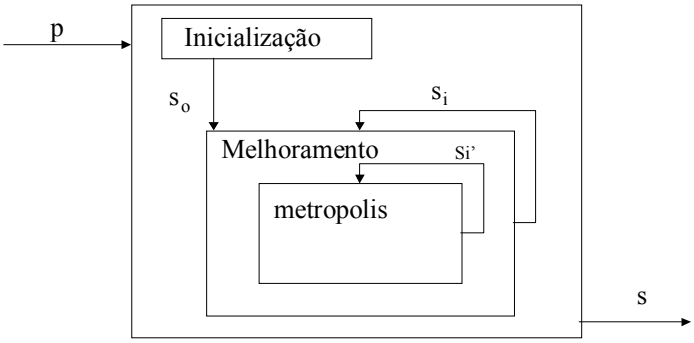

FIGURA 3.4 – Etapas do SA

A entrada para o algoritmo é a instância do problema. Os experimentos constróem uma seqüência de soluções que constituem um conjunto S' de soluções, onde $S' \subseteq S$ (conjunto de soluções possíveis para a instância do problema), dando como saída uma solução s^* quando as condições especificadas são satisfeitas. Os aspectos relacionados com a representação da instância do problema são apresentados na seção 3.3.2.

Assim, é construída uma especificação formal constituída de:

Definição dos domínios que serão utilizados pelo programa abstrato.

Definição das funções e predicados.

Diagrama sintático que representará, graficamente, a sintaxe e a aridade das funções e predicados especificados no programa abstrato, observando-se a seguinte notação: as funções são representadas por linhas com setas e os predicados por linhas.

Programa abstrato caracterizado por uma asserção de entrada φ com o objetivo de garantir a qualidade dos dados de entrada, uma asserção de saída ψ que estabelece condições para os dados de saída do procedimento, isto é, relaciona a saída com a entrada e invariantes que estabelecem condições que devem ser satisfeitas a cada vez que o programa atingir determinado ponto para garantir a consistência da solução a ser analisada a cada iteração e permitir a verificação da correção do programa.

Estabelecimento de axiomas que definem as propriedades das funções e predicados e permitem avaliar as condições descritas pelas invariantes.

3.5.2 Definição de Domínios

São considerados os seguintes domínios para utilização pelo programa abstrato:

P é o conjunto de instâncias p do problema.

$T \subset \Re^+$ é um conjunto de temperaturas.

S é o conjunto de soluções possíveis ou o conjunto de soluções viáveis de uma instância do problema. Este conjunto de soluções constitui o espaço de soluções a ser explorado. O tipo de solução é dependente da instância do problema. Cada elemento deste conjunto de soluções é construído pelo algoritmo pela função perturbação, portanto, cada solução é resultante da escolha desta função e esta função por sua vez é resultante do tipo de vizinhança projetado para a instância do problema.

Seja $p \in P$ um problema, então p é uma estrutura do tipo ($p_d, f_E, f_{vav}, \pi_p, \pi, n_{cad}, n_L, n_{aqc}, t_{ini}$, *indistinguível* , *indistinguível'*, ...) onde

- p_d é uma estrutura capaz de representar os dados que caracterizam a instância;
- f_E é uma função de avaliação de energia, sendo que esta função pode ser diferente da função objetivo do problema original pela inclusão de parcelas que representam penalidades;
- f_{vav} é uma função que permite construir, a partir de uma solução possível, uma solução viável de qualidade aproximada a esta solução possível (o significado de qualidade aproximada é dependente da instância do problema);

- π_p *é* um predicado que identifica se a solução é possível;
- π um predicado que identifica se a solução possível é viável;
- n_{aqc} é o número de experimentos na fase de aquecimento;
- n_{cad} é o número de procedimentos metropolis, ou seja, número de vezes que a temperatura é ajustada;
- n_L é o número de experimentos em cada procedimento metropolis;
- t_{ini} é um valor para temperatura inicial para o procedimento de aquecimento;
- *indistinguível* é um predicado que define quando duas soluções geradas em experimentos subsequentes são de energia indistinguíveis (o significado de energias indistinguíveis é dependente do problema e dado por algum critério de qualidade adequado para a instância do problema);
- *indistinguível'* é um predicado que define quando duas soluções geradas nos últimos procedimentos metropolis subsequentes são de energias indistinguíveis (o significado de energias indistinguíveis é dependente do problema e dado por algum critério de qualidade adequado para a instância do problema);

Outros elementos podem ser incluídos nesta lista, dependendo da forma como são implementados determinados procedimentos, assim como alguns elementos podem ser excluídos.

Seja $s \in S$ uma solução possível então s é uma estrutura do tipo (s_{est}, c) onde s_{est} é uma estrutura capaz de representar uma solução possível e c é a energia da solução avaliada por f_E.

N_r é o conjunto de números gerados por gerador de números aleatórios.

3.5.3 Procedimento SA

Considerando a introdução da seção 3.5.1 que descreve o algoritmo SA e os domínios definidos na seção 3.5.2 são identificados os predicados e funções que são usados na especificação do programa abstrato SA..

Diagrama Sintático

O Diagrama Sintático do procedimento SA é representado pela FIGURA 3.5.

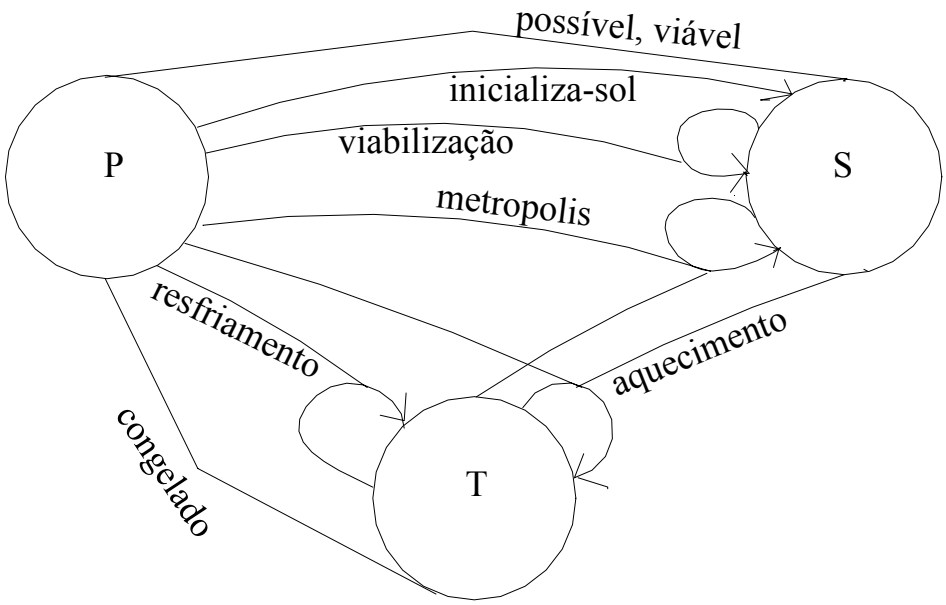

FIGURA 3.5 – Diagrama Sintático SA

Identificação dos predicados e funções

Os seguintes predicados são definidos para o programa abstrato SA:

possível: P x S → {V, F}

Quando verdadeiro indica que uma solução é uma solução possível para o problema.

viável: P x S → {V, F}

Quando verdadeiro indica que uma solução é uma solução viável para o problema.

congelado P x T → {V, F}

Corresponde a uma condição de término do algoritmo.

As seguintes funções são definidas para o programa abstrato SA:

inicializa-sol: $P \rightarrow S$

Gera uma solução inicial aleatória e avalia sua energia. Quando o problema tratar somente de soluções viáveis esta solução inicial deve ser viável.

aquecimento: $P \times S \times T \rightarrow T$

Atribui um valor inicial ao parâmetro temperatura. Esta atribuição pode ser realizada de diversas formas, sendo a mais simples a que faz uma atribuição direta de um valor arbitrário escolhido de acordo com a instância do problema.

resfriamento: $P \times T \rightarrow T$

Realiza o decréscimo da temperatura usando diferentes formulações. Para maiores detalhes sobre possíveis formas para o decréscimo de temperatura ver seção 3.4.1.

viabilização: $P \times S \rightarrow S$

Quando o algoritmo trata soluções possíveis, a solução obtida no final nem sempre é uma solução que satisfaz o predicado viável. Assim é necessário especificar o procedimento que constrói uma solução viável a partir de uma solução possível.

metropolis: $P \times S \times T \rightarrow S$

É um procedimento iterativo realizado a temperatura constante e que devolve uma solução que corresponde a uma situação de equilíbrio.

Programa Abstrato

O programa abstrato SA é caracterizado por:

(a) uma asserção de entrada φ com o objetivo de garantir a qualidade dos dados de entrada, restringindo o funcionamento do algoritmo para entradas que satisfazem φ(*p*);

(b) uma asserção de saída ψ que estabelece condições para os dados de saída do procedimento, isto é, relaciona a saída com a entrada;

(c) invariantes que estabelecem condições que devem ser satisfeitas a cada vez que o algoritmo atingir determinado ponto para garantir a consistência da solução a ser analisada a cada iteração e permitir a verificação da correção do programa.

O procedimento principal SA é representado pelo programa abstrato da Figura 3.6. A asserção de entrada garante a qualidade da entrada *p* e estabelece restrições aos valores dos parâmetros de entrada, como o número de iterações e temperatura inicial, e condições que devem satisfazer funções definidas para uma instância de um problema a ser tratada, como f_{vav}. O conjunto das entradas *P* foi definido na seção 3.1.

Assim, a asserção de entrada para uma entrada *p* a ser considerada para o programa abstrato da Figura 3.6, onde:

- devem ser especificados valores maiores que zero para os parâmetros n_{cad}, n_L, e n_{aqc} que estabelecem um limite para o número de chamadas do procedimento metropolis, o número de experimentos em cada procedimento metropolis e o número de experimentos no procedimento aquecimento. Estes números são usados isolados ou em combinação com outros critérios nas condições de parada, funcionando como um limite no número de iterações. Quando forem especificados outros critérios de parada, é desejável que sejam estabelecidos estes limites de número máximo de iterações como uma garantia extra de término. É difícil estabelecer qualquer outro critério para uso isolado que seja adequado para qualquer entrada *p*, pois o panorama de energia (seção 3.3.2) correspondente a uma determinada instância pode comprometer a satisfação de outros critérios em tempo computacional aceitável.

- O valor inicial t_{ini} do parâmetro temperatura deve ser maior que zero.

- A função f_{vav} é uma função que deve ser especificada quando as soluções tratadas pelo programa abstrato são soluções possíveis para a instância do problema *p*. Assim, a solução tratada na última iteração do processo de melhoramento é uma solução possível para esta instância *p*. Portanto, na asserção de entrada deve ser garantida a existência e a correção desta função que constrói uma solução viável a partir de uma solução possível ($(\forall p)\ (\forall s)$ possível $(p,s) \Rightarrow$ viável $(p, f_{vav}(p, s))$. A qualidade da solução viável construída pela função deve ser de qualidade semelhante a solução possível encontrada. Se o programa abstrato tratar somente soluções viáveis, a última solução tratada é viável e, portanto, esta função é a função identidade.

Procedimento SA

Entrada: p

Saída: s

$\varphi(p) = \{ (|S| > 0) \wedge (n_{cad} > 0) \wedge (n_L > 0) \wedge (n_{aqc} > 0) \wedge (t_{ini} > 0) \wedge ((\forall p) (\forall s)$ possível $(p,s) \Rightarrow$ viável $(p, f_{vav} (p, s))\}$

1 procedimento SA (p)

2 p: instância;

3 s, s_{ini}: solução;

4 t, t_{ini}: temperatura;

5 $s_{ini} \leftarrow$ inicializa-sol(p);

6 % {possível $(p, s_{ini}) \wedge t_{ini} > 0$}

7 t ← aquecimento(p, s_{ini});

8 % {quente $(p, t) \wedge t > 0$}

9 % início da fase de melhoramento:

10 enquanto ¬ congelado (p, t) faça

11 % {possível $(p, s) \wedge t > 0$}

12 s ← metropolis (p, s, t);

13 % (possível $(p, s) \wedge$ estável $(p, s_{ent}, t, s) \wedge t > 0$}

14 t ← resfriamento (p, t);

15 % {t > 0}

16 fim-enquanto;

17 % fim da fase de melhoramento;

18 % (congelado (p, t)}

19 % (possível $(p, s) \wedge$ estável$^+$ (p, s_0, s)}

20 s ← viabilização(p, s);

21 % {viável (p, s)}

22 fim-com-saída(s);

$\psi(p, s) = \{$(viável $(p, s) \wedge$ estável$^+$ (p, s_0, s)}

FIGURA 3.6 – Procedimento SA

A asserção de saída do procedimento SA estabelece duas características para a solução fornecida pelo procedimento. A primeira é que a solução fornecida seja uma solução viável e a segunda que esta solução seja uma que atenda determinados critérios de estabilidade. Estes critérios de estabilidade são estabelecidos pelos predicados que devem ser satisfeitas no decorrer da execução do programa (linhas 6, 8, 11, 13, 15, 18, 19, 21).

Estes predicados são analisados levando em consideração a asserção de entrada e os axiomas que estabelecem as características dos procedimentos que constituem o programa abstrato.

Os procedimentos metropolis e aquecimento são componentes essenciais do procedimento SA e são melhor detalhados nas seções 3.5.4 e 3.5.5, respectivamente.

3.5.4 Procedimento metropolis

Este é o procedimento que engloba os aspectos mais importantes do algoritmo. O procedimento metropolis corresponde a uma seqüência de iterações executadas a um valor constante para o parâmetro temperatura e termina quando uma condição de equilíbrio é satisfeita. Usando analogia com a termodinâmica, equilíbrio pode ser interpretado como estabilidade com relação a solução a ser retornada pelo procedimento. Esta estabilidade pode ser determinada por alguma quantidade macroscópica que possa ser medida. Uma das interpretações é considerar o número de iterações e relacionar este número de iterações com o tamanho da vizinhança. Assim, é comum encontrar que o equilíbrio é alcançado após L iterações, onde $L = k\Theta$, sendo $k = 1, 2, 3, ...$ uma constante e Θ o tamanho da vizinhança para a instância do problema.

Esta interpretação pode ser usada isolada ou combinada com outras mais elaboradas e, algumas vezes, levando em consideração aspectos dependentes do problema.

Uma especificação para o procedimento metropolis é apresentada na Figura 3.8.

Diagrama Sintático

A Figura 3.7 representa o diagrama sintático do procedimento metropolis.

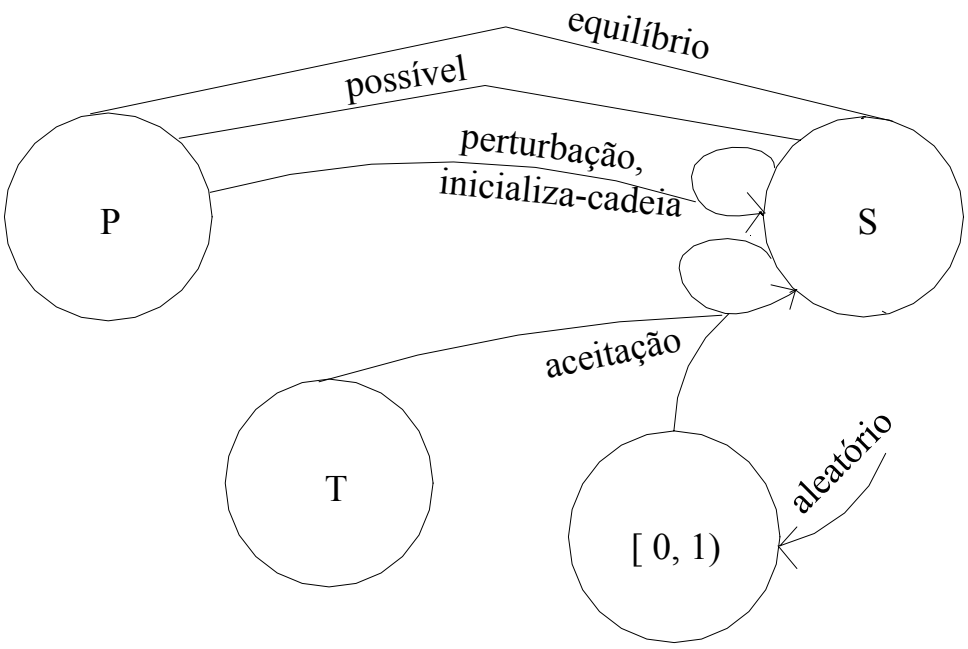

FIGURA 3.7 – Diagrama Sintático metropolis

Identificação dos predicados e funções

O seguinte predicado é definido para este procedimento metropolis:

equilíbrio: $P \times S \rightarrow \{V , F\}$

Corresponde a uma condição de término do procedimento.

As seguintes funções são definidas para este procedimento metropolis:

inicializa-cadeia: $P \times S \rightarrow S$

Esta função deve inicializar os aspectos relacionados com a solução de entrada para programa abstrato metropolis e, tratar outros aspectos de inicialização como um contador de iterações (experimentos).

aleatório: $[0, 1) \rightarrow N_r$

Esta função tem como finalidade gerar um número aleatório. As propriedades desejáveis para um gerador de números pseudo-aleatórios são descritas na seção 3.3.4.

perturbação: $P \times S \rightarrow S$

A função perturbação tem a finalidade de construir uma solução candidata a partir da solução corrente e calcular a energia desta solução pela função energia f_E. A solução construída deve pertencer ao conjunto de soluções vizinhas da solução corrente. A energia da solução pode ser calculado diretamente a partir dos elementos da solução candidata ou usar a diferença de energias, quando for possível o cálculo desta diferença a partir das modificações aplicadas a solução corrente para obter a solução candidata. No caso de ser utilizado o cálculo da diferença pode ser necessário, periodicamente, calcular a energia direto quando houver a possibilidade de degeneração do valor por algum motivo, por exemplo, algum arredondamento.

aceitação: $S \times S \times T \times N_r \rightarrow S$

A função aceitação compara as duas soluções e retorna a solução escolhida de acordo com um determinado critério de aceitação, que deve incluir um fator aleatório. A solução escolhida satisfaz o predicado aceita. Este predicado aceita pode ser interpretado como uma alternativa de escape do ótimo local. O critério de aceitação usualmente utilizado é o critério de metropolis (seção 3.3, expressões 3.03 e 3.04).

Cabe a seguinte observação: os procedimentos aleatório, perturbação e aceitação do procedimento metropolis e os procedimentos aleatório, perturbação e aceitação do procedimento aquecimento têm, respectivamente, as mesmas funções. Podem ser utilizadas as mesmas implementações ou implementações com pequenas diferenças.

Programa Abstrato

O programa abstrato metropolis é caracterizado por:

(a) uma asserção de entrada φ com o objetivo de garantir a qualidade dos dados de entrada, restringindo o funcionamento do algoritmo para entradas que satisfazem $\varphi(p, s_0, t)$;

(b) uma asserção de saída ψ que estabelece condições para os dados de saída do procedimento, isto é, relaciona a saída com a entrada;

(c) predicados (linhas 6, 8, 11, 13, 15) que estabelecem condições que devem ser satisfeitas a cada vez que o programa atingir determinado ponto para garantir a consistência da solução a ser analisada a cada iteração e permitir a verificação da correção do programa.

Procedimento metropolis

Entrada: *p*, *s*, *t*;

Saída; *s*;

$\varphi(p, s_0 ,t) = \{ \text{possível} (p, s_0) \wedge (t \geq 0)\}$

```
1    Procedimento metropolis ( p, s0, t)
2    p:                instância
3    s, s0:            solução
4    t:                temperatura
5    s ← inicializa-cadeia ( p, s0);
6                      % {possível (p, s)}
7    enquanto ¬ equilíbrio( p, s) faça
8                      % {possível (p, s)}
9              nr ← aleatório ();
10             sc ← perturbação ( p, s);
11                     % {possível (p, s) ∧ possível (p, sc)}
12             s ← aceitação (s, sc, t, nr);
13                     % {aceita (s, sant, sc, t, nr )}
14   fim-enquanto;
15                     % {estável (p, s0, t, s)}
16   fim-com-saída( s);
```

$\psi(p, s_0, t, s) = \{\text{possível} (p, s) \wedge \text{estável} (p, s_0, t, s)\}$

FIGURA 3.8 – Procedimento metropolis

3.5.5 Procedimento aquecimento

O procedimento aquecimento tem como finalidade elevar a temperatura até que o sistema esteja suficientemente quente.

O procedimento de aquecimento pode ser implementado de diversas formas. As mais simples são: (a) Pela atribuição direta de um valor t_o considerado alto ao parâmetro temperatura. (b) Pela execução iterativa de um procedimento que eleve a temperatura até que uma condição especificada seja satisfeita.

Uma especificação para o procedimento aquecimento é dada na Figura 3.10, conforme (b).

Diagrama Sintático

A Figura 3.9 representa o diagrama sintático do procedimento aquecimento.

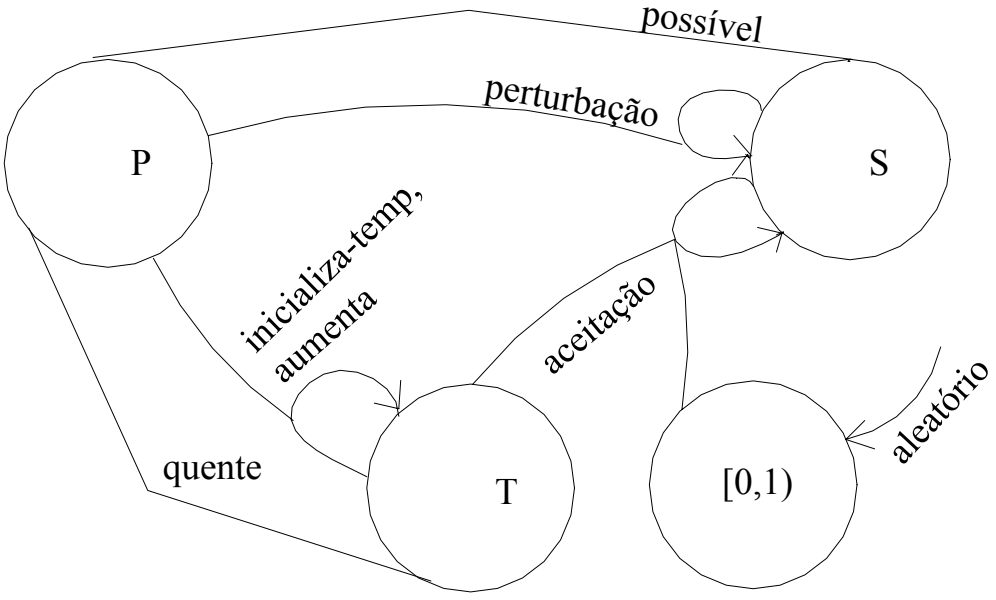

FIGURA 3.9 – Diagrama Sintático aquecimento

Identificação dos predicados e funções

Os seguintes predicados são definidos para o procedimento aquecimento:

quente: $P \times T \rightarrow \{V, F\}$

Quando verdadeiro indica que a condição especificada para determinar que um valor do parâmetro temperatura suficientemente alto foi alcançado.

As seguintes funções são definidas para o procedimento aquecimento:

inicializa-temp: $P \rightarrow T$

Esta é uma função de inicialização com a finalidade básica de inicializar a temperatura t com um valor de entrada para o enquanto 6-15. Na especificação dada este valor é t_{ini}, um parâmetro de entrada definido em p.

aleatório: $[0,1) \rightarrow N_r$

Gera um número aleatório. As propriedades desejáveis para um gerador de números pseudo-aleatórios são descritas seção na 3.3.4.

perturbação: $P \times S \rightarrow S$

A função perturbação tem a finalidade de construir uma solução candidata a partir da solução corrente e calcular a energia desta solução pela função energia f_E. Esta função é dependente da instância do problema. A solução construída deve pertencer ao conjunto de soluções vizinhas da solução corrente (ver seção 3.3). A energia da solução pode ser calculada diretamente a partir dos elementos da solução candidata ou usar a diferença de energias, quando for possível o cálculo desta diferença a partir das modificações aplicadas a solução corrente para obter a energia da solução candidata. No caso de ser utilizado o cálculo da diferença pode ser necessário, periodicamente, calcular a energia direto quando houver a possibilidade de degeneração do valor por algum motivo, por exemplo, algum arredondamento.

aceitação: $S \times S \times T \times N_r \rightarrow S$

A função aceitação compara as duas soluções e aceita ou não a solução candidata, que será a solução corrente para a próxima iteração. Se a solução candidata não for aceita permanece a atual como corrente para a próxima iteração. O critério utilizado usualmente é o critério de metropolis (seções 3.3 e 3.4).

aumenta: $P \times T \rightarrow T$

O objetivo desta função é elevar o valor da temperatura. Pode ser uma função simples como multiplicar o valor corrente por uma constante.

Como anteriormente observado, podem ser utilizadas as mesmas implementações ou implementações com pequenas diferenças para os procedimentos aleatório, perturbação e aceitação do programa metropolis e os do programa aquecimento.

Procedimento aquecimento

Entrada: p, s

Saída: s.

$\varphi(p, s, t) = \{ \text{possível } (p, s_{ini}) \wedge (t_{ini} > 0)\}$

1 Procedimento aquecimento (p, s_{ini});

2 t, t_{ini}: temperatura

3 s, s_c: soluções

4 t ← inicializa-temp (p, t_{ini}); s ← s_{ini};

5 % {possível (p, s) ∧ (t > 0)}

6 enquanto ¬ quente (p, t) faça

7 n_r ← aleatório ();

8 % {possível (p, s) ∧ (t > 0)}

9 s_c ← perturbação (p, s);

10 % {possível (p, s) ∧ possível (p, s_c)}

11 s ← aceitação (s, s_c, t, n_r);

12 % {possível (p, s)}

13 t ← aumenta(p, t);

14 % {(t > 0)}

15 fim-enquanto;

16 % {quente (p, t)}

17 fim-com-saída(t)

$\psi(p, t) = \{\text{quente } (p, t) \wedge (t > 0)\}$

FIGURA 3.10 – Procedimento aquecimento

3.5.6 Axiomatização

O estabelecimento de axiomas tem como objetivo dar uma semântica aos predicados e funções especificados para os procedimentos SA, metropolis e aquecimento. Esta axiomatização leva em consideração as seguintes condições básicas:

- Representação da solução – Cada solução é capaz de ser representada por uma codificação que depende do tipo de problema e ter sua energia avaliada pela função f_E. O tipo de representação não é particularizado. Considera-se que o programa abstrato deverá poder tratar soluções denotadas como soluções possíveis para a instância do problema. Uma solução possível pode ser uma solução viável ou não.
- Inicialização – Uma solução possível inicial é gerada de forma aleatória.
- Perturbação – Uma nova solução possível é construída a cada execução do procedimento perturbação. Esta solução é denotada como uma solução candidata. Esta solução deve ser consistente com o espaço de soluções que o algoritmo é capaz de explorar e respeitar os limites de "proximidade" estabelecidos por uma estrutura de vizinhança adequada à instância do problema. A construção da solução candidata deve incluir procedimentos que resultem em uma escolha aleatória desta solução vizinha da solução corrente.
- Aceitação – Um critério deve ser estabelecido para a escolha entre a solução corrente e a solução candidata. A solução escolhida será a corrente na próxima iteração. Este critério deve ser probabilístico, conforme o definido pela equação 3.03.
- Parâmetros – São considerados os parâmetros usuais para um algoritmo do tipo SA.

Axiomas do Procedimento SA:

SAAX01: (∀p) (∀s) viável (p,s) ⇒ [possível (p, s)]

Este axioma estabelece uma relação entre uma solução viável e uma solução possível.

inicializa-sol

SAAX02: (∀p) [possível (p, (inicializa-sol (p)))]

A solução gerada pelo procedimento inicializa-sol deve ser uma solução possível para a instância do problema. A escolha da solução inicial é aleatória e pode ser qualquer uma do espaço de soluções tratável pelo programa.

aquecimento

SAAX03: ($\forall$p) ($\forall$s) [possível (p, s) $\Rightarrow$ (aquecimento (p, s)) > 0)]

SAAX04: ($\forall$p) ($\forall$s) [possível (p, s) $\Rightarrow$ quente (p, aquecimento (p, s))]

Os dois axiomas acima estabelecem que o valor temperatura fornecido pelo procedimento aquecimento para temperatura deve ser positivo e alto. O significado do que é alto, ou seja, da condição que satisfaz o predicado quente, depende da instância do problema e do tipo de prescrição de resfriamento (seção 3.4) a ser utilizado. Uma interpretação é baseada na probabilidade de aceitação das soluções propostas, considerando que a mesma deve ser alta quando a temperatura estiver suficientemente elevada.

resfriamento

SAAX05: ($\forall$p) ($\forall$t) [t > 0 $\Rightarrow$ (resfriamento (p,t) < t) $\wedge$ (resfriamento (p,t) $\geq$ 0)]

Este axioma estabelece que a temperatura deve formar uma seqüência decrescente de valores (seção 3.3.3) [AZE 92]. Aceitando zero, esta condição deve ser usada como condição de término no predicado congelado.

viabilização

SAAX06: ($\forall$p) ($\forall$s) [viável (p, s) $\Rightarrow$ viabilização (p, s) = s]

SAAX07: ($\forall$p) ($\forall$s) [$\neg$ viável (p, s) $\Rightarrow$ viabilização (p, s) = f_{vav} (p, s)]

Estes dois axiomas estabelecem que o procedimento viabilização quando aplicado a uma solução viável s deve retornar a mesma solução viável s e quando aplicado a uma solução não viável deve retornar uma solução viável usando a função definida na entrada f_{vav}. Esta função f_{vav} é dependente do problema e deve garantir uma equivalência entre as soluções de entrada e saída para o procedimento.

SAAX08: ($\forall$p) ($\forall$s) ($\forall s_0$) [estavel$^+$ (p, s_0, s) $\Rightarrow$ estavel$^+$ (p, s_0, viabilização (p, s))]

O axioma acima estabelece que deve haver uma equivalência adequada, quando é considerada a instância do problema, entre a solução possível fornecida pela fase de melhoramento e solução viável obtida a partir da mesma.

congelado

O predicado congelado estabelece uma condição de término para o procedimento SA e pode usar uma ou mais condições de verificação. O significado teórico de congelado corresponde a uma situação de estabilidade que pode ser avaliada de alguma forma e que é dependente da instância do problema. Os axiomas abaixo apresentam algumas possibilidades que podem ser generalizadas com atribuição de valores adequados para os para os parâmetros utilizados.

Uma especificação possível é a dada pelos axiomas abaixo:

SAAX09: (∀p) (∀t) [(congelado (p, t)) ⇔ condição-congelado]

Diferentes interpretações podem ser analisadas para condição congelado. Uma interpretação possível é:

condição-congelado = ((cadeia (p, t) ≥ n_{cad})) ∨ (t ≤ 0)],

que leva em consideração dois aspectos relacionados com o parâmetro temperatura, o primeiro verifica uma condição de término relacionada com um limite, estabelecido por n_{cad} para o número de vezes que este parâmetro pode ser atualizado (número de resfriamentos, ou seja, número de cadeias metropolis) e o outro considera que o congelamento ocorre quando o valor da temperatura é menor ou igual a zero, o que corresponde a um significado teórico para congelamento. Outros dois axiomas são estabelecidos para suportar esta interpretação:

SAAX10: (∀p) (∀t) [(cadeia (p, aquecimento (p, t))) = 0]

SAAX11: (∀p) (∀t) [(cadeia (p, resfriamento (p, t))) = cadeia (p, t) + 1]

Outra interpretação possível é:

condição-congelado = ((cadeia (p, t) ≥ n_{cad})) ∨ (t ≤ 0) ∨ (indistinguível$'$ (h))

Esta é uma alternativa para a interpretação usada na especificação apresentada e considera que a estabilidade também é alcançada quando as últimas *n* soluções fornecidas pelo procedimento metropolis são indistinguíveis entre si, usualmente com relação ao valor da solução avaliado por f_E. A seqüência histórica das *n* últimas soluções é capaz de ser representada por *h* e congelado é capaz de inicializar, manter, avaliar e comparar os valores desta seqüência.

metropolis

SAAX12: (∀p) (∀s) (∀t) [possível (p,s) ⇒ (estável (p,s, t, metropolis (p, s ,t)) ∧ possível (metropolis(p, s, t)))]

O procedimento metropolis retorna uma solução que corresponde a uma situação estável a um valor constante de temperatura. A forma de avaliar esta estabilidade é dependente da instância do problema.

SAAX13: (∀p) (∀t) (∀s) (∀s') [estável (p, s, t, s') ⇔ ∃ (s_1, s_2, ..., s_n) ((s_1 = s) ∧ (s_{i+1} = metropolis (p, s_i ,t), para i =1,2, ..., n) ∧ (s_n = s') ∧ ((∃ (n'' < n) ∧ (n' > n'')) ⇒ indistinguível (s', $s_{n''}$)))]

Este axioma estabelece uma possível interpretação para estável em que os resultados obtidos a partir de um dado ponto na seqüência de experimentos são indistinguíveis entre si, ou seja, satisfazem o predicado *indistinguível'*. O significado de *indistinguível'* depende do problema e é usualmente relacionado com o valor da solução avaliado pela função f_E. Esta interpretação não foi usada na especificação proposta.

estável^{+}:

O predicado estável^{+} tem como finalidade identificar a característica da solução encontrada após a fase de melhoramento.

SAAX14: ($\forall p$) ($\forall s_0$) ($\forall s$) [estável^{+} (p, s_0, s) $\Leftarrow$ ($\exists$ (s_0, s_1, ..., s_{k+-1}) $\exists$ (t_0, t_1, ..., t_{k+1}) ($\wedge^{k}_{i=0}$ (possível (p, s_{i+1}) $\wedge$ possível (p, s_i) $\wedge$ estável (p, s_i, t_i, s_{i+1})) $\wedge$ (s_{k+1} = s)) $\wedge$ (congelado (p, s , t_{k+1}))]

Axiomas do Procedimento metropolis:

inicializa-cadeia

O procedimento inicializa-cadeia tem como finalidade preparar a execução do procedimento iterativo, incluindo as funções de inicialização necessárias.

MEAX01: ($\forall p$) ($\forall s$) [possível (p, s) $\Rightarrow$ possível (inicializa-cadeia (p, s))]

O axioma acima estabelece que a solução em tratamento deve manter o predicado possível satisfeito.

equilíbrio

O predicado equilíbrio estabelece uma condição de término para o procedimento metropolis. A forma mais simples de implementar é pela especificação de um número máximo de iterações. Esta condição pode ser implementada isolada ou combinada com outras mais elaboradas que permitam, por exemplo, verificar se as soluções fornecidas pelos procedimentos subsequentes são indistinguíveis (predicado indistinguível).

O axioma abaixo estabelece uma condição de equilíbrio:

MEAX02: ($\forall p$) ($\forall s$) [(equilíbrio (p, s) $\Leftrightarrow$ (condição-equilíbrio)]

Uma interpretação pode ser condição-equilíbrio = (experimento (p, s) $\geq$ L). Para esta interpretação os axiomas abaixo são estabelecidos:

MEAX03: ($\forall p$) ($\forall s$) ($\forall t$) [(experimento (p, inicializa- cadeia (p, s))) = 0]

MEAX04: ($\forall p$) ($\forall s$) ($\forall t$) [(experimento (p, perturbação (p, s))) = experimento (p, s) + 1]

Estes axiomas estabelecem os requisitos para inicialização e atualização de um contador de iterações.

O valor L pode ser uma constante parte da descrição da instância ou inicializado pelos procedimentos inicializa-sol (valor constante durante a execução) ou inicializa-cadeia (valor variável para cada execução do procedimento metropolis). Muitas vezes, na inicialização, é realizado um

processo de amostragem para obter o tamanho da vizinhança de uma solução, ou o tamanho médio desta vizinhança, se for o caso, e o valor L calculado a partir do tamanho desta vizinhança usando um expressão simples como, por exemplo, L = α(tamanho da vizinhança), onde (α = 1 ou 2 ou 3...).

Outras interpretações mais elaboradas podem ser dadas para a condição de equilíbrio, por exemplo, definindo de uma forma adequada o predicado indistinguível.

perturbação

O predicado possível tem como finalidade garantir que a perturbação retorne uma solução possível de ser tratada pelo problema.

MEAX05: $(\forall p) (\forall s) [possível (p, s) \Rightarrow possível (p, perturbação (p,s))]$

aleatório:

MEAX06: $[0 \leq aleatório () < 1]$

O axioma acima estabelece um intervalo para os números aleatórios gerados pelo procedimento aleatório.

aceitação:

A solução fornecida pelo procedimento aceitação é escolhida por um critério de aceitação, usualmente o critério de metropolis.

MEAX07: $(\forall p) (\forall s) (\forall t) [(possível (p,s) \wedge possível (p, perturbação (p, s))) \Rightarrow ((crit\text{-}metropolis(s, perturbação (p,s), t, aleatório())) \Rightarrow (aceitação(s, perturbação (p,s), t, aleatório()) = perturbação (p, s))) \wedge ((\neg crit\text{-}metropolis(s, perturbação (p,s), t, aleatório()) \Rightarrow (aceitação(s, perturbação (p,s), t, aleatório()) = s))]$

Este axioma caracteriza a função aceitação com relação ao critério de metropolis. Por exemplo, para uma minização o critério de metropolis pode ser (conforme definição dada pela equação 3.03):

crit-metropolis = $[(f_E (s_c) \leq f_E (s)) \vee (exp ((f_E (s) - f_E (s_c))/t) >$ aleatório)], onde s_c = perturbação (p, s).

MEAX08: $(\forall s) (\forall s') (\forall s'') (\forall t) (\forall n_r) [(aceita (s, s', s'', t, n) \Leftrightarrow (s = s'' \wedge crit\text{-}metropolis (s', s'', t, n)) \vee (s = s' \wedge \neg crit\text{-}metropolis (s', s'', t, n))]$

O axioma acima caracteriza quando uma solução satisfaz o predicado aceita.

MEAX09: ($\forall$p) ($\forall$s) ($\forall$s') ($\forall$t) ($\forall n_r$) [experimento (p, aceitação(s, s', t n_r)) = experimento (p, s)]

O axioma acima estabelece que experimento não sofre alteração pelo procedimento aceitação.

estável:

O predicado estável tem como finalidade identificar a característica da solução fornecida pelo procedimento metropolis.

MEAX10: ($\forall$p) ($\forall$s) ($\forall$t) ($\forall$s') [estável (p, s, t, s') $\Leftrightarrow$ ($\exists$ (n_1, s_{c1}, s_1) ... $\exists$ (n_k, s_{ck}, s_k)) (aceita (s_1, s_0, s_{c1}, t, n_1) $\wedge$ aceita (s_2, s_1, s_{c2}, t, n_2) $\wedge$ ... aceita (s_k, s_{k-1}, s_{ck}, t, n_k) $\wedge$ (s_k = s') $\wedge$ (s_0 = s)]

O axioma acima caracteriza a seqüência de soluções aceitas. Esta axioma estabelece que há um caminho entre a solução inicial s_0 e a solução obtida ao final do enquanto 7-14 da FIGURA 3.8.

Axiomas do Procedimento aquecimento:

inicializa-temp:

AQAX01: ($\forall$p) ($\forall t_{ini}$) [inicializa-temp (p, t_{ini}) $\Rightarrow$ t > 0]

O axioma acima estabelece que a temperatura inicial para o procedimento de aquecimento deve ser maior que zero.

quente:

O predicado quente estabelece uma condição de término para o procedimento aquecimento. A forma mais simples de implementar é pela especificação de um número máximo de iterações de aquecimento. Esta condição pode ser implementada isolada ou combinada com outras mais elaboradas que permitam, por exemplo, verificar se uma determinada probabilididade de aceitação alta de soluções foi alcançada.

Uma especificação possível é a dada pelos axiomas abaixo:

AQAX02: ($\forall$p) ($\forall$t) [(quente (p, t)) $\Leftrightarrow$ (mais-quente (p, t)) $\geq n_{aqc}$]

AQAX03: ($\forall$p) ($\forall$t) [mais quente (p, inicializa-temp (p, t_{ini})) $\Rightarrow$ 0]

AQAX04: ($\forall$p) ($\forall$t) [mais-quente (p, aumenta (p, t)) $\Rightarrow$ mais-quente (p, t) + 1]

Estes três axiomas estabelecem os requisitos para inicialização e atualização de um contador de iterações mais-quente e que tem como finalidade limitar o número de vezes que a temperatura pode aumentar.

aumenta:

AQAX05: (∀p) (∀t) [t > 0 ⇒ aumenta (p, t) ≥ t]

O axioma acima estabelece que vai ser construída uma seqüência crescente de temperatura pelo programa abstrato aquecimento.

3.5.7 Análise de Complexidade

A análise de complexidade é baseado na premissa de que Simulated Annealing pode ser visto como um método de desenvolvimento de algoritmos e que é possível analisar os aspectos de complexidade de um algoritmo específico gerado por este método a partir de uma expressão derivada do algoritmo abstrato especificado para o Simulated Annealing.

A complexidade aqui é vista como uma forma de avaliar o consumo de determinado tipo de recurso por um algoritmo para resolver uma instância de um problema. No caso, o recurso avaliado é o tempo computacional.

A forma mais comum de expressar este consumo de tempo é em termos de tamanho da instância do problema, que deve ser expressa através da utilização de um sistema de codificação fixo. Desta forma, a complexidade é vista como uma função do tamanho da instância do problema. Um estudo mais detalhado da complexidade e dos tipos de medida de complexidade pode ser encontrado em [AGU 98], [GAR 79], [AHO 74].

Para o cálculo de complexidade o algoritmo gerado pelo método é considerado uma instância do algoritmo abstrato SA e a complexidade total do algoritmo é resultado da complexidade dos procedimentos que compõe o algoritmo.

Como a complexidade de tempo de um algoritmo é expressa considerando o tamanho da entrada para este algoritmo, inicialmente é necessário estabelecer o que é a entrada para o algoritmo.

A entrada para o algoritmo SA foi definida genericamente como a instância do problema. Esta instância do problema consiste de uma estrutura cujos componentes foram descritos na seção 3.5.2. Analisando estes componentes é fácil de verificar que o componente importante e que caracteriza a variação em tamanho da entrada é o denotado como p_d e corresponde a uma estrutura de dados capaz de representar a instância original do problema a ser resolvido. Na expressão de avaliação de complexidade este tamanho será denotado por n. No entanto, observa-se, numa análise do programa abstrato apresentado, que esta complexidade também é dependente de outros parâmetros. Tendo em vista o cálculo da complexidade do pior caso, estes parâmetros são os que determinam o número máximo de iterações dos procedimentos, ou seja, n_{cad}, n_L e n_{aqc}. Na determinação de valores adequados para estes parâmetros devem ser considerados dois aspectos básicos: (a) o tamanho do espaço de solução a ser explorado pelo algoritmo; (b) o tamanho da vizinhança. Para uma

melhor expectativa de qualidade da solução fornecida pelo algoritmo é necessário que o mesmo explore uma parte significativa das soluções, e, para cada solução, é necessário que seja avaliada uma parte considerável da sua vizinhança, portanto, o arbítrio destes valores deve ser analisado com cuidado. Estes valores são, numa avaliação preliminar, claramente, função de *n*. No entanto, na especificação proposta para o programa abstrato estes parâmetros foram considerados como constantes e fornecidos com parte da entrada *p*.

Numa avaliação mais aprofundada da escolha destes parâmetros outros aspectos devem ser considerados como temperatura e prescrição de resfriamento (funções aumenta, resfriamento, predicados quente e congelado). Esta avaliação mais sofisticada deveria considerar a configuração do panorama de energia de uma determinada instância para determinar temperatura e prescrição de resfriamento. Uma possibilidade é usar um processo de amostragem para avaliar o panorama de energia, ou usar conceitos como entropia. A generalização da escolha destes parâmetros através de procedimentos computacionais, no entanto, não foi considerada no presente trabalho. Como uma primeira regra prática genérica [FRO 98] o número de iterações de aquecimento (n_{aqc}) não deve ultrapassar 100 iterações e o número total de iterações ($n_{aqc} + n_L$ x n_{cad}) não deve ultrapassar 1000 iterações para execuções em teste e 100000 iterações para execuções em produção.

Assim, a expressão de complexidade considera os parâmetros n, n_{cad}, n_L e n_{aqc}.

A expressão de avaliação de complexidade considera que a complexidade total do algoritmo é resultado da complexidade dos procedimentos que o compõe. Assim, a complexidade será expressa usando a notação C(x)(y), que significa complexidade do procedimento x é dependente da lista y.

Para um algoritmo SA, tem-se:

$$C(\text{SA})(n, n_{cad}, n_L, n_{aqc}) = C(\text{inicializa-sol})(n) + C(\text{aquecimento})(n, n_{aqc}) +$$
$$(n_{cad} + 1)\,[C(\text{congelado})(n)] +$$
$$(n_{cad})\,[C(\text{metropolis})(n, n_L) + C(\text{resfriamento})(n)] +$$
$$C(\text{viabilização})(n).$$

Assim, a expressão acima determina a complexidade de um algoritmo SA considerando como termos a complexidade dos seus procedimentos e o número máximo de vezes que estes procedimentos podem ser executados. A seguir, é apresentado a complexidade para cada destes procedimentos.

Complexidade dos procedimentos

inicializa-sol:

O principal aspecto a ser considerado é a construção da solução inicial. Como são utilizadas funções aleatórias na construção desta solução inicial, a complexidade do procedimento vai

depender da complexidade destas funções e do número de elementos que compõe a solução. Quando se optar pelo uso de uma outra heurística para construção da solução inicial, a complexidade será a complexidade desta heurística. A validade do uso de outra heurística pode ser avaliada por considerações de complexidade.

Neste procedimento também podem ser tratadas as inicializações de algumas variáveis utilizadas no procedimento. Entre estes procedimentos pode estar incluído um procedimento para inicializar o parâmetro que controla o número de iterações internas ao procedimento metropolis. Muitas vezes, este valor é baseado no tamanho da vizinhança. Estas inicializações, em geral, são de complexidade pequena e são absorvidas no total.

aquecimento

A complexidade deste procedimento é calculada pela equação:

$$C(\text{aquecimento})(n, n_{aqc}) = C(\text{inicializa-temp})(n) + (n_{aqc} + 1)[C(\text{quente})(n)] + n_{aqc}[C(\text{aleatório})(n) + C(\text{aceitação})(n) + C(\text{perturbação})(n) + C(\text{aumenta})(n)].$$

O procedimento perturbação é o mais importante a ser considerado para o cálculo de complexidade. Considerações sobre os procedimentos aleatório, perturbação, aceitação serão feitas à parte. As funções aumenta e inicializa-temp são, em geral, computacionalmente simples.

metropolis:

A complexidade do procedimento metropolis é expressa pela equação:

$$C(\text{metropolis})(n, n_L) = C(\text{inicializa-cad})(n) + (n_L + 1)[C(\text{equilíbrio})(n)] + n_L[C(\text{aleatório})(n) + C(\text{aceitação})(n) + C(\text{perturbação})(n)]$$

O procedimento perturbação é o mais importante a ser considerado para o cálculo de complexidade. Considerações sobre os procedimentos aleatório, perturbação, aceitação serão feitas à parte. A função inicializa-sol é, em geral, computacionalmente simples.

resfriamento:

A complexidade do resfriamento é, em geral, computacionalmente simples.

perturbação:

É o procedimento que , isoladamente, maior impacto tem na complexidade do algoritmo. É dependente do tipo de problema, mas, é importante salientar, que a perturbação é especificada em concordância com o tipo de estrutura de vizinhança projetada para o espaço de soluções. Assim, para uma mesma instância, estrutura de vizinhanças diferenciadas podem resultar em diferentes complexidades.

Há uma forte relação entre o tipo de estrutura de vizinhança, portanto, seu tamanho, e a complexidade e qualidade da solução obtida em algoritmo tipo o SA. Como exemplo, pode ser mencionado o trabalho de Johnson e McGeoch [JOH 97] para o problema do caixeiro-viajante, no qual são analisados os procedimentos iterativos de busca local para este problema, incluindo SA. Neste trabalho, os autores destacam que o primeiro e mais crucial aspecto a ser depurado é o relacionado com a estrutura de vizinhança.

Como regra prática procurar escolher estruturas de vizinhanças que resultem em um número pequeno de vizinhos (“pequeno”, é claro, é fortemente dependente do tamanho da instância) e uma perturbação de complexidade intrínseca baixa.

Outro fator a considerar é a avaliação da função energia. Se para cada solução construída for calculada diretamente o valor a partir dos elementos da solução a complexidade da perturbação é maior do que quando se calcula a diferença entre as soluções. Como regra prática, é importante procurar uma forma de cálculo da diferença baseada nas modificações sofridas pela solução perturbada, principalmente para funções energia mais complexas.

aceitação:

Neste procedimento o aspecto mais importante é avaliação da função exponencial, quando adotado o critério de metropolis original.

aleatório:

Deve ser verificada a complexidade do gerador de números pseudo-aleatório utilizado, uma vez que este procedimento é executado a cada iteração do algoritmo (seção 3.3.4).

Conclusões

Substituindo as complexidade dos procedimentos na fórmula da complexidade do programa abstrato SA, tem-se:

$C(SA) (n, n_{cad}, n_{L,} n_{aqc}) =$

$C(inicializa\text{-}sol) (n) + \{C(inicializa\text{-}temp) (n) +(n_{aqc} + 1) [C (quente) (n)] +$

$n_{aqc} [C (aleatório) (n) + C (aceitação) (n) +C (perturbação) (n) + C (aumenta) (n)] \} +$

$(n_{cad} + 1) [C (congelado) (n)] +n_{cad} \{C(inicializa\text{-}cad) (n) +(n_L + 1) [C (equilíbrio) (n)] +$

$n_L [C (aleatório) (n)+ C (aceitação) (n) +C (perturbação) (n)] +C(resfriamento) (n)\} +C(viabilização) (n);$

e,

$C(SA) (n, n_{cad}, n_{L,} n_{aqc}) =$

$C(inicializa\text{-}sol) (n) + \{C(inicializa\text{-}temp) (n) + (n_{aqc} + 1) [C (quente) (n)] +$

$n_{aqc} [C (aleatório) (n)] + C (aceitação) (n) +C (perturbação) (n)] +$

$n_{aqc} [C (aumenta) (n)] \} + (n_{cad} + 1) [C (congelado) (n)] +$

$n_{cad} \{C(inicializa\text{-}cad) (n) +(n_L + 1) [C (equilíbrio) (n)] +$

$(n_{aqc} \text{ x } n_L) [C (aleatório) (n)+ C (aceitação) (n) +C (perturbação) (n)] +$

$n_L [C(resfriamento) (n)]\} + C(viabilização) (n);$

e,

$C(SA) (n, n_{cad}, n_{L,} n_{aqc}) =$

$C(inicializa\text{-}sol) (n) + C(inicializa\text{-}temp) (n)$

$+ n_{cad} [C(inicializa\text{-}cad) (n)] + C(viabilização) (n) +$

$(n_{aqc} + 1) [C (quente) (n)] + (n_{cad} + 1) [C (congelado) (n)] +$

$(n_{cad} \text{ x } n_L) [C (equilíbrio) (n)]$

$(n_{aqc} + (n_{cad} \text{ x } n_L)) [C (aleatório) (n) + C (aceitação) (n) + C (perturbação) (n)];$

e, ainda, destacando as funções que atuam na construção ou modificação da solução,tem-se:

$C(SA)(n, n_{cad}, n_L, n_{aqc}) =$

$(n_L \times n_{cad})[C(\text{perturbação})(n)] + C(\text{inicializa-sol})(n) + C(\text{viabilização})(n) +$

$(n_L \times n_{cad})[C(\text{aleatório})(n) + C(\text{aceitação})(n)] + C(\text{inicializa-temp})(n) +$

$n_{cad}[C(\text{inicializa-cad})(n)] +$

$(n_{aqc} + 1)[C(\text{quente})(n)] + (n_{cad} + 1)[C(\text{congelado})(n)] +$

$(n_{cad} \times (n_L + 1))[C(\text{equilíbrio})(n)].$

Esta última expressão destaca a importância da função perturbação na avaliação da complexidade do programa abstrato SA. Entre os procedimentos que atuam na construção ou modificação de uma solução é o único que é executado ($n_L \times n_{cad}$) vezes, os outros são executados uma vez. Os procedimentos aleatório, aceitação e inicializa-temp, que também são executados ($n_L \times n_{cad}$) vezes, são usualmente de complexidade bem menor. As implementações dos predicados também são usualmente de complexidade menor.

Esta expressão mostra que a função perturbação e o número de soluções visitadas em uma execução é determinante da complexidade total. Como o número de soluções visitadas e a probabilidade de encontrar uma solução final de qualidade exigida pela aplicação interagem entre si, a implementação da função perturbação torna-se um fator crítico de complexidade. O esforço despendido numa codificação otimizada do procedimento que realiza esta função é recompensado com um desempenho melhor do algoritmo como um todo.

No entanto, esta é uma reflexão simplista, pois outros fatores estão envolvidos. A função perturbação é definida no projeto de uma estrutura de vizinhança e, esta estrutura de vizinhança determina o espaço de soluções possível de ser visitado. A escolha da função energia, uso de penalidades ou não, a escolha da prescrição de resfriamento são fatores que devem serem analisados em conjunto e que resultam em melhor ou pior desempenho do algoritmo.

Alguns aspectos simples podem ser considerados na escolha e codificação da função perturbação, como:

(a) estudar um espaço de soluções possíveis mais fácil de perturbar quando o espaço de soluções, constituídosomente por soluções viáveis, tornar difícil a perturbação de uma solução viável para outra (problemas do conjunto independente e coloração de grafos [AAR 89].

(b) avaliar, considerando as estruturas de vizinhanças que possam ser projetadas para o problema, os impactos entre complexidade da perturbação e tamanho da vizinhança, que influencia em n_L.

(c) implementar a perturbação de uma forma otimizada. Como exemplo, aproveitando o estudo de caso da seção 3.6, onde na perturbação é avaliada a diferença de energias entre solução corrente e candidata, aplicado o critério de metropolis e somente quando a solução candidata é aceita que se altera a solução (seq. 194-204).

(d) otimizar o cálculo da função energia.

Os parâmetros n_L e n_{cad} são decorrentes da prescrição de resfriamento escolhida, ou seja, resumindo do comportamento do parâmetro temperatura. A escolha da prescrição de resfriamento é, na maioria das vezes, feita de forma empírica. Como colocado por Catoni (cap. 3, [AZE 92]) o estudo teórico dos aspectos relacionados com prescrição de resfriamento para uma dada aplicação e uma dada disponibilidade de recursos deve ser encorajado, uma vez que não existe fórmula fechada para determinar esta prescrição e se está longe de encontrar uma prescrição de resfriamento genérica milagrosa. Atualmente, reconhecendo a dificuldade de determinar este tipo de parâmetro, estudos [MCA 97] [GEH 99] têm sido realizados, com maior profundidade teórica, para determinar propriedades que permitam uma melhor avaliação para estes parâmetros.

Portanto, a complexidade total de um algoritmo SA é fortemente dependente da instância do problema, da escolha da estrutura de vizinhança e da função energia usada para avaliar a solução. A estrutura de vizinhança e a função energia determinam o panorama de energia e influenciam na escolha da seqüência adequada para o parâmetro temperatura. O comportamento do algoritmo em termos de tempo computacional pode variar de uma execução para outra para uma mesma instância dependendo dos valores atribuídos aos diversos parâmetros. A escolha de valores adequados ao panorama de energia sendo tratado em conjunto com os predicados que determinam os critérios de parada vão determinar um tempo computacional aceitável ou não. Uma expectativa melhor com relação à qualidade da solução fornecida é também decorrência da escolha adequada destes valores.

Na avaliação da complexidade expressa pelas equações acima foram considerados como critérios de parada os limites máximo de iterações total e para cada procedimento iterativo. Isto implica em um limite de complexidade de tempo. Como o SA é um algoritmo aleatório o tempo de execução pode ser diferente para uma mesma instância quando critérios mais elaborados de término forem utilizados. Assim, no capítulo 4 são apresentados os aspectos relacionados com a convergência assintótica e complexidade [AAR 89] baseado no SA modelado como uma seqüência de cadeias de Markov, considerando conceitos da Física Estatística e para uma prescrição de resfriamento determinada.

Estudos de complexidade específicos mais atuais são citados em:

- Sasaki e Hajek [SAS 88] - analisam a complexidade de tempo média para o problema de cobertura máxima de grafos para o SA tradicional e uma variação onde a temperatura é uma constante arbitrária. Para o SA tradicional não encontraram nenhuma alternativa que resultasse em tempo médio polinomial. Para a variação do SA mostraram a possibilidade de encontrar coberturas máximas aproximadas em tempo polinomial.

- Nurmela [NUR 93] – apresenta um estudo de complexidade em espaço para o problema de cobertura.

- Jerrum e Sinclair [JER 97] - analisam os algoritmos do tipo Monte Carlo Markov Chain de uma forma geral, incluindo o algoritmo de Metropolis (a uma temperatura constante), modelado como uma cadeia de Markov irredutível, aperiódica, ergódica e satisfazendo a equação detalhada de balanceamento. Analisam com mais profundidade a aplicação do algoritmo para o problema de encontrar a cobertura máxima de grafos. Quanto ao SA (temperatura variando segundo uma prescrição de resfriamento)

observam que o mesmo pode ser analisado como uma cadeia de Markov não homogênea, salientando que mesmo a questão da convergência assintótica não é trivial;

- Rosenthal e outros [ROS 99] [ROB 97] – apresentam aspectos práticos e teóricos relacionados com a convergência dos algoritmos tipo MCMC, ampliando a abrangência para cadeias de Markov em espaços de soluções gerais (não só espaços finitos ou contáveis).

3.6 Estudo de Caso: "Algorithm of the Gods"

> "So keep your eyes open! Perhaps, while contemplating ripples of turbulence in cigarette smoke, investigating how individual bees organize themselves into a hive or studying the simultaneous turns of a flock of birds, you will discover the next major innovation in computer problem solving".
>
> *Shawn Carlson*

Neste anexo é apresentado um estudo de caso de um programa "Algorithm of The Gods" (PAG), desenvolvido por Carlson [CAR 98] como um programa exemplo, para o problema do caixeiro-viajante e disponível na WWW nas páginas da Society for Amateur Scientist. O código deste programa PAG é analisado em relação ao programa abstrato SA apresentado na seção 3.5. A aplicação em que Carlson utilizou Simulated Annealing [CAR 99] foi minimizar a movimentação de um telescópio quando examinando galáxias, ou seja, buscar uma ordenação das galáxias a serem observadas com um mínimo de movimentação do telescópio. Esta aplicação é semelhante ao problema do caixeiro-viajante.

3.6.1 Informações Gerais

O código PAG, desenvolvido em C (código na seção 3.6.4), trata uma instância do problema do caixeiro-viajante, considerando as cidades como pontos distribuídos em um plano e identificados por suas coordenadas cartesianas. A solução é uma rota, isto é, um caminho passando uma vez em cada cidade e iniciando e terminando numa mesma cidade. A energia entre duas cidades, segmento da rota, é a distância entre os pontos que representam as mesmas no plano. A energia da solução é a soma das energias destes segmentos da rota. No problema do caixeiro-viajante clássico a cidade início e fim da rota é usualmente fixa. O código PAG considera que esta cidade pode variar de uma solução para outra, mas é fornecida a modificação para que a cidade seja fixa. É fornecida também a modificação necessária para transformar o código numa busca local simples.

O código PAG está estruturado de forma a permitir modificações referentes à instância, como estrutura de dados e função.

3.6.2 Análise do Código

Estrutura

Os procedimentos de inicialização estão agrupados através de um procedimento setup (seq.113). Os procedimentos de melhoramento estão agrupados no for (seq. 58-78). O código inclui procedimentos auxiliares para saída de resultados.

O código PAG está estruturado de forma a permitir alterações em diversos aspectos como na estrutura de dados, no número de elementos que constituem a solução, na forma de avaliar a função energia. Estas modificações não devem alterar as características de funcionalidade do código. Neste aspecto, a observação dos axiomas é um auxílio importante para a garantia de correção.

A função perturbação usa dois tipos de estruturas de vizinhança e a escolha de uma delas é aleatória (seq. 242) com mesma probabilidade.

Parâmetros, Variáveis, Funções e Predicados

As TABELAS 3.2, 3.3 e 3.4 relacionam variáveis, funções e predicados definidos no programa abstrato SA (seção 3.5) e as definidas no código PAG, e tem como objetivo facilitar o entendimento da similaridade entre o programa abstrato e a implementação do algoritmo no código PAG. Como as funcionalidades não estão distribuídas da mesma forma nos diversos procedimentos não há uma equivalência entre os mesmos e as tabelas auxiliam na identificação destas funcionalidades.

TABELA 3.2 – Predicados

Programa Abstrato SA	**Algorithm of the God (PAG)**
possível (p,s)	"s é um circuito que passa exatamente uma vez em cada um dos NUMLISTITEMS pontos"
viável (p,s)	= possível (p,s)
congelado (p, t)	= (i ≥ numListItems) ∨ (numSuc = 0), i = cadeia (p,t) (seq. 58); (seq.75)
indistinguível'	= (numSuc ≤ runLimit) (seq. 70-73) seq. (205- 213)
quente (p, t)	t = energy/numListItems (seq. 122)
equilíbrio (p, s)	(i ≥ runLimit) ∨ (numSuc = sucLimit), i =experimento(p, t) (seq. 123-124) (seq. 193); (seq. 216)
estável (p, s_0, t, s)	for (seq. 123-221)
estável$^+$ (p, s_0, s')	for (seq. 58-78)
crit-metropolis	Edif < 0.0 || exp(-Edif/temperature) > (double)rand()/RAND_MAX) (seq. 230)

Entrada: Instância do Problema

A entrada para o programa constitui uma instância p do problema a ser resolvida.

Relacionando o que foi especificado nas seções 3.5.2 e 3.5.3 com o codificado neste programa PAG, tem-se:

- Estrutura de dados p_d: o programa considera que os dados podem ser armazenados em uma estrutura de lista das coordenadas cartesianas [ListElement (seq. 16)].
- Função energia f_E: A avaliação da função energia é estabelecida pelo código através dos procedimentos [GetEnergy (seq. 165), que calcula a energia da solução a partir da lista correspondente, e [EnergyDif (seq. 296], que calcula a diferença de energia entre a solução corrente e a candidata.

TABELA 3.3 – Parâmetros eVariáveis

Descrição	**Prog. SA**	**Algorithm of the God (PAG)**
Número máximo de experimentos na fase de aquecimento	n_{aqc}	-
Número máximo de resfriamentos	n_{cad}	numListItems (seq. 58)
Número máximo de perturbações em cada cadeia	n_L	runLimit (seq. 37) (seq.123)
Número máximo de perturbações com sucesso em cada cadeia	-	sucLimit (seq. 37) (seq. 124)
Número de elementos que compõe uma solução	-	numListItem (seq. 12) (seq. 46)
Fator de ajuste da temperatura	-	TEMP_FACTOR (seq. 7)
Parâmetro de controle temperatura	t	temperature (seq. 39) (seq. 77) (seq. 122)
Estrutura de armazenamento da instância	p_d	ListElement (seq.16) (seq.141)
Estrutura de armazenamento da solução	s_{est}	list (seq.36) (seq. 136)
Estrutura de armazenamento da melhor solução	s	bestList (seq. 30) (seq.138)
Armazenamento da energia da solução	c	ener (seq. 44)
Armazenamento da melhor energia solução	-	BESTENERGY (seq.29)
Armazenamento da diferença de energia entre as soluções	-	Edif (seq. 186)

- Função viabilização f_{vav}: o programa trata soluções viáveis, assim esta função não é definida, ou seja, toda solução possível é viável.
- Predicados π_p, π: estes predicados não são definidos explicitamente, mas π é assegurado pelas funções de construção das soluções.
- Parâmetro n_{aqc}: como é atribuído uma valor inicial diretamente a temperatura, este parâmetro não é especificado.
- Parâmetro n_{cad}: estabelecido através de um comando for (seq. 58).

- Parâmetro n_L: runLimit, estabelecido no procedimento setup (seq. 123).
- Parâmetro t_{ini}: é atribuído uma valor inicial diretamente a temperatura (temperature), no procedimento setup (seq. 122), em função da energia da solução inicial e do valor definido para NUMLISTITEMS.
- Predicado *indistínguivel*: não é estabelecido.
- Predicado *indistínguivel'*: estabelecido através de um fator de perturbações com sucesso (numSuc) com relação ao número limite de iterações (runLimit) (seq. 70-72); quando este fator é satisfeito passa a ser armazenado o melhor valor de energia encontrado.
- O programa estabelece um parâmetro de número de perturbações com sucesso sucLimit, no procedimento setup (seq. 124).
- O programa estabelece um parâmetro NUMLISTITEMS, (seq. 12), que define o número de coordenadas da instância (tamanho da instância) e é usado para estabelecer valores iniciais para outros parâmetros como temperatura, numSuc e runLimit.
- O programa estabelece um parâmetro TEMP_FACTOR, (seq. 7), usado para diminuir a temperatura.

TABELA 3.4 – Funções

Programa Abstrato SA	**Algorithm of the God (PAG)**
SA	(main)
inicializa-sol	[setup (seq. 113)]; [CreateInitialList (seq. 129)]; [InitializeElement (seq. 152)]; [GetEnergy (seq. 165)]
aquecimento	[setup (seq. 113)]; (seq. 122)
metropolis	[Anneal (seq. 179)]
resfriamento	[Anneal (seq. 179)]; (seq. 77)
viabilização	-
inicializa-cadeia	(seq. 192)
aleatório (metropolis)	(seq. 230)
perturbação (metropolis)	[GetSegment (seq. 250]; [PickAlteration (seq. 240)]; [EnergyDif (seq. 296] [trans_cost (seq. 304)] [reverse_cost (seq. 335)] [AlterList (seq. 371]
aceitação (metropolis)	[Oracle (seq. 226];
inicializa-temp	[setup (seq. 113)]; (seq. 122)

Como foi observado na seção 3.5.2 a entrada especificada para o programa abstrato é sujeita a modificações para permitir diferentes implementações. Observe que entrada não é entendida apenas como sendo feita pela passagem de parâmetros, mas pode ser também implementada por função de inicialização.

3.6.3 Verificação do Axiomas

Nesta seção é analisado o código do programa com relação aos axiomas estabelecidos para o programa abstrato SA (TABELA 3.5) e metropolis (TABELA 3.6), considerando as TABELAs 3.2, 3.3, 3.4. O programa abstrato aquecimento não é aplicável como um procedimento isolado, portanto, foram analisados somente os axiomas SAAX03 e SAAX04 de SA.

Axiomas do Programa Abstrato SA

A TABELA 3.5 relaciona os axiomas definidos para o programa abstrato SA (seção 3.5.6) para facilitar a verificação dos mesmos.

Suponha a asserção de entrada como satisfeita φ (p), considerando as observações com relação a entrada (seção 3.6.3).

SAAX01, SAAX02

Verifica-se pela análise que o código PAG trata soluções viáveis, conforme identificado pelos predicados possível e viável (TABELA 3.3). A função de inicialização da solução (inicializa-sol) é codificada nas seq. 120-121 (129-175). As estruturas ListElement e list são inicializadas com numListItems elementos, conforme seq. 146-147. Estas estruturas são modificadas nas seq. 202-203 (371-414), permanecendo com numListItems elementos, conforme pode ser observado. Portanto, SAAX01 e SAAX02 são satisfeitos.

SAAX03, SAAX04

Suponha possível (p,s), isto é, list aponta para os numListItems da estrutura ListElement. Suponha o aquecimento implementado conforme seq. 122. Observa-se que a temperatura é calculada considerando a energia da solução inicial dividida por numListItems, o que estabelece uma interpretação para quente. Portanto, SAAX04 é satisfeito. Verificando-se o código relativo a avaliação de energia [seq. 165-174 (357-368) (14)], observa-se que é um valor maior ou igual a zero e considerando a seq. 122, SAX03 é satisfeito.

TABELA 3.5 – Axiomas do Programa SA

Axioma	**Descrição**
SAAX01	(∀p) (∀s) viável (p,s) ⇒ possível (p, s)
SAAX02	(∀p) [possível (p, (inicializa-sol (p)))]
SAAX03	(∀p) (∀s) [possível (p, s) ⇒ (aquecimento (p, s)) > 0)]
SAAX04	(∀p) (∀s) [possível (p, s) ⇒ quente (p, aquecimento (p, s))]
SAAX05	(∀p) (∀t) [t > 0 ⇒ (resfriamento (p,t) < t) ∧ (resfriamento (p,t) ≥ 0)]
SAAX06	(∀p) (∀s) [viável (p, s) ⇒ viabilização (p, s) = s]
SAAX07	(∀p) (∀s) [¬ viável (p, s) ⇒ viabilização (p, s) = f_{vav} (p, s)]
SAAX08	(∀p) (∀s) (∀s_0) [estavel$^+$ (p, s_0, s) ⇒ estavel$^+$ (p, s_0, viabilização (p, s))]
SAAX09	(∀p) (∀t) [(congelado (p, t)) ⇔ condição-congelado] Uma interpretação usual é (condição-congelado = ((cadeia (p, t) ≥ n_{cad})) ∨ (t ≤ 0)). Outra interpretação pode ser condição-congelado = ((cadeia (p, t) ≥ n_{cad})) ∨ (t ≤ 0) ∨ (indistinguível$'$ (h)). Os axiomas SAAX10 e SAAX11 suportam a condição (cadeia (p, t) ≥ n_{cad}).
SAAX10	(∀p) (∀t) [(cadeia (p, aquecimento (p, t))) = 0]
SAAX11	(∀p) (∀t) [(cadeia (p, resfriamento (p, t))) = cadeia (p, t) + 1]
SAAX12	(∀p) (∀s) (∀t) [possível (p,s) ⇒ (estável (p, s, t, metropolis (p, s ,t)) ∧ possível (metropolis (p, s, t)))]
SAAX13	(∀p) (∀t) (∀s) (∀s') [estável (p, s, t, s') ⇐ ∃ (s_1, s_2, ..., s_n) ((s_1 = s) ∧ (s_{i+1} = metropolis (p, s_i ,t), para i =1,2, ..., n) ∧ (s_n = s') ∧ ((∃ (n'' < n) ∧ (n' > n'')) ⇒ indistinguível (s', $s_{n''}$)))]
SAAX14	(∀p) (∀s_0) (∀s) [estável$^+$ (p, s_0, s) ⇐ (∃ (s_0, s_1, ..., s_{k+1}) ∃ (t_0, t_1, ..., t_{k+1}) ($\wedge^{k}_{i=0}$ (possível (p, s_{i+1}) ∧ possível (p, s_i) ∧ estável (p, s_i, t_i, s_{i+1})) ∧ (s_{k+1} = s)) ∧ (congelado (p, s , t_{k+1}))]

SAAX05

Suponha TEMP_Factor > 0, t_1 > 0, pela atribuição inicial [seq. 122] e i = 1,2, ... , (numListItems –1) (seq. 58). Tem-se t_{1+1} > 0, pelo código da seq. 77. Portanto, SAAX05 é satisfeito.

SAAX06, SAAX07 e SAAX08

Suponha o axioma SAAX01 satisfeito. Tem-se viável (p,s) e viabilização (p, s) = s, portanto, os axiomas SAAX06, SAAX07 e SAAX08 são satisfeitos.

SAAX09, SAAX10, SAAX11

Estes axiomas definem o congelado. Genericamente o axioma SAAX09 define congelado usando uma condição-congelado, que pode ser interpretada de diferentes formas. No programa PAG a condição de congelado está codificada nas seq. 58 e seq. 75. Esta condição corresponde a interpretação condição-congelado = ((cadeia (p, t) $\geq n_{cad}$)) $\vee$ (indistinguível, (h)), com i(seq. 58) sendo a interpretação de cadeia (p, t). Pela análise da linha 58, observa-se que os axiomas SAAX10 e SAAX11 são satisfeitos. A interpretação de (indistinguível, (h)) é dada pelas (seq. 70-75) (seq. 216) e seq. (205-213).

SAAX12

Estes axiomas caracterizam o procedimento metropolis do programa abstrato SA. As funcionalidades do procedimento metropolis estão codificadas no procedimento Anneal (seq. 179-221) no programa PAG. A estabilidade é caracterizada no programa PAG por um número limite de perturbações (seq.193) ou um número limite de perturbações com sucesso(seq. 216). O axioma SAAX12 define que, para qualquer implementação de metropolis, a solução fornecida por este procedimento deve satisfazer possível (p, metropolis (p, s ,t)) e estável (p, s, t, metropolis (p, s ,t)) deve ser satisfeito. O predicado possível é analisado no axioma MEAX05. Uma interpretação para estável é dada através dos axiomas MEAX08 e MEAX09. Esta interpretação é analisada abaixo e considerada satisfeita.

SAAX13

Este axioma é uma interpretação alternativa não considerada no código PAG.

SAAX14

Este axioma estabelece uma interpretação para o predicado estável^{+}. Considerando a (seq.58-75) e os axiomas SAAX02 e SAAX12 tem-se [$\exists$ (s_0, s_1, ..., s_{k+-1})], considerando (seq. 77) e os axiomas SAAX03 e SAAX04 tem-se $\exists$ (t_0, t_1, ..., t_{k+1}), e, considerando o axioma SAAX12, tem-se (possível (p, s_{i+1}) $\wedge$ possível (p, s_i) $\wedge$ estável (p, s_i, t_i, s_{i+1}). Pelo axioma SAAX09, tem-se (congelado (p,s, t_{k+1}). Assim, o predicado SAAX14 é satisfeito.

Axiomas do Programa Abstrato metropolis

As funcionalidades do programa abstrato metropolis estão codificadas no procedimento Anneal e procedimentos executados a partir de Anneal.

A TABELA 3.6 relaciona os axiomas definidos para o programa abstrato metropolis definidos na seção 3.5.6.

Suponha a asserção de entrada satisfeita, considerando as observações acima sobre os axiomas SAAX01 a SAAX14 e asserção de entrada para SA satisfeita. A passagem de parâmetros (seq. 193) corresponde a passagem de parâmetros do programa abstrato SA quando executa metropolis.

MEAX01

A função correspondente a inicializa-cadeia estão codificadas nas (seq. 123-124) e (seq. 192-193). Pela asserção de entrada e considerando que não ocorre modificação na solução o axioma é sempre satisfeito.

MEAX02, MEAX03 e MEAX04

Estes axiomas definem o predicado equilíbrio. A interpretação para condição-equilíbrio é $(i \geq runlimit) \vee (numSuc = sucLimit)$, com i = experimento (p,s), conforme observa-se nas (seq. 123-124), (seq. 193) e (seq. 216). Portanto, os axiomas são satisfeitos.

MEAX05

Uma perturbação do programa abstrato corresponde às execuções do procedimentos chamados nas (seq. 194-197), (seq. 202), ou seja, GetSegment (seq. 248-294), PickAlteration (seq. 237-245), EnergyDiff (seq. 296-254) e AlterList (seq. 371-414). O código trata dois tipos de estruturas de vizinhança e, de forma aleatória, é escolhida uma ou forma de perturbação. Observa-se que sempre que o código constrói uma nova solução é mantido o mesmo número de elementos. Portanto, o axioma é satisfeito.

TABELA 3.6 – Axiomas do Programa metropolis

Axioma	**Descrição**
MEAX01	(∀p) (∀s) [possível (p, s) ⇒ possível (inicializa-cadeia (p, s))]
MEAX02	(∀p) (∀s) [(equilíbrio (p, s) ⇔ (condição-equilíbrio)] A interpretação para condição-equilíbrio = (experimento (p, s) ≥ L) é considerada para os axiomas MEAX03 e MEAX04.
MEAX03	(∀p) (∀s) (∀t) [(experimento (p, inicializa- cadeia (p, s))) = 0]
MEAX04	(∀p) (∀s) (∀t) [(experimento (p, perturbação (p, s))) = experimento (p, s) + 1]
MEAX05	(∀p) (∀s) [possível (p, s) ⇒ possível (p, perturbação (p,s))]
MEAX06	[0 ≤ aleatório () < 1]
MEAX07	(∀p) (∀s) (∀t) [(possível (p, s) ∧ possível (p, perturbação (p, s))) ⇒ ((crit-metropolis (s, perturbação (p, s), t, aleatório())) ⇒ (aceitação(s, perturbação (p, s), t, aleatório()) = perturbação (p, s))) ∧ ((¬ crit-metropolis (s, perturbação (p, s), t, aleatório()) ⇒ (aceitação (s, perturbação (p, s), t, aleatório()) = s))]
MEAX08	(∀s) (∀s') (∀s'') (∀t) (∀n_r) [(aceita (s, s', s'', t, n) ⇔ (s = s'' ∧ crit-metropolis (s', s'', t, n)) ∨ (s = s' ∧ ¬ crit-metropolis (s', s'', t, n))]
MEAX09	(∀p) (∀s) (∀s') (∀t) (∀n_r) [experimento (p, aceitação(s, s', t n_r)) = experimento (p, s)]
MEAX10:	(∀p) (∀s) (∀t) (∀s') [estável (p, s, t, s') ⇔ (∃ (n_1, s_{c1}, s_1) ... ∃ (n_k, s_{ck}, s_k)) (aceita (s_1, s_0, s_{c1}, t, n_1) ∧ aceita (s_2, s_1, s_{c2}, t, n_2) ∧ ... aceita (s_k, s_{k-1}, s_{ck}, t, n_k) ∧ (s_k = s') ∧ (s_0 = s)]

MEAX06

A geração de um número aleatório (seq. 230) é executado por um gerador de números pseudo-aleatório que atende o axioma.

MEAX07, MEAX08

A codificação de aceitação (seq. 201-202), que executa o procedimento Oracle (seq. 226-234), implementa o critério de metropolis (seq. 230), assim MEAX07 e MEAX08 são satisfeitos.

MEAX09

A codificação de aceitação não altera i (= experimento), portanto, MEAX09 é satisfeito.

MEAX10

Este axioma define uma interpretação para estável. Analisando os códigos implementados para (seq. 58-75), metropolis (seq. 194-221) e perturbação [GetSegment (seq. 248-294), PickAlteration (seq. 237-245), EnergyDiff (seq. 296-254) e AlterList (seq. 371-414)], tem-se [$\exists$ (n_1, s_{c1}, s_1) ... $\exists$ (n_k, s_{ck}, s_k)) (aceita (s_1, s_0, s_{c1}, t, n_1) $\wedge$ aceita (s_2, s_1, s_{c2}, t, n_2) $\wedge$ ... aceita (s_k, s_{k-1}, s_{ck}, t, n_k) $\wedge$ (s_k = s') $\wedge$ (s_0 = s)], onde $s_{c(i+1)}$ = perturbação (p, s_i) e $s_{(i+1)}$ = aceitação (p, t, s_i) e, portanto, considerando MEAX08, MEAX10 é satisfeito.

3.6.4 Código do Programa

Observação Importante: O código apresentado a seguir está disponível na WWW [CAR 98]. Para qualquer uso consultar a página referenciada na bibliografia.

```
1.      #include <stdio.h
2.      #include <math.h
3.      #include <stdlib.h
4.      #include <string.h
5.      #include <time.h
6.
7.      #define TEMP_FACTOR 0.9
8.      #define YES  1
9.      #define NO    0
10.     #define REVERSE             1
11.     #define TRANSPOSE           0
12.     #define NUMLISTITEMS 200          /* NUMBER OF ITEMS ON THE LIST.
13.             CHANGE THIS NUMBER TO SUIT YOUR APPLICATION.*/
14.     #define ALEN(a,b,c,d) sqrt(((b)-(a))*((b)-(a))+((d)-(c))*((d)-(c)))
15.
16.     struct ListElement
17.     {
18.     /* DEFINE YOUR DATA STRUCTURE HERE. WE'VE ASSUMED
19.     SOMETHING LISTED IN CARTESIAN COORDINATES.  IF YOU'RE DATA
20.     IS NOT IN THESE COORDINATES, YOU WILL NEED TO MODIFY THIS
21.     STRUCTURE AND ALSO MAKE THE NECESSARY CHANGES IN THE
22.     ENERGY FUNCTION. */
23.
24.             float x;
25.             float y;
26.     };
27.
28.     int CHECKBEST = NO;
29.     float BESTENERGY;
30.     struct ListElement **bestList;
31.
32.
33.
34.
35.     main()
36.     {       struct ListElement **list;
37.             int runLimit, sucLimit, numListItems, i, numSuc;
38.             int Anneal();
```

```
39.                 float temperature, energy;
40.                 struct ListElement **setup();
41.                 FILE *outFile, *energyFile;
42.                 void printList();
43.
44.                 float GetEnergy(), ener;
45.
46.                 numListItems = NUMLISTITEMS;
47.
48.                 list = setup(&energy, &temperature, &runLimit, &sucLimit,
49.                         numListItems);
50.
51.                 outFile = fopen("InitList.dat","w");
52.                 printList(outFile,list, numListItems);
53.                 fclose(outFile);
54.
55.                 energyFile = fopen("energy.dat", "w");
56.                 fprintf(energyFile,"Temp\tEnergy\tnumSuc\n");
57.
58.                 for(i = 1; i < numListItems; i++)
59.                 {
60.                         numSuc = Anneal(list, numListItems, runLimit,
61.                                 temperature,sucLimit, numListItems);
62.                         ener = GetEnergy(list,numListItems);
63.
64.                         printf("%f\t%f\t%d\n",temperature,ener,numSuc);
65.
        fprintf(energyFile,"%f\t%f\t%d\n",temperature,ener,numSuc);
66.
67.
68.                                 /* Start checking for the best solution?*/
69.
70.                         if(numSuc <= runLimit*0.01 && CHECKBEST == NO) {
71.                                 CHECKBEST = YES;
72.                                 BESTENERGY = ener;
73.                         }
74.
75.                         if(numSuc == 0) break;
76.
77.                         temperature *= TEMP_FACTOR;
78.                 }
79.
80.                 fclose(energyFile);
81.
82.                 outFile = fopen("FinalList.dat","w");
83.                 printList(outFile,list, numListItems);
84.                 fclose(outFile);
```

```
85.
86.                     outFile = fopen("BestList.dat","w");
87.                     printList(outFile,bestList,numListItems);
88.                     fclose(outFile);
89.
90.                     outFile = fopen("lowestEnergy.dat","w");
91.                     fprintf(outFile,"Lowest Energy = %f\n",BESTENERGY); /*
92.                     store energy. */
93.                     fclose(outFile);
94.
95.
96.             }
97.
98.
99.             void printList(outFile,list, numListItems)
100.            FILE *outFile;
101.            struct ListElement **list;
102.            int numListItems;
103.            {
104.                    int i;
105.
106.                    for(i=0; i< numListItems; i++)
107.                            fprintf(outFile,"%f\t%f\n",list[i]-x,list[i]-y);
108.
109.
110.            }
111.
112.
 struct ListElement **setup(energy, temperature, runLimit, sucLimit,
114.                                                    numListItems)
115.            int *runLimit, *sucLimit, numListItems;
116.            float *temperature, *energy;
117.            {       struct ListElement **CreateInitialList(), **list;
118.                    float GetEnergy();
119.
120.                    list = CreateInitialList(numListItems);
121.                    *energy = GetEnergy(list, numListItems);
122.                    *temperature = *energy/numListItems;
123.                    *runLimit = 100 * numListItems;
124.                    *sucLimit = 10 * numListItems;
125.
126.                    return list;
127.            }
128.
129.            struct ListElement **CreateInitialList(numListItems)
130.            int numListItems;
131.            {       int i;
```

```
132.                struct ListElement **list;
133.                void InitializeElement();
134.
135.                        /* Allocate memory for the array of pointers. */
136.               list = (struct ListElement **)malloc(sizeof(struct ListElement *) *
137.                numListItems);
138.                bestList = (struct ListElement **)malloc(sizeof(struct ListElement *) *
139.                numListItems);
140.
141.                /* Create the data structures and install pointers to them into the list. */
142.                for(i = 0; i < numListItems; i++)
143.                        list[i] = (struct ListElement *)malloc(sizeof(struct
144.                                ListElement));
145.
146.                for(i = 0; i < numListItems; i++)                /* Set initial values.
*/
147.                        InitializeElement(list[i]);
148.
149.                return list;
150.        }
151.
152.        void InitializeElement(element)
153.        struct ListElement *element;
154.        {
155.        /* NOTE: YOU WILL HAVE TO CHANGE THIS FUNCTION. THIS
156.        FUNCTION SETS THE INITIAL VALUES IN THE DATA STRUCTURE.
    HERE WE JUST SET THE COORDINATES TO NUMBERS BETWEEN
158.        ZERO AND 100. */
159.
160.                element-x = (float)rand()/RAND_MAX * 100;
161.                element-y = (float)rand()/RAND_MAX * 100;
162.        }
163.
164.
165.        float GetEnergy(list, numListItems)
166.        struct ListElement **list;
167.        int numListItems;
168.        {           int i;
169.                float ener, energy();
170.
171.                for(ener = 0, i = 0; i < numListItems - 1; i++)
172.                        ener += energy(list, i, i+1);
173.
174.                return ener;
175.        }
176.
177.        /* THIS FUNCTION CARRIES OUT THE ANNEALING PROCEDURE.  */
```

```
178.
179.    int Anneal(list, listSize, runLimit, temperature, sucLimit, numListItems)
180.    struct ListElement **list;
181.    int listSize, runLimit, sucLimit, numListItems;
182.    float temperature;
183.    {
184.            int numSuc, start, end, insertPoint, alteration, answer, i, j;
185.            int Oracle(), PickAlteration();
186.            float Ediff, EnergyDif();
187.            void GetSegment(), AlterList();
188.            FILE *bestFile;
189.
190.            float GetEnergy(), ener;
191.
192.            numSuc = 0;
193.            for(i = 0; i < runLimit; i++){
194.                    GetSegment(&start, &end, &insertPoint, numListItems);
195.
196.                    alteration = PickAlteration();
197.                    Ediff = EnergyDif(list, start, end, insertPoint, numListItems,
198.                                    alteration);
199.
200.
201.                    if(Oracle(Ediff, temperature) == YES){
202.                            AlterList(list,start,end, insertPoint,
203.                                    alteration, numListItems);
204.                            numSuc++;
205.                            if(CHECKBEST == YES) {
206.                            ener = GetEnergy(list,numListItems);
207.                            if(ener < BESTENERGY)
208.                                    {               /* Save best list. */
209.                                    BESTENERGY = ener;
210.                                    for(j = 0; j < numListItems;
j++) 211.                                           bestList[j] = list[j];
212.                            }
213.                    }
214.            }
215.
216.                    if(numSuc = sucLimit) break;
217.
218.            }
219.
220.            return numSuc;
221.    }
222.
223.
224.    /* THIS FUNCTION DECIDES IF A NEW PATH SHOULD BE KEPT. */
```

```
225.
226.        int Oracle(Edif, temperature)
227.        float Edif, temperature;
228.        {
229.
230.                if(Edif < 0.0 || exp(-Edif/temperature) > (double)rand()/RAND_MAX)
231.                                return YES;
232.
233.                return NO;
234.        }
235.
236.
 /* THIS FUNCTION DECIDES WHETHER TO TRY REVERSING THE
238.        PATH OR TRANSPOSING IT.  */
239.
240.        int PickAlteration(void)
241.        {
242.                if((double)rand()/RAND_MAX < 0.5) return REVERSE;
243.
244.                return TRANSPOSE;
245.        }
246.
247.
 /* THIS FUNCTION SELECTS A SEGMENT FOR POSSIBLE
249.                                ALTERATION. */
250.        void GetSegment(start, end, insertPoint, listSize)
251.        int *start, *end, *insertPoint, listSize;
252.        {       int shuffle[3], done = NO, i, j, hold;
253.
254.                do{                     /* Select three different numbers between 0 and
255.                        listSize - 1 */
256.                        for(i = 0; i <3; i++)
257.                          shuffle[i] =
258.                                (int)(((float)rand()/RAND_MAX)*(listSize - 1));
259.                          if(shuffle[0] == shuffle[1] ||
260.                                /* Make sure no two are the same. */
261.                                   shuffle[0] == shuffle[2] ||
262.                                   shuffle[1] == shuffle[2]) ;
263.                          else done = YES;
264.
265.                }while(done == NO);
266.
267.
268.                /* Sort these numbers. */
269.
270.                for(i = 0; i < 3; i++){
271.                        for(j = i + 1; j < 3; j++){
```

```
272.                                    if(shuffle[i]  shuffle[j]){
273.                                            hold = shuffle[i];
274.                                            shuffle[i] = shuffle[j];
275.                                            shuffle[j] = hold;
276.                                    }
277.                            }
278.            }
279.
280.    /* Decide whether to set the insertPoint above or below the block to be altered. */
282.
283.            if((double)rand()/RAND_MAX < 0.5 || shuffle[2] - 1 == shuffle[1]){
284.                    *insertPoint = shuffle[0];
285.                    *start = shuffle[1];
286.                    *end = shuffle[2];
287.            }
288.            else{
289.                    *start = shuffle[0];
290.                    *end  = shuffle[1];
291.                    *insertPoint = shuffle[2];
292.            }
293.    }
294.
295.
296.    float EnergyDif(list, start, end, insertPoint, listSize, alteration)
297.    struct ListElement **list;
298.    int start, end, insertPoint, alteration;
299.    {
300.            float trans_cost(), reverse_cost();
301.
302.            switch(alteration){
303.                    case TRANSPOSE:
304.                    return trans_cost(list,start,end,insertPoint,listSize);
305.
306.                    case REVERSE:
307.                    return  reverse_cost(list,start,end,listSize);
308.            }
309.
310.    }
311.
312.
313.    float trans_cost(list,start,end,insertPoint,listSize)
314.    struct ListElement **list;
315.    int start, end, insertPoint, listSize;
316.    {       float cost, energy();
317.
318.            cost = 0;
319.                                            /* Break the current connections. */
```

```
320.
    if(start  0) cost -= energy(list, start-1, start);
322.        if(end < listSize - 1) cost -= energy(list, end, end+1);
323         if(insertPoint > 0) cost -= energy(list,insertPoint, insertPoint -1);
324.
325.                                                 /* Build the new connections. */
326.
327.                 cost += energy(list, insertPoint , end);
328.                 if(insertPoint - 1 = 0 ) cost += energy(list,insertPoint - 1, start);
329.                 if(start > 0 && end < listSize - 1 )
330.                             cost += energy(list, start - 1, end + 1);
331.
332.                 return cost/listSize;
333.        }
334.
335.        float reverse_cost(list,start,end,listSize)
336.        struct ListElement **list;
337.        int start, end, listSize;
338.        {           float cost, energy();
339.
340.                 cost = 0;
341.
342.                                                 /*Subtract the old cost. */
343.
344.                 if(start > 0) cost -= energy(list, start - 1, start);
345.                 if(end < listSize- 1) cost -= energy(list,end,end+1);
346.
347.                                                 /* Add the new cost.  */
348.
349.                if(start > 0) cost += energy(list,start - 1, end);
350.                 if(end < listSize-1) cost += energy(list,start,end+1);
351.
352.                 return cost/listSize;
353.
354.        }
355.
356.
357.        float energy(list, v1, v2)
358.        struct ListElement **list;
359.        int v1, v2;
360.        {
361.                 /* YOU MUST CHANGE THIS FUNCTION TO CALCULATE THE
362.                 THE "ENERGY" BETWEEN TWO ELEMENTS IN YOUR LIST.
363.                 HERE WE JUST CALCULATE THE PATH LENGTH BETWEEN
364.                 TWO POINTS ON A PLANE */
365.
366.                 return  (float)ALEN(list[v1]-x,list[v2]-x,list[v1]-y,list[v2]-y);
```

```
367.
368.    }
369.
370.
371.    void AlterList(list, start, end, insertPoint, alteration, listSize)
372.    struct ListElement **list;
373.    int start, end, insertPoint, alteration;
374.    {       struct ListElement *holder, **dumList;
375.            int i, j;
376.
377.            switch(alteration)
378.            {       case REVERSE:
379.                    for(i = start, j = end; i < j; i++, j--){
380.                            holder = list[i];
381.                            list[i] = list[j];
382.                            list[j] = holder;
383.                    }
384.                    return;
385.
386.                    case TRANSPOSE:
387.                    dumList = (struct ListElement **)malloc(sizeof(struct
388.                            ListElment *)*listSize);
389.
390.
391.                    for(j = 0, i = start; i <= end; i++, j++)    /* Copy group. */
392.                            dumList[j] = list[i];
393.
394.                    if(insertPoint  end){                        /* Move downward.
*/
395.                      for(j=end+1, i = start; j < insertPoint; i++, j++)
396.                            list[i] = list[j];
397.
398.                       for(j = insertPoint - (end - start + 1), i = 0; j < insertPoint;
399.                                       i++, j++)
400.                    list[j] = dumList[i];                        /* Reinstall group. */
401.                    }
402.                    else {
403.                      for(j=start - 1, i = end; j = insertPoint; i--, j--)
404.                            list[i] = list[j];                   /* Move upward. */
405.
406.                       for(j = insertPoint, i = 0; i <= (end - start); i++, j++)
407.                            list[j] = dumList[i];
408.
409.                    }
410.
411.                    free(dumList);
412.
```

```
413.                        return;
414.                }
415.
416.
417.        }
```

4 Aplicações do Simulated Annealing

Nesta capítulo são apresentadas algumas propostas alternativas ao Simulated Annealing tradicional e uma relação de aplicações. Estas propostas alternativas serviram para a obtenção de um melhor entendimento do paradigma e uma melhor visualização das diversas características do mesmo. Não há o objetivo de ser um relacionamento exaustivo, pois o paradigma tem sido utilizado em diversas áreas e com inúmeras variações.

4.1 Propostas Alternativas

Modificações ao algoritmo Simulated Annealing são encontradas com freqüência documentadas na literatura, algumas apontando resultados satisfatórios em relação ao Simulated Annealing tradicional.

Seleção de Movimentos para Rejeição

Na forma clássica do Simulated Annealing não é realizada nenhuma seleção com relação aos movimentos propostos. Isso pode resultar em um tempo de execução a baixas temperaturas muito grande porque muitos movimentos candidatos são rejeitados. A proposição é manter uma lista de cada movimento candidato contendo informações sobre os efeitos deste movimento e usar esta informação para influenciar a próxima seleção.

Greene e Suppovit [GRE 89] apontam como vantagem desta alternativa o tempo de execução independente da taxa de aceitação e relatam uma experiência de aplicação na etapa de partição lógica do projeto de microprocessador, usando solução inicial aleatória e solução inicial construída por heurística. Em qualquer caso, numa primeira fase foi usado Simulated Annealing tradicional e, numa segunda fase, uma variação com menor número de rejeições. A comparação foi favorável para o método sugerido pelos autores.

Simulated Annealing com Reinicialização Periódica

Nesta proposição periodicamente o processo é reiniciado tendo como solução de reinicio a melhor encontrada até o momento (apresentada por Charon e Olivier, cap. 35) [OSM 96].

Simulated Annealing com Exploração Sistemática da Vizinhança

Nesta proposta é realizada um exploração sistemática da vizinhança da solução corrente até que uma solução seja aceita (apresentada por Charon e Olivier, cap. 35) [OSM 96].

Simulated Annealing Adaptativo com Relação à Temperatura

Existem várias propostas de Simulated Annealing onde o processo de escolha da temperatura é adaptativo, como nas propostas apresentadas por:

(a) Charon e Olivier (cap. 35 [OSM 96]), onde é utilizada uma função, baseada no número de transições desfavoráveis, para uso na atualização da temperatura, aumentando a temperatura quando o número de transformações não desejáveis é menor que um valor arbitrário estipulado e diminuindo em caso contrário.

(b) Ingber (Adaptative Simulated Annealing - ASA) [ING 99] com o objetivo de ajudar no manuseio de modelos não-lineares complexos. Conhecido inicialmente como VFSR (Very Fast Simulated Annealing), desenvolvido em 1987. Os problemas são visualizados com um terreno geográfico, com montanhas e vales. O objetivo é encontrar o vale mais baixo neste terreno. A principal modificação é relacionada com o parâmetro temperatura, permitindo que a temperatura seja aumentada dentro de determinadas condições (reannealing). É uma proposta bem documentada, desenvolvida em C e com código disponível.

Mean-field Annealing (MFA)

Utilizada por Elmohamed e outros [ELM 97] em um estudo comparativo de Simulated Annealing para o problema de escalonamento de cursos (timetabling) para uma grande universidade. O problema encontrado foi o tratamento complexo do tipo de redes neuroniais envolvidas.

Two-Stage Simulated Annealing (TSSA)

Esta proposta, apresentada por Varanelli e Cohoon [VAR 93], apresenta um algoritmo de duas etapas. Na primeira etapa é usada uma heurística diferente do Simulated Annealing para aproximar uma solução inicial. Esta etapa serve também como uma etapa de amostragem, onde dados estatísticos são recolhidos e verificado uma aproximação da distribuição das soluções. Na segunda etapa é utilizado Simulated Annealing tradicional, possibilitando a escolha da prescrição de resfriamento, exponencial ou logarítmica. A escolha da temperatura inicial, para a segunda etapa, é baseada nos dados recolhidos por amostragem na primeira.

Os resultados apresentados pelos autores referentes a testes com três problemas importantes de otimização combinatória são satisfatórios, com manutenção da qualidade da solução obtida com relação ao Simulated Annealing tradicional e ganho significativo de tempo

computacional. A contribuição importante deste trabalho é um cálculo da temperatura inicial baseado em uma base mais formal.

Híbridos

As técnicas de otimização que fazem uso de hibridização procuram explorar as vantagens de diferentes enfoques e evitar suas desvantagens. Através da hibridização, uma estratégia de otimização pode ser adaptada às características específicas de um problema e, assim, aprimorar a robustez e eficiência do processo de otimização como um todo. Seja um problema cujo espaço de soluções pode conter múltiplos mínimos. Um processo de otimização poderia ser utilizado para identificar regiões promissoras, usando um algoritmo adequado para isso (por exemplo, algoritmos genéticos). Como estes algoritmos podem ser dispendiosos uma vez que requerem avaliações de muitas funções, numa segunda fase poderia ser utilizada uma técnica de busca local, aplicada às regiões promissoras, com convergência mais rápida a um ótimo local.

O Simulated Annealing tem sido utilizado em processos de otimização híbridos, como:

- Combinado com técnicas de busca local determinísticas, numa heurística denominada Chained Local Optimization (C-L-O) [MAR 93], como bons resultados comparativos com outras heurísticas para os problemas do caixeiro-viajante e particionamento de grafos. Outra proposição neste enfoque é a inclusão periódica de busca local descendente no algoritmo SA (apresentada por Charon e Olivier, cap. 35) [OSM 96].

- Combinado com algoritmo genético para o problema de particionamento de grafos [AGJ 95], com o algoritmo genético utilizado para identificar boas regiões e o Simulated Annealing na busca local de um ótimo.

- Combinado com técnicas das heurísticas busca tabu e algoritmo genético, constituindo um SA sofisticado proposto por Fox [FOX 94]. A proposição inclui o uso de vizinhanças dinâmicas, estado constituído por uma população de soluções, ou seja, cada estado a ser percorrido é constituído de várias soluções em vez de uma e manutenção de uma lista das soluções visitadas recentemente.

- Combinado com técnicas de algoritmo genético e construção da solução vizinha a partir da solução corrente empregando critérios parcialmente determinísticos. Esta proposta é apresentada por Wendt [WEN 98].

4.2 Levantamento de Aplicações

A relação, resumida e não exaustiva, tem como objetivo apontar para documentação existente. Os comentários são observados a partir do documento consultado e referenciado.

1. **Aplicação**: Problema do caixeiro-viajante
   1.1. [PRE 88] - exemplo codificado em C
   Prescrição de resfriamento: $T_{i+1} = \alpha \times T_i$
   1.2 [CAR 98] - exemplo codificado em C, disponível na WEB
   Prescrição de resfriamento: $T_{i+1} = \alpha \times T_i$
2. **Aplicação**: Particionamento de grafos
   2.1. [JOH 89] - estudo comparativo de alternativas de prescrições de resfriamento.
   2.2. [AGJ 95] - híbrido com algoritmo genético
   A utilização do método híbrido resultou em ganhos de performance.
   2.3. [SHR 91] - teste da performance do algoritmo para grafos aleatórios e geométricos.
   Dado um grafo G = (V,A), onde V é o conjunto de vértices e A o conjunto de arestas, existem dois conjuntos V_1 e V_2 tais que $V = V_1 \cup V_2$ e $V_1 \cap V_2 = \varnothing$.
   Função objetivo: $c(V_1 \text{ e } V_2) = |\{(m,v) \in A: m \in V_1 \text{ e } v \in V_2\}| + \beta(|V_1| \text{ e } |V_2|)^2$, onde m e v são vértices distintos do grafo e β é um fator de desbalanceamento para penalizar partições de tamanho desigual.
   Vizinhança: duas partições são consideradas vizinhas quando uma pode ser obtida da outra pela movimentação de um vértice de um conjunto para outro.
   Perturbação: movimentação de um vértice de um conjunto para outro.
   Temperatura inicial: determinada pela condição de aceitar maus movimentos com uma probabilidade de 0.4.
   Prescrição de resfriamento: $T_{i+1} = \alpha \times T_i$ ($\alpha = 0.95$).
   Critério de término (congelamento): quando não há modificação do valor da função objetivo nas últimas 5 temperaturas e a porcentagem de movimentações aceitas não excede 2%.
   Comprimento da cadeia de Markov (número de iterações numa mesma temperatura): $16 \times n_v$ onde n_v é o tamanho da vizinhança.
3. **Aplicação**: Projeto de Cobertura
   3.1. [NUR 93]
   Apresenta um forma alternativa para cálculo da temperatura inicial e uso uma forma aproximada para o cálculo da função exponencial do critério de metropolis. Código em C disponível via ftp.
4. **Aplicação**: Poker solitário
   4.1. [ROD 93]
   Temperatura inicial: estimada usando o conceito de entropia.
   Prescrição de resfriamento: $T_{i+1} = \alpha \times T_i$ ($\alpha = 0.85$).
   Critério de término (congelamento): 0.1.

5. **Aplicação**: VLSI
   5.1. [SAI 95] - particionamento(two-way partitioning) de circuitos
   Função objetivo: gerar partições balanceadas.
   Perturbação: troca de pares, ou seja, troca um nodo $a \in A$ por um nodo $b \in B$.
   Temperatura inicial: 10.
   Prescrição de resfriamento: $T_{i+1} = \alpha \times T_i$ ($\alpha = 0.9$).
   Temperatura final: quando T reduz 30% da temperatura inicial
   5.2. [SAI 95] - posicionamento de células, algoritmo TimberWolf.
   Função objetivo: minimizar comprimento total da fiação, penalizando sobreposições e excesso de comprimento numa linha
   Perturbação: mover uma célula para uma nova posição, ou trocar duas células de posição entre si, ou espelhar uma célula sobre um eixo. É utilizada uma função dependente da temperatura para limitar a distância do movimento, que decresce logaritmicamente com a temperatura..
   Temperatura inicial: 4×10^6
   Prescrição de resfriamento: $T_{i+1} = \alpha(T_i) \times T_i$ (inicialmente $\alpha(T) \approx 0.8$, a temperaturas intermediárias $\alpha(T) \approx 0.95$ e no final $\alpha(T) \approx 0.8$)
   Temperatura final: $T < 1$
   Comprimento da cadeia de Markov: número de movimentos fixo a cada temperatura e dependente do tamanho do circuito (100 movimentos por célula para um circuito de 200 células, o que corresponde a cerca de 125 etapas de temperaturas com avaliação de 2.34×10^6 configurações e 700 movimentos por célula por um circuito de 3000 células, o que corresponde a 247.5×0^6 configurações).
6. **Aplicação**: Job Shop Scheduling
   6.1. [LAA 92] - Seja um conjunto de tarefas e um conjunto de máquinas. Cada tarefa consiste em uma cadeia de operações, onde cada operação deve ser processada durante um intervalo ininterrupto de tempo em uma dada máquina. Cada máquina só pode processar uma operação num determinado tempo. Um sequenciamento (schedule) é uma alocação de operações nas máquinas em intervalos de tempo.
   Função objetivo: encontrar um sequenciamento (schedule) de tamanho mínimo.
   Perturbação: mover uma célula para uma nova posição, ou trocar duas células de posição entre si, ou espelhar uma célula sobre um eixo.
7. **Aplicação**: Alocação de Tarefas Imprecisas em Ambiente Distribuído
   7.1. [OLI 97]
   Função objetivo: $E = k_e E_e + k_b E_b$, onde E_e e E_b são, respectivamente, a energia associada com a escalonabilidade das tarefas e a energia associada com o balanceamento de carga no sistema e k_e e k_b constantes de valores adequados ($k_e >> k_b$) de forma a atribuir uma importância maior para o aspecto escalonabilidade do que para o aspecto balanceamento de carga.
   Perturbação: de dois tipos, ou seja, uma tarefa é escolhida aleatoriamente e movida para outro processador também escolhido aleatoriamente (70% dos casos) ou duas tarefas são escolhidas aleatoriamente e trocam de processador entre si (30% dos casos).

Temperatura inicial: 10 x número de processadores x número de tarefas.
Prescrição de resfriamento: $t_{k+1} = \alpha$ x t_k ($\alpha = 0.99$).
Critério de equilíbrio: a temperatura é reduzida sempre que uma das seguintes condições for satisfeita: número máximo de movimentos igual ao (número de processadores * número de tarefas)/2 ou número máximo de tentativas igual ao número de processadores * número de tarefas.
Critério de término (congelamento): quando uma das seguintes condições for satisfeita: temperatura final igual ao número de tarefas ou número máximo de tentativas sem sucesso igual a 2 * número de processadores* número de tarefas.
Observação: as experiências realizadas indicaram que o SA é uma abordagem viável para o problema

8. **Aplicação:** Escalonamento de cursos (Timetabling)
   8.1. [ELM 97] comparativo de várias alternativas do SA para esta aplicação para uma grande universidade
9. **Aplicação**: Reconstrução de imagens PET ("Positrom Emission Tomografy")
   9.1. [SUN 96]
   Função objetivo: expressa a diferença entre os dados que representam a imagem original e os dados que representam a imagem reconstruída.
   Temperatura inicial: escolhida considerando que cerca de 80% dos movimentos que resultam em aumento da função objetivo são aceitas. O método prático utilizado usa a fórmula $t_0 = 5\ (\Delta E_i)_{\text{médio}}$ calculado a partir dos movimentos com aumento da função objetivo após a execução de *n* iterações arbitrárias.
   Prescrição de resfriamento: $T_{i+1} = \alpha$ x T_i ($0{,}8 \leq \alpha \geq 0{,}98$).
   Observação: resultados considerados favoráveis em relação algoritmo determinístico geralmente usado ("filtered backprojection algorithm").
10. **Aplicação**: Genéricos

    10.1. EBSA (Ensemble Based Simulated Annealing) [FRO 98]: conjunto de procedimentos desenvolvidos em C, permitindo a entrada de parâmetros pelo usuário e interface para procedimentos definidos pelo usuário de acordo com a instância do problema a ser resolvido.

    10.2. ASA (Adaptative Simulated Annealing) [ING 99]: conjunto de rotinas disponíveis na WEB, propondo uma prescrição de resfriamento adaptativa para problemas complexos.

5 Conclusão Final

O presente trabalho, a partir de um levantamento prévio de como o paradigma Simulated Annealing está sendo utilizado e de suas origens históricas, identificou as principais características do Simulated Annealing seqüencial, verificando a importância das mesmas e a forma como interagem, buscando as diferentes interpretações para estas características.

Usando o conhecimento adquirido o Simulated Annealing foi formalizado como um método de desenvolvimento de algoritmos. Este método Simulated Annealing identifica procedimentos básicos, como funções e predicados, assim como o relacionamento entre os mesmos. Apresenta programas abstratos para a estes procedimentos e axiomas, que estabelecem, de forma clara, as propriedades que as funções e predicados devem satisfazer. Foi apresentada uma equação simples de cálculo de complexidade para um algoritmos desenvolvidos utilizando o método, identificando os aspectos críticos que influenciam o bom desempenho do algoritmo.

Inicialmente, no capítulo 1, Introdução, foi apresentado o tema do trabalho, Simulated Annealing, destacando como principais vantagens do Simulated Annealing a aplicabilidade geral e a habilidade de fornecer soluções bastante próximas da ótima, identificando de forma abreviada as diferentes áreas de aplicação. Foi apresentado também os objetivos do trabalho e as principais fontes de consulta sobre os diferentes assuntos envolvidos.

O capítulo 2, Otimização, teve como finalidade situar o Simulated Annealing em um contexto mais amplo. Assim, foram apresentados aspectos teóricos referentes a otimização, identificando conceitos adequados para problema de otimização e seus elementos, em especial, os conceitos de espaço de solução e vizinhança, e introduzindo noções referentes as técnicas de resolução baseadas em busca local. Como “otimização” é uma área ampla de pesquisas, os aspectos introduzidos pelo trabalho são aqueles considerados importantes para o entendimento do mesmo. Buscou-se uma definição de problema de otimização adequada aos propósitos do trabalho e apresentou-se as características essenciais de um espaço de soluções baseado no estabelecimento de uma estrutura de vizinhança para o problema de otimização. Foram introduzidos os aspectos básicos das chamadas técnicas de busca local e o conceito de perturbação como função básica de suporte para estas técnicas. Como o Simulated Annealing é visto como uma metaheurística foi introduzido este conceito e como o Simulated Annealing utiliza técnicas de aleatoriedade foi apresentado um conceito de algoritmo aleatório.

No capítulo 3, Simulated Annealing, apresentou–se, inicialmente, os aspectos que suportam o paradigma, identificando suas origens históricas, introduzindo o algoritmo de Metropolis original [MET 53] e o modelo clássico de Kirkpatrick [KIR 83], baseado nos conceitos de mecânica estatística. Como o Simulated Annealing tem sido modelado por muitos autores [AAR 89] [AZE 92] usando a teoria de cadeia de Markov, foi apresentada uma introdução as métodos Monte Carlo Markov Chain (MCMC) [KAL 86] [JER 97].

A partir do modelo clássico do Simulated Annealing, com a finalidade de permitir uma melhor visualização das características, foi elaborada a idéia de um algoritmo genérico seqüencial, constituído de procedimentos independentes, mas que se relacionam entre si e identificadas as principais funções destes procedimentos. Foi destacado a importância da atribuição adequada de valores aos principais elementos envolvidos na construção de um algoritmo SA, como temperatura, e a especificação de funções adequadas para perturbação e energia. Foram apresentados os modelos básicos de prescrições de resfriamento[KIR 83] [AAR 89] [ROD93], identificando suas características usando os procedimentos do algoritmo genérico.

O conhecimento das características e dos procedimentos constituintes deste algoritmo genérico permitiram a construção de uma especificação formal, representando o algoritmo através de um programa abstrato e estabelecendo axiomas que definem os requisitos que as principais funções e predicados devem satisfazer. O formalismo utilizado, ou seja, a axiomatização, foi escolhido por permitir um estabelecimento claro destes requisitos sem definir uma forma de implementação. O programa abstrato é utilizado para descrever os elementos que influenciam na complexidade do algoritmo, ou seja, na performance do mesmo. Uma demonstração da correção do programa abstrato proposto é apresentada no Anexo1.

A especificação do Simulated Annealing, sua definição como um método e sua representação através de um algoritmo abstrato, permitiu a análise da complexidade de algoritmos. Em resultado desta análise, verificou-se que a função perturbação e o número de soluções visitadas em uma execução é determinante da complexidade total. Como o número de soluções visitadas e a probabilidade de encontrar uma solução final de qualidade exigida pela aplicação interagem entre si, a implementação da função perturbação torna-se um fator crítico de complexidade. O esforço despendido numa codificação otimizada do procedimento que realiza esta função é recompensado com um desempenho melhor do algoritmo como um todo. No entanto, esta é uma reflexão simplista, pois outros fatores estão envolvidos. A função perturbação é definida no projeto de uma estrutura de vizinhança e, esta estrutura de vizinhança determina o espaço de soluções possível de ser visitado. A escolha da função energia, uso de penalidades ou não, a escolha da prescrição de resfriamento são fatores que devem serem analisados em conjunto e que resultam em melhor ou pior desempenho do algoritmo. Como colocado por Catoni (cap. 3, [AZE 92]) o estudo teórico dos aspectos relacionados com prescrição de resfriamento para uma dada aplicação e uma dada disponibilidade de recursos deve ser encorajado, uma vez que não existe fórmula fechada para determinar esta prescrição e se está longe de encontrar uma prescrição de resfriamento genérica milagrosa.

Ainda, no capítulo 3, é apresentado um estudo de caso. Neste estudo de caso é analisado o código de um programa exemplo [CAR 98] em relação ao programa abstrato apresentado, verificando como os procedimentos foram implementados e a aplicabilidade dos axiomas definidos. A busca da identificação das funções e predicados em um código implementado e a verificação dos axiomas mostrou-se uma ferramenta interessante para análise deste código. Esta análise permite verificar tanto aspectos de correção do código como estudos de modificações para outras aplicações.

Complementando o trabalho dois aspectos importantes são apresentados: o primeiro, no capítulo 4, aprofunda o conhecimento teórico do Simulated Annealing, através da apresentação de um modelo do baseado em Cadeias de Markov [AAR 89], detalhando aspectos da propriedade de convergência do algoritmo e apresentando uma dedução empírica da complexidade do algoritmo considerando este modelo; o segundo, no capítulo 5, mostra o resultado resumido do levantamento das utilizações do algoritmo tradicional e propostas alternativas com a finalidade de detalhar como o Simulated Annealing está sendo utilizado na prática. A análise destas utilizações foi base para a especificação do programa abstrato apresentado no capítulo 3. A identificação do essencial permitiu a especificação de procedimentos básicos e o estabelecimento de axiomas que caracterizam o algoritmo.

Duas metas principais guiaram o desenvolvimento do trabalho: primeiro propiciar conhecimento básico sobre o assunto e, segundo, permanecer em um nível teórico para permitir a observação dos aspectos genéricos de forma adequada sem o comprometimento de uma implementação específica. A formalização proposta não abrange todas as alternativas de desenvolvimento de algoritmos Simulated Annealing, no entanto, procurou identificar de forma clara as características essenciais, possibilitando uma base sólida que pode ser modificada e ampliada quando necessário.

Os aspectos de complexidade do programa abstrato proposto foram analisados, onde, a perturbação, mostrou-se como peça fundamental na avaliação da complexidade de um algoritmo SA. Como a perturbação é decorrente da escolha de uma estrutura de vizinhança destaca-se a importância da escolha adequada da mesma.

As principais contribuições do trabalho são: (a) um guia para o estudo do Simulated Annealing, considerando fatores empíricos e teóricos, permitindo o esclarecimento de conceitos e o uso adequado dos mesmos; (b) uma base teórica para implementações baseadas na forma como estes conceitos estão sendo utilizados na prática; (c) uma formalização, estabelecendo um método que especifica um programa abstrato e define axiomas, permite uma melhor visualização das propriedades de um algoritmo e de seus elementos constituintes, a análise destas propriedades e, propicia um melhor entendimento dos aspectos de complexidade.

A apresentação do trabalho foi organizada de forma a permitir diferentes leituras dependendo do interesse do leitor: um conhecimento em nível de introdução pode ser obtido pela leitura do capítulo 2 e seções 3.1, 3.2 e 3.3; um conhecimento mais detalhado dos aspectos de implementação podem ser obtidos pela leitura da seção 3.4, 3.5, 3.6 e do capítulo 4; um conhecimento teórico mais aprofundado e detalhado requer também a leitura do anexo A2.

Entre os temas para trabalhos futuros relacionados com o Simulated Annealing, incluindo os estudos de otimização e complexidade de algoritmos, podem ser mencionados: estudos detalhados teóricos e práticos dos aspectos relacionados com o espaço de soluções para problemas de otimização, empregando técnicas de amostragem adequadas, para permitir uma determinação mais científica dos parâmetros do algoritmo e possibilitar o estudo da complexidade média a partir da equação definida, que é mais interessante no estudo de heurísticas; estudo de tipos de perturbação aplicáveis a determinados grupos de problemas, construindo uma base teórica para a escolha do tipo de estrutura de vizinhança e um formalismo para representar de forma adequada e genérica os tipos de perturbação; estudos

dos aspectos de paralelização do algoritmo Simulated Annealing, baseada nas técnicas já existentes, permitindo um melhor conhecimento e representação das mesmas e identificando os aspectos empíricos e teóricos relacionados com a performance.

Em conclusão, este trabalho apresentou o Simulated Annealing de forma genérica e completa, possibilitando que a partir do mesmo possam ser desenvolvidos e analisados aspectos específicos.

Anexo A1 Verificação da Especificação Proposta

Neste anexo é verificada a correção do programa abstrato SA proposto como um método de desenvolvimento na seção 3.05. A estrutura do programa abstrato SA especificado é simples. Assim, inicialmente é feita a demonstração da correção da especificação do programa abstrato SA (Fig. 3.06), considerando as especificações dos programas abstratos chamados (aquecimento, Fig. 3.10, e metropolis, Fig. 3.08) como corretas e, posteriormente, é feita a demonstração para estes programas. As especificações dos procedimentos dependentes do problema não foram detalhadas e são consideradas corretas pelo estabelecimento dos axiomas que as mesmas devem satisfazer.

A1.1 Programa Abstrato SA

A demonstração do programa abstrato abaixo considera a especificação dada na Fig. 3.06, diagrama sintático da Fig. 3.05 e definições de funções de axiomas, funções e predicados da seção 3.5.

Suposição inicial:

Suponha a asserção de entrada, φ (p) = (($|S| > 0$) $\wedge$ ($n_{cad} > 0$) $\wedge$ ($n_L > 0$) $\wedge$ ($n_{aqc} > 0$) $\wedge$ ($t_{ini} > 0$) $\wedge$ (($\forall$p) ($\forall$s) possível (p,s) $\Rightarrow$ viável (p, f_{vav} (p, s))), verdadeira logo antes do início da execução do procedimento.

Demonstração da invariante da linha 6:

Pelo axioma SAAX02, após a execução da linha 5, tem-se a invariante [possível (p, s)] e pela asserção de entrada [t > 0].

Demonstração da invariante da linha 8:

A invariante da linha 6 satisfaz a asserção de entrada do programa abstrato aquecimento. Portanto, considerando a semântica da especificação do programa abstrato aquecimento, tem-se a asserção de saída do mesmo, [quente (p, t) $\wedge$ t > 0], verdadeira após a execução da linha 7, o que garante a invariante da linha 8.

A invariante da linha 8 também pode ser considerada para qualquer outra especificação para o procedimento aquecimento diferente da proposta, que satisfaça os axiomas SAAX03 e SAAX0A2.

Demonstração das invariantes das linhas 11, 13 e 15:

Entre as linhas 10 e 16 tem-se a seqüência de instruções do enquanto que são repetidas *n* vezes, constituindo uma fase de melhoramento da solução tratada, até que que as condições estabelecidas pelo predicado congelado sejam satisfeitas.

Será considerada inicialmente a situação das invariantes na primeira iteração(i =1). A invariante da linha 11 é válida, pois [possível (p, s)] mantém-se sem alteração desde a linha 6, pela semântica de aquecimento e congelado, e [t > 0], uma vez que pela semântica do predicado congelado, não há modificação do valor de t neste trecho. A asserção de entrada do programa abstrato metropolis, [possível (p, s) e t > 0], corresponde a invariante da linha 8. Como é válida a asserção de entrada do programa abstrato metropolis e pela semântica do mesmo tem-se a invariante da linha 13, pois [possível (p, s) $\wedge$ estável (p, s_{ent}, t, s)] é garantido pela asserção de saída e t não é alterada. Pelo axioma SAAX05, tem-se a invariante da linha 15.

Em cada iteração i > 1 e i < n, as seguintes considerações são válidas: sejam s' e t' os valores das variáveis s e t logo após a execução $i^{ésima}$ iteração da linha 11 e {s''= metropolis (p, s', t')} e {t'' = resfriamento (p, t')} e supondo verdadeiras as invariantes da linha 11, 13 e 15 nesta $i^{ésima}$ iteração. Então, na $(i+1)^{ésima}$ iteração a invariante da linha 11 é valida pelas invariantes da linha 13 e 15 da $i^{ésima}$ iteração. O raciocínio do parágrafo anterior para as invariantes da linha 13 e 15, válido para a primeira iteração, é válido para as iterações seguintes.

Demonstração da invariante da linha 18:

A invariante da linha18 é resultado da condição de término [congelado (p, t)] do enquanto 10-16.

A invariante da linha 18 também pode ser considerada para qualquer outra especificação para o procedimento metropolis diferente da proposta, que satisfaça os axiomas SAAX12.

Demonstração da invariante da linha 19:

Sejam as seqüências s_0, s_1, ... s_{k+1} e t_0, t_1, ... t_{k+1} onde s_0 é a solução inicial definida na linha 5, t_0 a temperatura definida na linha 7, s_i é a solução encontrada na $i^{ésima}$ execução (linha 12) da iteração 10-16, t_i é a temperatura encontrada na $i^{ésima}$ execução (linha 14) da iteração 10-16 e i = 1, 2, ..., k+1, onde k+1 é o número de vêzes que a iteração 10-16 é executada. Então, pelas invariantes das linhas 11 e 13 de cada iteração i = 1,2,... k, a condição [congelado (p, t)] (linha 10) no final da execução da iteração k+1 e o axioma SAAX14 tem-se [$estável^{+}$ (p, s, t)]. Pelas invariantes das linhas 6 e 13, tem-se, em qualquer caso, [possível (p, s)], o que prova a validade da invariante da linha 19.

Demonstração da invariante da linha 21:

Dois casos devem ser considerados. No primeiro caso, logo antes da execução da linha 20 tem-se uma solução s para a qual [viável (p, s)] é satisfeito, então, pelo axioma SAAX06 o valor de s não é alterado na linha 20 e, portanto, [viável (p, s)] na linha 21. No segundo caso, logo antes da execução da linha 20 tem-se uma solução s para a qual [$\neg$ viável (p, s)] é satisfeito, então, pelo axioma SAAX07, após a execução da linha 20

o valor de s será f_{vav}(p, s) e pela asserção de entrada a função f_{vav} é tal que [(($\forall$p) ($\forall$s) possível (p,s) $\Rightarrow$ viável (p, f_{vav} (p, s))] e, assim, [viável (p, s)] na linha 21.

Verificação da asserção de saída:

A asserção de saída é garantida pelas invariante das linhas 19 e 21 e pelo axioma SAAX09.

Prova da terminação do programa

Para garantir o término do programa abstrato SA é suficiente provar que o enquanto 10-16 termina.

Dois casos podem ser analisados, considerando

[condição-congelado = ((cadeia (p, t) $\geq n_{cad}$) $\vee$ (t $\leq$ 0)].

No primeiro caso, o término ocorre pelo número máximo de resfriamentos permitidos. Ou seja:

Seja t_i o valor de t na $i^{ésima}$ iteração do enquanto 10-16. Tem-se

[cadeia (p, t_{i+1}) > cadeia (p, t_i)] pelos axiomas SAAX11, e

[cadeia (p, t_1) = 0] pelo axioma SAAX10.

Então, [cadeia (p, s_k) $\geq n_{cad}$] para k = n_{cad} e pelo axioma SAAX09 tem-se congelado (p, t_{k+1}), o que mostra que após n_{cad} passos o enquanto 10-16 termina.

No segundo caso, o término ocorre quando t_{i+1} = 0 e, então, pelo axioma SAAX09, tem-se congelado (p, t_{i+1}).

A1.2 Programa Abstrato metropolis

A demonstração do programa abstrato abaixo considera a especificação dada na Fig. 3.08, diagrama sintático da Fig. 3.07 e axiomas, funções e predicados descritos na seção 3.5 para o procedimento metropolis..

Suposição inicial:

Suponha a asserção de entrada, φ (p) = [possível (p, s_o) $\wedge$ (t $\geq$ 0)], verdadeira logo antes do início da execução do programa.

Demonstração da invariante da linha 6:

Pela asserção de entrada, execução da linha 5 e axioma MEAX01 tem-se a invariante da linha 6.

Demonstração das invariantes das linhas 8, 11 e 13:

Entre as linhas 7 e 14 tem-se uma seqüência de instruções do enquanto que são repetidas *n* vezes, constituindo uma fase de melhoramento da solução tratada, até que as condições estabelecidas pelo predicado equilíbrio sejam satisfeitas.

Será considerada inicialmente a primeira iteração.

A invariante da linha 8 é satisfeita pois não há modificação na solução desde a linha 6.

Pela invariante da linha 8, execução da linha 10 e axioma MEAX05 tem-se a invariante da linha 11.

Pela invariante da linha 11 e axioma MEAX07 tem-se

[(crit-metropolis (s, s_c, t , n_r) $\Rightarrow$ aceitação (s, s_c, t , n_r) = s) $\wedge$ ($\neg$crit-metropolis (s, s_c, t , n_r) $\Rightarrow$ aceitação (s, s_c, t , n_r) = s_c)]

que é equivalente a

[((aceitação (s, s_c, t , n_r) = s) $\wedge$ crit-metropolis (s, s_c, t , n_r)] $\vee$ [((aceitação (s, s_c, t , n_r) = s_c) $\wedge$ $\neg$crit-metropolis (s, s_c, t , n_r)]. A demonstração da equivalência é descrita na seção Anexo1.A2.

Então, pelo axioma MEAX08 tem-se

[aceita (aceitação (s, s_c, t, n_r), s, s_c, t n_r)] que é a invariante da linha 13.

Para as iterações seguintes do enquanto 7-14 tem-se:

Seja s' o valor de s na iteração anterior e s'' o valor de s para a iteração atual; pela invariante da linha 11 tem-se

[possível (p, s') $\wedge$ possível (p, perturbação (p, s')]

e

s'' = aceitação (s, perturbação (p, s'), t, aleatório());

Pelo axioma MEAX07 s'' = s' ou s'' = perturbação(p, s'), ou seja, em qualquer caso, possível (p, s''), que é a invariante da linha 8.

A demonstração das invariantes das linhas 11 e 13 é feita usando raciocínio análogo ao utilizado para a primeira iteração.

Demonstração da invariante da linha 15:

Considere as k iterações do enquanto 7-14 e faça s_{i-1} o valor de s na entrada da $i^{ésima}$ iteração. Pela invariante da linha 13 tem-se

[aceita (s_i, s_{i-1}, perturbação (p, s_{i-1}), t, aleatório())] para i = 1,2, ..., k.

Então, pela condição de parada dada por equilíbrio (s_k) e pelo axioma MEAX10, tem-se a invariante da linha 15.

Verificação da asserção de saída:

Tem-se a asserção de saída, {possível (p, s) $\wedge$ estável (p, s_0, t, s)}, em decorrência da invariante da linha 15 e usando raciocínio análogo ao da demonstração da invariante da linha 8 nas iterações subsequentes à primeira.

Prova da terminação do programa

Para garantir o término do programa abstrato metropólis é suficiente provar que o enquanto 7-14 termina.

Seja s_i o valor de s na $i^{ésima}$ iteração do enquanto 7-1A2. Tem-se

[experimento (p, s_{i+1}) > experimento (p, s_i)] pelos axiomas MEAX04 e MEAX09, e

[experimento (p, s_1) = 0] pelo axioma MEAX03.

Então, [experimento (p, s_k) $\geq$ n_L] para k = n_L e pelo axioma MEAX02 tem-se equilíbrio (p, s_{k+1}), o que mostra que após n_L passos o enquanto 7-14 termina.

A1.3 Programa Abstrato aquecimento

A demonstração do programa abstrato abaixo considera a especificação dada na Fig. 3.10, diagrama sintático da Fig. 3.09 e definições de funções de axiomas, funções e predicados da seção 3.5. Os procedimentos aleatório, perturbação e aceitação do programa abstrato aquecimento são considerados como tendo as mesmas funções dos respectivos procedimentos do programa abstrato metropolis e, em decorrência, são válidos os mesmos axiomas (MEAX05 e MEAX07).

Suposição inicial:

Suponha a asserção de entrada , φ (p) = (possível (p, s_{ini}) $\wedge$ ($t_{ini} > 0$)), verdadeira logo antes do início da execução do programa. A asserção de entrada estabelece as condições que devem ser satisfeitas no início da execução para garantir a correção do programa.

Demonstração da invariante da linha 5:

Considerando a asserção de entrada, execução da linha 4 e axioma AQAX01 tem-se a invariante da linha 5.

Demonstração da invariante da linha 8, 10, 12 e 14:

Entre as linhas 6 e 15 tem-se a seqüência de instruções do enquanto que são repetidas *n* vezes, constituindo uma fase de aquecimento, até que as condições estabelecidas pelo predicado quente sejam satisfeitas.

Será considerada inicialmente a primeira iteração.

A invariante da linha 8 é satisfeita pois não há modificação na solução e na temperatura desde a linha 6.

Pela invariante da linha 8, execução da linha 9 e axioma MEAX05 tem-se a invariante da linha 10.

Seja s' o valor de s na iteração anterior e s" o valor de s para a iteração atual; pela invariante da linha 10 tem-se

[possível (p, s') $\wedge$ possível (p, perturbação (p, s')]

e pela execução da linha 11 tem-se

s'' = aceitação (s, perturbação (p, s'), t, aleatório());

Pelo axioma MEAX07 s'' = s' ou s'' = perturbação(p, s'), ou seja, em qualquer caso, possível (p, s''), que é a invariante da linha 12.

Pela invariante da linha 8, execução da linha 13 e axioma AQAX05 tem-se a invariante da linha 1A2.

Nas iterações do enquanto 7-15 subsequentes a primeira tem-se:

Da invariante da linha 12 tem-se [possível (p, s)] e da invariante da linha 14 tem-se [t > 0], demonstrando a invariante da linha 8.

As provas para as invariantes das linhas 10, 12 e 14 são análogas às provas para a primeira iteração.

Demonstração da invariante da linha 16:

Em decorrência da condição da linha 6 tem-se a invariante da linha16.

Verificação da asserção de saída:

A asserção de saída é satisfeita pois [quente (p, s)] é decorrência da invariante da linha 15, [posssível (p, s)] é decorrência das invariantes das linhas 5 e 12 e [t > 0] é decorrente das invariantes da linha5 e 8 e axioma AQAX05.

Prova da terminação do programa

Para garantir o término do programa abstrato metropólis é suficiente provar que o enquanto 6-15 termina.

Seja s_i o valor de s na $i^{ésima}$ iteração do enquanto 6-15. Tem-se

[mais-quente (p, s_{i+1}) > mais-quente (p, s_i)] pelos axiomas AQAX04, e

[mais-quente (p, s_1) = 0] pelo axioma AQAX03.

Então, [mais-quente (p, s_k) $\geq n_L$] para k = n_L e pelo axioma AQAX02 tem-se quente (p, s_{k+1}), o que mostra que após n_L passos o enquanto 6-15 termina.

A1.4 Demonstração de Equivalência

Seja a expressão

[(crit-metropolis (s, s_c, t , n_r) $\Rightarrow$ aceitação (s, s_c, t , n_r) = s_c) $\wedge$ ($\neg$crit-metropolis (s, s_c, t , n_r) $\Rightarrow$ aceitação (s, s_c, t , n_r) = s)],

faça

p = (crit-metropolis (s, s_c, t , n_r)),

q = (aceitação (s, s_c, t , n_r) = s_c),

q'= (aceitação (s, s_c, t , n_r) = s).

Substituindo na expressão acima, tem-se

(p $\Rightarrow$ q) $\wedge$ ($\neg$ p $\Rightarrow$ q').

Como p $\Rightarrow$ q $\equiv$ $\neg$ p $\vee$ q (teorema da lógica), substituindo tem-se

(p $\Rightarrow$ q) $\wedge$ ($\neg$ p $\Rightarrow$ q') $\equiv$ ($\neg$ p $\vee$ q) $\wedge$ (p $\vee$ q')

e aplicando a lei da distribuição, tem-se

($\neg$ p $\vee$ q) $\wedge$ (p $\vee$ q') $\equiv$ [$\neg$ p $\wedge$ (p $\vee$ q')] $\vee$ [q $\wedge$ (p $\vee$ q')],

($\neg$ p $\vee$ q) $\wedge$ (p $\vee$ q') $\equiv$ ($\neg$ p $\wedge$ p) $\vee$ ($\neg$ p $\wedge$ q') $\vee$ (q $\wedge$ p) $\vee$ (q $\wedge$ q'),

($\neg$ p $\vee$ q) $\wedge$ (p $\vee$ q') $\equiv$ ($\neg$ p $\wedge$ q') $\vee$ (q $\wedge$ p) $\vee$ (q $\wedge$ q'),

($\neg$ p $\vee$ q) $\wedge$ (p $\vee$ q') $\equiv$ [($\neg$ p $\wedge$ q') $\vee$ (q $\wedge$ p)] $\vee$ (q $\wedge$ q')

e pela lei da absorção

($\neg$ p $\vee$ q) $\wedge$ (p $\vee$ q') $\equiv$ ($\neg$ p $\wedge$ q') $\vee$ (q $\wedge$ p) $\equiv$ (q $\wedge$ p) $\vee$ (q' $\wedge$ $\neg$ p).

Então, [(crit-metropolis (s, s_c, t , n_r) $\Rightarrow$ aceitação (s, s_c, t , n_r) = s_c) $\wedge$ ($\neg$crit-metropolis (s, s_c, t , n_r) $\Rightarrow$ aceitação (s, s_c, t , n_r) = s)] é equivalente a

[((aceitação (s, s_c, t , n_r) = s_c) $\wedge$ crit-metropolis (s, s_c, t , n_r)] $\vee$ [((aceitação (s, s_c, t , n_r) = s) $\wedge$ $\neg$crit-metropolis (s, s_c, t , n_r)],

conforme se queria demonstrar.

Anexo A2 Aspectos de Convergência

Os estudos relacionados com os aspectos de convergência do SA levam em consideração, na sua grande maioria, um modelo teórico fundamentado em cadeias de Markov. Assim, é incluído neste capítulo uma pequena introdução da teoria de cadeias de Markov e, a seguir, são apresentados alguns estudos de convergência.

A2.1 Introdução à Teoria de Cadeias de Markov

Uma Cadeia de Markov é um tipo de processo estocástico discreto onde o comportamento futuro depende só do presente, e não do passado. Um estudo mais detalhado pode ser realizado através de [KOV 96] [SIE 92].

A seguir serão apresentadas algumas definições baseadas no trabalho Aarts e Korst [AAR 89].

Uma cadeia de Markov *M* é uma seqüência de experimentos, onde a probabilidade do resultado de um dado experimento depende somente do resultado do experimento anterior. Seja $X(k)$ uma variável estocástica denotando o resultado do k-ésimo experimento, então a probabilidade de transição no k-ésimo experimento para cada par i, j de resultados é definido como

$$P_{ij}(k) = \mathrm{P}\{\mathrm{X}(k) = j \mid \mathrm{X}(k-1) = i\}. \tag{A2.01}$$

A matriz *P(k)* dos elementos dados pela A2.01 é chamada de matriz de transição.

Se $a_i(k)$ denota a probabilidade do resultado i na k-ésima experiência, isto é,

$$a_i(k) = \mathrm{P}\{\mathrm{X}(k) = i\}, \tag{A2.02}$$

Então $a_i(k)$ é dada pela seguinte recursão:

$$a_i(k) = \sum_l a_l(k-1) P_{li}(k). \tag{A2.03}$$

Uma cadeia de Markov é considerada finita se é definida para um conjunto finito de resultados. Uma cadeia de Markov é homogênea se as probabilidades de transição associadas são independentes do número do experimento e não-homogênea se as probabilidades de transição associadas são dependentes do número do experimento.

Diz-se que um vetor a é estocástico se seus componentes a_i satisfazem as seguintes condições:

$$a_i \geq 0, \text{ para todo } i; \quad \sum_i a_i = 1. \tag{A2.04}$$

Uma matriz P é estocástica se seus componentes P_{ij} satisfazem as seguintes condições:

$$P_{ij} \geq 0, \text{ para todo } i, j; \quad \sum P_{ij} = 1, \text{ para todo } i. \tag{A2.05}$$

Uma cadeia de Markov com matriz de transição P é *irredutível*, se para cada par de soluções $i, j \in S$ existe uma probabilidade positiva de encontrar j a partir de i em um número finito de experiências, isto é,

$$\forall i, j \quad \exists n \geq 1 : \left(P^n\right)_{ij} > o. \tag{A2.06}$$

Uma cadeia de Markov com matriz de transição P é *aperiódica*, se para cada solução $i \in S$ o maior divisor comum $mdc(D_i) = 1$, onde o conjunto D_i consiste de todos os inteiros $n > 0$, com

$$\left(P^n\right)_{ii} > 0 \tag{A2.07}$$

e o inteiro $mdc(D_i)$ é chamado o *período* da solução i.

Portanto, aperiodicidade requer que todas as soluções tenham período 1.

Uma cadeia de Markov com matriz de transição P é aperiódica se

$$\exists j \in S : P_{jj} > 0. \tag{A2.08}$$

A2.2 Análise de Convergência

Aarts e Korst [AAR 89] apresentam um dos enfoques mais simples e completos sobre a convergência de um algoritmo SA e referenciam uma série de outros estudos sobre o assunto. A seguir, serão apresentados os aspectos básicos deste enfoque. O algoritmo SA analisado segue os passos descritos pela figura A2.1:

```
Procedimento SA;
(s0, L0) ← inicializa_solução;
t0 ← aqueça;
k ← 0;
i ← s0;
ciclo_externo: enquanto não critério-término faça
  l ← 0;
  ciclo_interno: enquanto l < Lk faça
                 gera  j vizinha de i;
                    se f(j) ≤ f(si) então i ← j;
                    senão
                    se exp (f(i) - f(j))/t) > aleatório[0, 1)
                       então i ← j
                    l ← l + 1;
  fim_enquanto;
  k ← k + 1;
  Lk ← calcula-comprimento;
  tk ← calcula-temperatura;
fim_enquanto;
```

FIGURA A2.1 – Algoritmo SA

Equilíbrio Estatístico

Considerando a analogia com os conceitos da física estatística (seção 3.1.2) é estabelecida a seguinte conjetura que associa uma distribuição estacionária à execução do algoritmo SA para resolução de uma instância de um problema de otimização combinatória, sendo S o espaço de soluções, f a função energia e $i, j \in S$:

Conjetura A2.01 [AAR 89]: Dado uma instância de um problema de otimização combinatória (S, f) e uma estrutura de vizinhança apropriada então, após um número suficientemente grande de transições a um valor fixo de temperatura t, aplicando a probabilidade de aceitação de (3.03), o algoritmo SA encontrará uma solução $i \in S$ com uma probabilidade

$$\mathrm{P}_t\{\mathrm{X} = i\} \overset{def}{=} q_i(t) = \frac{1}{\mathrm{N}_o(t)} \exp\left(-\frac{f(i)}{t}\right), \qquad \text{(A2.09)}$$

onde X é uma variável estocástica denotando a solução corrente obtida pelo algoritmo e

$$\mathrm{N}_o(t) = \sum_{j \in S} \exp\left(-\frac{f(j)}{t}\right) \qquad \text{(A2.10)}$$

denota uma constante de normalização.

A distribuição de probabilidade acima definida é chamada de estacionária ou de equilíbrio e é equivalente a distribuição de Boltzmann (3.01). A constante de normalização é equivalente a função de partição (3.02).

Uma estrutura de vizinhança apropriada é estabelecida pela definição (2.3) e, ainda, deve satisfazer a propriedade de simetria, ou seja, $j \in S_i \Leftrightarrow i \in S_j$. A especificação de uma estrutura de vizinhança apropriada é dependente do problema.

O corolário abaixo estabelece a garantia de convergência assintótica do algoritmo SA para o conjunto de soluções globalmente ótimas desde que a distribuição estacionária seja alcançada a cada valor de t.

Corolário A2.01 [AAR 89]: Considerando a distribuição estacionária $q_i(t)$ dada pela (A2.09), então

$$\lim_{t \downarrow 0} q_i(t) \stackrel{def}{=} q_i^* = \frac{1}{|S_{opt}|} \chi_{(S_{opt})}(i), \tag{A2.11}$$

onde S_{opt} denota o conjunto de soluções ótimas globais e onde, sendo A e $A' \subset A$ dois conjuntos, $\chi_{(A')} : A \rightarrow \{0, 1\}$ é a função característica do conjunto A' definida como $\chi_{(A')}(a)=1$ se $a \in A'$, senão 0.

A prova deste corolário pode ser encontrada em [AAR 89].

Considerando a distribuição estacionária $q_i(t)$ dada pela (A2.09) é possível definir outras quantidades para um problema de otimização combinatória, estendendo a analogia com física estatística. Estas quantidades são:

custo esperado em equilíbrio, definido como

$$\mathrm{E}_t(f) \stackrel{def}{=} \langle f \rangle_t = \sum_{i \in S} f(i) \mathrm{P}_t\{\mathrm{X} = i\} = \sum_{i \in S} f(i) q_i(t); \tag{A2.12}$$

quadrado do custo esperado em equilíbrio, definido como

$$\mathrm{E}_t(f^2) \stackrel{def}{=} \langle f^2 \rangle_t = \sum_{i \in S} f^2(i) \mathrm{P}_t\{\mathrm{X} = i\} = \sum_{i \in S} f^2(i) q_i(t); \tag{A2.13}$$

variância do custo esperado em equilíbrio, definido como

$$Var_t(f) \stackrel{def}{=} \sigma_t^2 = \sum_{i \in S} (f(i) - \mathrm{E}_t(f))^2 \mathrm{P}_t\{\mathrm{X} = i\} = \sum_{i \in S} (f(i) - \langle f \rangle_t)^2 q_i(t) = \langle f^2 \rangle_t - \langle f \rangle_t^2; \tag{A2.14}$$

entropia em equilíbrio, definido como

$$\mathsf{S}_t = -\sum q_i(t)\ln q_i(t). \tag{A2.15}$$

A entropia pode ser vista como uma medida natural da quantidade de desordem ou de informação em um sistema.

A partir destas definições pode ser derivado o seguinte corolário:

Corolário A2.2 [AAR 89]: Considerando a distribuição estacionária $q_i(t)$ dada pela (A2.09) as seguintes relações são estabelecidas:

$$\frac{\partial}{\partial t}\langle f\rangle_t = \frac{\sigma_t^2}{t^2} \tag{A2.16}$$

$$\frac{\partial}{\partial t}\mathsf{S}_t = \frac{\sigma_t^2}{t^3}. \tag{A2.17}$$

A prova pode ser encontrada em [AAR 89].

As expressões definidas entre (A2.12) e (A2.17) são derivadas de expressões bem conhecidas em Física Estatística e são importantes para análise dos mecanismos envolvidos no equilíbrio de grandes ensembles físicos. Usando estas expressões um outro corolário pode ser estabelecido:

Corolário A2.3 [AAR 89]: Considerando a distribuição estacionária $q_i(t)$ dada pela (A2.09) e as expressões dadas por (A2.12) a (A2.17) tem-se

$$\lim_{t\to\infty}\langle f\rangle_t \overset{def}{=} \langle f\rangle_\infty = \frac{1}{|S|}\sum_{i\in S} f(i) \quad , \tag{A2.18}$$

$$\lim_{t\downarrow 0}\langle f\rangle_t = f_{opt} \tag{A2.19}$$

$$\lim_{t\to\infty}\sigma_t^2 \overset{def}{=} \sigma_\infty^2 = \frac{1}{|S|}\sum_{i\in S}\left(f(i) - \langle f\rangle_\infty\right)^2 \quad , \tag{A2.20}$$

$$\lim_{t\downarrow 0}\sigma_t^2 = 0 \quad , \tag{A2.21}$$

$$\lim_{t\to\infty}\mathsf{S}_t \overset{def}{=} \mathsf{S}_\infty = \ln|S| \quad , \tag{A2.22}$$

e

$$\lim_{t \downarrow 0} \mathsf{S}_t \overset{def}{=} \mathsf{S}_0 = \ln\left|S_{opt}\right| \quad . \tag{A2.23}$$

Prova: A prova pode ser encontrada em [AAR 89].

Na expressão (A2.23) ,que estabelece a entropia quando a temperatura tende a zero, assumindo que haja um único estado fundamental tem-se

$$\mathsf{S}_0 = \ln|1| = 0 \quad , \tag{A2.24}$$

que corresponde a terceira lei da termodinâmica.

Lembrando que a entropia pode ser interpretada como uma medida natural de ordem de um sistema físico:

valores altos para entropia correspondem a caos; e

valores baixos para entropia correspondem a ordem (segundo Toda, Kubo e Saitô,1983 citado por [AAR 89].

A definição de entropia similar a dada pela (A2.15) é conhecida na Teoria da Informação, onde é vista como uma medida da quantidade de informação de um sistema [SHA 48].

No caso do SA, ou para otimização em geral, a entropia pode ter o significado de uma medida quantitativa do grau de otimalidade [AAR 89]. O uso do conceito de entropia é melhor detalhado nos trabalhos de Rodrigues e Anjo [ROD 93] [ANJ 98].

Considerando as expressões (A2.16) e (A2.17) observa-se que na execução do algoritmo SA o custo esperado e a entropia decrescem monotonicamente para f_{opt} e $\ln|S_{opt}|$, desde que o equilíbrio seja alcançado a cada valor de temperatura.

Ainda é possível estabelecer que a probabilidade de encontrar uma solução ótima aumenta monotonicamente com o decréscimo de t e que para cada solução não ótima, há um valor positivo t'_i, tal que para $t < t'_i$, a probalidade de encontrar esta solução decresce monotonicamente com o decréscimo de t. Isso pode ser estabelecido considerando o seguinte corolário:

Corolário A2.4 [AAR 89]: Seja um problema de otimização combinatória denotado por (S, f) com $S_{opt} \neq S$ e seja a distribuição estacionária associada ao algoritmo SA denotada por $q_i(t)$ e dada pela (A2.09), então tem-se

$\forall i \in S_{opt}$:

$$\frac{\partial}{\partial t} q_i(t) < 0, \tag{A2.25}$$

$\forall i \notin S_{opt}$ e $f(i) \geq \langle f \rangle_\infty$:

$$\frac{\partial}{\partial t} q_i(t) > 0, \tag{A2.26}$$

$\forall i \notin S_{opt}$ e $f(i) < \langle f \rangle_\infty$, $\exists$ t'$_i > 0$:

$$\frac{\partial}{\partial t} q_i(t) > 0 \text{ se } t < t_i^{'}, \tag{A2.27}$$

$$\frac{\partial}{\partial t} q_i(t) = 0 \text{ se } t = t_i^{'}, \tag{A2.28}$$

$$\frac{\partial}{\partial t} q_i(t) < 0 \text{ se } t > t_i^{'}. \tag{A2.29}$$

Prova: A prova pode ser encontrada em [AAR 89].

SA como cadeia de Markov

O algoritmo SA pode ser modelado como uma cadeia de Markov finita, onde um experimento corresponde a uma transição, o conjunto de experimentos é dado pelo conjunto finito de soluções e o resultado de um dado experimento depende somente do resultado do experimento anterior, ou seja o comportamento futuro depende somente do estado corrente e não é considerado como chegou-se a este estado ("memorylessness property" [STO 98]).

Como visto anteriormente, uma transição no algoritmo SA corresponde à aplicação de um mecanismo de geração seguido da aplicação de um critério de aceitação. A seguir, serão definidas probabilidades de transição, geração e aceitação levando em consideração o modelo original do algoritmo SA, o qual segue a analogia com o processo físico anteriormente descrito. Estas definições podem ser usadas virtualmente para qualquer problema de otimização combinatória [AAR 89].

Definição A2.01 [AAR 89]: Seja uma instância de um problema de otimização combinatória denotada por (S, f). Então a probabilidade de transição para o algoritmo SA pode ser definida como:

$$\forall i, j \in S : P_{ij}(k) = P_{ij}(t_k) = \begin{cases} G_{ij}(t_k) A_{ij}(t_k) & \text{se} \quad i \neq j \\ 1 - \sum_{'l \in S, l \neq i} P_{il}(t_k) & \text{se} \quad i = j \end{cases} \tag{A2.30}$$

onde

$G_{ij}(c_k)$ denota a probabilidade de geração, isto é, a probabilidade de gerar a solução j a partir da solução i e é definida como

$$\forall i, j \in S : G_{ij}(c_k) = G_{ij} = \frac{1}{\Theta} \chi_{(S_i)}(j) \tag{A2.31}$$

onde $\Theta = |S_i|$, para todo $i \in S$; e

$A_{ij}(c_k)$ denota a probabilidade de aceitação, isto é, a probabilidade de aceitar j uma vez gerada a partir de i, definida como

$$\forall i, j \in S : A_{ij}(t_k) = \exp\left(- \frac{(f(j) - f(i))^+}{t_k} \right) \tag{A2.32}$$

onde, para todo $a \in \Re$, $a^+ = a$ se $a > 0$, e senão $a^+ = 0$.

As probabilidades de geração e aceitação assim definidas são probabilidades condicionais e as matrizes correspondentes são denominadas matriz de geração e matriz de aceitação, respectivamente.

Observa-se que as probabilidades de geração são escolhidas independente do parâmetro de controle t_k e uniformemente na vizinhança S_i, assumindo que todas as vizinhanças são do mesmo tamanho, isto e', $|S_i| = \Theta$, para todo $i \in S$ As probabilidades de aceitação são baseadas no critério de aceitação, no caso, critério de Metropolis.

Observa-se ainda que as matrizes de transição e de geração são estocásticas e a matriz de aceitação não é estocástica.

Convergência Assintótica

A probabilidade do algoritmo SA encontrar uma solução ótima após um número possivelmente grande de experiências é igual a 1 se:

$$\mathrm{P}\{\mathrm{X}(k) \in S_{opt}\} = 1 \tag{A2.33}$$

Nas seções seguintes será provado que sob determinadas condições o algoritmo SA converge assintoticamente para o conjunto de soluções ótimas, ou seja:

$$\lim_{k\to\infty} \mathrm{P}\{\mathrm{X}(k) \in S_{opt}\} = 1 \tag{A2.34}$$

Distribuição Estacionária

A prova da convergência assintótica do algoritmo SA é baseada na existência de uma única distribuição estacionária. Esta distribuição estacionária existe somente quando a cadeia de Markov associada com o algoritmo obedece determinadas condições. Aarts e Korst [AAR 89] analisam estas condições e provam que, para uma cadeia de Markov homogênea associada ao algoritmo SA, a distribuição estacionária assume a forma exponencial dada pela (A2.09), ou seja, provam a conjetura (A2.01).

Nas definições seguintes é assumido que o valor do parâmetro de controle é independente de k, isto é, $t_k = t$ para todo k. Assim, tem-se $P(k) = P$, para todo k, o que corresponde a uma cadeia de Markov homogênea.

Definição 3.08 [AAR 89] A distribuição estacionária de uma cadeia de Markov finita e homogênea com matriz de transição P é definida como o vetor $\boldsymbol{q}$, cujo i-ésimo componente é dado por

$$q_i = \lim_{k\to\infty} \mathrm{P}\{\mathrm{X}(k) = i \mid \mathrm{X}(0) = j\}, \quad \text{para} \quad \text{qualquer} \quad j. \tag{A2.35}$$

Se esta distribuição estacionária $\boldsymbol{q}$ existe tem-se

$$\lim_{k\to\infty} a_i(k) = \lim_{k\to\infty} \mathrm{P}\{\mathrm{X}(k) = i\} = \lim_{k\to\infty} \sum_j \mathrm{P}\{\mathrm{X}(k) = i \mid \mathrm{X}(0) = j\}\mathrm{P}\{\mathrm{X}(0) = j\} = q_i \sum_j \mathrm{P}\{\mathrm{X}(0) = j\} = q_i \tag{A2.36}$$

Desta maneira, a distribuição estacionária é a distribuição de probabilidade das soluções após um número infinito de experiências.

Além disso, tem-se

$$\mathbf{q}^T = \lim_{k\to\infty} a^T(0)\prod_{l=1}^{k} P(l) = \lim_{k\to\infty} a^T(0)P^k = \lim_{k\to\infty} a^T(0)P^{k-1}P = \lim_{l\to\infty} a^T(0)P^l P = \mathbf{q}^T P \tag{A2.37}$$

então, q é o autovetor de P com autovalor 1. Claramente, no caso do algoritmo SA, como P depende de t, $\mathbf{q}$ depende de t, isto é, $\mathbf{q} = \mathbf{q}(t)$.

Teorema A2.01 (Feller, 50 citado por [AAR 89]: Seja P uma matriz de transição associada a uma cadeia de Markov homogênea e finita e seja esta cadeia irredutível e aperiódica. Então existe um vetor estocástico $\mathbf{q}$ no qual cada componente q_i é unicamente determinado pela seguinte equação (equação de balanceamento global):

$$\sum_j q_j P_{ji} = q_i, \quad \text{para qualquer } i. \tag{A2.38}$$

Claramente, o vetor q é a distribuição estacionária da cadeia de Markov, porque satisfaz (A2.37).

Lema A2.01 [AAR 89]: Seja P uma matriz de transição associada a uma cadeia de Markov homogênea e finita e seja esta cadeia irredutível e aperiódica. Então uma dada distribuição é a distribuição estacionária da cadeia de Markov se seus componentes satisfazem a seguinte equação(equação de balanceamento detalhada):

$$q_i P_{ij} = q_j P_{ji}, \quad \text{para qualquer } i, j \in S. \tag{A2.39}$$

Teorema A2.02 [AAR 89]: Faça (S, f) denotar uma instância de um problema de otimização combinatória e P(t) a matriz de transição associado com o algoritmo SA definido por (A2.30), (A2.31), e (A2.32). Além disso, faça com que a seguinte condição seja satisfeita:

$$\begin{aligned} &\forall i, j \in S\ \exists p \geq 1, \exists l_0, l_1, \ldots\ldots l_p \in S, \\ &com \quad l_0 = i, l_p = j, \quad e \\ &G_{l_k l_{k+1}} > 0, k = 0,1,2,\ldots\ldots, p-1. \end{aligned} \tag{A2.40}$$

Então, a cadeia de Markov tem uma distribuição estacionária $\mathbf{q}$(T), cujos componentes são dados pela

$$q_i(t) = \frac{1}{\mathrm{N}_o(t)} \exp\left(-\frac{f(i)}{t}\right), \text{ para qualquer } i \in S, \tag{A2.41}$$

onde

$$\mathrm{N}_o(t) = \sum_{j \in S} \exp\left(-\frac{f(j)}{t}\right). \tag{A2.42}$$

A prova é feita mostrando que a cadeia de Markov assim definida é irredutível e aperiódica e que a distribuição estacionária definida pela (A2.41) e (A2.42) é um vetor estocástico e cujos componentes satisfazem a equação detalhada de balanceamento (A2.39).

A distribuição estacionária assim definida é idêntica a obtida anteriormente (por conjetura) a partir da analogia física.

Viu-se anteriormente que

$$\lim_{t \downarrow 0} \mathbf{q}(t) = \mathbf{q}^*, \tag{A2.43}$$

onde os componentes de $\mathbf{q}^*$ são dados por

$$q_i^* = \frac{1}{|S_{opt}|} \chi_{(S_{opt})}(i), \tag{A2.44}$$

Portanto, obtém-se

$$\lim_{t \downarrow 0} \lim_{k \to \infty} \mathrm{P}_t\{\mathrm{X}(k) = i\} = \lim_{t \downarrow 0} q_i(t) = q_i^*, \tag{A2.45}$$

ou

$$\lim_{c \downarrow 0} \lim_{k \to \infty} \mathrm{P}_c\{\mathrm{X}(k) \in S_{opt}\} = 1. \tag{A2.46}$$

O que basicamente reflete a garantia de que o algoritmo encontra assintoticamente uma solução ótima.

Aarts e Korst provam que a garantia de convergência assintótica existe também para probabilidades de transição definidas como uma classe mais genérica do que a estabelecida anteriormente e citam outros autores.

A convergência assintótica, como estabelecida nesta seção, indica que o algoritmo SA requer um número infinito de transições para aproximar uma distribuição estacionária. Isso implica que o algoritmo assim definido pode requerer uma seqüência de cadeias de Markov homogênas infinitamente longa a valores decrescentes do parâmetro temperatura.

Assim, uma alternativa é descrever o algoritmo usando um modelo associado de cadeia de Markov não homogênea. Aarts e Korts [AAR 89] estabelecem as condições de convergência e os aspectos de ergodicidade para este modelo. Entretanto, a conclusão dos autores é:

(a) um algoritmo SA pode ser considerado um algoritmo exato somente se for aceitável um número infinito de transições;

(b) a aproximação assintótica dentro de um limite arbitrário requer para a maioria dos problemas um número de transições maior que o tamanho do espaço de solução, resultando em tempo exponencial para o algoritmo.

(c) o comportamento assintótico para o algoritmo pode ser aproximado em tempo polinomial, usando prescrições de resfriamento adequadas como as descrita na seção 3.A2.2.

SA como um MCMC

Estudos mais recentes analisam o comportamento do SA como um algoritmo do tipo Monte Carlo Markov Chain, MCMC (seção 3.1.1).

Entre estes estudos pode ser mencionado:

(a) Jerrum e Sinclair [JER 97] - analisam os algoritmos do tipo Monte Carlo Markov Chain de uma forma geral, incluindo o algoritmo de Metropolis modelado como uma cadeia de Markov irredutível, aperiódica, ergódica e satisfazendo a equação detalhada de balanceamento e analisam com mais profundidade a aplicação do algoritmo para o problema de encontrar a cobertura máxima de grafos. Quanto ao SA observam que o mesmo pode ser analisado como uma cadeia de Markov não homogênea, salientando que mesmo a questão da convergência assintótica não é trivial;

(b) Rosenthal e outros [ROB 97] [ROS 99] - apresentam aspectos práticos e teóricos relacionados com a convergência dos algoritmos tipo MCMC, ampliando a abrangência para cadeias de Markov em espaços de soluções gerais (não só espaços finitos ou contáveis).

A2.3 Convergência e Complexidade

São poucos os estudos relacionados diretamente com a complexidade do algoritmo SA, mas a complexidade está relacionada com os estudos de convergência.

A complexidade pode ser analisada para a implementação do algoritmo a um problema específico ou então estabelecida para um caso mais geral usando um determinado tipo de prescrição de resfriamento, ou ainda, genericamente como na seção 3.5.

Sasaki e Hajek [SAS 88] analisam a complexidade de tempo média para o problema de cobertura máxima de grafos para o SA tradicional e uma variação onde a temperatura é uma constante arbitrária. Para o SA tradicional não encontraram nenhuma alternativa que resultasse em tempo médio polinomial. Para a variação do SA mostraram a possibilidade de encontrar coberturas máximas aproximadas em tempo polinomial.

Aarts e Korst [AAR 89] apresentam um estudo para um caso geral usando a prescrição de resfriamento que foi denotada como prescrição de resfriamento de AARTS na seção 3.4.2. O resultado deste estudo, no entanto, pode ser visto como uma estimativa do tempo de computação, pois não avalia a complexidade em função do tamanho da entrada (no caso, a entrada é a instância do problema). A seguir são apresentadas as características desta implementação do algoritmo SA que resulta em uma aproximação em tempo polinomial e considerando o que apresentado para convergência do algoritmo SA na seção 4.1:

Esta prescrição de resfriamento (prescrição de resfriamento de AARTS na seção 3.4.2), baseada em fatos empíricos, mas com fundamentação teórica, é especificada através de:

(a) Seqüência finita de valores descendentes do parâmetro de controle t;

(b) Valor inicial t_0 para o parâmetro de controle;

(c) Uma função para decrescer o parâmetro de controle;

(d) Um valor final para o parâmetro de controle especificado por um critério de parada;

(e) Um número finito de transições a cada valor do parâmetro de controle, ou seja, um comprimento finito para cada cadeia de Markov homogênea.

Um conceito de quase-equílíbrio é estabelecido,como:

Faça L_k denotar o comprimento da k-ésima cadeia de Markov e t_k o valor correspondente do parâmetro de controle. Um quase-equilíbrio é alcançado se $a(L_k, t_k)$, isto é, a distribuição de probabilidades das soluções depois de L_k experimentos da cadeia de Markov, é 'suficientemente próxima' a $q(t)$, distribuição estacionária a este valor de t_k, definido pelas expressões (A2.40) e (A2.41), isto é,

$$\left\| a\left(L_k, t_k\right) - q\left(t_k\right) \right\| < \varepsilon \tag{A2.47}$$

para um valor positivo especificado para ε.

É demonstrado em [AAR 89] que o requerimento de que a expressão (A2.47) seja satisfeita para valores pequenos de ε implica na necessidade de um número de transições que é quadrática em relação ao tamanho do espaço de soluções, o que resulta para a maioria dos problemas de otimização combinatória em um tempo exponencial de execução para o algoritmo SA.

Na prática, é necessário quantificar de forma menos rígida o que é quase-equilíbrio. Isso leva a diferentes interpretações do conceito de quase-equilíbrio, resultando em uma variedade de prescrições de resfriamento.

O número total de iterações são influenciados pela forma de realizar o decréscimo do parâmetro de controle para a prescrição de resfriamento em estudo e pelo número de iterações a cada cadeia de Markov. Assim, considerando a condição de quase-equilíbrio dada pela (A2.47) e que o processo de quase-equilíbrio é restabelecido durante a resolução a cada valor de t, uma vez que o mesmo tenha sido alcançado com t_0, é assumido que esta expressão pode ser substituída por:

$$\forall k \geq 0 : \left\| a\left(L_k, t_k\right) - q\left(t_k\right) \right\| < \varepsilon \tag{A2.48}$$

Como para dois valores sucessivos do parâmetro de controle é esperado distribuições estacionárias 'próximas', isto pode ser quantificado por:

$$\forall i \in S : \frac{1}{1+\delta} < \frac{q_i\left(t_k\right)}{q_i\left(t_{k+1}\right)} < 1+\delta, \quad \text{para } k = 0,1,\ldots\ldots, \tag{A2.49}$$

Para um valor positivo arbitrário de δ, que pode estar relacionado com ε.

A partir destas condições e, ainda, considerando que a região próxima a solução ótima pode ser aproximada por uma distribuição exponencial e a região próxima a solução de custo aproximado à média de todas as soluções pode ser aproximada por uma distribuição normal, chega-se a expressão de decréscimo de temperatura

$$t_{k+1} = \frac{t_k}{1 + \dfrac{t_k . \ln\left(1+\delta\right)}{3\sigma_{t_k}}}, \quad k = 0,1,\ldots\ldots \tag{A2.50}$$

O aspecto do comprimento da cadeia de Markov pode ser deduzido baseado nas seguintes considerações: pelo conceito de quase-equilíbrio, estabelecido acima, um número pequeno de transições deve ser executado para restabelecer o quase-equilíbrio a cada valor de temperatura; a suposição do que é este "pequeno" pode ser especificada considerando que deve haver uma probabilidade grande do algoritmo visitar ao menos a maior parte das soluções vizinhas de uma dada solução.

Na prescrição de resfriamento em estudo o comprimento da cadeia de Markov é especificado com valor igual ao tamanho da vizinhança e é demonstrado que, para vizinhanças suficientemente grandes (tamanho da vizinhança maior que 100), a cada três cadeias de Markov subsequentes é possível visitar aproximadamente toda a vizinhança de uma solução.

O tempo de computação para esta prescrição de resfriamento é estabelecida pelo teorema (A2.03) que estabelece um limite superior para o número de cadeias de Markov:

Teorema A2.03 [AAR 89]: Seja a função de decréscimo de temperatura dada pela expressão

$$t_{k+1} = \frac{t_k}{1+\alpha_k t_k}, \text{ para } k = 0,1,... \tag{A2.51}$$

onde

$$\alpha_k = \frac{\ln(1+\delta)}{3\sigma_{t_k}}, \text{ para } k = 0,1,... \tag{A2.52}$$

e seja K o primeiro inteiro para o critério de término seja satisfeito, ou seja:

$$\frac{t_K}{\langle f \rangle_\infty} \left| \frac{\partial \langle f \rangle_t}{\partial t} \right|_{t=t_K} < \varepsilon_s . \tag{A2.53}$$

Então, se obtém

$$K = O(\ln|S|), \tag{A2.54}$$

sob certas condições da derivação com relação a t do valor esperado $\langle f \rangle_t$ e da entropia S_t.

A prova do teorema pode ser encontrada em [AAR 89], mas basicamente consiste de duas etapas: na primeira o número total de iterações K é expresso em função de t_K e, na segunda, um limite inferior é derivado para t_K.

Assim, o tempo de computação total satisfaz a seguinte relação:

$$\mathrm{T} = O(w(L(\ln|S|))) \tag{A2.55}$$

onde w o tempo computacional referente a uma transição, portanto, depende da instância do problema, L denota o comprimento de cada cadeia de Markov individual, $\ln|S|$ um limite superior para o número de cadeias de Markov.

Portanto, para os problemas onde for possível estabelecer w e L em tempo polinomial (segundo Aarts e Korst [AAR 89] isso é possível para uma boa parte dos problemas de otimização combinatória) e se $\ln|S|$ for polinomial com relação ao tamanho da instância, o algoritmo SA executa em tempo polinomial.

Bibliografia

[AAR 85] AARTS, E. H L.;LAARHOVEN, P. J. M. VAN. A new-Polinomial Time Cooling Schedule. In: IEEE International Conference on Computer-Aided Design, ICCAD, 1985. **Proceedings...** Santa Clara: IEEE, 1985. p. 206-208.

[AAR 89] AARTS, E.H. L.; KORST, J. **Simulated Annealing and Boltzmann Machines**. Essex: John Wiley & Sons, 1989. 272p.

[ABR 97] ABRAMSON, D.; RANDALL, M. **A Simulated Annealing Code for General Integer Linear Programs**. Disponível por e-mail em davida@dgs.monash.edu.au e M.Randall@eas.gu.edu.au. (1997)

[AGJ 95] AGUILAR, J. A Hybrid Approach Based on Genetic Algorithms and Simulated Annealing for Combinatorial Optimization Problems**.** In: Seminário Integrado de Software e Hardware, SEMISH, 22., 1995. **Anais...** Porto Alegre: Instituto de Informática da UFRGS, 1995. p. 1401-1411.

[AGU 96] AGUIAR, M. S. **Tratamento de Problemas NP-Completo** trabalho individual. Porto Alegre: CPGCC da UFRGS, 1996. (TI - 570).

[AGU 98] AGUIAR, M. S.; TOSCANI, L. V. **Análise Formal da Complexidade de Algoritmos Genéticos**. Porto Alegre: CPGCC da UFRGS, 1998. Dissertação de Mestrado.

[AHO 74] AHO, A. V.; HOPCROFT, J. T., ULLMAN J. D. **The Design and Analisys of Computer Algorithms**. Readings: Addison Wesley, 1974, 470 p

[ALI 94] ALIMONTI, P. New Local Search Approximation Techniques for Maximum Generalized Satisfiability Problems. In: Bonuccelli, M.; Crescenzi, P.; Petreschi, R. (Eds). **Algorithms and Complexity**. Berlin: Springer-Verlag, 1994. p. 40-53. (Lectures Notes in Computer Science, n. 778).

[ALI 97] ALIMONTI, P.; KANN, V. Hardness of Approximating Problems on Cubic Graphs. In: Giancarlo, B., Bovet, D. P., Battista, G. D. (Eds) **Algorithms and Complexity**. Berlin: Springer-Verlag, 1997. p. 288-298. (Lectures Notes in Computer Science, n. 1203).

[ALL 96] ALLEN, R. C. et al. Monte Carlo Algorithms and ``Random" Numbers**.** In: **Computacional Science Education Project.** USA: Energy Departament, 1996 Disponível por WWW em http://csep1.phy.ornl.gov/. (Nov 1999)

[AND 98] ANDRZEJAK, A. Introduction to Randomized Algorithms. In: Mayr, W., Promel, H. J., Steger, A. (Ed.) **Lectures on Proof Verification and Approximation Algorithms**. Berlim: Springer-Verlag, 1998. p. 30-39. (Lectures Notes in Computer Science, n. 1367)

[AND 99] ANDREATTA, A. A.; CARVALHO, S. E. R.; RIBEIRO, C. C. **A Framework for Local Search Heuristics for Combinatorial Otimization Problems**. Disponível por WWW em http://www-di.inf.puc-rio/~celso/publicações.html. (Ago 1999)

[ANJ 98] ANJO, A. J. B.; RODRIGUES, M. R. D. **Entropia, Seqüências Típicas e Máxima Entropia**. Aveiro: Universidade de Aveiro, 1998. Disponível por WWW em http://www.mat.ua/batel/maxentropia.html. (Dez 1998)

[AZE 92] AZENCOTT, R. **Simulated Annealing Parallelization Techniques**. New York:

John Wiley & Sons, 1992. 242p.

[BOV 94] BOVET, D. P.; CRESCENZI, P. **Introduction to the Theory of Complexity**. New York: Prentice Hall, 1994. 282p.

[BRD 89] BRUSCHI, D.; JOSEPH, D.; YOUNG, P. A Structural Overview of NP Optimization Problems. In: International Symposium on Optimal Algorithms, 2., 1989. **Proceedings...** Berlin: Springer-Verlag, 1989. p. 205-228. (Lectures Notes in Computer Science, n. 401).

[BRU 95] BRUCKER, P. **Schedulling Algorithms**. Berlin: Springer - Verlag, 1995. 326p.

[CAM 94] CAMPELL, R. E.; MACULAN, N. **Algoritmos e Heurísticas** Desenvolvimento e Avaliação de Performance. Niterói: Universidade Federal Fluminense, 1994. 227p.

[CAR 98] CARLSON, S. **Algorithm of the Gods**. Disponível por WWW em http://earth.thesfhere.com/SAS/SciAm/SimAnneal/SimAnneal.html. (set 1998).

[CAR 99] CARLSON, S. **Algorithm of the Gods**. Disponível por WWW em http://www.sciam.com/0397issue/0397amsci.html. (dez 1999).

[CEP 98] CEPERLEY, D. Generation of Randon Numbers. In: **Atomic-Scale Simulation**. Urbana: University of Illinois, 1998. Disponível por WWW em http:\\bguy.mse.uiuc.edu/matse390.

[CRE 97] CRESCENZI, P.; KANN, V. **A compendium of NP optimization Problems**. Disponível por WWW em http://www.nada.kth.se/theory/problemlist.html. (dez 1998).

[ELM 97] ELMOHAMED, S.; CODDINGTON, P.; FOX, G. A Comparison of Annealing Techniques for Academic Course Scheduling. **NPAC Technical Report SCCS-777**, Syracuse, Syracuse University, 1997.

[FOX 94] FOX, B. **Simulated Annealing**: Folklore, Facts, and Directions. Denver: Universidade de Colorado, 1994. Disponível por e-mail em bfox@cudenver.bitnet. (Mar/98)

[FRO 98] FROST, R. **SDSC EBSA C Library Documentation**. San Diego: San Diego Super Computer Centre e General Atomics, 1994. Disponível por ftp em ftp.sdsc.edu, diretório pub/sdsc/parallel/ParTools (Mar1998).

[GAR 79] GAREY, R. G.; JOHNSON, D. S.**Computers and Intractability - A guide to the Theory of NP-Completeness**. San Francisco: W. H. Freeman and Company, 1979. 340p.

[GEH 99] GEHRING, W. et al. **Correlation Structure of Landscapes of NP-complete Optimization Problems at Finite Temperatures**. Odense: Odense University, 1999. Disponível por WWW em http://www.sdu.dk.(Nov 1999).

[GRE 89] GREENE, J.W.; SUPOWIT, K.J. Simulated Annealing Without Rejected Moves. **IEEE Transactions on CAD**, New York, v. 3, n. 1, p. 221-228, 1989.

[GRE 99] GREENBERG, H. J. **Mathematical Glossary**. Denver: University of Colorado, 1999. Disponível por WWW em http://www.cudenver.edu/~hgreenbe/glossary/glossary.html. (Mar 1999).

[HEL 88] HELMAN, P. An Álgebra for Search Problems and Their Solutions. In: Kanal, L.; Kumar, V. (Eds.) **Search in Artificial Intelligence**. New York: Springer-Verlag, 1988. p. 29-90.

[HOR 78] HOROWITZ, E.; SAHNI, S. **Fundamentals of Computers Algorithms**. California: Computer Science Press, 1978. 626p.

[ING 93] INGBER, A. L. **Simulated Annealing: Practice versus Theory**. Oxford: Journal of Mathematical Computer Modelling, v. 18, n. 11, p. 29-57, 1993.

[ING 99] INGBER, A. L. **Adaptive Simulated Annealing (ASA)**. Disponível por ftp em ftp.alumni.calctech.edu, diretório /pub/ingber/ASA. (Mar 1999)

[IZQ 97] IZQUIERDO, V. B.; TOSCANI, L. Introdução aos Métodos Heurísticos para Resolução de Problemas de Otimização Combinatorial: trabalho individual. Porto Alegre: CPGCC da UFRGS, 1997. 41p. (TI-720)

[IZQ 98] IZQUIERDO, V. B.; TOSCANI, L. V. Aspectos da Fundamentação Teórica do Simulated Annealing. In: Semana Acadêmica do CPGCC, 3., 1998. **Anais...** Porto Alegre: p. 193-196.

[JAN 98] JANSEN, T. Introduction to the Theory of Complexity and Approximation Algorithms**.** In: Mayr, W., Promel, H. J., Steger, A. (Eds) **Lectures on Proof Verification and Approximation Algorithms**. Berlim: Springer-Verlag, 1998. p. 5-28. (Lectures Notes in Computer Science, n.1367).

[JER 97] JERRUM, M.; SINCLAIR, A. The Markov Chain Monte Carlo Method: An Approach to Approximate Counting and Integration. In: HOCHBAUM, D. S. (Ed.). **Approximation Algorithms for NP-Hard Problems**. Boston: PWS, 1997. p. 483-518.

[JOH 89] JOHNSON, D. S. et al. Optimization by Simulated Annealing: An Experimental Evaluation; Part I, Graph Partitioning. **Operations Research**, Baltimore, v. 37, n. 6, p. 865-891, 1989.

[JOH 91] JOHNSON, D. S. et al. Optimization by Simulated Annealing: An Experimental Evaluation; Part II, Graph Coloring and Number Partitioning. **Operations Research**, Baltimore, v. 39, n. 3, p. 378-405, 1991.

[JOH 97] JOHNSON, D. S.; MCGEOCH, L. A. The Travelling Salesman Problem: A Case Study in Local Otimization. In: Aarts, E. H. L.; Lenstra, J. K. (Eds) **Local Search in Combinatorial Optimization**. London: John Wiley and Sons, 1997. p. 215-310.

[KAL 86] KALOS, M. H.; WHITLOCK, P. A. **Monte Carlo Methods** Volume I: Basics. New York: John Wiley & Sons, 1986. 186p.

[KAR 86] KARP, R. M. Combinatorics, Complexity, and Randomness. **Communications of the ACM**, New York, v. 29, n. 2, p. 97-118, 1986.

[KIR 83] KIRKPATRICK, S.;GELLATT , C. D.;VECCHI, M. P. Optimization by Simulated Annealing**. Science**, Washington, v. 220, n. 4598, p. 671-680, 1983.

[KNU 81] KNUTH, D. E. **The Art of Computer Programming**. Massachusetts: Addison-Wesley, 1981. v. 2, p. 1-160.

[KNU 83] KNUTH, D. E. Algorithm and Program, Information and Data. **Communications of the ACM**, New York, v. 26, p. 56-98, 1983.

[KOV 96] KOVÁCS, Z. L. **Teoria da Probabilidade e Processos Estocásticos**. São Paulo: Escola Politécnica da USP, 1996.

[LAA 92] LAARHOVEN, P. J. M. VAN; AARTS, E. H L.; LENSTRA, J. K. Job Shop Scheduling by Simulated Annealing. **Operations Research**, Baltimore, v. 40, n. 1, p. 113-125, 1992.

[MAR 93] MARTIN, O.; OTTO, S. W. **Combining Simulated Annealing with Local Search Heuristics**. 1993. Disponível por e-mail em martin_o@ipacls.in2p3.fr. (1998).

[MCA 97] MCAALLESTER, D.; SELMAN, B.; KAUTZ, H. **Evidence for Invariants in Local Search**. Disponível por WWW em http://www.cornell.edu, 1997.

[MET 53] METROPOLIS, N. et al. Equation of State Calculation by Fast Computer Machines. **The Journal of Chemical Physics**, New York, v. 1, n. 6, p. 1087-1092, 1953.

[MUL 93] MULLER, F. M. **Algoritmos Heurísticos e Exatos para Resolução do Problema de Sequenciamento em Processadores Paralelos**. Campinas: UNICAMP, 1993. Tese de Doutorado.

[MUL 96] MULLER, F. M. **Heurísticas e Metaheurísticas**. In: V Escola Regional de Informática da SBC Regional Sul. **Anais...** Santa Maria: Departamento de Eletrônica e Computação da UFSM, 1995.

[NIE 95] NIEVERGELT, J. et al. All the Needles in a Haystack: Can Exhaustive Search Overcome Combinatorial Chaos? In: Leeuwen, V. J. (Ed.) **Computer Science Today,** Recent Trends and Development. Berlin: Springer-Verlag, 1995. p 254-274. (Lectures Notes in Computer Science, n. 1000)

[NUR 93] NURMELA, K. J.; OSTERGARD, P. R. J. Constructing Covering Designs by Simulated Annealing. **Helsinki University of Technology, Digital Systems Laboratory Technical Reports**, Helsinki, s. B, n. 10, 1993.

[OCH 94]

[OLI 97] OLIVEIRA, R. S. **Escalonamento de Tarefas Imprecisas em Ambiente Distribuído**. Florianópolis: UFSC, 1997. Tese de Doutorado.

[OSM 96] OSMAN, I. H.; KELLY, J. P. **Meta-heuristics**: Theory & Applications., Boston: Kluwer Academic, 1996.

[PRE 88] PRESS, W. H. et al. **Numerical Recipes in C, The Art of Scientific Computing**, Cambridge: Cambridge University, 1988. 735 p.

[RAY 96] RAYWARD-SMITH, et al. **Modern Heuristics Search Methods**. Chichester: John Wiley & Sons, 1996. 294p.

[RIB 96] RIBEIRO, C. C. **Metaheuristics and Applications** (material de apresentação), Advanced School on Artificial Intelligence, Estoril, Portugal, 1996. Disponível por WWW em http://inf.puc.rio.br/~celso/~poggi.(Março 1999).

[RIB 98] RIBEIRO, C. C.; ARAGÃO, M. V. S. P. **Metaheuristicas**. Material de apresentação, Escola Brasileira de Computação, 1998. Disponível por WWW em http://inf.puc.rio.br/~celso/~poggi. (Março 99).

[ROB 97] ROBERTS, G. O., ROSENTHAL, J. S. **Markov Chain Monte Carlo**: Some Practical Implications of Theoretical Results. Toronto: Universidade de Toronto, 1997. Disponível por WWW em http://utstat.toronto.edu/jeff/research.html. (Set 1999)

[ROD 93] RODRIGUES, M. R. D.; ANJO, A. J. B. On Simulating Thermodynamics**.** In: Vidal, R. (Ed.). **Applied Simulated Anneal**. Berlim: Springer Verlag, 1993. p. 45-58, (Lectures Notes in Economics and Mathematical Systems, n. 369)

[ROG 99] ROGLIA, I. B.; MULLER, F. M.; TOSCANI, L. V. **Método Heurístico para Solução do Problema de Sequenciamento Cíclico de N Tarefas em M Processadores Paralelos Idênticos**. Porto Alegre: CPGCC da UFRGS, 1999. Dissertação de Mestrado.

[ROS 93] ROSE, J. S.; KLEBSCH, W.; WOLF, J. Temperature Measurement and Equilibrium Dynamics of Simulated Annealing Placements**. IEEE Transaction on CADICS**, New York, v. 9, p. 253-259, 1990.

[ROS 99] ROSENTHAL, J. S. **A Review of Asymptotic Convergence for General State Markov Chains**. Toronto: Universidade de Toronto, 1999. Disponível por WWW em http://utstat.toronto.edu/jeff/research.html. (Set 1999)

[SAI 95] SAIT, S. M.; YOUSSEF, H. **VLSI Phisical Design Automation-Theory and Practice**. Piscataway: IEEE, 1995. 426p.

[SAS 88] SASAKI, G. H.; HAJEK, B. The Time Complexity of Maximum Matching by Simulated Annealing. **Journal of Association of Computing Machinery**, New York, v. 35, n. 2, p. 387-403, 1988.

[SHA 48] SHANNON, C. E. A Mathematical Theory of Communication. **Bell System Technical Journal**, n. 27, p. 379-423, 623-656, 1948. Disponível por WWW em http://cm.bell-labs.com/cm/ms/what/shannonday/paper.html. (Jun 1999)

[SHR 91] SHRAGOWITZ, E.; RUNG-BIN, L. Combinatorial Optimization, Markov Chains, and Stochastic Automata. In**:** Stewart, W. J. (Ed.) **Numerical Solution of Markov Chain** (Probability, Pure and Applied; n. 8). New York: Marcel Dekker, 1991. p. 543-563.

[SIE 92] SILVA, E. A. S.; MUNTZ, R. R. **Métodos Computacionais de Solução de Cadeias de Markov: Aplicações a Sistemas de Computação e Comunicação**. In: VIII Escola de Computação, Gramado, ago/1992. **Anais...** Porto Alegre: Instituto de Informática da UFRGS, 1992. 206p.

[SIL 99] SILVEIRA, C. M. D.; TOSCANI, L.V. **GRASP**. Porto Alegre: CPGCC da UFRGS, 1999. 41p. (TI-801)

[STA 92] STADLER, P. F.; SCHNABL, W. The Lanscape of the Traveling Salesman Problem. **Phisycs Letters A**, North-Holland, n. 161, p. 337-344, 1992.

[STO 98] STOUGIE, L. **M3: Algoritmos Aleatórios**. Buenos Aires : Departamento de Computacion/Universidad de Buenos Aires, 1998. 391p.

[STV 94] STOLTENBERG-HANSEN, V. **Mathematical Theory of Domains**. Cambridge: Cambridge University Press, 1994. 349p.

[SUN 96] SUNDERMANN, E. **PET Image Reconstruction Using Simulated Annealing**. Leeds: University of Leeds, 1996. Disponível por WWW em nup://petaxp.rug.ac.be/~erik/research/welcome.html. (Jan 1999).

[SWZ 84] SWZARCFITER,J. **Grafos e Algoritmos Computacionais**. Rio de Janeiro: Campos, 1984. 216p.

[TOR 92] TORREÃO, J. R. A. Métodos Estocásticos em Computação Visual. In: Escola de Computação, 8., 1992. **Anais...** Porto Alegre: Instituto de Informática da UFRGS, 1992.

[TOS 86] TOSCANI, L. V.; SZWARCFITER, J. L. **Algoritmos Aproximativos**. Porto Alegre: CPGCC – UFRGS, 1986. (Rp - 58).

[TOS 88] TOSCANI, L.V. **Métodos e Desenvolvimento de Algoritmos**: Especificação Formal, Análise Comparativa e de Complexidade. Rio de Janeiro: PUC, 1988. Tese de Doutorado.

[VAR 93] VARANELLI ,J. M.; COHOON, J. P. **A Two-Stage Simulated Annealing Methodology**. Virginia: Universidade de Virginia, 1993. Disponível por e-mail cohoon@cs.virginia.edu. (1998).

[VEL 84] VELOSO, P. A. S. Aspectos de uma Teoria Geral de Problemas. **Cadernos de História e Filosofia Da Ciência**, n.7, 1984. p. 21-42.

[WEN 98] WENDT, O. **Combinatorial Otimization and Local Search**. Frankfurt: Frankfurt University, 1998. Disponível por WWW em http://caladan.wiwi.uni-frankfurt/IWI/cosa/cop/cop.html. (Set 1999)

[WHI 84] WHITE, S. R. Concepts of Scale in Simulated Annealing. In: IEEE International Conference on Circuit Design, ICCAD, 1984. **Proceedings...** New York: Watson Research Center, 1984. p. 646-651.