Uma Avaliação Experimental de uma Versão Paralela de Simulated Annealing

Maria Cristina Silva Boeres

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Prof. Valmir C. Barbosa, Ph. D.
(presidente)

Prof. Claudio L. Amorim, Ph. D.

Profa. Nair Maria Maia de Abreu, D. Sc.

RIO DE JANEIRO, RJ – BRASIL
JULHO DE 1990

BOERES, MARIA CRISTINA SILVA

Uma Avaliação Experimental de uma Versão Paralela de Simulated Annealing [Rio de Janeiro] 1990

VI, 72 p., 29.7 cm, (COPPE/UFRJ, M. Sc., ENGENHARIA DE SISTEMAS E COMPUTACÃO, 1990)

TESE – Universidade Federal do Rio de Janeiro, COPPE

1 – *NP*-completo 2 – Paralelo 3 – Simulated Annealing

I. COPPE/UFRJ II. Título(Série).

A meus pais

Agradecimentos

A todas as pessoas que tiveram, ao longo deste deste período, não só participação direta como também, de alguma maneira, influenciaram no andamento e desempenho deste trabalho.

Inicialmente, agradeço ao meu orientador Valmir C. Barbosa, pela oportunidade de trabalhar com uma pessoa que tanto admiro, principalmente pela sua vasta experiência e competência profissional.

Aos meus pais, Léa e José, pelo incentivo a minha vida profissional, sempre dando apoio a todas minhas decisões. Agradeço a minha mãe, a sua paciência de me ajudar a escrever o texto e ao meu pai, pela boa vontade de fazer os desenhos.

À Claudinha, minha irmã, pela sua amizade e companheirismo e pela pronta ajuda em tantos momentos que precisei, que não foram somente problemas e dificuldades com o trabalho. A sua ajuda também, na revisão do texto foi de grande valia.

Ao meu irmão, Déo, agradeço a sua admiração, torcendo sempre para que eu terminasse bem este trabalho. Não obstante, agradeço pela sua ajuda nos contra-tempos ocorridos ao longo do tempo.

À minha família, principalmente à vovó, pela sua eterna preocupação.

À Claudia Linhares, pela sua grande amizade, me ajudando tanto com a sua competência e responsabilidade, além da sua força e incentivo em tantos momentos.

À amiga de anos, Lúcia, pelo seu companheirismo tão necessário em todas as horas.

À Mau, Anna e Rui Marinho, por sempre estarem prontos a prestarem as suas ajudas.

A Carlos, que no início deste curso, me compreendeu e incentivou no trabalho, sem falar na sua ajuda incontestável.

Às secretárias Suely, Denise e Claudia, pela paciência de atenderem prontamente a tantos pedidos.

Aos amigos da COPPE, Clícia, Delfim, Wamberto, Rui, Márcia, Rose, Graça, Lélio, Juarez, Rita, Hércules, Maurício, Mónica, Max, Ricardo, Ildemar e Júlio, pelo grande companheirismo.

Ao pessoal do laboratório, pela grande ajuda.

A não menos, a Ricardo Citro e Eliseu, pela franca ajuda nos problemas surgidos na implementação do trabalho.

Aos amigos de coração, que me acompanharam e incentivaram durante todo esse tempo, o que foi muito importante e construtivo para mim.

Finalmente, à UFF, pela compreensão e entusiasmo em relação ao meu trabalho.

Resumo da Tese apresentada à COPPE como parte dos requisitos necessários para a obtencão do grau de Mestre em Ciências (M. Sc.)

Uma Avaliação Experimental de uma Versão Paralela do Simulated Annealing

Maria Cristina Silva Boeres

Julho de 1990

Orientador: Valmir Carneiro Barbosa

Programa: Engenharia de Sistemas e Computação

É apresentada neste trabalho, uma versão paralela do *Simulated Annealing*, aplicado sobre um problema de otimização combinatória, onde as variáveis do problema correspondem aos processos do sistema. O método é aplicado no Problema do Cobrimento de Vértices Mínimo, que é *NP*-completo e possui uma heurística implementada na sua forma centralizada. Esse tipo de problema pode ser formalizado de tal maneira que torna a utilização da busca estocástica, na sua forma paralela, bem propícia. A aplicação do método em grafos aleatórios, gerou resultados mais significativos do que outras heurísticas até então usadas. Foi observado, através do Método de Fluxo Máximo, que *Simulated Annealing* chega sempre, no final da computação, ao melhor resultado dentre todos os gerados durante sua execução . Os testes foram desenvolvidos em uma rede de *transputers*, utilizando a linguagem *Occam* como linguagem de programação paralela.

Abstract of Thesis presented to COPPE as partial fulfillment of the requirements for the degree of Master of Science (M. Sc.)

An Experimental Evaluation of a Parallel Version of Simulated Annealing

Maria Cristina Silva Boeres

July, 1990

Thesis Supervisor: Valmir C. Barbosa

Department: Programa de Engenharia de Sistemas e Computação

It is presented in this work, an implementation of Simulated Annealing, which utilizes up to as many processing elements as there are variables in the problem. The method is applied to the Minimum Vertex Cover Problem, which is *NP*-complete and admits a centralized heuristic. This type of problem can be formalized in a way, such that the parallel stochastic search can be very well suitable. The application of the method on random graphs, give better quality solutions than other known heuristic. It is observed, through the Maximum Flow Method, that the Simulated Annealing at the end of the computation, returns the best solution over all the ones calculated during the execution. The tests have been performed on a transputer network, utilizing the Occam language for parallel programming.

Índice

Lista de Figuras

Capítulo I

Introdução

Podemos fazer uma classificação de problemas em Problemas de *Decisão*, de *Localização* e de *Otimização*. Em um problema de Decisão, o objetivo consiste em decidir a resposta *Sim* ou *Não* a uma questão. Em um problema de Localização, o objetivo é localizar uma certa estrutura S que satisfaça um conjunto de propriedades dadas. Se as propriedades a que S deve satisfazer envolverem critérios C de otimização, o problema é de Otimização. É possível formular um problema de decisão cujo objetivo é indagar se existe ou não a mencionada estrutura S, satisfazendo as propriedades dadas. Isto é, um problema de decisão pode ser transformado em um problema de localização e também em um problema de otimização.

Um problema pode ser definido como uma questão a ser resolvida, onde geralmente, parâmetros são especificados. Uma instância de um problema é obtida quando se especifica valores para os seus parâmetros. Após a definição do problema, o importante é especificar um algoritmo eficiente para resolvê-lo. Pode-se entender como o mais eficiente, aquele que é o mais rápido dentre os existentes que resolvem o problema.

O tempo de execução de um algoritmo é o número de passos necessários para seu processamento e é expresso em relação ao tamanho de sua entrada. Por sua vez, a complexidade de um algoritmo expressa um limite superior, ou seja, o maior tempo de execução necessário para resolver um problema. A complexidade, logicamente, só pode ser definida depois que o algoritmo é especificado.

Diferentes algoritmos possuem diferentes funções de complexidade. Cada função caracteriza o algoritmo como sendo eficiente ou ineficiente. Um algoritmo é considerado eficiente, quando possui o número de seus passos limitado por uma função polinomial, ou seja, ele soluciona um determinado problema em tempo polinomial. Qualquer algoritmo cuja função de complexidade não é limitada polinomialmente, é chamado de algoritmo ineficiente ou exponencial. Pode-se dizer mais precisamente, que esses algoritmos não resolvem os respectivos problemas em tempo polinomial.

Considere um algoritmo que resolva um problema p. Se o algoritmo é eficiente, p é denominado tratável, se é ineficiente, intratável.

Vários algoritmos podem resolver um problema. Mas, se dentre esses algoritmos existe pelo menos um, cuja função de complexidade é polinomial, então, o problema pertence à classe P dos problemas polinomias. Se não foi encontrado até então, nenhum algoritmo de complexidade polinomial, o problema pertence à classe NP dos problemas Não determinísticos Polinomiais.

Dado uma solução, é possível verificar a corretude de um problema NP através de um algoritmo polinomial. É importante observar que o algoritmo de reconhecimento da solução será polinomial, se o tamanho da solução for também polinomial. No entanto, se o tamanho da solução, não for polinomial, o algoritmo pode ser exponencial.

Não se pode afirmar que um problema não pertence à classe P, apenas pelo fato de não ter sido encontrado até os dias atuais um algoritmo que o resolva em tempo polinomial. É preciso uma prova de que não existe tal algoritmo para fazer a afirmação.

Uma questão que se deseja resolver é saber se $P = NP$. No entanto, se acha pouco provável que isto seja verdade, já que NP possui diversos problemas extremamente difíceis, que até hoje não se conseguiu construir um algoritmo que os resolva em tempo polinomial.

Alguns problemas da classe NP podem ser reduzidos a outros pro-

blemas da mesma classe, através de uma transformação feita em tempo polinomial. Se um problema p_1 pode ser reduzido a um outro problema p_2, então p_2 é tão difícil quanto p_1. Além do mais, se for encontrado um algoritmo que resolva p_2 em tempo polinomial, este algoritmo também resolverá p_1.

Em 1971, Cook, através de um teorema, demonstrou que todo problema *NP* pode ser reduzido a um problema da lógica, chamado Problema da Satisfatibilidade, considerado o problema mais difícil da classe *NP*.

Karp, em 1972, reduziu este problema da lógica a outros problemas *NP*, mostrando que todos têm o mesmo tipo de dificuldade.

Existe assim, outra classe de problemas que são os mais difíceis dentre todos contidos na classe *NP*. São os *NP*-completo, cuja definição se segue.

Um problema p é *NP*-completo se, e somente se,

(i) $p \in NP$

(ii) todo problema $p' \in NP$ pode ser transformado no problema p

Observe que (ii) implica que todo problema da classe *NP* pode ser transformado em tempo polinomial em um problema *NP*-completo. Isto é, se um problema *NP*-completo puder ser resolvido em tempo polinomial então, todo problema *NP* admite também algoritmo polinomial.

A área de otimização combinatória contém vários problemas para os quais, até o momento, não foi construído nenhum algoritmo que os resolva em tempo polinomial.

I.1 Uma Visão Geral

Neste trabalho vamos nos concentrar no universo da otimização combinatória. Toma-se como exemplo, um problema de tal área. Dados um conjunto $X = \{X_1, X_2, \ldots, X_n\}$ de variáveis que tomam valores dentro de um domínio $D = \{0, 1\}$ e um conjunto

PV de pontos viáveis ou praticáveis, pertencentes à D^n, o problema consiste em achar o mínimo global de uma função f, onde

$$f : D^n \longrightarrow \mathcal{R},$$

sobre o conjunto de pontos viáveis PV. Isso é equivalente ao fato de achar o ponto ótimo $x^* \in PV$, tal que $f(x^*) \leq f(x), \forall x \in PV$. É tratado neste contexto, um problema formulado dessa maneira que é *NP*-completo, pois é de difícil resolução, já que a maioria dos métodos existentes não chegam a um mínimo global e sim, um mínimo local.

Heurísticas existentes tentam solucionar tais problemas. Surge uma heurística como uma nova proposta, que provém de um método usado na física, para que se possa observar as propriedades de um material em estudo. Para tal, este último deve estar no seu estado básico de energia, seu estado de equilíbrio térmico. O método propõe que o material seja submetido inicialmente a uma temperatura bem alta e que posteriormente, essa temperatura sofra um decréscimo lento e gradual. Assim, a energia do material vai diminuindo. É esse decréscimo na temperatura que leva o material ao seu estado de energia básico, de maneira uniforme e não somente em alguns pontos. Denomina-se esse método de *Annealing*.

Para resolver problemas de otimização combinatória, é proposto uma simulação do método físico, utilizando a heurística denominada *Simulated Annealing*, que faz uma busca estocástica ao ponto ótimo no conjunto de pontos viáveis PV. A busca é guiada pela distribuição de Boltzmann. Assim, aumentos na função objetivo f, que simula a energia do material no processo físico, podem ser aceitos. Utiliza-se um parâmetro T para simular o comportamento da temperatura. T começa com um valor bem alto e vai sendo decrementada lentamente segundo uma função pré-estabelecida. *Simulated Annealing* tenta achar o mínimo global da função e não um ponto ótimo local.

É suposto que duas variáveis de $X = \{X_1, X_2, \ldots, X_n\}$ possam ser consideradas vizinhas ou não vizinhas, de acordo com algum critério do problema a ser resolvido. Esse conjunto de variáveis podem ser consideradas como sendo um *Campo Aleatório de Markov*, devido as suas propriedades probabilísticas, onde o

valor de uma variável X_i depende somente dos valores das variáveis vizinhas.

O interesse no *Campo Aleatório de Markov* nos problemas a serem resolvidos pelo *Simulated Annealing*, recai no fato de que variáveis vizinhas não podem calcular seus valores independentemente. Por outro lado, variáveis não vizinhas podem processar em paralelo.

É proposto então, um algoritmo paralelo para *Simulated Annealing*. Com isto, cada variável está associada a um processador e processadores vizinhos se comunicam através de canais bidirecionais.

Simulated Annealing foi aplicado no problema de **Cobrimento de Vértice Mínimo** de um grafo $G = (N, A)$. O objetivo do problema é achar o menor subconjunto I de vértices de N, de tal maneira que pelo menos um vértice de cada arco do grafo esteja contido em I.

Outra heurística foi utilizada com o intuito de comparar resultados com os gerados pelo *Simulated Annealing*: o emparelhamento maximal de um grafo $G = (N, A)$.

Um ponto a ser observado no método *Simulated Annealing* é o seguinte: como é feito uma busca estocástica sobre o conjunto de pontos viáveis, são aceitos não só decréscimos, como aumentos na função objetivo f. Assim, durante a simulação, vários estados globais são formados e para cada um, é calculado o valor de f. Ao final de toda a computação distribuída, pode-se chegar a um estado global em que f não foi o menor dentre todos já calculados durante o processamento.

Um estado global nada mais é do que um corte no grafo de precedência, definido no capítulo V, que por sua vez, representa todos os eventos ocorridos durante a computação. Achar o corte mínimo, equivale a procurar o estado global em que f foi o menor. Para tal, aplica-se o algoritmo de Fluxo Máximo no grafo de precedência para comparar com o resultado gerado pelo *Simulated Annealing*.

A implementação distribuída do *Simulated Annealing* foi executada em uma rede *transputers* (processadores), na linguagem *OCCAM* que possui direti-

vas de paralelismo e os tempos de execução foram avaliados.

I.2 Organização do Texto

Este trabalho está dividido em seis capítulos e dois apêndices, sendo o primeiro capítulo a introdução, que apresenta a classificação dada aos problemas de otimização combinatória e descreve brevemente todo o trabalho.

A descrição do método físico chamado *Annealing* é feito no Capítulo II. É mostrado como se processa o método e qual seu objetivo no campo da física. Mostra-se também que a simulação deste método pode ser aplicada para resolver problemas de otimização combinatória. É descrito finalmente, *Simulated Annealing.*

No Capítulo III, descrevem-se as características dos problemas sobre os quais *Simulated Annealing* pode operar, propondo uma implementação distribuída do método. Apresenta-se a implementação do método na sua forma distribuída, através de um algoritmo paralelo.

O problema tratado neste contexto é apresentado no Capítulo IV. São descritos todos os parâmetros do algoritmo paralelo que foram utilizados para sua execução. É descrita uma outra heurística usada para comparar seus resultados com os gerados pelo *Simulated Annealing*, baseada no conceito de Emparelhamento Maximal.

No Capítulo V, é dada uma visão de estado global e grafos de precedência, para que sejam utilizados na recuperação da melhor solução e verificar se o *Simulated Annealing* converge.

Finalmente, no Capítulo VI, são avaliados os resultados e apresentadas conclusões sobre o trabalho.

Junto a esse trabalho, existem dois apêndices. O Apêndice A, descreve o algoritmo de Fluxo Máximo utilizado para recuperar a melhor solução. Por fim, no Apêndice B, é feita uma descrição sumária da linguagem *OCCAM* e do processador *transputer*

Capítulo II

Simulated Annealing

Em muitos casos da física, ao analisar as propriedades de um determinado material, é necessário que ele esteja em um estado de equilíbrio térmico, um estado onde as propriedades mais comuns possam ser observadas.

Cada estado do material (líquido, sólido ou gasoso) caracteriza uma configuração definida pela posição de cada um de seus átomos. Essa configuração se modifica com a mudança de temperatura a qual o material se submete. As várias configurações são balanceadas pelo fator de probabilidade de Boltzmann :

$$exp(-E(\{p_i\})/k_B T)$$

onde

- $\{p_i\}$ representa o conjunto de posições dos átomos, definindo assim a configuração
- $E(\{p_i\})$ representa sua energia
- k_B é uma constante de Boltzmann
- T é a temperatura a qual a configuração é submetida.

Uma questão fundamental na física, é o que acontece a um material quando se chega a uma temperatura baixa. Estados básicos e energia de configurações próximo a esses estados, são raros, levando-se em consideração todas as

configurações do material. As propriedades do material são bem observadas em baixa temperatura, pois conforme T vai decrescendo, a distribuição de Boltzmann atinge o mais baixo estado de energia.

O sistema não chega ao seu estado de equilíbrio aplicando-se imediatamente uma temperatura baixa. Deve-se submetê-lo inicialmente a uma temperatura alta e decrementá-la gradualmente, como se fosse um resfriamento bem lento. O importante é o fato do decréscimo lento da temperatura, feito segundo uma determinada função, até baixá-la sufucientemente e ficar por algum tempo resfriando para que o material chegue uniformemente ao estado de equilíbrio, não sendo encontrado no material, partes em equilíbrio e outras não. Esse processo denomina-se *Annealing.*

O processo *Annealing* nada mais faz do que tentar chegar a um estado ótimo de energia. Essa propriedade é muito poderosa, quando se trata da solução de problemas de otimização combinatória, pois sua finalidade é chegar ao valor ótimo de uma função que representa o objetivo do problema. Surge então, um novo meio de conseguir melhores resultados: o uso do método conhecido como *Simulated Annealing.*

O algoritmo de Metropolis *et al.* [14] descreve *Simulated Annealing* para um sistema físico. Cada átomo que compõe o sistema pode mover-se, produzindo uma variação de energia. Como o objetivo é chegar a uma energia que represente o equilíbrio do sistema, o deslocamento do átomo pode ser aceito ou não, pois seu deslocamento pode significar uma aproximação ou não do estado de equilíbrio. Seja ΔE, a variação de energia provocada pelo deslocamento. Se $\Delta E \leq 0$, o deslocamento é aceito, pois a energia está diminuindo, e assim o sistema pode estar chegando perto de seu estado de equilíbrio. No entanto, se $\Delta E > 0$, a aceitação do deslocamento é probabilística, pois neste caso, a energia do sistema vai aumentar ao invéz de diminuir. A probabilidade é dada por:

$$P(\Delta E) = exp(-\Delta E/k_B T).$$

Posteriormente, é gerado um número aleatório x (os números gerados durante todo o processo estão distribuídos uniformemente dentro do intervalo $(0, 1)$

), que é comparado com $P(\Delta E)$:

Se $x < p(\Delta E)$ então

o deslocamento é aceito,

caso contrário

ele não será aceito.

A configuração resultante é usada para dar início a um novo passo.

Em um problema de otimização combinatória, a função objetivo equivale à energia do sistema físico. Os parâmetros $\{x_1, ..., x_n\}$ do problema equivalem às posições dos átomos $\{p_i\}$. A temperatura é um parâmetro de controle que vai diminuindo segundo uma função pré-estabelecida. O decréscimo sendo lento, simula um resfriamento no sistema em questão. Se o decréscimo da temperatura não for lento, o sistema otimizado pode chegar a um estado ótimo local.

O método empregado, análogo ao processo físico, essencialmente faz uma busca estocástica sobre o conjunto de pontos viáveis, como se estivesse "saltando" entre pontos deste conjunto. Em alguns momentos, a função objetivo pode aumentar, apesar da finalidade ser minimizá-la. Porém, é esta possibilidade de aumento quando necessário, que assegura o resultado mínimo global ao final do processo. A distribuição de Boltzmann, como foi vista no modelo físico, dirige a busca, tendo T como parâmetro que simula a temperatura empregada ao sistema.

Simulated Annealing aplicado a um problema de otimização combinatória, pode ser descrito da seguinte forma:

- PV é o conjunto de pontos viáveis onde a busca estocástica pode ser feita.
- x, um ponto que pertence a PV, mas não ainda o melhor ponto.

O método aplicado a este ponto, faz com que seja dado um **"salto"** para um outro ponto, digamos y, onde $y \in PV$. Se a função objetivo f for menor no ponto y do que no ponto x, ou seja,

$$f(y) \leq f(x)$$

o deslocamento é aceito com probabilidade 1, pois f decresce. Caso contrário, a probabilidade de aceitação é dada por

$$exp\left[\frac{f(x)-f(y)}{T}\right] \tag{II.1}$$

No começo do processo aplica-se ao sistema uma temperatura inicial alta. Tendo em vista essa iniciativa, a procura pelo melhor ponto do conjunto de pontos viáveis se comporta como **"saltos"** disformes, fazendo com que f cresça com uma certa frequência. Consequentemente, com decréscimo de T, a aceitação dos aumentos de f vai diminuindo. Quando T tende a zero, a procura se torna cada vez mais próxima do ponto viável onde f é mínimo global, ou seja, o ponto ótimo. Isto é demonstrado em um teorema descrito em [8].

Observando II.1, conclui-se que no início do processo, aumentos na função objetivo são geralmente aceitos. T toma valores iniciais altos, acarretando que a expressão II.1 tenda a 1. Será mais provável, neste ponto, que o número aleatório gerado seja menor que a probabilidade de aceitação. Daí f pode aumentar.

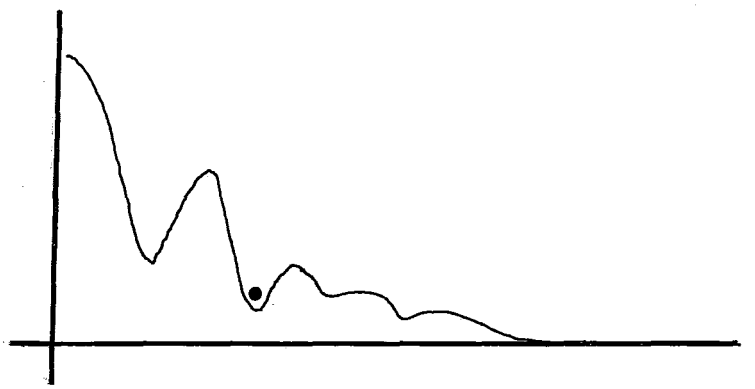


Figura II.1: *Simulated Annealing*: a procura pelo ótimo

Na figura II.1. é observado os **"saltos"** que podem ocorrer no valor de f, durante toda a computação.

Capítulo III

Implementação Paralela

III.1 Motivação para o paralelismo

O problema de otimização combinatória aqui abordado, é um problema ocorrido em grafos. Tomemos um grafo qualquer, onde existe um conjunto de vértices V e um conjunto de arcos A, definindo o grafo $G = (V, A)$. Cada vértice i do grafo pode ou não possuir vizinhos $j \in V$, sendo definido o conjunto de vizinhos $\{V_j/(i,j) \in E\}$.

A cada vértice de V é associada uma variável de $X = \{X_1, \ldots, X_n\}$. Logo, duas variáveis X_i e X_j pertencentes a X, são vizinhas se, e somente se, os vértices correspondentes i e j formam um arco $(i,j) \in A$.

Seja um subconjunto $Y \subseteq X$, uma clique, que é definido como um subconjunto de variáveis de X, onde todas são vizinhas entre si [6]. Dado um problema de otimização combinatória, suponha que para cada subconjunto $Y \subseteq X$, a seguinte função pode ser escrita

$$f = \sum_{Y \subseteq X} g_Y(x) \tag{III.1}$$

onde g_Y só depende das coordenadas de x que correspondem às variáveis de Y. g_Y também é não constante se Y não for um subconjunto com cardinalidade alta.

A procura estocástica tem a seguinte forma. Para $1 \leq i \leq n$ associe

o valor x_i à variável X_i com a seguinte probabilidade

$$\pi(X_i = x_i \mid X_j = x_j, j \neq i)$$

onde π é a distribuição de Boltzmann, que é dada como

$$\pi(x) = \frac{1}{Z} exp[-f(x)/T], \qquad \forall x \in D^n$$

Z é uma constante de normalização e T, a temperatura.

Como conseqüência da forma de f, é visto que a probabilidade condicional vista acima é dada por

$$\frac{1}{1 + \sum_{\substack{d \in D \\ d \neq x_i}} exp\left[-\frac{1}{T} \sum_{\substack{Y \subseteq X \\ X_i \in Y}} (g_Y(x^d) - g_Y(x^{x_i}))\right]}$$

onde, $\forall d \in D$, x^d é um ponto em D^n em que $X_i = d$ e todos os vizinhos de X_i têm o mesmo valor, ou seja, é só X_i que pode mudar seu valor. Pode-se dizer que

$$\pi(X_i = x_i \mid X_j = x_j, j \neq i) = \pi(X_i = x_i \mid X_j = x_j, X_j \text{ vizinho de } X_i)$$

Dentro desse contexto de busca estocástica sobre o conjunto D^n, se o conjunto de variáveis de X são variáveis aleatórias e se acontecem as seguintes propriedades

$$Prob(x) > 0, \forall x \in D^n$$

$$Prob(X_i = x_i \mid X_j = x_j, j \neq i) = Prob(X_i = x_i \mid X_j = x_j, X_j \text{ vizinho de } X_i) \tag{III.2}$$

essas variáveis formam um *Campo Aleatório de Markov.*

As propriedades III.2 são verdadeiras para X se, e somente se, existe a função III.1, onde $g_Y(x)$ só depende das variáveis que estão na clique Y e tal que

$$\text{Prob}(x) = \frac{1}{Z} exp[\frac{-f(x)}{T}]$$

Essa é a distribuição de *Gibbs*, onde Z é uma constante de normalização e T é o parâmetro que simula a temperatura.

Conclui-se então, que para a variável X_i assumir um valor x_i vai depender somente dos valores de seus vizinhos. Cada variável não precisa consultar todos os outros vértices de G por questão de cálculo. Surge neste ponto, um bom motivo para utilizar o paralelismo no cálculo da função objetivo f, pela sua própria forma em certos tipos de problemas.

Variáveis que não são vizinhas, podem processar os seus valores ao mesmo tempo, pois uma não depende da outra para tal. Já não acontece para variáveis que são vizinhas. Essas não podem fazer seus cálculos em paralelo, pois o valor de uma depende do valor da outra. Caso isso acontecesse, provavelmente se chegaria a resultados inconsistentes, pois uma determinada variável poderia utilizar um valor desatualizado de um determinado vizinho.

Com a possibilidade de paralelizar a computação, é associado a cada variável X_i, um processador. Se duas variáveis X_i e X_j são vizinhas, os processadores associados se comunicam através de canais bidirecionais, correspondendo aos arcos. Cada variável tem a visão imediata dos valores de seus vizinhos.

III.2 Implementação

A cada variável $X_i \in X$, é associado um processo p_i. Dois processos p_i e p_j se comunicam por canais, se as variáveis X_i e X_j são vizinhas. É por meio destes canais, que o processo p_i manda o valor de sua variável correspondente X_i e também, recebe os valores de seus vizinhos.

Considere o grafo $G = (V, A)$. A cada vértice i do grafo, está associada uma variável X_i. Por sua vez, a cada variável tem-se um processo p_i correspondente.

G é um grafo não orientado aleatório, onde um arco existe segundo uma certa probabilidade, como visto a seguir:

Para $i := 1$ até n

    Para $j := 1$ até n

$w :=$ número aleatório

Se $w < p$ então

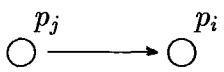
$(i,j) \in A$

caso contrário

$(i,j) \notin A$

onde p é a probabilidade do grafo e n o seu número de vérices.

Para iniciar a computação distribuída, foi associado a cada vértice um processo e a cada arco uma orientação para envio inicial de mensagem. Essa orientação foi especificada somente com o intuito de determinar os processos que iniciam a computação. A orientação é a seguinte: sendo p_i e p_j dois processos vizinhos, se $j > i$, o arco está orientado de p_j para p_i.



É garantida a aciclicidade do novo grafo formado pelos processos e pelos arcos orientados. Seja G' este novo grafo. Sendo G' orientado e acíclico, existe pelo menos um vértice fonte e um vértice sumidouro. Fonte é um vértice cujos arcos são divergentes. Por sua vez, sumidouro só possui arcos incidentes a ele.

As variáveis associadas a cada processo são inicializadas com valores aleatórios dentro do domínio $D = \{0,1\}$. O parâmetro de controle T que representa a temperatura, é inicializada com valor alto.

No início da computação, cada processo p_i contabiliza o número de processos vizinhos. Com isto, p_i sabe quantos valores deve esperar receber caso seja um vértice sumidouro e para quantos vizinhos deve mandar seu valor, quando for um fonte. Cada processo verifica de quais processos, do total de vizinhos, saem arcos para sua direção, para que ele possa esperar os valores desses vizinhos.

A operação de cada processo continua enquanto a temperatura T não

alcançar uma temperatura final pré-determinada T_{final}, que deve ser baixa.

Um processo pronto para operação é um sumidouro. Neste caso, ele pode calcular o seu novo valor que irá ser armazenado em sua variável, de acordo com os valores recebidos dos seus vizinhos.

Terminado o cálculo do novo valor de sua variável, o processo manda este valor para seus vizinhos. Para tal, ele reverte seus arcos e consequentemente outros processos poderão se tornar sumidouros, prontos para operação.

Terminada a reversão, é feito o cálculo da nova temperatura. O processo só poderá operar outra vez, se a nova temperatura for maior ou igual que a temperatura final T_{final}. Não atendendo este dispositivo, a computação deste processo terminará.

Antes de ser extinto, o processo ser tem que esperar as mensagens de seus vizinhos, que no início de toda a computação, tinham os respectivos arcos para sua direção. Isto é, se p_i operar primeiro que seu vizinho p_j, p_i terminará primeiro. O próximo a operar será p_j, que terminará logo depois. Mas, pelo algoritmo, p_j mandará seus valores para todos os seus vizinhos, inclusive p_i, que já terminou. Uma vez que p_i já tenha terminado, ele não receberia mais nenhuma mensagem. Mas p_j ficaria esperando indefinidamente que p_i as receba e o processamento nunca terminaria.

Inicialmente, são criados tantos processos em paralelo, quantos forem os números de vértices do grafo. Os que operam concorrentemente, são os processos não são vizinhos e assim não dependem um do outro para o cálculo do novo valor de suas variáveis.

Ao dar início ao processamento como um todo, pelo menos um dos processo é o sumidouro. Com a reversão dos arcos no final de sua operação, o sumidouro se torna fonte, e outros processos vizinhos a ele podem se tornar sumidouros.

Na Figura III.2(a), o sumidouro é o processo 0. Ao acabar sua operação, a reversão de seus arcos, implica na emissão de seu novo valor para seus vizinhos, fazendo com que o processo 2 seja o novo sumidouro (Figura III.2(b)).

X_i := valor aleatório;
envia X_i pelos canais direcionados para fora;
marca os canais direcionados para dentro;
espera receber as mensagens de todos os canais marcados armazenando
    cada valor recebido na variável correspondente X_j;
$k := 0$; T_k := temperatura inicial;
Enquanto $T_k \geq T_{final}$ faça
    espere se tornar um sumidouro, armazenando cada valor recebido
        na variável correspondente X_j;
    Para todo $x_i \in D$ faça
        calcular as probabilidades condicionais;
    atualizar X_i;
    reverter os canais, enviando X_i para todos os vizinhos;
    $k := k + 1; T_k := r(k)$;
espera pelas mensagens dos canais marcados ignorando-as;

Figura III.1: Algoritmo para um processo p_i

Este, por sua vez, ao reverter seus arcos no final de sua operação, para o envio de seu novo valor, faz com que os processos vizinhos 4 e 3, se tornem os sumidouros. Estes dois, por sua vez, processam em paralelo (Figura III.2(c)), pois eles não dependem um do outro.

O algoritmo que representa todo este procedimento é descrito na Figura III.1, onde k representa o número de vezes que um processo opera. Ele vai sendo incrementado até que T_k, a temperatura corrente, atinja um valor menor que T_{final}, que nada mais é do que a temperatura final pré-estabelecida.

Cada processo começa a operar com a mesma temperatura inicial T_o. A cada iteração, a temperatura decresce segundo uma mesma função, calculada por cada p_i, cujos parâmetros são os mesmos para todos os processos. Assim, todos eles operam o mesmo número de vezes, até chegar a T_{final}.

Se o processo p_i, correspondente ao vértice i, for o primeiro sumidouro a operar, ele será o primeiro processo a atingir a temperatura T_{final} e terminará seu processamento. Posteriormente, chegarão ao final seus vizinhos que, no início foram os segundos a operar. Isto pode ser vizualizado como uma onda que propaga desde os primeiros sumidouros, até os últimos (Figura III.3).

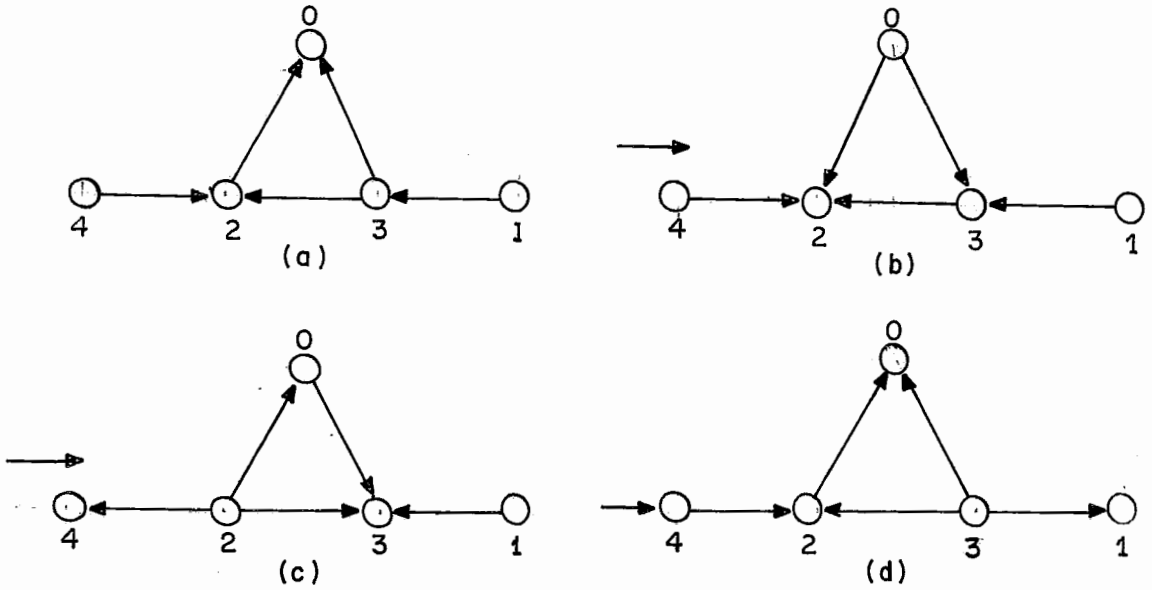


Figura III.2: Processamento distribuído

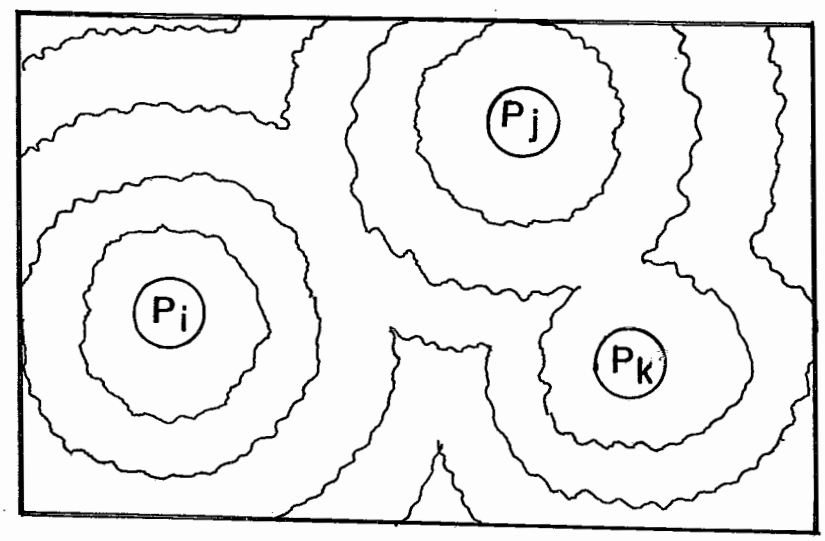


Figura III.3: Onda propagada no término

Capítulo IV

Avaliação Experimental

IV.1 Problemas Tratados

Dizemos que uma expressão lógica está na sua forma normal conjuntiva, quando ela representa uma conjunção de cláusulas, e estas por sua vez, representam uma disjunção de literais. Um exemplo seria

$$(x_1) \wedge (\sim x_1 \vee x_2 \vee \sim x_3) \wedge (x_3)$$

onde $x_1, x_2, x_3, \sim x_1 \text{e} \sim x_3$ são literais.

Chamamos de Problema da Satisfatibilidade , aquele que determina se uma expressão lógica na sua forma normal conjuntiva é verdadeira para valores dados a seus literais. Neste caso, cada cláusula tem que assumir valor verdadeiro.

A classe dos problemas *NP*-completo contém os problemas mais difíceis do universo de problema *NP*. O problema da Satisfatibilidade é *NP*-completo. No entanto, uma família ou classe de instâncias para o problema da Satisfatibilidade cujas cláusulas possuem no máximo dois literais, não é *NP*-completo, pois já foi feito um algoritmo que o resolve em tempo polinomial. Então, dado $n \geq 3$, os problemas de Satisfatibilidade com n literais em cada cláusula, são *NP*-completo. Sua forma geral com r cláusulas:

$$A_1 \wedge A_2 \wedge ... \wedge A_r$$

onde cada cláusula A_i é dada por:

$$L_{i,1} \vee L_{i,2} \vee L_{i,3} \ldots L_{i,n}$$

Os literais serão representados, neste contexto, pelas variáveis $X = \{X_1, X_2, ..., X_n\}$, vistas anteriormente, que podem assumir valores dentro do domínio $D = \{VERDADEIRO, FALSO\}$.

Os literais podem assumir os valores de X_i,ou sua negação $\sim X_i$, ou a constante *FALSO*. A última hipótese faz com que a cláusula possa ter menos de três literais.

Para que a fórmula seja satisfeita, deve-se associar valores às variáveis de tal maneira que, as r cláusulas tenham valor verdadeiro.

Representando o problema através de um grafo, problemas em grafos, cada literal corresponderá a um vértice. Sendo assim, cada literal também corresponde à variável $X_i \in X$ associada ao vértice.

Transformando cada cláusula de um problema de Satisfatibilidade em uma clique, que é um conjunto de vértices que são vizinhos entre si, cada clique gera um valor resultante da disjunção de suas variáveis. Para cada clique $Y \subseteq X$, é definido a função $g_Y(x) = c$, onde o valor c, depende somente das variáveis de $X_i \in Y$. Seja a função

$$f(x) = \sum_{Y \subseteq X} g_Y(x)$$

que se quer minimizar. Ela assume o valor u, se x for um ponto que pertence a D^n, de modo que u cláusulas sejam satisfeitas. Assim, o problema de decisão pode ter como objetivo achar um ponto ótimo $x^* \in D^n$, onde $-f$ é mínimo e verificar se $f(x^*) = r$, ou seja, todas as r cláusulas que constituem o problema foram satisfeitas.

Como exemplo, observe a fórmula na sua forma normal conjuntiva

$$(X_1 \vee X_3) \wedge (\sim X_1 \vee X_3 \vee X_4) \wedge (\sim X_3 \vee X_5) \wedge (X_2 \vee X_4) \wedge (\sim X_2 \vee X_4)$$

é um problema de Satisfatibilidade, onde só será satisfeita se a função f for tal que, o mínimo global $-f$ tiver o valor $-r = -5$, isto é, todas as r cláusulas foram

satisfeitas. Associando cada cláusula a uma clique, cada $g_Y(x)$ é dado por

$$g_{\{X_1,X_3\}}(x) = \begin{cases} 1 & \text{se } A_1 \text{ for verdadeira,} \\ 0 & \text{caso contrário} \end{cases}$$

$$g_{\{X_1,X_3,X_4\}}(x) = \begin{cases} 1 & \text{se } A_2 \text{ for verdadeira,} \\ 0 & \text{caso contrário} \end{cases}$$

$$g_{\{X_3,X_5\}}(x) = \begin{cases} 1 & \text{se } A_3 \text{ for verdadeira,} \\ 0 & \text{caso contrário} \end{cases}$$

$$g_{\{X_2,X_4\}}(x) = \begin{cases} 2 & \text{se } A_4 \text{ e } A_5 \text{ forem verdadeiras,} \\ 1 & \text{se somente } A_4 \text{ ou } A_5 \text{ for verdadeira} \end{cases}$$

Transformando as variáveis em vértices e associando um arco a cada par de variáveis, pertencentes a uma cláusula, o grafo resultante é visto na Figura IV.1. O grafo é geralmente esparso, pois cada clique está associado a uma cláusula que possui no máximo três literais.

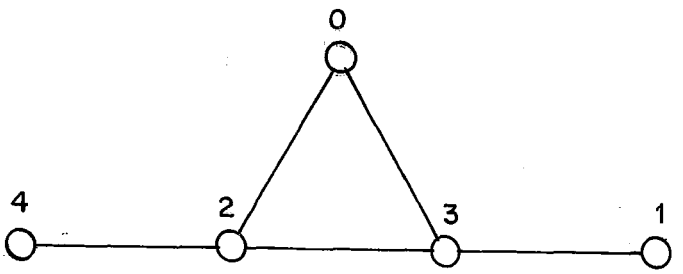


Figura IV.1: grafo $G = (V, A)$

Ao invés dos valores assumidos pelas variáveis estarem dentro do domínio $D = \{VERDADEIRO, FALSO\}$, os valores estarão agora, dentro do domínio $D = \{0, 1\}$. Formula-se desta maneira, problemas de interesse neste contexto e que são *NP*-completo.

IV.2 Problema do Cobrimento de Vértices Mínimo

O cobrimento de vértices mínimo de um grafo $G = (V, A)$, onde V é o conjunto de vértices e A de arcos, é o menor subconjunto $I \subset< V$ de vértices do grafo, tal que,

para cada arco $(i,j) \in A$, pelo menos um dos vértices do arco pertençe à I [6]. É interessante ressaltar o fato de que $\mid I \mid \leq \mid V \mid$ e que I sendo mínimo, não existe um subconjunto $I' \subseteq V$ com estas características e com cardinalidade menor que I.

Para este problema, a função objetivo f é definida como a soma das funções

$$\begin{aligned} &\forall i \in V: \quad g_{\{X_i\}}(x) = X_i \\ &\forall (i,j) \in A: \quad g_{\{X_i,X_j\}}(x) = \begin{cases} 2 & \text{se } X_i = X_j = 0, \\ 0 & \text{caso contrário} \end{cases} \end{aligned} \tag{IV.1}$$

Cada variável X_i associada ao vértice i, pode assumir os valores dentro do domínio $D = \{0,1\}$. No problema do cobrimento de vértices mínimo, o vértice $i \in I$ se, e somente se, $X_i = 1$. Fica agora claro a análise das funções IV.1. Quando as variáveis X_i e X_j possuem seus valores iguais a 0, significa que o arco $(i,j) \in A$ ainda não foi coberto. $g_{\{X_i,X_j\}}(x)$, neste caso, adiciona 2 à função f, pois não se chegou ao objetivo:cobrir todos os arcos. Por isso, f não assumiu o menor valor. Quando se atinge o objetivo, todos os arcos estão cobertos. Assim, f será a soma das funções $g_{\{X_i\}}(x)$, pois para cada arco, pelo menos uma extremidade terá sua variáveis com o valor 1. No final do processamento, $f = \mid I \mid$.

IV.3 O cálculo do novo valor de uma variável

Um processo p_i se torna sumidouro quando recebe os valores das variáveis de todos os seus vizinhos, guardando estes valores em um vetor V (V_j contém o valor do vizinho j).

Posteriormente, é calculado *f(x)* para $X_i = 0$. Ou seja, *f(x)* é a soma das funções

$$g_{\{X_i\}}(x) = 0$$

e para todo vizinho de p_i

$$g_{\{X_i,X_j\}}(x) = \begin{cases} 2 & \text{se } X_i = X_j = 0, \\ 0 & \text{caso contrário} \end{cases}$$

Neste caso, o valor calculado para *f(x)*, só vai depender dos valores dos vizinhos, já que ele está com o valor 0.

O próximo passo, é calcular *f(x)* para X_i contendo o valor 1. Pela especificação de *f(x)*, a contribuição que os arcos terão sobre *f(x)* é nula, pois $X_i = 1$. Assim, *f(x)* se resume no somatório das funções

$$g_{\{X_i\}} = 1$$

Então, sejam f_0 e f_1 definidos por

$$f_0 = exp(-\frac{1}{T_k} g_{\{X_i=0\}}(x))$$

e

$$f_1 = exp(-\frac{1}{T_k} g_{\{X_i=1\}}(x)) = exp(-\frac{1}{T_k})$$

onde T_k é a temperatura da *k-ésima* operação do processo p_i.

Seja

$$p = \frac{f_0}{f_0 + f_1}$$

a expressão de probabilidade condicional, que nada mais é do que a simplificação da expressão de probabilidade vista anteriormente.

Foi descrito por Metropolis *et al.*[14], um algoritmo que simula o método *Annealing* de um material para que este chegue ao seu estado de equilíbrio. Analogamente, isto é feito para o cálculo de *f* mínimo global. É calculado a variação de *f*, correspondente à variação da energia *E* no sistema físico.

A variação de *f* é tratada probabilísticamente, onde *p* é a probabilidade da nova configuração ser aceita ou não. Números aleatórios são gerados uniformemente distribuídos dentro do intervalo $(0, 1)$, sendo esta parte caracterizada como a parte aleatória do algoritmo. Se este número aleatório for menor que *p*, o novo valor para a variável X_i será 0, caso contrário, será 1.

A parte aleatória do algoritmo caracteriza a busca estocástica do melhor ponto *x*, para que *f* seja mínimo global. O novo ponto escolhido vai depender do número aleatório. Ser menor que *p*, significa que a busca estocástica é feita mais próxima a f_0.Como f_o foi calculado com $X_i = 0$, a variável fica com o valor 0. Por outro lado, o número aleatório ser maior que *p*, significa que a busca está mais próxima do ponto em que $f = f_1$. Então, X_i assume o valor 1.

Inicialmente, a temperatura T_k assume valores altos. Consequentemente, tanto f_0 quanto f_1 assumem valores bem próximos de 1. Deste modo, a probabilidade é aproximadamente igual a $1/2$. Neste caso, a busca se equivale a **"pulos"** disformes entre o espaço de pontos viáveis, na procura de uma região próxima ao mínimo global, pois X_i pode assumir valor 0 ou 1 com a mesma probabilidade.

Conforme as iterações vão se processando, T_k vai adquirindo valores cada vez menores. Quando T_k tem um valor muito pequeno, tanto f_0 quanto f_1, recebem valores pequenos e consequentemente p também. Assim, a probabilidade do número aleatório ser menor que p é bem menor, o que equivale a dizer que a probabilidade de X_i assumir valor 0 é menor. Neste ponto, a busca já está em uma região próxima ao mínimo global, pois quanto maior a possibilidade de X_i receber o valor 1, maior é a possibilidade de f decrescer. Os **"saltos"** vão diminuindo e f não aumenta com tanta frequência.

IV.4 Avaliação e Desempenho

IV.4.1 A melhor Heurística

Para que um problema de otimização combinatória seja definido, é preciso declarar:

a) o conjunto L de instâncias

b) para cada instância $I \in L$, um conjunto de candidatas à solução de I, representado por *S(I)*.

c) uma função $F(I, \sigma)$, que associa a cada instância $I \in L$ e a cada candidata à solução $\sigma \in S(I)$, um valor solução.

Em um problema de otimização combinatória, cuja meta é minimizar a função objetivo, a solução ótima é uma candidata à solução $\sigma^* \in S(I)$, onde

$$\forall \sigma \in S(I), F(I, \sigma^*) \leq F(I, \sigma).$$

Em um problema de maximização, σ^* é a solução ótima de uma dada instância I, se

$$\forall \sigma \in S(I), F(I, \sigma^*) \geq F(I, \sigma).$$

Surge a idéia de um algoritmo de aproximação [6] para um problema de otimização combinatória. Este algoritmo nada mais faz do que achar uma candidata σ à solução , do problema em questão, onde $\sigma \in S(I)$.

Os algoritmos de aproximação são uma forte ferramenta, em se tratando dos problemas *NP*-completo, pois esta classe de problemas de otimização combinatória, não possui um algoritmo que ache sua solução ótima, em tempo polinomial. O melhor caminho a seguir é utilizar algoritmos que tenham como resultado, soluções perto da ótima. Estes algoritmos são os de aproximação e uma vez dado um problema, sua execução é feita em tempo polinomial.

Como visto anteriormente, o problema do cobrimento de vértices mínimo é um problema *NP*-completo. Foi estudado então, um algoritmo de aproximação para resolvê-lo, que é baseado no problema de Emparelhamento Maximal de um grafo.

Um emparelhamento de um grafo $G = (V, A)$, consiste em um subconjunto $M \subseteq A$, onde não existe dois arcos pertencentes a M que tenham um vértice em comum. Um emparelhamento é maximal, se todo arco que pertence ao conjunto resultante $A - M$, possui um vértice em comum com um arco de M [6].

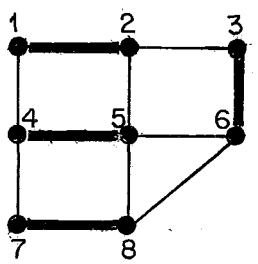


Figura IV.2: Os arcos enfatizados constituem um Emparelhamento Maximal

Sendo M maximal, o conjunto de todos os vértices que compõem seus arcos, constituem um cobrimento de vértice para o grafo G. Isto se observa pelo fato dos arcos de M não possuirem vértice em comum e sendo M, um conjunto maximal,

os arcos restantes de G, $A - M$,possuem um vértice em comum com os arcos do emparelhamento. Assim, todo o arco possui pelo menos um vértice em M. Isto nada mais é do que a definição de cobrimento de vértice de um grafo G. Em relação ao emparelhamento M, o cobrimento de vértices é dado por $2 \cdot \mid M \mid$.

Como já foi visto, um cobrimento de um grafo é um subconjunto I de V, tal que todo arco de G tem pelo menos uma extremidade em I. Um cobrimento I é mínimo, se G não possui um cobrimento I', com $\mid I' \mid < \mid I \mid$. Se I é um cobrimento e M um emparelhamento, então I contém pelo menos uma extremidade de cada arco de M. Assim, para qualquer emparelhamento M e qualquer cobrimento I, tem-se $\mid M \mid \leq \mid I \mid$.

Caso M não fosse maximal, $A - M$ conteria um arco que não tem um vértice em comum com algum arco em M. Logo, existiria pelo menos um arco sem vértices em M. Consequentemente, M não constituiria um cobrimento de vértices do grafo G. Concluindo, o cobrimento de vértices construído a partir de M, nunca é maior que duas vezes o cobrimento de vértices mínimo correspondente.

Além do mais, este algoritmo de aproximação é eficiente, já que pode ser resolvido em tempo polinomial na $O(\mid A \mid)$, pelo simples fato de adicionar-se um arco de cada vez a M.

IV.4.2 Descrição da implementação da heurística usada

O algoritmo implementado, não adiciona cada arco conforme a ordem que aparece no grafo, pois esta não é a maneira mais eficiente para resolução do problema. Seja o grafo $G = (V, A)$ da figura IV.3. Adicionando um arco de cada vez em M, se necessário, o emparelhamento maximal $M \subseteq A$, seria $\{(1,4),(5,6)\}$. Assim o cobrimento de vértice seria $2 \cdot \mid M \mid = 4$. No entanto, a implementação usada obteve resultados mais eficientes.

O grau de um arco (i, j) é definido como sendo a soma dos graus de seus vértices extremos menos duas unidades, pois cada vértice conta o arco (i, j) como sendo uma unidade.

O algoritmo implementado, possui os seguintes passos:

1. Os arcos são ordenados em ordem crescente de grau.
2. O arco de maior grau é colocado no emparelhamento M.
3. Para cada próximo arco (i,j) de maior grau, é verificado se (i,j) já não possui um vértice em comum com arcos do emparelhamento. Se possui, (i,j) não entra no emparelhamento, senão ele é adicionado a M.
4. A cobertura de vértices é dada por $2 \cdot \mid M \mid$.

Este algoritmo aplicado ao grafo da Figura IV.3, resulta em um cobrimento de vértice igual a 2, já que a ordenação dos arcos seria

$$\{(4,5),(1,4),(2,4),(3,4),(5,6),(5,7),(5,8)\}.$$

O primeiro arco escolhido é o arco $(4,5)$ e mais nenhum vai pertencer a M, já que todos os outros têm um vértice em comum com este arco.

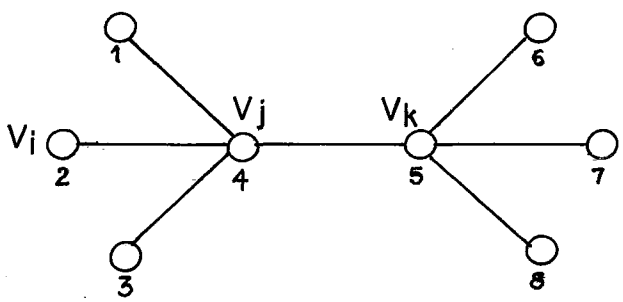


Figura IV.3: Aplicação da heurística no problema do cobrimento de vértice mínimo

IV.4.3 Comparações Experimentais

Simulated Annealing, o método empregado para resolver o problema do cobrimento mínimo de vértice de um grafo $G = (V, A)$, foi executado de uma forma paralela, onde um processo foi gerado para cada vértice do grafo. Os processos vizinhos se comunicam através de canais.

O grafo sobre o qual foi feita a execução do sistema, foi um grafo aleatório, gerado segundo uma certa probabilidade. Quanto menor a probabilidade, mais esparso é o grafo e quanto maior a sua esparsidade, menor o cobrimento mínimo de vértice, pois menos arcos ele possui.

Para os testes, foram gerados três conjuntos de grafos aleatórios, onde cada conjunto foi definido segundo uma probabilidade p. Os valores usados de p foram 0.1, 0.5 e 0.7. Além do mais, cada conjunto de grafos aleatórios com probabilidade p, continha grafos com número de nós $N = 5, 10, 15, 20, 25, \ldots, 290, 295, 300$.

Cada processo gerado, é submetido a uma temperatura inicial T_0 e uma temperatura final T_{final}.

É importante que, na implementaçaõ de *Simulated Annealing*, a temperatura em cada processo diminua lentamente conforme vai aumentando o número de operações do processo. De acordo com o problema, a função que define como a temperatura deve diminuir, pode ser desenvolvida pelo método de tentativa e erro, ou simplesmente resfriando o sistema lentamente até que se chegue ao estado básico de equilíbrio, que no caso corresponde ao mínimo global.

É dito em [7] que a função que manipula a temperatura deve ser logarítmica para que o algoritmo tenha uma grande probabilidade de convergir, isto é, achar o mínimo global. Acredita-se que a função logarítmica é a melhor, pois faz a temperatura decrescer muito lentamente. Mas, como um processo opera até chegar a temperatura final T_{final} e a diferença entre as temperaturas inicial e final é muito grande, se a função logarítmica for usada, cada processo vai operar muitas vezes, tornando a execução de todo o sistema demorada.

Assim, foi escolhido a função que *Kirkpatrick*, em seu artigo original sobre a *Simulated Annealing*, primeiramente utilizou. Seja k, o número da iteração correspondente a uma operação de um determinado processo. Para $k \geq 0$, temos

$$T_k = T_0 \alpha^k,$$

onde T_k é a temperatura que o processo é submetido em sua iteração k, T_0 é a temperatura inicial e α, uma constante que pode assumir valores dentro do intervalo

$(0,1)$, a fim de que a temperatura possa decrescer com o aumento de k. Obviamente, quanto maior α, mais lentamente a temperatura vai decrescer.

Cada grafo foi submetido à execução do sistema com $\alpha = 0.3$ e $\alpha = 0.7$. Neste último, o decréscimo foi mais lento, mas o resultado não foi tão melhor para ser levado em consideração o valor de α.

A temperatura inicial T_0, é uma função do número de nós do grafo: $T_0 = N^2$. A temperatura final T_{final} é constante para qualquer grafo: $T_{final} = 0.1$. Em [8], é especificado que a temperatura final deve assumir um valor próximo a 0 para que o objetivo do problema seja alcançado.

Pode ser que a solução final não seja um cobrimento de vértice como um todo, pois podem existir arcos do grafo, cujos vértices não pertençam ao conjunto de cobrimento mínimo de vértices, ou seja, as variáveis associadas a cada vértice do arco (i,j), X_i e X_j, têm valor 0. Isto se deve ao fato de que, como a temperatura final é pré-fixada, o mínimo global pode ter sido achado em iterações anteriores à iteração final ou ainda, não ter atingido o mínimo global. Para resolver este problema, ao final de todo o processamento paralelo, todos os arcos são percorridos e se for encontrado algum arco (i,j) com $X_i = X_j = 0$, faz-se $X_i = 1$,ou $X_j = 1$. A função objetivo do problema, decresce ainda mais, chegando assim a um resultado melhor. Experimentalmente, poucas vezes ocorreu a existência de arcos não cobertos.

As Figuras IV.4, IV.5 e IV.6, mostram os resultados alcançados. Em todos os casos, é observado que a Simulação do Resfriamento obteve resultados melhores que o método que utiliza o conceito de emparelhamento.

Na Figura IV.4, é observado os resultados com grafos aleatórios com probabilidade $p = 0.1$. A diferença entre o resultado da simulação e o da heurística é maior, neste caso, pois sendo os grafos de menor probabilidade, eles são mais esparsos, logo o cobrimento mínimo de vértice é menor, tendendo a ser igual ao número de arcos do emparelhamento maximal.

É observado também, que com $\alpha = 0.7$, os resultados são um pouco melhores, pois a temperatura decresce mais lentamente, equivalendo a um resfria-

mento mais lento, fazendo com que a simulação chegue a uma solução mais satisfatória. No entanto, a diferença entre os dois tipos de resultado não é tão crítica, para que neste caso, o valor de α seja levado em consideração.

Pelas outras figuras, os grafos aleatórios tendem a ser menos esparsos. A simulação proporciona resultados mais significativos que o emparelhamento, porém, os resultados se aproximam. Isto decorre do fato de que quanto mais denso o grafo, maior o número de arcos. Consequentemente maior será o cobrimento mínimo de vértices.

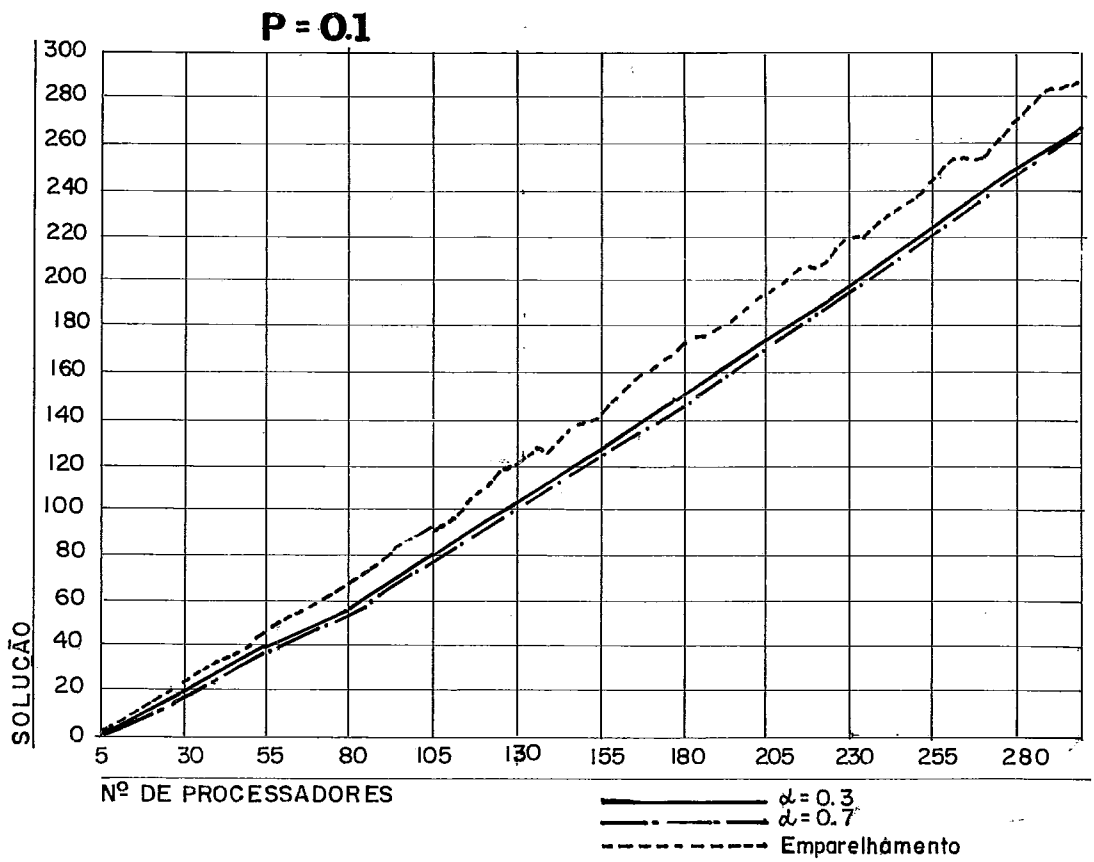


Figura IV.4: Resultados sobre grafos aleatórios com $p = 0.1$

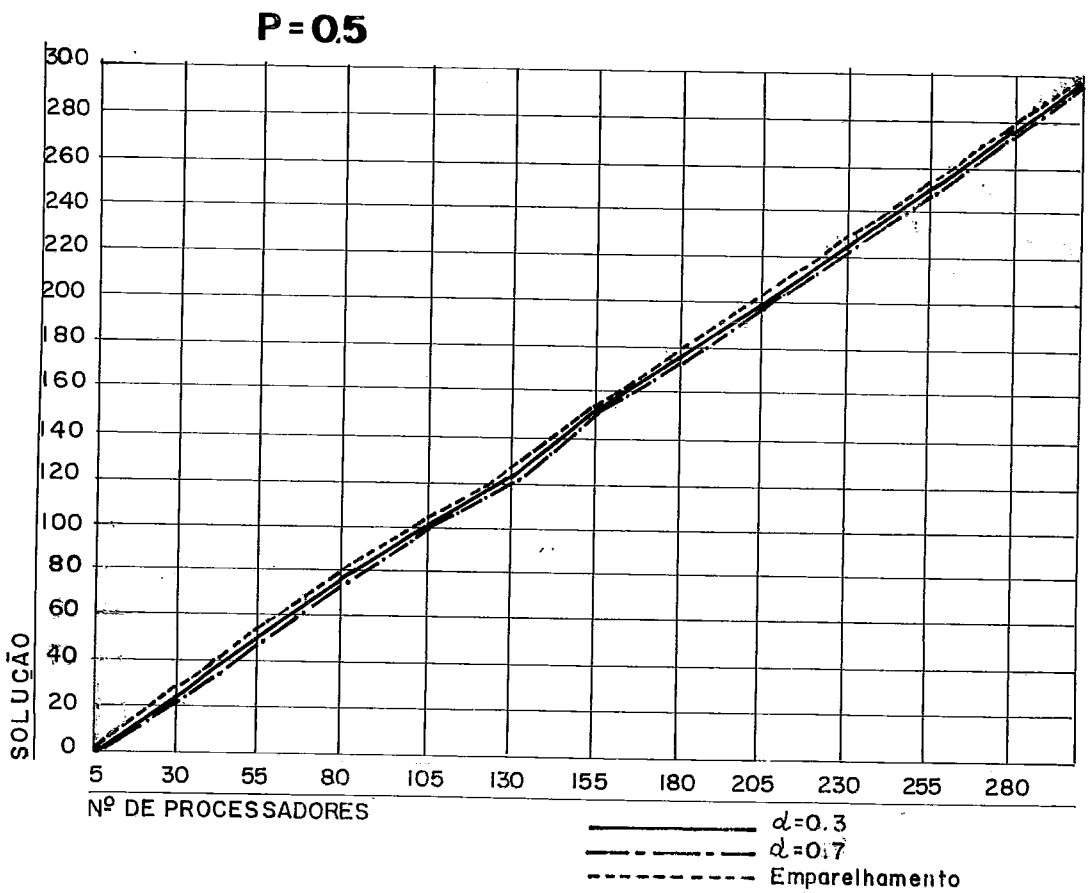


Figura IV.5: Resultados sobre grafos aleatórios com $p = 0.5$

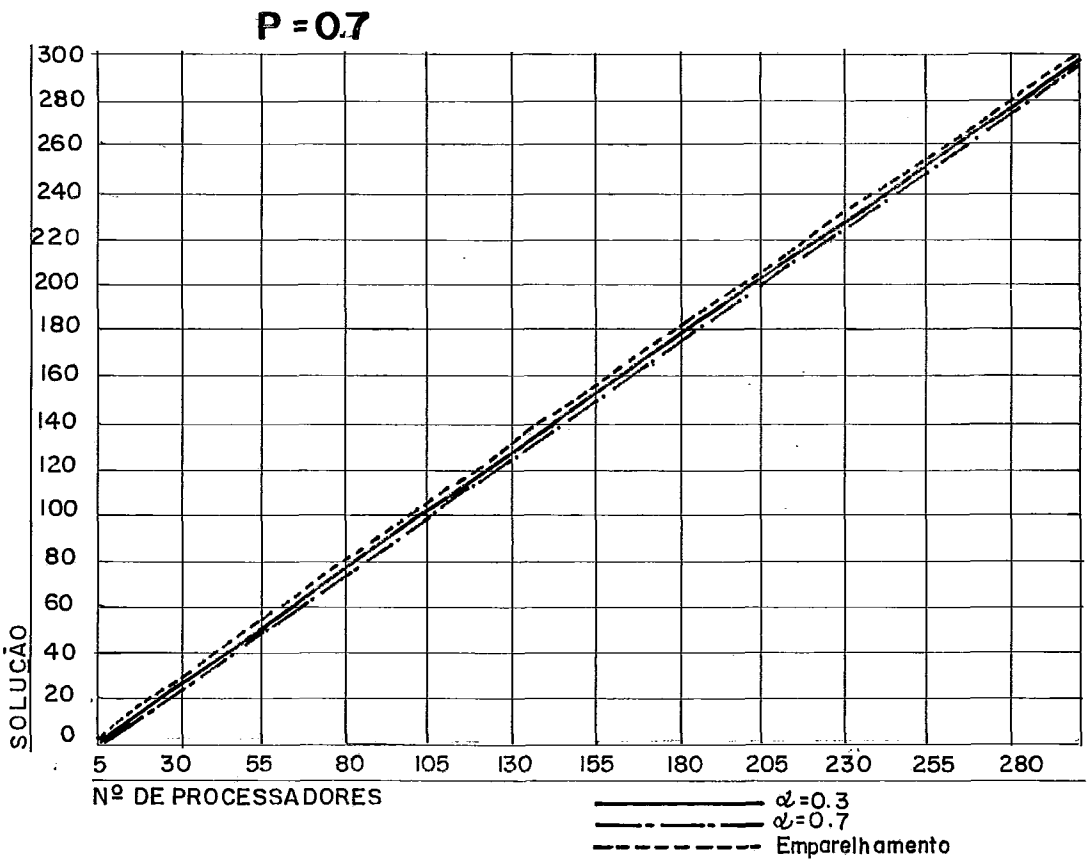


Figura IV.6: Resultados sobre grafos aleatórios com $p = 0.7$

Capítulo V

Recuperando a Solução Ótima

V.1 Eventos, Estados Globais e Grafos de Precedência

Um sistema distribuído consiste de um conjunto finito de processos e um conjunto finito de canais que ligam alguns destes processos. Um processo é definido por :

- seu conjunto de estados locais
- seu estado inicial
- um conjunto de eventos ocorridos durante a computação

Um evento e ocorrido em um processo p é uma ação atômica que muda o estado do processo. Já o estado de um canal pode mudar pelo recebimento ou o envio de mensagens, que também é um evento. Assim, um evento só fica definido quando se especifica:

- o processo p onde ocorreu
- os estados locais de p, anterior e posterior ao evento
- os canais que mudaram de estado, devido ao recebimento e envio de mensagens.

Durante toda a computação, olhando somente para um nó i, vários eventos vão ocorrer, pois um nó se torna sumidouro várias vezes, entrando em operação várias vezes. Seja E_i o conjunto de eventos que ocorrem durante toda a computação no nó i. Então,

$$E_i = \{e_{i,1}, e_{i,2}, \ldots, e_{i,k_i}\}$$

onde k_i é o número de operações do nó i e $e_{i,l}$ é o evento que ocorre no nó i na *l-ésima* operação. Seja E o conjunto de todos os eventos ocorridos na computação, generalizando para todos os nós, ou seja,

$$E = E_1 \cup E_2 \cup \ldots E_n$$

sendo n o número de nós.

Dentro do universo de eventos ocorridos em um sistema distribuído, podemos definir uma relação entre dois eventos, que se chama *ANTES*. Seja e e $e' \in E$. Pode se afirmar que $eANTESe'$, se e somente se:

- e e e' ocorrem no mesmo nó, e e precede e'
- e e e' ocorrem em dois nós vizinhos i e j, respectivamente, e a atualização de X_j depende do valor de X_i, se i opera imediatamente antes de j e assim, e ocorre antes de e'.

Seja um fecho transitivo A da relação *ANTES*, ou seja, se $(e, e') \in A$, então $e, e' \in E$ se, e somente se existem k eventos, tais que

$$eANTESe_1, \quad e_1ANTESE_2, \ldots, \quad e_kANTESe'$$

. A relação A é de ordem parcial sobre os eventos de E, pois é:

1. irreflexiva: $eANTESe$ não pode acontecer
2. transitiva: se $eANTESe_1$ e $e_1ANTESe'$ então, $eANTESe'$

Já no caso de dois nós i e j não vizinhos, não se pode afirmar nada sobre a relação entre os eventos neles ocorridos, ou seja, afirmar que $eANTESe'$ ou que $e'ANTESe$, se torna inválido (e e e' são eventos ocorridos em i e j, repectivamente).

Um grafo de precedência pode ser usado para representar a relação *ANTES*, no qual cada nó , representa um evento e. Um arco que liga dois eventos ocorridos no mesmo processo representa o estado do processo: é o arco de estado. Arcos que ligam eventos que ocorreram em processos diferentes, são os arcos mensagens: é o envio de mensagens entre dois processos.

O estado do sistema é definido se especificando o estado de cada processo e o estado de cada canal. Quando um estado do sistema é consistente, ele é chamado de estado global.

R_T uma relação de ordem total quando, para dois eventos e e e', ou eR_te' ou, $e'R_Te$ e ainda, R_T obedece à ordem parcial.

Um estado do sistema é global se existe uma ordem total.

Seja φ um estado global. Se ocorrer um evento e, de tal maneira que mude o estado de um processo p e um canal c, o novo estado global será então, φ', que só difere de φ devido aos novos estados do processo p e do canal c, que mudaram com a ocorrência de e.

Fica aqui definido que cada canal só pode mandar mensagem em uma direção, por isto, entre dois processos são colocados dois canais unidirecionais, para envio e recebimento de mensagens. É suposto então, que um sistema é constituído de dois processos p_0 e p_1, que se comunicam através dos canais $c_{0,1}$ e $c_{1,0}$ como mostra a Figura V.1.

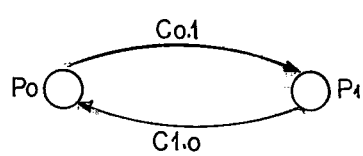


Figura V.1: Dois processos com canais de comunicação

É observado na figura V.1:

1. Inicialmente, p_0 contém a mensagem m a ser enviada para p_1.

2. p_1 está esperando pela mensagem e $c_{0,1}$ e $c_{1,0}$ estão vazios. Este conjunto de estados define o estado global inicial.

3. Ocorre um evento: p_0 manda a mensagem m para p_1, caracterizando um novo estado global. Agora p_0 está vazio, $c_{0,1}$ contém m, p_1 ainda vazio espera por m e $c_{1,0}$ está vazio.

4. A seguir, ocorre um outro evento: p_1 recebe m, resultando em um novo estado global com a seguinte configuração: p_0 está vazio, a espera de m, $c_{0,1}$ está vazio, pois a mensagem já foi enviada a p_1, p_1 contém m e $c_{1,0}$ está vazio.

5. Mais um estado global é criado, caracterizado pela ocorrência de mais um evento: o envio da mensagem m a p_0 por p_1. Este estado global é formado pelos estados de p_0 vazio, $c_{0,1}$ vazio, $c_{1,0}$ com m e p_1 vazio, pois este já enviou a mensagem.

Voltando à computação distribuída deste trabalho, é sabido que o seu objetivo, é de minimizar a função f de tal maneira que se ache o mínimo global no sistema. Associado ao sistema distribuído, está o grafo $G = (V, A)$, onde cada nó contribui no cálculo do valor de f, atualizando o valor da variável X_i, associada ao nó. É a atualização desta variável que caracteriza um evento ocorrido em um nó i. A partir do estado global inicial definido pelos estados iniciais dos nós (associados aos processos) e dos arcos (associados aos canais), são gerados estados globais para cada evento ocorrido no sistema distribuído.

Neste sistema, podem existir dois nós que não são vizinhos, porque, não existe um arco que os ligue. Esses nós associados a processos , podem operar ao mesmo tempo se eles se tornarem sumidouros simultaneamente, podendo potencialmente operar em paralelo. Mas não se pode afirmar nada sobre a relação entre os eventos em cada nó não vizinho, pois as atualizações de suas variáveis podem não ocorrer exatamente no mesmo instante. Já em nós vizinhos, como a atualização de suas variáveis são interdependentes, seus eventos não podem ocorrer em paralelo. Com isto, um estado global difere de um estado global imediatamente anterior de, no mínimo, um estado de um nó (processo) e de um arco (canal).

Surge então, a idéia de grafos de precedência, que representa as relações entre todos os eventos que ocorreram durante toda a computação distribuída. Cada evento ocorrido será representado no grafo de precedência por um nó e se $e_i ANTES e_{i+1}$, então, os dois nós que representam os eventos são ligados por um arco com origem no nó e_i e destino no nó e_{i+1}.

Suponha $G = (V, A)$ um grafo orientado acíclico e $GP = (E, ANTES)$, o grafo de precedência correspondente, como mostra a Figura V.2. Cada eixo horizontal representa o eixo do tempo de cada processo, correspondendo aos eventos ocorridos no nó, durante a computação. As setas representam a relação *ANTES*. É importante observar que, um evento em um nó i ocorre antes do evento ocorrido em seu vizinho j, se i opera imediatamente antes de j, e se com a reversão dos arcos de i, o próximo a se tornar sumidouro é o j.

Cada primeiro evento ocorrido em cada nó, será ligado a um nó s, chamado nó origem, que representa um único evento: o início da computação. A ligação para cada evento $e_{i,1}$ será feita com um arco com direção $s \longrightarrow e_{i,1}$. Da mesma forma, depois de cada último evento e_{i,k_i} ocorrido em cada nó, um evento ocorre: o final da computação. Este evento será representado pelo nó t, chamado de nó destino, e todo evento e_{i,k_i} será ligado a t por um arco com direção $e_{i,k_i} \longrightarrow t$. É definido assim, sobre o grafo de precedência mais um grafo $H = (J, A')$, onde

$$J = \{s, t\} \cup E$$

e

$$A' = ANTES \cup \{(s \longrightarrow e_{i,1}), (e_{i,k_i} \longrightarrow t), \forall i \in V\}$$

como mostra a Figura V.3.

Então, em H existem vários caminhos que ligam s a t. Para cada nó $i \in V$, é definido um caminho:

$$s \longrightarrow ei, 1 \longrightarrow \ldots \longrightarrow e_{i,k_i} \longrightarrow t$$

onde $(s \longrightarrow e_{i,1})$ caracteriza o estado inicial do nó i e $(e_{i,k_i} \longrightarrow t)$ o seu estado final. Aos arcos que ligam eventos que ocorrem em um mesmo nó, damos o nome de arcos de estado. Aos outros arcos de H, que ligam eventos que ocorrem em nós vizinhos,

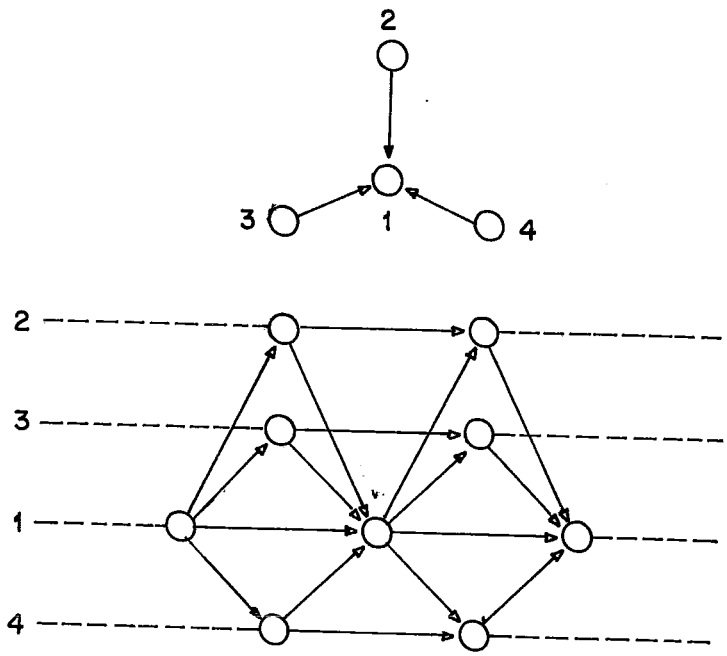


Figura V.2: Grafo orientado acíclico $G = (V, A)$ e seu grafo de precedência $GP = (E, ANTES)$

caracterizando o envio de mensagens entre os nós e consequentemente a transição dos estados dos canais, damos o nome de arcos de mensagens.

Em um grafo $H = (J, A')$ com s e t como nós origem e destino, respectivamente, um corte é definido como um subconjunto de arcos de A', onde cada caminho de s a t usa pelo menos um membro do subconjunto. Esse corte é chamado de corte s-t. Todos os caminhos de s a t são separados por um arco pertencente ao corte s-t.

O corte s-t divide o conjunto J de vértices de H em dois subconjuntos S e $\overline{S}$ onde, $s \in S$ e $t \in \overline{S}$. $S : \overline{S}$ representa o conjunto de arcos que têm origem em vértices de S e destino em vértices pertencentes a $\overline{S}$. Já o conjunto $\overline{S} : S$ representa os arcos que saem de vértices de $\overline{S}$ e terminam em vértices de S.

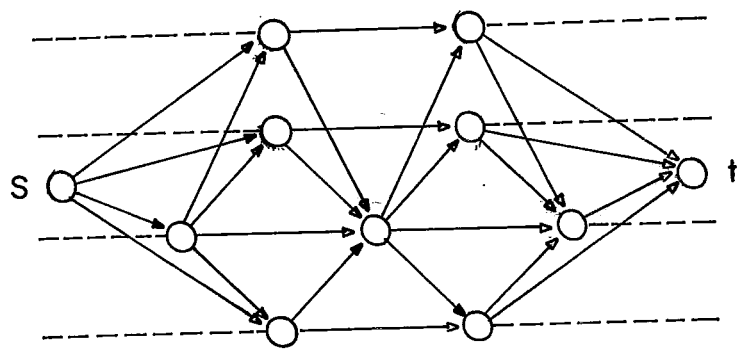


Figura V.3: Grafo $H = (J, A')$

Em um corte *s-t* de um grafo de precedência representado por H, é observado que os arcos de estado que pertencem ao corte representam o estado local do nó. Os arcos mensagens que estão no corte, correspondem aos estados de canais.

Dado um evento $v \in J$, é definido o passado e o futuro de v, como sendo os eventos que ocorreram antes e depois do evento v, respectivamente, isto é:

$$PASSADO(v) = \{v'/v' \in J \text{ e } \quad v'ANTESv\}$$

$$FUTURO(v) = \{v'/v' \in J \text{ e } \quad vANTESv'\}$$

Da mesma forma, um corte c, pode repartir um grafo de precedência em dois conjuntos de eventos: $PASSADO_c$ e $FUTURO_c$. Um corte c é consistente, se para um evento qualquer $e \in FUTURO_c$ e $eANTESe'$, onde $e \neq e'$, então, $e' \in FUTURO_c$.

Um estado global é caracterizado pelo fato de que nenhum evento do futuro anteceda um evento pertencente ao passado. Logo, podemos dizer que o estado global é consistente.

Um corte c em um grafo de precedência H, define um estado global consistente, onde nenhum evento pertence ao $FUTURO_c$ anteceda a um evento do $PASSADO_c$. Isto é equivalente a dizer que um corte em H, não conterá nenhum arco que saia do lado direito e vá para o lado esquerdo.

O corte s-t representa um estado global no grafo de precedência $H = (J, A')$, quando $\overline{S} : S = \emptyset$ (nenhum arco do futuro e vai para o passado).

É observado que o corte que caracteriza um estado global terá exatamente n arcos de estado, onde n representa o número de processos que deram origem ao grafo de precedência. Se o número de arcos de estado presentes no corte for menor que n, significa que os nós s e t, ou pertencem a S ou pertencem a $\overline{S}$, não caracterizando assim, o corte. Neste caso, pode ainda existir algum nó cujo arco de estado não se apresenta no corte e consequentemente, o nó não participa do estado global.

Por outro lado, se o corte possuir mais de n arcos de estado, ficará caracterizado a presença de um arco pertencente a $\overline{S} : S$, pois o corte passará por mais de uma vez em arcos de estados pertencentes a um mesmo nó. Isto não representará um estado global, pois existirão arcos que saem do futuro e vão para o passado.

Seja φ um estado global. Na verdade, φ é um corte no grafo de precedência em questão. Pode-se separar exatamente o passado e o futuro do estado global. Além do mais, já é sabido que o estado global vai possuir o estado atual de cada processo, pois o corte que corresponde a este estado global, contém o estado atual de cada nó do grafo que deu origem ao grafo de precedência. Então,

$$PASSADO(\varphi) = S - \{s\}$$

ou seja, o conjunto de todos os eventos que ocorreram antes do estado global φ. O evento representado pelo nó s não é considerado como o primeiro evento ocorrido dentro de um nó qualquer do grafo. Por sua vez,

$$FUTURO(\varphi) = \overline{S} - \{t\}$$

representa todos os eventos ocorridos após o estado global φ, até o final da computação distribuída. O evento representado pelo nó t, não é considerado como sendo o último evento ocorrido em um nó qualquer, por isto é excluído.

O conjunto de todos os estados globais será representado por Φ.

V.2 O Problema - Recuperando a Melhor Solução

V.2.1 Definição do Problema

A cada estado global φ, está associado um conjunto de estados correntes de cada nó pertencente ao grafo $G = (V, A)$, que deu origem ao grafo de precedência $H = (J, A')$. O estado de cada nó $i \in V$ é caracterizado pelo valor corrente da variável X_i, associada ao nó. Um ponto $x \in D^n$, onde a componente i do ponto contém o valor da variável X_i, representa o estado global φ.

A atualização da variável associada a cada nó, caracteriza a ocorrência de um evento. A contribuição dada pelo nó ao sistema é definido como sendo a diferença entre o valor de f depois da atualização e o valor de f antes da atualização.

Ao final de todo o processo, tem-se armazenado todas as contribuições a f resultante da ocorrência de cada evento. Deste modo, pode-se calcular o valor de f até um estado global φ, partindo de um estado global inicial. Durante todo este processo, os eventos contribuem para o acréscimo ou decréscimo valor de f.

Seja φ_o o estado global inicial, onde as variáveis associadas a cada nó, têm os seus valores iniciais. Deve-se de imediato calcular o valor de f inicial, a partir dos valores iniciais das variáveis. O valor de f, calculado até o estado global φ, vai ser a soma das contribuições desde o estado inicial do sistema até φ, ou seja,

$$f(\varphi) = f(\varphi_o) + \sum_{e \in PASSADO(\varphi)} \delta_f(e)$$

onde $\delta_f(e)$, é a contribuição dada a f por um determinado evento $e \in PASSADO(\varphi)$.

Da mesma forma que f pode ser calculado localmente, a contribuição também poderá. Seja e um evento ocorrido durante a computação, em um determinado nó i. φ_i e φ_j são dois estados globais imediatamente anterior e posterior , ao evento e, tal que,

$$PASSADO(\varphi_j) - PASSADO(\varphi_i) = \{e\}$$

Como f pode ser calculado localmente, e a contribuição depende exclusivamente do valor de f, tem-se

$$\delta_(e) = \sum_{\substack{Y \subseteq X \\ X_i \in Y}} g_Y(\varphi_j) - \sum_{\substack{Y \subseteq X \\ X_i \in Y}} g_Y(\varphi_i)$$

Cada nó pode calcular sua contribuição a f, pois somente envolve informação local.

Sabe-se que *Simulated Annealing*, fazendo uma busca estocástica sobre o conjunto de pontos viáveis, permite que em determinados momentos, sejam aceitos *aumentos* no valor de f. No final da computação distribuída, pode-se ter passado de um estado global φ em que f foi menor que todos os outros estados globais. Em resumo, o problema é: recuperar o menor valor de f, que foi calculado em algum ponto durante a computação distribuída.

Quando f aumenta, a contribuição do evento ocorrido em um determinado nó ao sistema é positiva. Recuperar o menor valor de f, corresponde a recuperar o estado global φ, cuja soma das contribuições até aquele ponto tenha sido o menor, ou seja, achar o estado global φ tal que

$$\sum_{e \in PASSADO(\varphi)} \delta_f(e)$$

seja a mínima, para que f neste estado, seja o mínimo.

Como a contribuição pode ser calculada localmente, o somatório acima pode ser escrito como sendo:

$$\sum_{i \in V} \sum_{e \in E_i \cap PASSADO(\varphi)} \delta_f(v)$$

Fazendo, $g_i(\varphi)$, como sendo o somatório da contribuição de cada evento ocorrido no nó i até o estado global φ, tem-se

$$g_i(\varphi) = \sum_{e \in E_i \cap PASSADO(\varphi)} \delta_f(v)$$

Pode-se dizer então, que o estado global ótimo φ^* é tal que

$$f(\varphi^*) \leq f(\varphi), \qquad \forall \varphi \in \Phi$$

se e somente se

$$\sum_{i \in V} g_i(\varphi^*) \leq \sum_{i \in V} g_i(\varphi), \qquad \forall \varphi \in \Phi.$$

O Fluxo na Rede

Seja o seguinte caminho de s até t

$$s \longrightarrow ei, 1 \longrightarrow ... \longrightarrow e_{i,k_i} \longrightarrow t$$

através do nó i.

Cada arco de estado deste caminho representa o estado do nó i em cada evento ocorrido durante a computação. Então, cada arco estado representa a contribuição, que é calculada localmente,a f até aquele momento.

O grafo $H = (J, A)$, equivale a uma rede, e pode-se então especificar para cada arco da rede uma capacidade máxima de fluxo.

Problema do Fluxo Máximo na Rede

O que se pretende fazer, é achar o estado global onde f seja mínimo. Solucionando o problema através do método de fluxo máximo, o objetivo então, é maximizar $-f$. Para tal, os arcos de estado recebem $g_i(\varphi)$, como capacidade máxima de fluxo que pode passar por eles. No entanto, determinados $g_i(\varphi)$ podem ser negativos, e o método de Fluxo Máximo não resolve o problema com arcos com fluxos negativos.

É necessário então, substituir estes fluxos , através de funções não negativas, por valores positivos. Para isto, acha-se o menor $g_i(\varphi)$, (geralmente é um valor negativo), subtraindo este valor do fluxo de cada arco de estado do nó i correspondente. Assim, tem-se

$$h_i(\varphi) = g_i(\varphi) - \overline{g_i},$$

$\forall i \in N$ e $\forall \varphi \in \Phi$, onde

$$\overline{g_i} = \min\{0, \min_{\varphi \in \Phi} g_i(\varphi)\}$$

Ocasionalmente pode existir um corte s-t em H, com um arco pertencente ao conjunto $\overline{S} : S$, por exemplo, $(v \longrightarrow v')$, sendo i, o nó em que este arco se processa. Observa-se aqui, que este corte não caracteriza um estado global. As mensagens, através de v, vão para os vizinhos de i e são recebidas por v' pelos mesmos vizinhos. Pelo menos uma mensagem também corresponde a um arco pertencente a $\overline{S} : S$. Com isto, fica caracterizado então, um estado global em H, através de um corte s-t sem arcos mensagens em $\overline{S} : S$.

Daí, é formado um outro grafo $\overline{H} = (J, \overline{A})$, onde $\overline{A}$ resulta da reversão de todos os arcos mensagens pertencentes a A.

Na rede $\overline{H} = (J, \overline{A})$, os arcos de estado têm como capacidade máxima de fluxo, $h_i(\varphi)$. Já os arcos mensagens recebem um fluxo muito alto, ou seja, ∞, pois o que interessa é achar os arcos de estados que participam do estado global ótimo φ^*.

O resultado do método de Fluxo Máximo que passa por uma rede é definido como sendo todo o fluxo que chega ao nó destino t, caracterizado pelo valor de um corte s-t, ou seja, a soma dos fluxos resultantes dos arcos que saem de S e chegam a $\overline{S}$.

Como já demonstrado em [2], cortes s-t com valores finitos em $\overline{H}$, têm uma correspondência biunívoca com estados globais. Assim, achar um estado global ótimo, corresponde a achar um corte s-t, cujo valor é mínimo.

No teorema de Fluxo Máximo-Corte Mínimo [13], é demostrado que

o valor máximo de fluxo que pode passar em uma rede, corresponde ao valor do corte s-t mínimo.

Aplicando o algoritmo de Fluxo Máximo ao grafo $\overline{H} = (J, \overline{A})$, chega-se ao valor do maior fluxo que pode passar na rede. Conseqüentemente, pelo teorema citado, este valor, é o valor do corte s-t mínimo. Por sua vez, ele corresponde ao estado global φ^*, cujo valor é mínimo dentre todos os estados globais gerados durante a computação distribuída.

Resultados Práticos

Foi observado que, *Simulated Annealing*, utilizada para achar uma boa solução para um problema *NP*-completo, do Cobrimento de Vértices Mínimo, acha o menor resultado dentre todos aqueles calculados durante a computação.

Dentre todos os valores de f, que a simulação calcula, o valor final poderia não ser o melhor, sendo por isto utilizado o algoritmo de Fluxo Máximo para recuperar a melhor solução. Porém, o que foi verificado na prática, que o resultado das execuções sobre os grafos de precedência $\overline{H} = (J, \overline{A})$, foi exatamente o valor final calculado pela computação distribuída sobre os grafos $G = (V, A)$.

Conclui-se, que não há necessidade de executar o algoritmo de Fluxo Máximo sobre os resultados da computação distribuída para recuperar o melhor estado global, pois o último estado gerado é o estado ótimo φ^*.

Capítulo VI

Análise de Desempenho e Conclusões

VI.1 Divisão entre Processadores

A implementação distribuída do algoritmo *Simulated Annealing* foi testada em uma rede de *transputers*. A rede é constituída de quatro processadores que se comunicam através de ligações bidirecionais, conforme descrito no Apêndice B.

A título de comparação de divisão de tarefas entre processadores, os testes foram executados sobre grafos formados aleatoriamente com probabilidade $p = 0.1$. Foram processados grafos com número de nós $N = 5, 10, 15, \ldots, 65, 70$. A limitação do número de nós está associada às limitações do ambiente de trabalho.

Cada nó do grafo está associado a uma tarefa. A divisão de tarefas foi feita segundo um método de alocar várias tarefas em diferentes processadores (tese desenvolvida na COPPE). As tarefas foram divididas em 2, 3 e 4 processadores, além da execução de todas as tarefas em um só processador.

Sendo o número de processadores muito menor do que o número de tarefas executadas, logicamente, a divisão em vários processadores resultou na alocação de várias tarefas em cada processador.

Tarefas vizinhas, alocadas em um mesmo processador, utilizam canais

lógicos. Por outro lado, tarefas vizinhas alocadas em processadores distintos, fazem as comunicações utilizando as ligações físicas entre os processadores. Neste caso, quando os nós vizinhos trocam mensagens, o custo de comunicação vai ser maior do que se eles estivessem no mesmo processador.

O gráfico da figura VI.1 apresenta a comparação entre os tempos de processamento de cada grafo em 1, 2, 3 e 4 processadores. Levando em consideração os tempos de execução, verifica-se na maioria dos grafos testados, que é melhor executar todos os processos em um só processador.

Em resumo, o motivo para tal problema, está no fato de que o custo de comunicação é muito alto, se for comparado com o custo de processamento. Para grafos de pequena dimensão, o tempo de processamento não é tão alto. No entanto, o interessante é tentar resolver o problema sobre grafos de dimensões bem maiores.

VI.2 Emparelhamento × Simulated Annealing

O gráfico da figura VI.2 mostra a comparação entre os tempos de execução da implementação distribuída do *Simulated Annealing* e da heurística implementada para resolver o problema do Cobrimento de Vértices Mínimo, utilizando o conceito de emparelhamento. Nos grafos aleatórios, cujos números de nós são $N = 5, 10, 15, \ldots, 290, 295, 300$, com probabilidade $p = 0.1$, verificou-se que a heurística é mais rápida que o *Simulated Annealing*.

É importante salientar, que as execuções aqui descritas foram feitas em um só processador.

A figura VI.3, mostra a comparação, para grafos aleatórios com probabilidade $p = 0.5$ e a figura VI.4, para grafos com probabilidade $p = 0.7$.

Observando os três gráficos, concluímos que os grafos mais esparsos são resolvidos mais rapidamente pela implementação distribuída. Isso é decorrente do fato de menos processos serem vizinhos entre si e consequentemente, o custo de comunicação é menor.

VI.3 Conclusões

O método físico conhecido como *Annealing*, se aplica de maneira eficiente à procura de um estado básico de energia de materiais, para que seja possível observar propriedades e características destes. O método leva o material a um estado de equilíbrio térmico global e uniforme.

Fazendo uma analogia com problemas de otimização combinatória, verificou-se a possibilidade de simulação do método para ser aplicado à solução de um problema desse tipo e chegou-se a resultados consideráveis. O problema de otimização combinatória estudado é *NP*-completo mapeado em grafo. Cada vértice do grafo corresponde a uma variável do problema.

Simulated Annealing faz uma busca estocástica dentro de um conjunto de pontos viáveis à solução. Foram estudados problemas cujas variáveis formam um *Campo Aleatório de Markov*, ou seja, o valor de uma variável está exclusivamente relacionado com os valores das variáveis vizinhas.

O método, ao ser aplicado a tais tipos de problemas, não precisa conhecer os valores de todas as outras variáveis para calcular o valor de uma determinada variável. Por esta razão, tornou-se viável a implementação de um algoritmo de forma distribuída para *Simulated Annealing*.

O problema estudado, do Cobrimento de Vértices Mínimo, já possuía uma heurística eficiente para resolvê-lo, implementada em uma versão seqüencial. Seus resultados foram comparados com os resultados produzidos pelo *Simulated Annealing* e este último obteve melhores soluções do que o primeiro, em todos os testes.

Em relação ao tempo de execução, a implementação do *Simulated Annealing* não se comportou conforme o desejado. Um motivo para tal fato, é o gerenciamento feito pelo sistema sobre o qual foram implementados os algoritmos distribuído e centralizado. É feita uma troca de contexto de processos que concorrem a um único processador, o que não acontece em processos seqüenciais, que é o caso da heurística [Apêndice B].

Finalmente, como o *Simulated Annealing* faz uma busca estocástica, foi preciso recuperar a melhor solução dentre todas as calculadas durante o processamento. Através do método de Fluxo Máximo, foi observado que o *Simulated Annealing* sempre chega à menor solução no final da computação. Assim, não se faz necessário a utilização do algoritmo de Fluxo Máximo para recuperar tal solução.

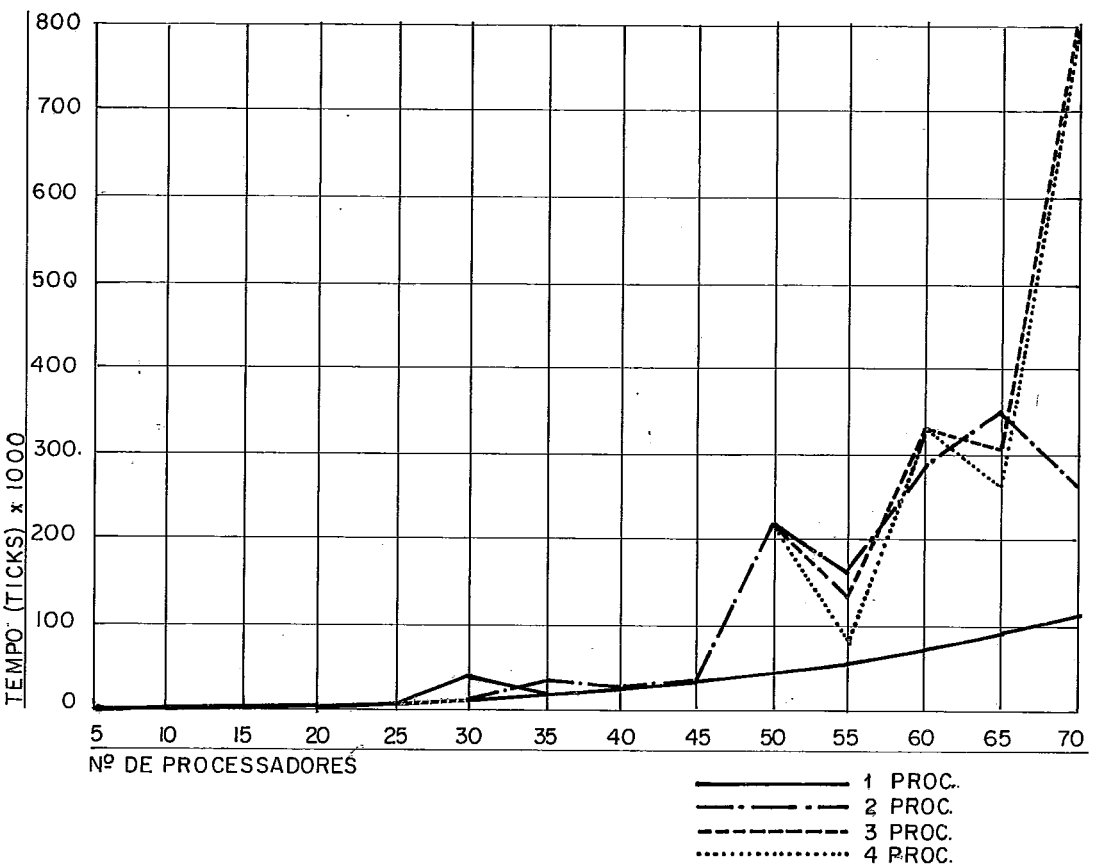


Figura VI.1: Divisão de Tarefas em 1, 2, 3 e 4 processadores

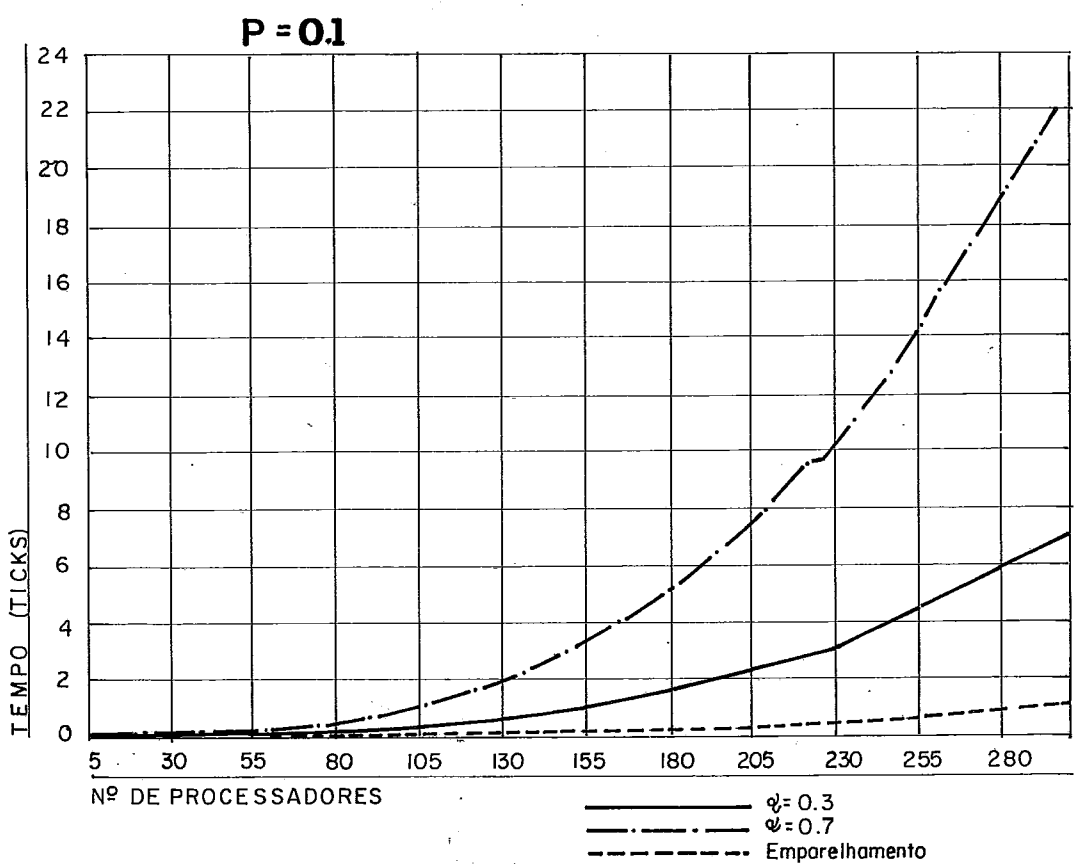


Figura VI.2: Emparelhamento × *Simulated Annealing* - Grafos com $p = 0.1$

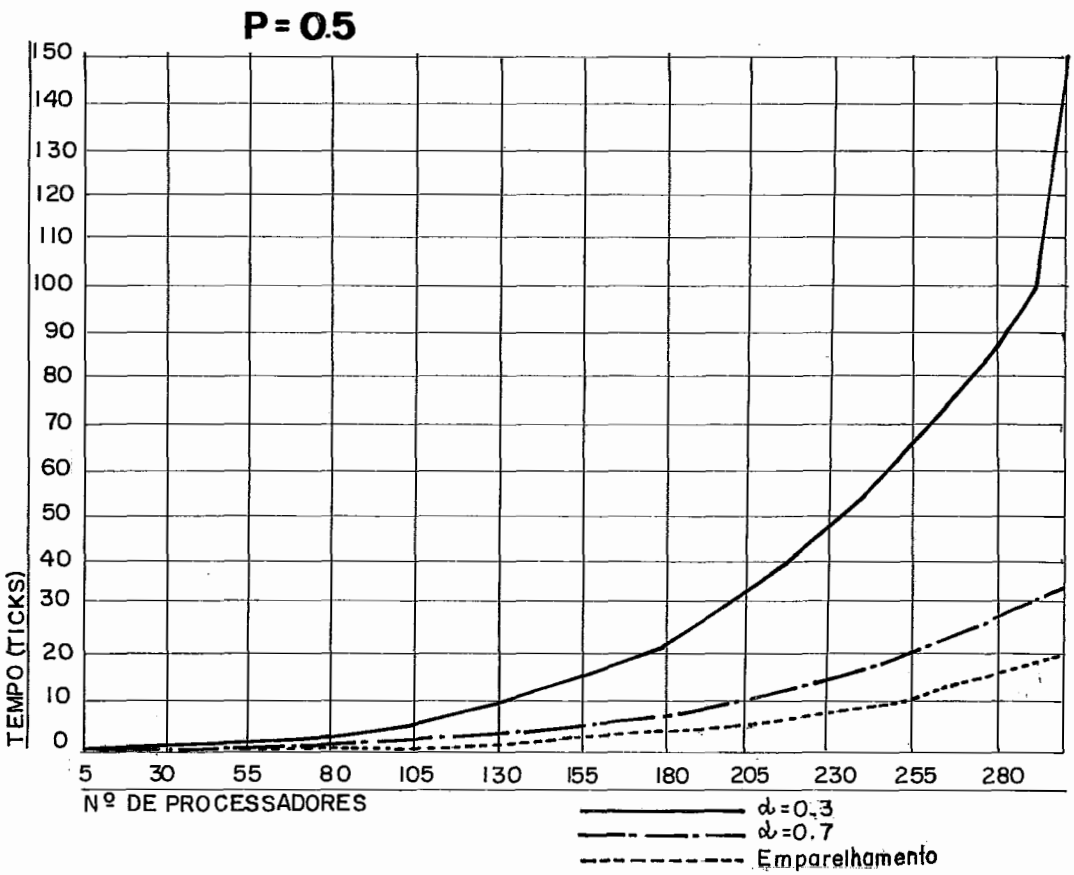


Figura VI.3: Emparelhamento × *Simulated Annealing* - Grafos com $p = 0.5$

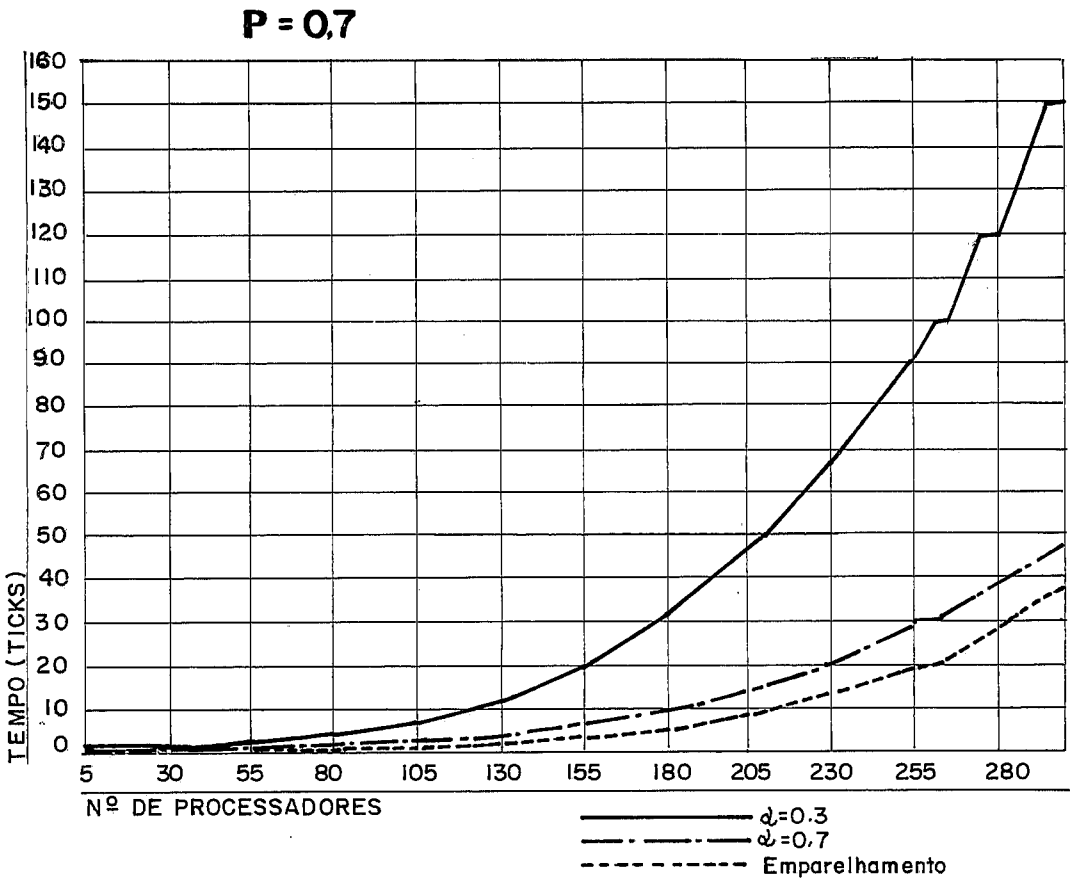


Figura VI.4: Emparelhamento × *Simulated Annealing* - Grafos com $p = 0.7$

Bibliografia

[1] Aragon, C. R., Johson, D. S., McGeoch, L. A., and Schevon, C., "Optimization by simulated annealing: an experimental evaluation," *Wokshop on Statistical Physics in Engineering and Biology* (1984).

[2] Barbosa, V., C. and, Gafni, E., "A distributed implementation of simulated annealing", submited to *Journal of Parallel and Distributed Computing*

[3] Bondy, J. A., Murty, U. S. R., "Graph Theory with Applications", *University of Waterloo*

[4] Burns, A., *Programming in Occam 2*, Addison-Wesley Plubishing Company (1988)

[5] Chandy, K. M., and Lamport, L., "Distributed snapshots: determining global states of distributed systems". *ACM Transactions on Computer Systems* **3**, 1 (February 1985), **63-75**

[6] Garey, M. R., Johnson, D. S., *Computers and and Intractability: A Guide to the Theory of NP-completeness*, Freeman, San Francisco, CA(1979).

[7] Gelatt Jr., C. D., and Vecchi, M. P., "Optimization by simulated annealing," *Science* **220** (4598), pp. **671-680** (May 13, 1983)

[8] Gemman, S., and Gemman, D., "Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images", *IEEE Transactions on Pattern Analysis and Machine Intelligence* **PAMI-6** (6), pp. **721-741** (November 1984).

[9] Goldberg, A. V., and Tarjan, R. E., "A new approach to the maximum flow problem". *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, Berkeley, CA (May 1986), **136-146**

[10] INMOS Limited, **TDS** - *Transputer Development System*, Prentice Hall (1988)

[11] Kinderman, R., and Snell, J. L., "Markov Random Fields and their Applications", *American Mathematical Society*, Providence, RI(1980).

[12] Lamport, L., "Time, clocks, and the ordering of events is a Distributed System". *Comunications of the ACM* **21, 7** (July 1978), 558-565.

[13] Lawer, E. L. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinchart and Winston, New York, NY(1976).

[14] Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., Teller, E., *J. Chem. Phys.*, 21, 1087(1953)

[15] Papadimitriou, C. H., and Steiglitz, K. *Combinatorial Optimization.* Prentice-Hall, Englewood Cliffs, NJ(1982).

Appendix A

O algoritmo utilizado no cálculo do Fluxo Máximo

A.1 Introdução

O problema consiste em achar o fluxo máximo de um grafo orientado acíclico dado o número de nós do grafo, os vizinhos de cada nó, determinando a orientação dos arcos e as capacidades destes arcos.

Vários algoritmos já foram propostos, como os de *Even*, *Ford* e *Fulkerson*, *Lawer*, *Papadimitriou* e *Steiglitz*, e *Tarjan*.

Dentre os tipos de algoritmos propostos, nenhum deles se destaca por ter melhor eficiência sobre os outros. Eles diferem em eficiência dependendo do grafo sobre o qual estão sendo aplicados.

A.2 Descrição do algoritmo

O algoritmo usado neste trabalho se baseia numa nova idéia especificada por *Tarjan* em [9]. É usado no algoritmo a idéia de um *pré-fluxo*, que é a mesma coisa que um fluxo, com a diferença que, a quantidade de fluxo que entra em um vértice pode diferir da quantidade de fluxo que sai do mesmo vértice, contrariando a Lei de Conservação de fluxo.

O algoritmo mantém em cada fase, um pré-fluxo sobre cada arco de

um grafo orientado acíclico, determinando um novo grafo acíclico residual para a próxima fase. Este grafo acíclico residual, é formado pela capacidade restante de fluxo de cada arco do grafo.

Não há necessidade de cada vértice ter o conhecimento global do grafo em cada fase. Um vértice só precisa saber da quantidade de fluxo que ele recebeu e quais são seus vizinhos, para que ele escolha um vizinho cujo o arco ainda é residual e que faça parte do menor caminho em direção ao destino.

Se em algum momento existe um nó com excesso de fluxo, mas não existe caminho residual em direção ao destino, este excesso é retornado ao nó origem, também escolhendo dentre os arcos a ele ligados, aquele que oferece o menor caminho em direção à origem.

Quando o algoritmo termina, o pré-fluxo se torna fluxo, ou seja, o que chega ao nó é igual ao fluxo que sai dele. Este fluxo é o máximo que pode passar nos arcos.

Este algoritmo, pelo detalhe da localidade descrito, pode ser implementado não só sequencialmente, como também em paralelo. O modo escolhido para a implementação entretanto, foi o mais simples: seqüencial.

A.3 Implementação

O algoritmo foi executado sobre grafos de precedência $H = (J, A')$. Este grafo é uma rede com fluxo, pois existe um nó origem s, e um nó destino t e para cada arco $(v, w) \in A'$, está associada a sua capacidade máxima de fluxo $c(v, w)$. É importante observar que, se $(v, w) \notin A'$, então $c(v, w) = 0$. Seja $f(v, w)$, o fluxo que passa pelo arco (v, w). Este fluxo deve obedecer às seguintes restrições:

1. $f(v, w) \leq c(v, w), \quad \forall (v, w) \in J \times J$
2. $f(v, w) = -f(w, v), \quad \forall (v, w) \in J \times J$
3. $\sum_{w \in J} f(v, w) = 0, \quad \forall v \in J - \{s, t\}$

O valor do fluxo f da rede é definido como sendo todo fluxo que chega ao destino:

$$| f |= \sum_{v \in J} f(v,t)$$

O fluxo máximo é o fluxo de maior valor.

Neste algoritmo, o fluxo máximo é achado manipulando um pré-fluxo f que respeita as restrições 1 e 2 citadas acima e difere da restrição 3 da seguinte forma

$$\sum_{v' \in J} f(v,v') \geq 0, \ \forall v \in J - \{s\}$$

pois, o fluxo não obedece à Lei da Conservação até o final do algoritmo.

O excesso de fluxo de um nó v, $e(v)$, é todo fluxo que chega a este nó:

$$e(v) = \sum_{v' \in J} f(v',v)$$

e a capacidade residual de um arco $(v,v') \in A'$ é definido como sendo

$$r(v,v') = c(v,v') - f(v,v')$$

Cada vértice do grafo mantém uma lista indicando quem são os seus vizinhos. Para isto, cada arco do grafo é visto como sendo não direcionado, ou seja, se existe um par $\{v, w\}$, ou $(v,w) \in A'$, ou $(w,v) \in A'$. Se $\{v,w\}$ é um par formando um arco não direcionado, v pertence à lista de w e w pertence à lista de v. Assim, cada vértice tem a visão dos arcos que saem dele e são caminho para o destino, e dos arcos que chegam a ele , e são caminho para a origem.

Considere $v \in J$ um vértice que possui excesso de fluxo para mandá-lo em direção ao vértice destino t, através do menor caminho a ser escolhido. Se existe um arco residual $(v,v') \in A'$, onde $r(v,v') > 0$, então o fluxo mandado a v' por v é o mínimo entre $e(v)$ e $r(v,v')$. Explica-se isto, através fato de não se poder mandar mais fluxo que a capacidade residual do arco ou então, não poder mandar mais fluxo que o vértice possui em excesso.

Esta ação pode ter dois resultados: o arco é saturado $(r(v,v') = 0)$, ou não $(r(v,v') > 0)$. Nestes dois casos, o fluxo que passa pelo arco (v,v'), aumenta.

Pode ainda acontecer de um vértice ainda ter excesso de fluxo, mas não existir nenhum arco (v, v') em direção ao destino com $r(v, v') > 0$. É usado então, o arco *(v, v')* que tem resíduo positivo. Na verdade $(v, v') \notin A'$, então $c(v, v') = 0$. Assim,

$$r(v, v') = c(v, v') - f(v, v') = f(v', v).$$

Isto resulta no decréscimo de fluxo no arco (v', v). Na verdade, a direção do arco é de v' para v, mas o arco (v, v') é considerado a fim de que, se for necessário, retorne fluxo em direção à origem s.

Em relação à distância de s a t, é importante que o fluxo vá pelo menor caminho, para ter um resultado mais eficiente. É definido a distância d, onde

1. $d(s) = n$
2. $d(t) = 0$
3. $\forall v \in J - \{s, t\}, \quad d(v) = d(w) + 1, \forall (v, w) \in A'$ e $r(v, w) > 0$

É importante saber quando os vértices vão entrar em operação, mandando fluxo através de seus arcos ou atualizando suas distâncias d. Para esta finalidade, é definido o conceito de vértice ativo. Um vértice $v \in J - \{s, t\}$ é ativo quando $e(v) > 0$ e $d(v) < n$.

O algoritmo de fluxo máximo tem os seguintes valores iniciais. Para cada arco que deixa o nó origem s, o pré-fluxo f é igual à capacidade máxima do arco, pois todo fluxo da origem deve sair inicialmente. Para os arcos restantes, o fluxo f é zero.

A distância inicial usada é

- $d(s) = n$
- $d(v) = 0, \quad \forall v \in J - \{s\}$

Para cada vértice, o excesso inicial é definido como sendo

- $e(v) = f(s,v), \quad \forall v \in J - \{s\}$
- $e(v) = 0$

O algoritmo aplica as operações básicas *push(v,w)* e *relabel(v)*, enquanto houver vértices ativos. Quando não houver mais vértices ativos, o algoritmo termina e o valor do fluxo máximo é o excesso no destino $e(t)$.

A operação básica *push (v,w)*, manda fluxo de v para w, saturando ou não o arco residual (v,w), quando a distância de v é $d(v) = d(w)+1$. A operação básica *relabel (v)* , atualiza a distância de v para o maior valor válido que pode ser assumido, que nada mais é do que, o mínimo das distâncias de seus vizinhos, incrementado de 1.

Push (v,w)
α = mínimo entre $e(v)$ e $r(v,w)$
o envio de α de v para w atualiza os valores:
$f(v,w) := f(v,w) + \alpha$
$f(w,v) := f(w,v) - \alpha$
$e(v) := e(v) - \alpha$
$e(w) := e(w) + \alpha$
atualização de $r(v,w)$ e $r(w,v)$

Relabel (v)
$\forall w \in J$, $r(v,w) > 0$ e $d(v) \leq d(w)$, faça:
$d(v)$ = mínimo $\{d(w) + 1/(v,w) \in A'\}$

Programa Principal
Enquanto $\exists v \in J - \{s,t\}$ é ativo, faça:
Se $\exists w \in J/(v,w) \in A'$ e $r(v,w) > 0$ e $d(v) = d(w) + 1$, então:
faça a operaçao básica push (v,w);
senão
faça a operação relabel (v);

Figura A.1: Algoritmo de Fluxo Máximo

Appendix B

Occam e Transputer

B.1 Introdução

A necessidade de melhorar a velocidade de computação provocou o surgimento da computação paralela. Já não é mais suficiente a rapidez que o *hardware* oferece, mesmo com o desenvolvimento de máquinas com um só processador cada vez mais eficientes.

Foram as linguagens paralelas, que introduziram o conceito de concorrência. Elas são muito necessárias devido ao fato de certas aplicações, ao serem desenvolvidas seqüencialmente, se tornarem menos eficientes, ao contrário de quando estão paralelizadas.

Surge então, a linguagem paralela *Occam*, que está fortemente associada com as placas *transputers*, que são componentes *VLSI* programáveis. *Occam* foi inspirada na linguagem *CSP*, estudada e especificada durante muitos anos por *Hoare*, desde 1978 até 1985. Por sua vez, os *transputers* foram desenvolvidos para que *Occam* fosse aproveitado na sua essência, sendo o resultado deste conjunto, um sistema concorrente simples e poderoso.

É observado que muitas aplicações podem ser decompostas em processos a serem executados concorrentemente. Essas aplicações, implementadas em *Occam* e particularmente sobre os *transputers*, resultam em uma maior eficiência e economia.

B.2 Occam e Transputer

Occam é uma linguagem que pode trabalhar sobre um *transputer* ou sobre uma rede de *transputers*. A rede é formada através de conecções dos canais de comunicação existentes em cada *transputer*.

Fica transparente para o programador, se ele está trabalhando sobre um único *transputer* ou sobre uma rede, devido às estruturas que *Occam* oferece. Além do mais, *Occam* pode ser considerada uma linguagem de baixo nível em relação ao *transputer*, pois muitas de suas funções são implementadas trabalhando diretamente sobre o *hardware*. Em consequência, quando possível, uma boa escolha é trabalhar com esta linguagem, para melhor eficiência da aplicação.

B.3 Transputer

A tecnologia *VLSI* é bem mais barata. Por esta razão, o *transputer* foi projetado como sendo um componente *VLSI*, onde a comunicação no mesmo circuito é bem menos custosa do que a comunicação em circuitos diferentes. O *transputer* possui a memória local ao circuito do processador.

Quando vários processos estão sendo executados em um só *transputer*, eles irão compartilhar o tempo do processador. É um microcódigo que controla esta tarefa, deixando o processador livre deste encargo.

Um canal entre dois processos que executam em um mesmo *transputer*, é implementado como sendo uma palavra de memória. Já no caso de processos sendo executados em *transputers* distintos, o canal é uma ligação ponto-a-ponto.

B.4 Sincronização e Comunicação

A comunicação entre processos em *Occam*, é feita através de canais de comunicação. Os canais em *Occam* são unidirecionais e só podem ser usados pelos processos origem e destino do canal.

Seja c, um canal. Para mandar o conteúdo de uma variável *var* através do canal c, é dado o **comando de saída** com a seguinte sintaxe:

c ! var

Para *var* receber um valor através do canal c, ou seja, para representar um **comando de entrada**, é dado:

c ? var

É através da comunicação de dados que se faz a sincronização entre processos. Por exemplo, sejam p_1 e p_2 dois processos que executam em paralelo e se comunicam através do canal c. Se em um dado instante p_1 precisa mandar um valor através de c para p_2, é dado o seguinte comando:

*c ! var*1

O processo p_1 só continuará sua execução depois que o processo p_2 receber este valor, ou seja, p_2 deve conter o comando:

*c ? var*2

É necessário que para cada saída exista uma entrada correspondente e vice-versa, senão como no exemplo anterior, p_1 poderia esperar para sempre, se p_2 nunca estiver pronto para receber a mensagem. Neste caso, o sistema entraria em *deadlock*.

A sincronização entre processos impede com que dados sejam perdidos na comunicação . Esta, por sua vez, é sem armazenamento, ou seja, não guarda as mensagens que vão chegando, fazendo com que não haja necessidade de nenhuma memória associada ao canal.

B.5 Definição de processo em Occam

Um sistema em *Occam* pode ser definido como um conjunto de processos concorrentes que se comunicam entre canais. Um processo pode consistir de um simples comando, ou ainda, possuir outros processos internos.

B.5.1 Processos Primitivos

Processos em *Occam* são construídos a partir dos processo primitivos. *SKIP* é um processo de execução imediata e sem efeito. *STOP* é um processo sem efeito, mas que nunca termina, impedindo que o processo que o contém prossiga. A atribuição é um processo que tem a seguinte forma:

variável := expressão

Os processos de entrada e de saída também são processos primitivos.

B.5.2 Construções

Occam possui construções para fazer melhor uso dos processos primitivos.

Para que os processos sejam executados sequencialmente, é usado a construção *SEQ*, como no exemplo:

SEQ
  P_1
  P_2

A construção *WHILE* executa um processo enquanto a expressão lógica for verdadeira.

WHILE $T_0 \geq T_final$
  SEQ
    P_1
    P_2

A construção *IF* tem a seguinte forma:

IF
  $cond_1$
    P_1

...

$cond_n$

  P_n

TRUE

  SKIP

onde $cond_1$, ..., $cond_n$ são expressões lógicas testadas e a primeira verdadeira tem o processo correspondente executado. É necessário que pelo menos uma condição seja verdadeira, caso contrário o processo é suspenso, se comportando como o processo *STOP*. Para assegurar sua corretude, é usada a condição *TRUE*.

A construção *PAR* especifica quais os processos serão executados em paralelo. A construção termina, quando todos os processos que a compõem terminam suas execuções. No exemplo:

SEQ

  P_1

  PAR

    P_2

    P_3

  P_4

os processos P_2 e P_3 executam em paralelo.

Finalmente, a construção *ALT* executa o processo correspondente ao comando guardado que esteja pronto.

ALT

  $guard_1$

    P_1

  ...

  $guard_n$

    P_n

A forma mais simples de um comando guardado é o processo de entrada. Se outro processo manda algum objeto de modo que a entrada possa ser feita, então o comando guardado está pronto. Se não houver nenhum comando guardado pronto, *ALT* fica suspenso até que algum fique pronto. Por outro lado, se houver mais de um comando pronto, a escolha de qual será atendido é arbitrária.

B.6 Mapeamento da Rede

Os grafos $G = (V, A)$ utilizados como entrada para o sistema distribuído, tiveram seus nós (processos) divididos em uma rede de *transputers* do tipo T800, projetada para trabalhar em um IBM PC XT/AT ou compatível. O mapeamento da rede usada é mostrado na Figura B.1.

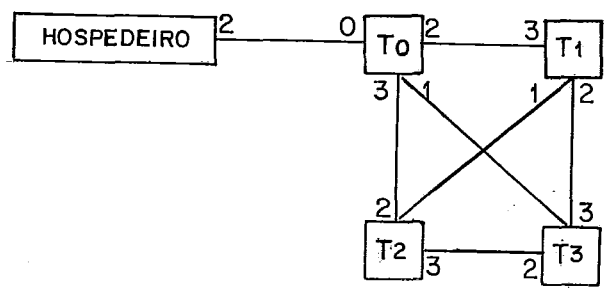


Figura B.1: Rede de Quatro *Transputers*

T0, T1, T2 e T3 identificam cada um dos *transputers* que compõem a rede. O sistema TDS (*Transputer Development System*)[10], é executado no *transputer* T0, o chamado *raíz*. O TDS se comunica com a máquina hospedeira através de uma ligação física (Figura B.1), por onde se faz as comunicações com a tela, teclado, disco e outras operações de entrada e saída.

Para executar vários processos distribuídos na rede, são necessários dois tipos de blocos de código, os chamados *folds* [10]:

EXE - neste *fold*, deve estar todo o código que será executado no *transputer* T0, pois é o local onde o TDS é executado. Também deve ser incluído o mapeamento dos canais lógicos sobre as ligações físicas com os outros *transputers* da rede.

PROGRAM - é o *fold* que contém as diretivas necessárias para o mapeamento nos outros *transputers* da rede e também a descrição das interconecções entre os *transputers*. Este *fold* consta da especificação da configuração da rede.

Para carregar um determinado código a um processador, os comandos usados são:

PLACED PAR (B.1)

PROCESSOR *número tipo-do-transputer* (B.2)

Com essas contruções e o mapeamento dos canais lógicos sobre as ligações físicas, o configurador do TDS [10] pode identificar o destino de cada código e verificar se a rede mapeada pode ser carregada.

O código a ser executado em cada processador deve constar de um único procedimento especificado em um outro tipo de *fold*, o chamado SC (*Separate Compilation*) [10]. Este procedimento é que vai ser executado em paralelo com os outros procedimento dos outros *transputers*.

Na diretiva B.2, *número* é o identificador lógico do processador (no caso os valores podem ser 1, 2 ou 3). Já *tipo-do-transputer*, pode ser T2, T4 ou T8, de acordo com o *transputer* utilizado.

O mapeamento dos canais lógicos sobre as ligações físicas, é feito através do comando PLACE. Um canal que é mapeado sobre uma ligação duas vezes, deve ser colocado no endereço físico de entrada e de saída.

A seguir, é mostrado como se pode fazer um programa que seja executado na rede:

{{{ EXE programa (no T0)

```
... declarações de variáveis e constantes
... declarações de protocolos

– declarações dos endereços lógicos das ligações físicas :
VAL link0out IS 0:
VAL link1out IS 1:
VAL link2out IS 2:
VAL link3out IS 3:
VAL link0in IS 4:
VAL link1in IS 5:
VAL link2in IS 6:
VAL link3in IS 7:

– declaração dos canais lógicos:
CHAN OF Protocolo in1:
CHAN OF Protocolo in2:
CHAN OF Protocolo in3:
CHAN OF Protocolo out1:
CHAN OF Protocolo out2:
CHAN OF Protocolo out3:

– Mapeamento:
PLACE in1 AT link1in:    - - de T1 para T0
PLACE in2 AT link2in:    - - de T2 para T0
PLACE in3 AT link3in:    - - de T3 para T0
PLACE out1 AT link1out:    - - de T0 para T1
PLACE out2 AT link2out:    - - de T0 para T2
PLACE out3 AT link3out:    - - de T0 para T3

... código executado no T0

}}}
```

Não se pode utilizar um vetor ou matriz de canais lógicos. É necessário se declarar elemento a elemento para que o mapeamento sobre os canais físicos possa ser feito.

O esboço do *fold* PROGRAM, conteria as seguintes diretivas:

... declarações de Protocolos
... declarações dos endereços das ligações físicas(como feitas no *fold* EXE)
... declarações dos canais lógicos
... SC procedimento (os procedimentos executados em cada processador)

```
PLACED PAR
  PROCESSOR 1 T8
    PLACE canal[0][1] AT link3in:
    PLACE canal[2][1] AT link2in:
    PLACE canal[3][1] AT link1in:
    PLACE canal[1][0] AT link3out:
    PLACE canal[1][2] AT link2out:
    PLACE canal[1][3] AT link1out:
    procedimento ( canal[0][1], canal[2][1], canal[3][1],
                   canal[1][0], canal[1][2], canal[1][3] )

  PROCESSOR 2 T8
    PLACE canal[1][2] AT link3in:
    PLACE canal[0][2] AT link1in:
    PLACE canal[3][2] AT link2in:
    PLACE canal[2][1] AT link3out:
    PLACE canal[2][0] AT link1out:
    PLACE canal[2][3] AT link2out:
    procedimento ( canal[1][2], canal[0][2], canal[3][2],
                   canal[2][1], canal[2][0], canal[2][3] )

  PROCESSOR 3 T8
```

```
PLACE canal[0][3] AT link2in:
PLACE canal[1][3] AT link1in:
PLACE canal[2][3] AT link3in:
PLACE canal[3][0] AT link2out:
PLACE canal[3][1] AT link1out:
PLACE canal[3][2] AT link3out:
procedimento ( canal[0][3], canal[1][3], canal[2][3],
                    canal[3][0], canal[3][1], canal[3][2] )

}}}
```

Os canais lógicos, devem ser passados elemento a elemento, como parâmetro. Só é necessário se declarar e mapear os canais que serão utilizados.

B.7 Multiplexadores e Demultiplexadores

Vários processos podem ser executados em cada processador. Alguns destes processos podem se comunicar com outros processos em outros processadores. No entanto, como ele são divididos na rede e entre os processadores só existe uma ligação física,e ainda, não é possível mapear vários canais lógicos sobre uma ligação física, surge a necessidade de se especificar um multiplexador e um demultiplexador.

No caso da aplicação desenvolvida, em cada procedimento executado em cada processador, um multiplexador e demultiplexador foi implementado para manipular cada ligação física. Com isto se aproveitou o máximo de paralelismo, evitando a ocorrência de *deadlock*. Caso fosse declarado um só multiplexador que manipulasse todas as ligações físicas de um processador, acarretaria a existência de um "gargalo" nas comunicações, eliminando grande parte do paralelismo. E como o paralelismo é vital no processamento, sua falta provocaria *deadlock*.

Os processos (nós) executados em um processador, se comunicam através de canais, com o multiplexador. Este, por sua vez, ao receber uma mensagem de algum processo, envia para o processador destino via o canal lógico que foi

mapeado sobre a ligação física que ele manipula.

Por outro lado, o demultiplexador espera por uma mensagem vinda do canal lógico, que também foi mapeado sobre uma ligação fisica. Recebendo, ele envia a mensagem para o processo destino, residente naquele processador.

B.8 Gerenciamento de processos Seqüenciais e Paralelos

Um processo é definido como um conjunto de instruções. Em um processador *transputer* vários processos podem ser executados concorrentemente. O processador contém um escalonador microprogramado que torna viável a execução de vários processos concorrentes, que compartilham o tempo do processador.

Pode-se definir dois estados de um processo:

1. *Ativo*: o processo está sendo executado ou está em uma fila esperando para ser executado.

2. *Inativo*: o processo está pronto para realizar entrada ou saída, ou está esperando chegar uma determinada hora.

O escalonador gerencia a utilização do processador de tal maneira que, processos inativos não gastam tempo de processador. Para processos ativos, o escalonador aloca uma fatia de tempo do processador. Os processos ativos podem então, ficar esperando para serem executados.

Um processo permanece em execução até completar uma ação. Ele perde o processador quando executa instruções de comunicação ou entra em espera por um determinado tempo (*delay time*).

Quando um processo é retirado de processamento, seu ponteiro para a próxima instrução é guardado em um espaço de trabalho,que cada processo tem associado, e outro processo da fila de prontos ganha o processador. É definido assim, a troca de contexto.

No caso de processos seqüenciais, especificados pela construção SEQ, não se gasta tempo em certos tipos de tarefas , como a troca de contexto. Por outro lado, os processos paralelos, especificados pela construção PAR, perdem tempo com esse tipo de tarefa.