UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

DEPARTAMENTO DE ENGENHARIA DE SISTEMAS

MINIMIZAÇÃO DE PERDAS EM REDES DE DISTRIBUIÇÃO DE ENERGIA ELÉTRICA ATRAVÉS DE MÉTODOS DE BUSCA INTELIGENTES COM PROCESSAMENTO PARALELO

WELFANE KEMIL TÃO

Orientador
Prof. Dr. Christiano Lyra Filho

Tese apresentada à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas – UNICAMP, como parte dos requisitos exigidos para a obtenção do título de Mestre em Engenharia Elétrica.

– Março 99 –

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

T159m

Tão, Welfane Kemil
Minimização de perdas em redes de distribuição de energia elétrica através de métodos de busca inteligentes com processamento paralelo / Welfane Kemil Tão.-- Campinas, SP: [s.n.], 1997.

Orientador: Christiano Lyra Filho.
Dissertação (mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Processamento paralelo (Computadores). 2. Sistemas de energia elétrica 3. Energia elétrica - Distribuição. 4. Otimização combinatória. 5. Otimização matemática. 6. Inteligência artificial. I. Lyra Filho, Christiano. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Aos meus pais,
Feres e Ana

"As soluções, eu já as possuo há muito tempo. Mas ainda não sei como cheguei a elas."

Carl Friedrich Gauss

Este trabalho teve o apoio da Fundação Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) e do Fundo de Apoio ao Ensino e a Pesquisa da UNICAMP (FAEP).

AGRADECIMENTOS

- Ao Prof. Christiano, pelos ensinamentos e pela grande amizade;
- A Celso Cavellucci, pela amizade e pela sua essencial contribuição para a realização deste trabalho;
- À equipe do Centro Nacional de Processamento de Alto Desempenho (CENAPAD) de São Paulo (Edmundo, Ana Drummond, Ricardo Küsel, Ana Seixas, Fernando Whitaker, Fábio, Érico e Léo), pelo suporte dado à implementação do processamento paralelo e pela paciência;
- Ao CENAPAD do Nordeste pela disponibilização dos recursos computacionais e pelo suporte;
- Ao Prof. Eleri Cardoso pela disponibilização dos recursos computacionais do laboratório do Departamento de Computação e Automação (DCA).
- Aos meus pais, pelo amor, pelo apoio, pela compreensão e pela vida;
- A Sandra Regina Mane, pelo apoio e pelos bons momentos que vivemos juntos;
- Ao Walcir e à Luciana, pelo apoio computacional e pela paciência;
- Aos professores e funcionários do DENSIS, pela amizade;
- Aos amigos do DENSIS (Vitória, Débora, Renata, Fran, Cintia, Regina, Jeanne, Leonardo, Hamilton, Marcos Trevisan, Marcão, Mauro, Gelson, Cássio, Makoto e todos os demais), que sempre fizeram do laboratório um ambiente de trabalho muito agradável.
- À CAPES e à FAEP pelo suporte financeiro;

RESUMO

Esse trabalho trata o problema da minimização das perdas em sistemas de distribuição de energia elétrica. Considerando a restrição da operação radial da rede de distribuição, o problema pode ser formulado como uma generalização da árvore recobridora de custo mínimo. A solução de mínimas perdas é obtida em duas etapas. A restrição de radialidade é relaxada na primeira etapa, obtendo-se uma solução otimista para o problema. Na segunda etapa, utiliza-se uma estratégia de busca para encontrar a solução ótima global factível do problema, guiada pelas informações da solução otimista. A solução otimista é obtida através de técnicas de otimização de fluxos não lineares em redes. A estratégia de busca usa procedimentos da área de inteligência artificial. Técnicas de processamento paralelo auxiliam a obtenção mais rápida da solução ótima.

ABSTRACT

This thesis addresses the problem of loss minimization for electric energy distribution system. As distribution networks operates radially, the problem can be formulated as a generalization of minimum spanning tree problem. The minimum loss solution is obtained in two steps. The constraint of radial operation is relaxes in the first step, leading to an optimistic solution. Information from optimistic solution is used to guide search strategies for obtain the optimal feasible solution. Non-linear network flow methods are adopted to find the optimistic solution. The search strategies is based on concepts from the field of artificial intelligence. Parallel processing speeds the search of optimal solution.

ÍNDICE

CAPÍTULO 1

INTRODUÇÃO

1.1 Apresentação do Problema

Em toda a extensão de um sistema de energia elétrica, desde a geração até os pontos de demanda, uma parcela da energia elétrica é dissipada em forma de energia calorífera devido a um fenômeno conhecido como Efeito Joule. Como qualquer outra forma de dissipação, esta também não é desejável, e portanto existe interesse em se estudar meios de minimizá-la.

Este trabalho propõe uma nova metodologia de resolução para o problema de reconfiguração de redes primárias de distribuição de energia elétrica para minimização de perdas por Efeito Joule. Esta metodologia se baseia nos métodos de busca advindos da inteligência artificial e em processamento paralelo, visando redução no tempo computacional necessário para obtenção da solução ótima global.

A obtenção da solução ótima global tem sido considerada inviável para redes de tamanho real devido ao enorme tempo computacional requerido (Shirmohammadi e Hong, 1989; Wagner, Chikhani e Hackam,1991). E mesmo que fosse viável, não teria muita utilidade se o tempo requerido para sua obtenção fosse muito alto, pois para que a reconfiguração da rede possa funcionar como uma ferramenta de controle em tempo real, a obtenção da solução precisa ser a mais rápida possível, de preferência inferior a 10 minutos (Sárfi, Salama e Chikhani, 1996). Em função disto, os métodos de solução atualmente existentes para o problema são métodos heurísticos que demandam menor esforço computacional e podem fornecer boas soluções, embora sem garantir a otimalidade das mesmas.

Considerando a constante evolução dos algoritmos e da capacidade de processamento dos computadores, fica a seguinte pergunta: Será que a obtenção da solução ótima global para problemas de tamanho real já se tornou viável? E em caso afirmativo, a que distância estaremos de obtê-la em tempos considerados aceitáveis para o controle em tempo real?

Para uma melhor compreensão de como este problema se enquadra no contexto do sistema de energia elétrica, é feito a seguir uma pequena explanação sobre este.

1.2 O Sistema de Energia Elétrica

O percurso feito pela energia elétrica entre a geração e os consumidores pode ser dividido basicamente em duas etapas: transmissão e distribuição.

A transmissão é o trecho do sistema elétrico entre as usinas geradoras e as subestações dos centros urbanos e industriais. Opera normalmente com tensões muito elevadas, por ser este o principal recurso para atenuar as perdas por Efeito Joule no transporte de energia elétrica a longas distâncias.

A distribuição é o trecho do sistema elétrico entre as subestações urbanas e os pontos de demanda. O sistema de distribuição é dividido em duas partes: rede primária e rede secundária. Rede primária é o trecho entre as subestações e os transformadores de distribuição, responsáveis por abaixar a tensão para níveis adequados ao consumo residencial e comercial. Rede secundária é o trecho entre os transformadores de distribuição e os pontos de demanda.

A maior parte das perdas por Efeito Joule, ou perdas resistivas, ocorrem na transmissão e na rede primária de distribuição. Portanto estas fases merecem maior atenção no estudo de otimização de perdas resistivas. Para se ter noção da dimensão destes valores, cerca de 2% da energia gerada é dissipada na transmissão, e de 5% a 13% é dissipada na rede primária de distribuição (Bunch, Miller e Wheeler, 1982). Para ilustrarmos a importância do estudo do problema de minimização das perdas por Efeito Joule, basta lembrarmos que nos últimos dez anos, o governo federal estabeleceu o horário de verão para reduzir o consumo, e consequentemente a necessidade de geração, em apenas 1%.

1.3 O Problema de Minimização de Perdas

Os métodos de otimização consistem, em primeiro lugar, na busca de um modelo matemático para o problema, ou seja, na busca de um conjunto de equações que representem matematicamente o sistema real. Para deduzir estas equações, pode-se partir de dados experimentais e

processos estatísticos, ou de leis físicas conhecidas, como no caso deste trabalho. Em segundo lugar deve-se procurar um método capaz de encontrar soluções ótimas para o problema representado por este conjunto de equações.

Apesar dos modelos matemáticos que representam a transmissão e a rede primária de distribuição serem similares, no segundo existe uma restrição adicional decorrente da necessidade da rede primária de distribuição operar em configuração radial, ou seja, sem a existência de malhas. Isto é necessário devido a razões de segurança, como por exemplo, uma melhor coordenação do sistema de proteção e maior rapidez na localização de falhas no sistema, além de propiciar menor corrente de curto-circuito (Shirmohammadi e Hong, 1989; Taleski e Rajicic, 1997).

A característica de operação radial aumenta consideravelmente o grau de complexidade para resolução do problema de minimização de perdas, pois transforma-o num problema de otimização combinatorial.

1.4 A Rede Primária de Distribuição de Energia Elétrica

Como já foi dito, rede primária de distribuição é o trecho do sistema elétrico entre as subestações urbanas e os transformadores de distribuição. Neste trecho, para que ocorra um bom funcionamento do sistema, deve-se minimizar as perdas de potência ativa e reativa e atender aos níveis estabelecidos para a tensão na rede.

Este trabalho parte da premissa que a rede de distribuição está balanceada, e portanto possui fator de potência aproximadamente constante. Desta forma, os fluxos de potência ativa e reativa são uniformemente proporcionais, e portanto, embora a reconfiguração seja feita em função apenas do fluxo de potência ativa, ela também será ótima para o fluxo de potência reativa.

A análise de queda de tensão, por sua vez, tem como ponto de partida uma configuração de rede que seja radial. E encontrar esta configuração radial é o objetivo aqui presente, e portanto análise de queda de tensão está fora do escopo deste trabalho.

A Fig. 1.1 ilustra uma rede primária de distribuição através de um diagrama unifilar. Nela podemos visualizar todas as conexões existentes entre as subestações (SS1 e SS2) e os blocos de carga (L1,L2,..., L10), e entre os próprios blocos de carga.

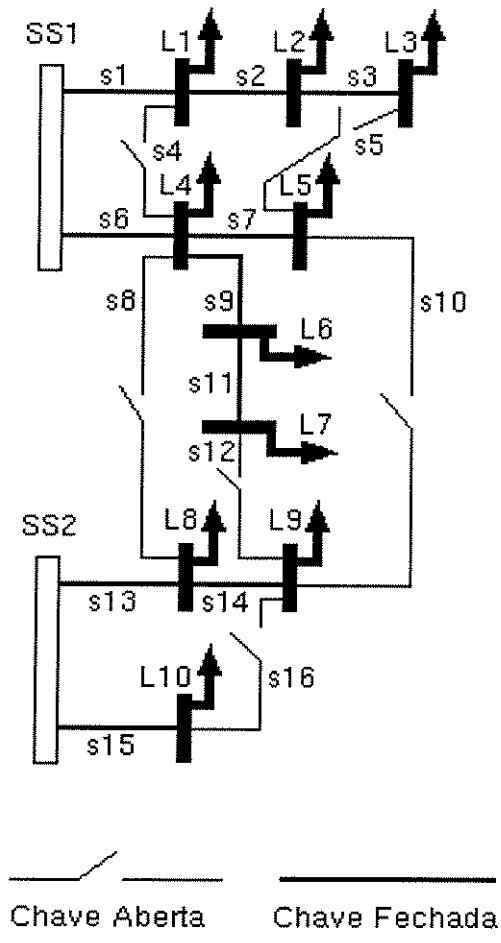


Fig. 1.2 Diagrama unifilar de uma rede de distribuição de energia elétrica

Em cada uma destas conexões existe uma chave seccionadora (s1,s2,...,s16), sendo que algumas operam normalmente abertas (NA) e outras operam normalmente fechadas (NF). A existência destas chaves permite que existam inúmeros possíveis caminhos de alimentação de energia aos consumidores. No caso de alguma anormalidade, algumas chaves NF são abertas com a finalidade de isolar a área onde ocorreu a anormalidade, a algumas chaves NA são fechadas com a finalidade de propiciar energia elétrica à maior parte possível da área atingida.

Cada bloco de carga representa um conjunto de transformadores de distribuição que não possuem chaves seccionadoras entre si. Portanto, cada bloco de carga está completamente energizado ou completamente desenergizado.

1.5 Representação por Grafos para Redes de Distribuição

Neste trabalho a rede de distribuição é representada por um grafo orientado (Ahuja, Magnanti e Orlin, 1993). Quando um grafo $G = [N,A]$ representa uma rede de distribuição, os *nós* do conjunto N correspondem às subestações ou aos blocos de carga. Um *nó raiz* é incluido para contornar dificuldades desnecessárias, decorrentes do requisito de conectividade da rede. Os *arcos* do conjunto A representam as conexões onde existe chave seccionadora, com exceção aos arcos que conectam as subestações ao *nó raiz*, denominados *arcos artificiais*. Estes arcos possuem perdas nulas e capacidade de fluxo equivalente à capacidade das subestações. O *nó raiz* e os *arcos artificiais* podem ser associados ao sistema de transmissão, ou seja, à fonte de energia de todo o sistema de distribuição. O grafo correspondente à Fig. 1.1 é apresentado na Fig. 1.2.

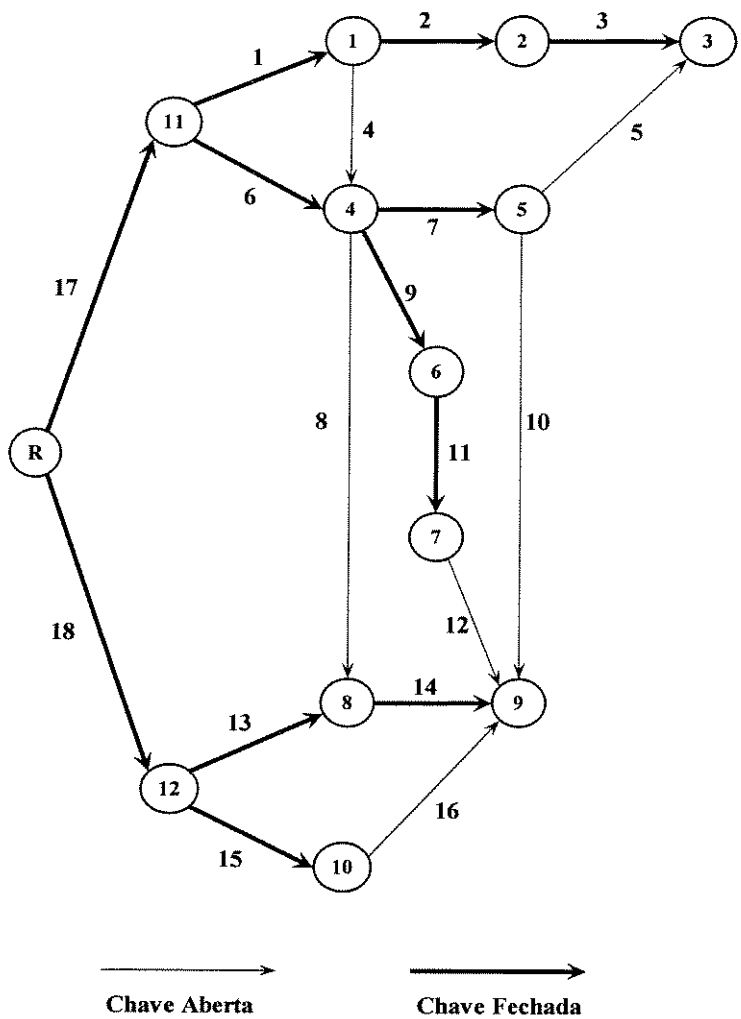


Fig. 1.2 Grafo correspondente ao diagrama da Fig. 1.1

Na Fig. 1.2 o nó R representa o *nó raiz*, enquanto os nós 11 e 12 representam as subestações.

Pode-se mostrar facilmente que o número de chaves que precisam ficar abertas em um configuração radial é dado por c_a,

$$c_a = a - n + 1 \tag{1.1}$$

Onde:

a é o número de arcos no conjunto A;

n é o número de nós no conjunto N.

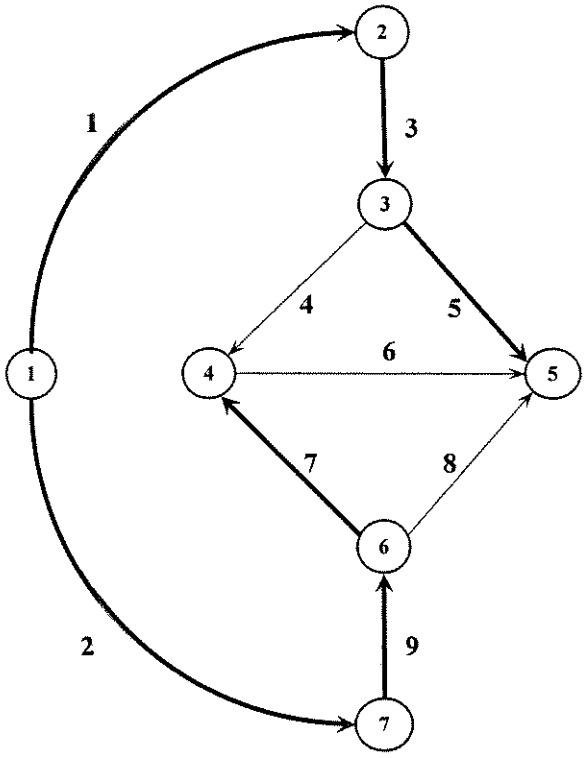


Fig. 1.3a Configuração Radial I

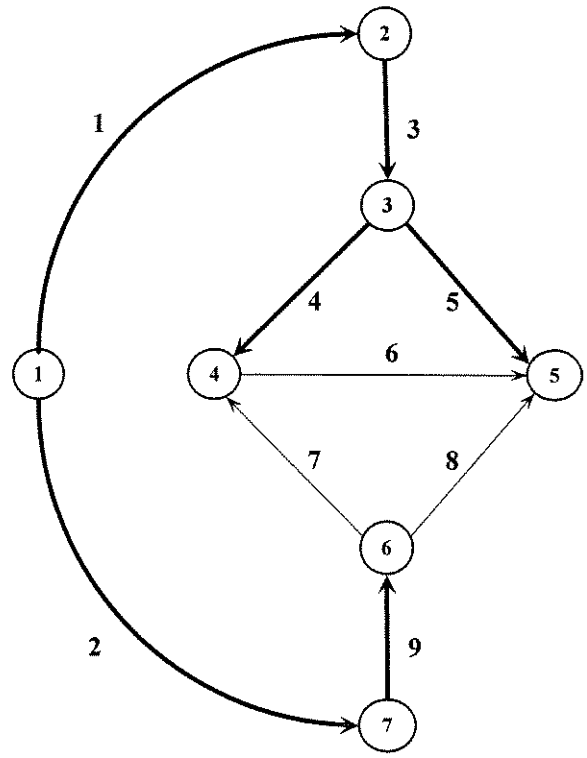


Fig. 1.3b Configuração Radial II

Fig. 1.3 Duas possíveis configurações radiais para um mesmo sistema

1.6 Aspecto Combinatório Decorrente da Restrição de Radialidade

A Fig. 1.3 apresenta duas possíveis configurações radiais para um mesmo sistema, onde o nó 1 representa o *nó raiz*, os nós 2 e 7 representam subestações, e os demais nós representam blocos de carga. Na Fig. 1.3a, a configuração radial é obtida com as chaves 4, 6 e 8 no estado aberto. Outra possível configuração radial pode ser obtida com as chaves 6, 7 e 8 no estado aberto, conforme ilustrado na Fig. 1.3b. No total, a rede representada nesta figura, com apenas duas subestações, quatro blocos de carga e nove arcos fornece vinte e quatro possíveis configurações radiais. E esse número aumenta exponencialmente com c_a (1.1), ou seja, com o número de chaves que necessitam ficar abertas para obtermos configurações radiais.

Para ilustrar a dificuldade de resolver o problema de minimização de perdas resistivas, podemos tomar como exemplo a rede de distribuição da cidade de Bauru, utilizada em estudos de caso neste trabalho. Bauru é uma cidade média, com cerca de 400.000 habitantes, e sua rede de distribuição de energia elétrica possuia em 1990 quatro subestações, aproximadamente 200 blocos de carga e 300 chaves (Carneiro da Silva, 1990). Uma rede deste tamanho pode fornecer milhões de possíveis configurações radiais, e em cada uma das configurações, as perdas por Efeito Joule possuem valores distintos, pois os fatores que influenciam nas perdas, como a corrente que flui através de cada arco, dependem da configuração.

Portanto, a tarefa de minimizar perdas resistivas é equivalente a encontrar qual é a configuração radial ótima, ou seja, aquela que é capaz de atender a todos os consumidores com as menores perdas possíveis por Efeito Joule; isto equivale a determinarmos qual deve ser o estado (aberta/fechada) de cada chave seccionadora existente na rede.

O fato de existir um número muito alto de configurações radiais (exponencialmente crescente com a dimensão da rede) e a necessidade de se calcular as perdas em um grande número destas configurações (posteriormente será demonstrado que o cálculo não precisa ser feito para todas elas), explica porque o problema requer um grande esforço computacional para ser solucionado.

Além disso, uma rede primária de distribuição possui cargas residenciais, comerciais e industriais, que variam de intensidade ao longo de cada dia, o que faz com que a configuração ideal para um determinado momento possa não ser a ideal para algumas horas mais tarde. Logo, o problema de minimização de perdas precisa ser resolvido várias vezes por dia, o que enfatiza a necessidade de se procurar reduzir o tempo necessário à obtenção de soluções ótimas.

1.7 Revisão Bibliográfica

Existem quatro principais estratégias de abordagem para o problema de reconfiguração de redes primárias de distribuição de energia elétrica para minimização de perdas resistivas.

A primeira consiste em partir de uma configuração radial (solução factível), e através de métodos heurísticos tentar detectar como se pode alterar esta configuração inicial sem perder a factibilidade da solução obtendo uma redução no valor das perdas. Alguns métodos propõem uma forma de estimar a variação das perdas quando um grupo de cargas é transferido de um alimentador para outro (Civanlar et al., 1988; Baran e Wu, 1989; Goswami e Basu, 1992; Taleski e Rajicic, 1997). Outros métodos buscam a melhor alteração da configuração através de métodos de busca heurística (Taylor e Lubkeman, 1990), algoritmos genéticos (Nara et al., 1992) ou de redes neurais (Kim, Ko e Jung, 1993).

A segunda estratégia é dividida em duas etapas. Na primeira determina-se qual seria a configuração se não houvesse a restrição de radialidade (solução otimista). Muito provavelmente isto corresponde à configuração de rede que possui todas as chaves no estado fechado. A segunda etapa consiste na elaboração de mecanismos para detectar quais as chaves que devem ser abertas para a eliminação das malhas (Shirmohammadi e Hong, 1989; Wagner, Chikhani e Hackam, 1991; Borozan, Rajicic e Ackovski, 1995; Cavellucci e Lyra, 1997).

A terceira estratégia consiste em construir o grafo a partir dos nós correspondentes às subestações e detectar qual dentre os inúmeros caminhos possíveis para conectar cada bloco de carga ao grafo deve ser utilizado (Glamocanim, 1990).

A quarta estratégia é particionar o sistema em vários sub-sistemas e aplicar uma das três estratégias anteriores para resolver cada sub-sistema separadamente (Sárfi, Salama e Chikhani, 1996).

1.8 Estratégia de Abordagem Adotada

A estratégia de abordagem adotada neste trabalho corresponde a segunda estratégia apresentada no item anterior. Porém, os trabalhos acima citados implementam a segunda etapa através de métodos heurísticos, enquanto neste trabalho isto é feito através de um método formal de busca baseado em conceitos da área de inteligência artificial. Esta técnica permite a obtenção da solução ótima global. Também é implementada uma versão do método que incorpora informação heurística, porém como a solução ótima global é determinada pelo método exato, pode-se variar os parâmetros

heurísticos de forma a não perder a condição de otimalidade. Além disso, a implementação deste trabalho utiliza processamento paralelo, visando uma exploração mais rápida das configurações de rede possíveis.

O capítulo 2 contém o método de minimização de perdas utilizado para determinar qual a melhor distribuição de fluxo em cada configuração de rede.

O capítulo 3 explana os diversos métodos de busca existentes para a exploração de um espaço de estados, e justifica o método escolhido para este caso.

O capítulo 4 contém informações sobre processamento paralelo, que são importantes para o bom entendimento da implementação feita.

O capítulo 5 expõe como implementar o método de busca escolhido no capítulo 3 utilizando o processamento paralelo exposto no capítulo 4 e apresenta os resultados e conclusões.

CAPÍTULO 2

O MÉTODO DE MINIMIZAÇÃO DAS PERDAS

2.1 A Formulação Matemática

O primeiro passo para solucionarmos um problema de otimização é desenvolver um modelo matemático para o problema. Este modelo matemático consiste de uma função objetivo, que estabelece a nossa meta, ou seja, o que deve ser otimizado, e um conjunto de equações e condições que as variáveis da função objetivo devem respeitar, ou seja, as restrições da nossa otimização. Neste trabalho, o objetivo é minimizar as perdas por Efeito Joule nas linhas de uma rede primária de distribuição de energia elétrica; portanto, a função objetivo será a soma das perdas de cada um dos arcos do grafo que representa a rede. Supondo que o grafo possua n arcos, a função objetivo torna-se:

$$Min \sum_{k=1}^{n} r_k x_k^2 \tag{2.1}$$

Nesta equação, r_k representa a resistência ôhmica no arco k, e x_k o fluxo de corrente do mesmo arco k. Com a nossa função objetivo estabelecida, resta-nos estabelecermos as restrições. A primeira delas será a necessidade de atendermos aos consumidores. Para isto, devemos respeitar a Primeira Lei de Kirchhoff em cada nó do grafo; em outras palavras, a diferença entre o fluxo de corrente que entra e o fluxo de corrente que sai de cada nó equivale ao consumo ou geração de energia

do nó. No caso do fluxo de corrente que entra no nó ser maior que o fluxo de corrente que sai do nó, é porque este nó está consumindo energia (é um bloco de carga). Se o fluxo de corrente que entra no nó for menor que o fluxo de corrente que sai do nó, este nó está fornecendo energia, (é uma subestação). E no caso da diferença ser nula, é porque este nó não consome nem fornece energia (trata-se de um nó de passagem, ou transbordo).

Para exemplificarmos, tomemos o grafo da Fig. 1.3a, convencionando sinal positivo para os fluxos que saem de cada nó e sinal negativo para os fluxos que entram em cada nó. Temos:

$$\begin{aligned}
x_1 + x_2 &= b_1 \\
-x_1 + x_3 &= b_2 \\
-x_3 + x_4 + x_5 &= b_3 \\
-x_4 + x_6 - x_7 &= b_4 \\
-x_5 - x_6 - x_8 &= b_5 \\
+x_7 + x_8 - x_9 &= b_6 \\
-x_2 + x_9 &= b_7
\end{aligned} \tag{2.2}$$

No sistema de equações acima, as variáveis b_i representam o consumo ou geração de energia de cada nó. Nos nós correspondentes ao nó raiz e às subestações, teremos $b_i > 0$, nos nós correspondentes aos blocos de carga teremos $b_i < 0$, e nos nós de passagem teremos $b_i = 0$. Podemos observar que este sistema de equações pode ser representado matricialmente da seguinte forma:

$$\begin{bmatrix}
+1 & +1 & & & & & & & \\
-1 & & +1 & & & & & & \\
 & & -1 & +1 & +1 & & & & \\
 & & & -1 & & +1 & -1 & & \\
 & & & & -1 & -1 & & -1 & \\
 & & & & & & +1 & +1 & -1 \\
 & -1 & & & & & & & +1
\end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} \tag{2.3}$$

Escrevemos então a restrição de suprimento da demanda como $\boldsymbol{Ax} = \boldsymbol{b}$, onde $\boldsymbol{A}$ representa a **matriz de incidência** do grafo (Ahuja, Magnantie e Orlin, 1993), $\boldsymbol{x}$ representa o vetor coluna das variáveis x_k e $\boldsymbol{b}$ representa o vetor coluna das variáveis b_i. As colunas da matriz fornecem informações

sobre os arcos do grafo; o elemento "+1" aparece na linha correspondente ao nó origem do arco, e o elemento "–1" aparece na linha correspondente ao nó destino do arco. As linhas informam quais arcos incidem sobre o nó e em que sentido (entrando ou saindo); o elemento "+1" aparece nas colunas correspondentes aos arcos que entram no nó, enquanto o elemento "–1" aparece nas colunas correspondentes aos arcos que saem.

A próxima restrição diz respeito à capacidade física das linhas de distribuição de transmitirem fluxo de corrente. Se lembrarmos que fluxo de corrente significa taxa de transmissão de cargas elétricas, o número de cargas elétricas que pode fluir através de um cabo em um determinado período de tempo é, portanto, limitado pelo tamanho da secção transversal, ou bitola, deste cabo.

Como estamos trabalhando com arcos orientados, e portanto existe a possibilidade de existirem valores de fluxo de corrente negativos, esta restrição é escrita da seguinte forma

$$\underline{x} \leq x \leq \overline{x} \tag{2.4}$$

Finalmente, tem-se a restrição de operar a rede em configuração radial, aconselhável devido a razões de segurança citadas no item 1.3.

O modelo matemático acima descrito, é formulado a seguir:

$$P \begin{cases} \textit{Min} \sum_{k=1}^{n} r_k x_k^2 \\ \textbf{sa} \\ \quad Ax = b \\ \quad \underline{x} \leq x \leq \overline{x} \\ \quad \textit{Radialidade} \end{cases}$$

Estabelecido o modelo matemático do problema ***P***, o passo seguinte é encontrar um método para encontrar sua solução ótima. Como já foi mencionado no Capítulo 1, o método de resolução será dividido em duas fases. Na primeira, resolveremos o problema desconsiderando a restrição (4), referente à radialidade da rede em operação; na segunda fase, faremos uma busca no espaço de estados (Pearl, 1984), onde cada estado representa uma possível configuração da rede, até

determinarmos a configuração radial ótima. Portanto, em primeira instância relaxamos a condição de radialidade e passamos a ter o seguinte modelo matemático para o problema relaxado P_r:

$$P_r \begin{cases} Min \sum_{k=1}^{n} r_k x_k^2 \\ s.a. \\ \quad Ax = b \\ \quad \underline{x} \le x \le \overline{x} \end{cases}$$

O problema relaxado P_r possui função objetivo não-linear e restrições lineares; pode ser solucionado por qualquer técnica para programação não-linear com restrições lineares. Neste trabalho é adotado o método do gradiente reduzido especializado à resolução de problemas de fluxos em redes. Estes aspectos são discutidos nos próximos itens.

2.2 O Método do Gradiente Reduzido

O Gradiente Reduzido (Luemberger, 1984) é um dos métodos primais clássicos para resolução de problemas de otimização com função objetivo não linear e restrições lineares.

Considere o problema P_2:

$$P_2 \begin{cases} Min\ f(x) \\ s.a. \\ \quad Ax = b \\ \quad \underline{x} \le x \le \overline{x} \end{cases}$$

Onde:

$f(x)$: função contínua em C^2;

A: matriz de dimensão m x n;

x: vetor coluna de dimensão n;

b: vetor coluna de dimensão m;

Devemos ressaltar que as m linhas da matriz A devem ser linearmente independentes.

O método do gradiente reduzido separa o vetor x em dois grupos. Um deles será denominado conjunto das variáveis dependentes x_I, ou variáveis básicas, possuindo as seguintes características:

1) tem dimensão m;

2) as m respectivas colunas na matriz A são linearmente independentes;

3) as variáveis em x_I estão totalmente imersas no intervalo de factibilidade

$$\underline{x}_I \leq x_I \leq \overline{x}_I \tag{2.5}$$

O outro será denominado conjunto das variáveis independentes x_J, ou variáveis não básicas, com dimensão n-m. Quando não for possível obter esta repartição sem dificuldades, adota-se um procedimento semelhante a Fase 1 no método simplex (Bazaraa, 1990).

Portanto, podemos reescrever o problema P_2 da seguinte forma:

$$P_3 \begin{cases} \textit{Min } f(x_I, x_J) \\ \textit{s.a.} \\ \quad A^I x_I + A^J x_J = b \\ \quad \underline{x_I} \leq x_I \leq \overline{x_I} \\ \quad \underline{x_J} \leq x_J \leq \overline{x_J} \end{cases}$$

No problema P_3, A^I é a matriz formada pelas colunas de A correspondentes às variáveis dependentes, e possui a característica de ser não singular; A^J é a matriz formada pelas colunas de A correspondentes às variáveis independentes.

A partir da restrição (2) de P_3, e da não singularidade da matriz A^I, podemos concluir que:

$$x_I = (A^I)^{-1} [b - A^J x_J] \tag{2.6}$$

Logo, podemos obter todos os x_I através dos x_J, ou seja podemos reescrever o problema em termos apenas das variáveis x_J.

Para obtermos uma função objetivo em função de x_J, basta fazermos a seguinte

substituição:

$$F(x_J) = f(x_I, x_J) = f(h(x_J), x_J) \tag{2.7}$$

$$h(x_J) = (A^I)^{-1} b - (A^I)^{-1} A^J x_J \tag{2.8}$$

Logo, podemos reescrever o problema P_3 da seguinte forma:

$$P_4 \begin{cases} Min\ F(x_J) \\ s.a. \\ \quad \underline{x_I} < (A^I)^{-1} b - (A^I)^{-1} A^J x_J < \overline{x_I} \\ \quad \underline{x_J} \leq x_J \leq \overline{x_J} \end{cases}$$

O método denomina-se gradiente reduzido porque, ao invés de nos basearmos no gradiente da função f(x) para procedermos a otimização, nos basearemos no gradiente da função F(x_J). Portanto trabalharemos apenas com as variáveis independentes.

O próximo passo será determinar o gradiente reduzido, que será representado pelo vetor ρ. Considerando a definição de F(x_J), temos:

$$\rho = \nabla_J F(x_J) = \nabla_J f(h(x_J), x_J) = \nabla_J f(x) + \nabla_I f(x)\ (\delta h / \delta x_J) \tag{2.9}$$

$$\rho = \nabla_J F(x_J) = \nabla_J f(x) + \nabla_I f(x)\ [\ -(A^I)^{-1} A^J\ ] \tag{2.10}$$

O fator $\nabla_I f(x)\ [\ (A^I)^{-1}\ ]$ é definido como o vetor multiplicador associado à base I (também denominado de vetor de potenciais) no ponto x, representado por $\lambda(x)$. Portanto:

$$\lambda(x) = \nabla_I f(x)\ [\ (A^I)^{-1}\ ] \tag{2.11}$$

e

$$\rho = \nabla_J f(x) - \lambda(x) A^J \tag{2.12}$$

Quando pretendemos minimizar uma função utilizando um método primal, como é o caso do gradiente reduzido, devemos iniciar o processo em uma solução factível $\mathbf{x}^0$. Procura-se melhorar a solução numa direção obtida a partir do sentido negativo do gradiente da função em $\mathbf{x}^0$. O tamanho do passo será restringido pelos limites superior e inferior de cada variável. Se uma variável atinge um dos

limites, para manter a factibilidade do problema, devemos projetar o gradiente reduzido sobre a restrição de canalização alcançada. Para isso, anula-se as componentes do gradiente reduzido que nos levariam à região infactível. Quando todas as componentes forem anuladas, significa que o ótimo foi atingido (Luemberger, 1984). Caso contrário, determina-se o passo máximo do conjunto das variáveis dependentes, e do conjunto das variáveis independentes, para que seus respectivos limites inferiores e superiores não sejam violados. A seguir deve-se fazer uma busca unidimensional dentro da faixa delimitada pelo valor do passo máximo para determinar o passo t que nos fornecerá o valor mínimo da função $f(x - t\Delta x)$. Se o passo obtido coincidir com o passo máximo das variáveis dependentes, uma variável dependente atingiu um de seus limites. Neste caso, não está mais totalmente imersa no seu intervalo de factibilidade e, portanto, não pode mais pertencer ao conjunto das variáveis dependentes; e a repartição entre variáveis dependentes e independentes precisa ser refeita. Determina-se o novo gradiente da função e repete-se o processo.

O gradiente reduzido pode ser resumido na seqüência de passos a seguir:

1° Passo – Calcula-se o gradiente da função $f(x)$ no ponto x^0, $\nabla f(x^0)$;

2° Passo – Calcula-se o vetor de potenciais $\lambda(x) = \nabla_I f(x)\ [\ (A^I)^{-1}\ ]$;

3° Passo – Calcula-se o gradiente reduzido $\rho = \nabla_J f(x) - \lambda(x) A^J$;

4° Passo – Faz-se, $\forall j \in J$,

$$\Delta x_j = \rho_j, \text{ se } \underline{x}_j < x_j < \overline{x}_j$$

$$\text{ou } x_j = \underline{x}_j \text{ e } \rho_j < 0$$

$$\text{ou } x_j = \overline{x}_j \text{ e } \rho_j > 0$$

$$\Delta x_j = 0, \text{ se } x_j = \underline{x}_j \text{ e } \rho_j \geq 0$$

$$\text{ou } x_j = \overline{x}_j \text{ e } \rho_j \leq 0$$

5° Passo - Se $\Delta x_J = 0$, a solução é ótima;

6° Passo – Calcula-se a variação unitária das variáveis básicas,

$$\Delta x_I = - (A^I)^{-1} A^J \Delta x_J;$$

7° Passo – Calcula-se

$$t_1 = \max \left\{ t \; t.q. \; \underline{x}_I \le x_I^0 - t\Delta x_I \le \bar{x}_I \right\}$$

$$t_2 = \max \left\{ t \; t.q. \; \underline{x}_J \le x_J^0 - t\Delta x_J \le \bar{x}_J \right\}$$

$$t_L = \min \{t_1, t_2\} = \max \left\{ t \; t.q. \; \underline{x} \le x^0 - t\Delta x \le \bar{x} \right\}$$

$$t_3 = \arg \left\{ \underset{0 \le t \le t_L}{Min} \; f(x - t\Delta x) \right\}$$

8° Passo – Compara-se t_3 com t_1. Se $t_3 < t_1$, volta-se para o 1° passo (as variáveis básicas continuam estritamente no interior do intervalo de factibilidade). Caso contrário vai-se ao 9° passo;
9° Passo – Troca-se de base, ou seja, redefine-se a partição ***I,J***. Volta-se ao 1° passo.

O método acima descrito tem como finalidade a resolução de qualquer problema de otimização que pertença à família de problemas de funções não lineares e restrições lineares. No caso particular de otimização de fluxos em rede algumas características permitem um aprimoramento do método (Lyra Filho, 1984). Este é o objeto do próximo item.

2.3 Especialização do Método do Gradiente Reduzido para a Otimização de Funções não Lineares de Fluxos em Redes

Os problemas de otimização de fluxos em redes nos proporcionam facilidades para a utilização da seqüência de passos descrita no item anterior, principalmente no processamento dos passos 2 e 6, onde se torna desnecessário a obtenção da inversa da matriz A^I. Neste caso, deve-se lembrar, A é a matriz de incidência do grafo, que possui características muito particulares.

Vejamos então as facilidades incorporadas aos passos 2 e 6:

2.3.1 Cálculo do Vetor Multiplicador

Considere o vetor multiplicador $\lambda(x) = \nabla_I f(x) \; [\; (A^I)^{-1} \;] \equiv \lambda(x) \, A^I = \nabla_I f(x)$.

Em primeiro lugar devemos ressaltar que $\lambda(x)$ possui dimensão m, e cada um dos seus componentes está associado a um dos m nós do grafo.

Em segundo lugar devemos observar que: uma matriz de incidência de um grafo possui a

característica de conter somente dois elementos não nulos por coluna, sendo "+1" (na linha correspondente ao nó origem do arco) e "–1" (na linha correspondente ao nó destino do arco). Portanto, a soma das linhas desta matriz é nula, o que indica que não se trata de uma matriz linearmente independente. No entanto, a retirada de qualquer uma das linhas produz uma matriz com *m–1* linhas linearmente independentes (Ahuja, Magnanti e Orlin, 1993).

Como uma matriz de incidência possui *m-1* linhas linearmente independentes, o sistema $\lambda(x)\, A^I = \nabla_I f(x)$ possui um grau de liberdade, ou seja, torna-se necessário conhecer o valor de algum componente do vetor $\lambda(x)$ para podermos determinar os outros componentes de forma única.

Como A é uma matriz de incidência, se o arco correspondente a uma coluna k conecta um nó origem i a um nó destino j, tem-se:

$$\lambda_i(x) - \lambda_j(x) = \nabla_{ij} f(x), \forall\ \mathbf{k} \in \mathbf{I} \tag{2.13}$$

Fazendo analogia com um circuito elétrico, esta é a mesma situação de quando conhecemos as diferenças de potencial elétrico entre cada nó do circuito e desejamos determinar o potencial elétrico de cada nó. Para resolver este problema, é necessário conhecermos o potencial elétrico de pelo menos um nó do circuito, para determinarmos os potenciais elétricos dos demais nós de forma única. Este problema é resolvido aterrando-se um dos nós, ou seja, atribuindo-se potencial elétrico igual a zero para o referido nó.

De forma análoga, esta também é a solução para o nosso sistema, $\lambda(x)\ A^I = \nabla_I f(x)$. Devemos atribuir potencial nulo para um dos nós do grafo, para determinarmos os potenciais dos demais nós e obtermos o vetor $\lambda(x)$.

2.3.2 Cálculo das Variações Unitárias das Variáveis Básicas

Considere o vetor de variações unitárias das variáveis básicas, $\Delta x_I = -(A^I)^{-1} A^J \Delta x_J$.

Para uma melhor compreensão das vantagens obtidas quando trabalhamos com grafos, devemos chamar a atenção para algumas das suas particularidades.

A primeira delas está na fácil visualização da existência ou não de dependência linear, pois em grafos ela é apontada pela existência de ciclos. Lembrando que as colunas de uma matriz de incidência de um grafo representam arcos, caso existam dois caminhos entre dois pontos do grafo, significa que um deles pode ser expresso como uma combinação linear, dos arcos que constituem o

outro caminho. O grafo de uma matriz cujas colunas sejam linearmente independentes não possuirá arcos fechando ciclos.

Como as colunas da matriz A^I são independentes o grafo associado a esta matriz não possui ciclos, ou seja, existe apenas um único caminho entre quaisquer dois pontos do grafo. Um grafo que cobre todos os nós da rede, é conexo e não possui ciclos, é definido na teoria de grafos como uma árvore (Ahuja, Magnanti e Orlin, 1993). Como o grafo associado a matriz A^I satisfaz estas condições, podemos dizer que o conjunto das variáveis básicas sempre estará associado a uma árvore.

As variáveis não básicas (associadas às colunas da matriz A^J) correspondem aos arcos que não pertencem à árvore. A inclusão de qualquer um destes arcos à árvore gera um grafo com um único ciclo; esta é a característica que nos interessa para processarmos o 6° passo.

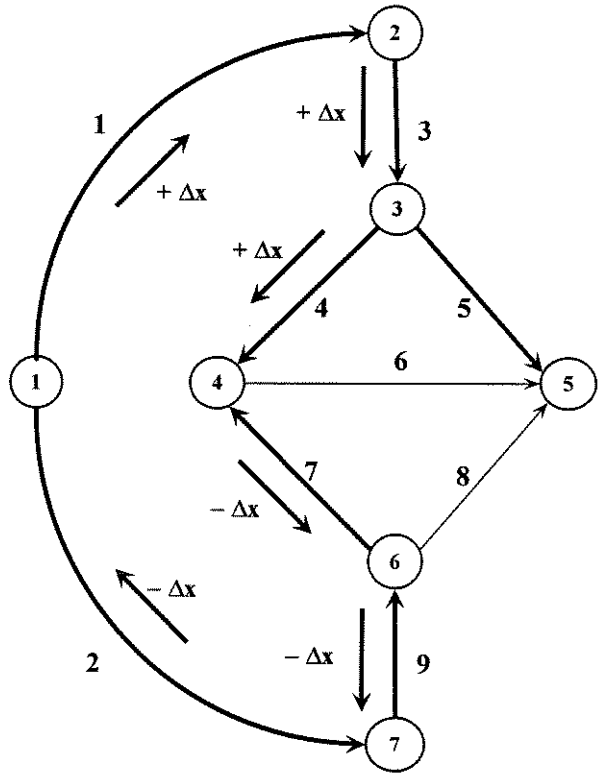


Fig. 2.1 Variação Unitária das Variáveis Básicas

Partindo da característica acima citada, podemos determinar facilmente a variação unitária das variáveis básicas em função de cada uma das variáveis não básicas, pois se cada uma delas faz circular um fluxo incremental Δx no ciclo formado, a variação unitária das variáveis básicas será de Δx nos arcos pertencentes ao ciclo e orientados no mesmo sentido que o arco não básico e de $-\Delta x$ nos arcos pertencentes ao ciclo e orientados no sentido contrário ao do arco não básico. Aplicando o

princípio da superposição, obtemos a variação unitária total de cada variável básica. Este fato pode ser melhor compreendido observando-se a Fig. 2.1, que corresponde à Fig. 1.3a com a inclusão do arco não básico *4*.

2.4 Observação

Ao final do processo de minimização com o método do gradiente reduzido, o grafo formado pelos arcos que possuem fluxo de corrente raramente é uma árvore, pois nem todos os arcos correspondentes às variáveis não básicas possuirão fluxo de corrente nulo. Como nosso objetivo é obter uma solução ótima radial, a partir deste grafo deve-se descobrir qual das inúmeras árvores que podem ser obtidas através da retirada dos arcos excedentes minimizará a nossa função objetivo. Esta técnica é o tema do próximo capítulo.

CAPÍTULO 3

BUSCA DA SOLUÇÃO FACTÍVEL ÓTIMA

3.1 Introdução

O objetivo deste capítulo é expor a metodologia utilizada para obtermos a configuração radial ótima da rede a partir da solução do problema relaxado P_r, obtida ao final do capítulo anterior.

A solução do problema relaxado P_r nos fornece os fluxos de corrente ideais em cada arco da rede para que as perdas por Efeito Joule fossem minimizadas, caso pudéssemos operar com todos os arcos simultaneamente fechados, mas isto não representa uma situação real; como já foi exposto anteriormente a rede deve operar de forma radial.

Portanto, a solução de P_r é otimista, visto que a rede precisa ser alterada até obtermos soluções radiais factíveis. A cada alteração (abertura de arcos), estaremos nos afastando da solução otimista.

A configuração radial ótima é aquela que, entre todas as possíveis configurações radiais, menos se afasta da solução otimista no que se refere a perdas.

Para encontrar esta configuração será necessário estabelecermos a metodologia de busca adequada para identificarmos qual dentre as inúmeras possíveis configurações radiais será a ótima. Isto é feito através da exploração de um espaço de estados (Pearl, 1984), onde cada estado representa uma possível configuração de rede.

3.2 Espaço de Estados

O espaço de estados, representado por um grafo, é o conjunto de todos os possíveis estados existentes em um problema, e de como eles se relacionam entre si, ou seja, quais estados podem ser obtidos a partir de cada estado e através de qual operação. A Fig. 3.1 ilustra o espaço de estados para a busca da configuração radial ótima em redes de distribuição.

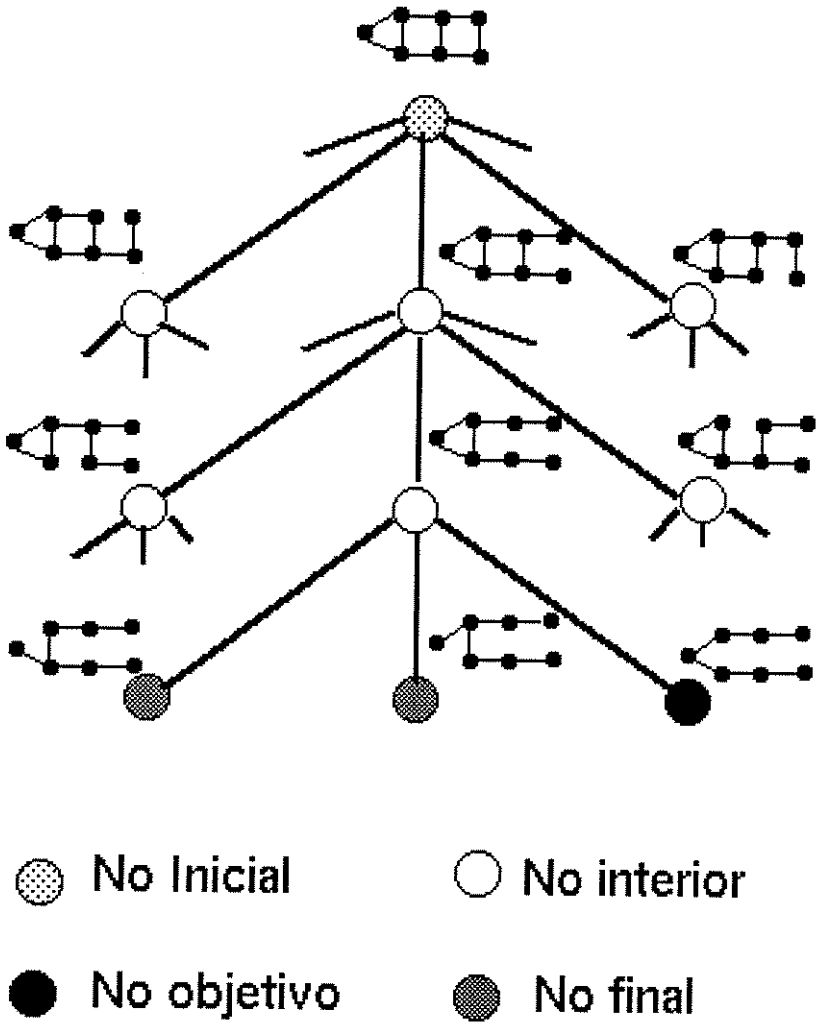


Fig. 3.1 Grafo do espaço de estados

Cada estado é representado por um nó, e cada operação é representada por um arco conectando os estados anterior e posterior à respectiva operação. Os estados posteriores de um determinado estado são denominados estados filhos, ou estados sucessores imediatos. Quando todos os estados filhos de um estado já foram gerados, o nó correspondente a este estado é denominado *expandido*. Ao conjunto de estados que além dos estados filhos, engloba os filhos dos filhos, e todos os demais descendentes, denomina-se conjunto dos estados sucessores.

Neste trabalho, cada estado é uma possível configuração de rede, e cada operação corresponde a manobra de chave seccionadora da rede.

Na Fig. 3.1, o nó origem representa o estado inicial, ou seja, a configuração otimista obtida com a resolução de $\boldsymbol{P_r}$. Por definição, este nó está no nível de profundidade *zero*. Os nós que constituem o nível de profundidade *um* representam os estados gerados a partir de uma única operação sobre o estado inicial – as configurações de rede que podem ser obtidas através da retirada de algum arco que formava ciclo no nó origem.

Para iniciar o processo de busca da solução radial ótima, devemos identificar quais são os arcos que formam ciclos no estado inicial e armazená-los em uma pilha. Se esta pilha for composta por m arcos, isto significa que a expansão do nó origem produzirá m estados filhos no nível de profundidade *um*.

Cada um destes m estados filhos corresponde à configuração de rede do nó origem após a retirada de algum dos m arcos que formavam ciclo. Logo, o número de arcos que formam ciclo em cada um destes estados passará a ser de m' $(m' < m)$; conseqüentemente, a expansão de cada um destes m estados produzirá m' filhos no nível de profundidade *dois*.

Repetimos o processo de expansão dos nós até o nível de profundidade que nos forneça as configurações radiais.

Como a passagem de um nível de profundidade para o nível seguinte é feita através da abertura de uma chave, o nível de profundidade que fornece configurações radiais é caracterizado pelo número de chaves que necessitam estar abertas nestas configurações (definido em 1.1).

Devido ao elevado número de estados em problemas deste gênero, foram desenvolvidos métodos de busca, para que o estado correspondente à solução desejada possa ser detectado o mais brevemente possível, ou seja, para que não exista a necessidade de percorrermos todo o espaço de estados. Os estados percorridos pelo método de busca constituem a árvore de busca.

A seguir são expostos alguns métodos de busca, seus princípios de funcionamento, vantagens e desvantagens.

3.3 Métodos de Busca

Os métodos de busca (Pearl, 1984) podem ser classificados em duas categorias: buscas desinformadas e buscas informadas.

As buscas desinformadas, como o próprio nome sugere, não utilizam informações do problema para guiar o procedimento de busca, enquanto as buscas informadas utilizam informações do problema, disponíveis a priori ou adquiridas durante a solução, para aprimorar o processo de busca.

3.3.1 Métodos de Busca Desinformada

Os três principais métodos de busca desinformada, busca em profundidade, backtracking e busca em largura, são discutidos a seguir:

3.3.1.1 Busca em Profundidade

A busca em profundidade é aquela que possui a característica de priorizar a expansão (geração de todos os estados filhos) dos nós que estão no maior nível de profundidade.

Expandimos o nó origem, e armazenamos seus filhos em uma pilha denominada ABERTO, que conterá todos os estados da árvore a serem processados. *Processar* o nó significa resolver o problema relaxado $\boldsymbol{P_r}$, e conseqüentemente determinar qual a distribuição ótima de fluxo e o valor das perdas por Efeito Joule associado para a configuração de rede representada por este nó no espaço de estados. Quando um nó e todos os seus sucessores já foram processados, este nó é considerado *explorado*. A cada expansão de um nó, seus filhos serão acrescentados no topo da pilha ABERTO. Conseqüentemente, aquele que estiver no topo da pilha será o último nó gerado – possui o maior nível de profundidade dentre os elementos desta pilha.

O próximo nó a ser expandido na busca em profundidade será aquele que estiver no topo da pilha. No entanto, se o nó origem possui m sucessores que foram gerados simultaneamente, qual deles ficará no topo da pilha, ou em outras palavras, como fazer a ordenação? Para esta pergunta existem duas possíveis respostas:

A primeira resposta seria armazená-los aleatoriamente, o que nos levaria a uma busca totalmente desinformada.

A segunda resposta seria ordená-los fazendo uso de alguma heurística, de forma que o estado, que segundo a heurística seria o mais promissor ficasse no topo, ou seja, que fosse armazenado

por último, e o estado menos promissor fosse o primeiro a ser armazenado. A utilização desta heurística torna esta busca do tipo parcialmente informada.

Em alguns casos é possível obtermos profundidade infinita, ou seja, sempre existirão estados sucessores. Neste caso, há o risco de continuarmos indefinidamente a busca, sem obtenção de um estado objetivo. Por isso, no procedimento genérico de busca em profundidade, é conveniente limitarmos a profundidade da busca. Convém, no entanto, enfatizar que a profundidade da árvore de busca para o problema estudado neste trabalho é finito, e determinável a priori. A busca em profundidade genérica pode ser resumida na seqüência de passos a seguir (Pearl, 1984):

1º Passo: Coloque o nó origem na pilha ABERTO;
2º Passo: Se ABERTO estiver vazio, saia indicando insucesso; caso contrário continue;
3º Passo: Remova o nó do topo da pilha ABERTO e coloque-o na pilha FECHADO; denomine-o de nó *n*;
4º Passo: Se a profundidade de *n* for igual ao limite de profundidade, limpe FECHADO e volte ao 2º passo; caso contrário continue;
5º Passo: Expanda *n* gerando todos os seus filhos; coloque estes filhos no topo da pilha ABERTO e providencie para cada um deles um ponteiro que aponte para *n*;
6º Passo: Se algum destes filhos for um estado objetivo, saia com a solução obtida percorrendo o caminho de volta deste estado para o estado inicial através dos ponteiros; caso contrário continue;
7º Passo: Se algum destes filhos for um fim-de-linha, ou seja, não possui sucessores, remova-o de ABERTO e limpe FECHADO;
8º Passo: Volte ao 2º passo.

A operação limpar a pilha FECHADO é realizada nos passos 4 e 7 apenas para que sejam retirados de FECHADO todos os nós que não possuem sucessores em ABERTO. Esta operação é opcional e tem como finalidade salvar espaço de memória.

3.3.1.2 Backtracking

Backtracking é uma versão da busca em profundidade, ou seja, também prioriza o caminho de busca que passa pelo nó gerado mais recentemente. A diferença está no fato de que ao contrário da busca em profundidade onde todos os filhos de um nó são gerados simultaneamente, os filhos na busca backtracking são gerados um de cada vez, e apenas depois do primeiro já ter sido explorado é que o segundo sucessor é gerado para também ser explorado.

O procedimento backtracking requer a precaução de marcarmos quais sucessores já foram gerados, para que quando o segundo sucessor já tiver sido explorado não corramos o risco de gerarmos novamente o primeiro sucessor ao invés de gerarmos o terceiro sucessor, caso ele exista. Este procedimento pode ser descrito pela seguinte seqüência de passos (Pearl, 1984):

1° Passo: Coloque o nó origem na pilha ABERTO;
2° Passo: Se ABERTO estiver vazio, saia indicando insucesso; caso contrário continue;
3° Passo: Examine o nó do topo da pilha ABERTO, e denomine-o de n;
4° Passo: Se a profundidade de n for igual ao limite de profundidade, ou todos os seus sucessores já tiverem sido percorridos, remova n de ABERTO e volte ao 2° passo; caso contrário continue;
5° Passo: Gere um novo sucessor de n, e denomine-o de n'; coloque n' no topo de ABERTO e providencie um ponteiro que aponte para n;
6° Passo: Marque n para indicar que o arco (n,n') já foi percorrido;
7° Passo: Se n' for um estado objetivo, saia com a solução obtida percorrendo o caminho indicado pelos ponteiros. Caso contrário continue;
8° Passo: Se n' for um fim-de-linha, remova-o de ABERTO;
9° Passo: Volte ao 2° passo.

A busca backtracking tem o atraente aspecto (em relação à busca em profundidade) de economia de espaço de memória, pois não mais estaremos armazenando todos os sucessores de cada nó independentemente do fato deles serem processados ou não, mas apenas aqueles que de fato são processados.

3.3.1.3 Busca em Largura

No procedimento da busca em largura, ao contrário do que é feito na busca em profundidade, a prioridade de expansão é dada aos nós que estão no menor nível de profundidade entre aqueles já gerados e não processados; de forma análoga à busca em profundidade, os nós estarão armazenados na pilha ABERTO. Portanto, a diferença entre a seqüência de passos destas duas buscas está no fato de que agora os novos estados gerados entram no fundo da pilha ABERTO, e não mais no topo. Quando a existência da solução é garantida, deixa de existir a necessidade de estabelecermos um limite de profundidade.

A seqüência de passos pode ser escrita como abaixo (Pearl, 1984):

1° Passo: Coloque o nó origem na pilha ABERTO;
2° Passo: Se ABERTO estiver vazio, saia indicando insucesso; caso contrário continue;
3° Passo: Remova o nó do topo da pilha ABERTO e coloque-o na pilha FECHADO; denomine-o de nó *n*;
4° Passo: Expanda *n* gerando todos os seus sucessores; coloque estes sucessores no fundo da pilha ABERTO e providencie para cada um deles um ponteiro que aponte para *n*;
5° Passo: Se algum destes sucessores for um estado objetivo, saia com a solução obtida percorrendo o caminho de volta deste estado para o estado inicial através dos ponteiros; caso contrário continue;
6° Passo: Se algum destes sucessores for um fim-de-linha, remova-o de ABERTO;
7° Passo: Volte ao 2° passo.

A busca em largura garante que a solução será encontrada, caso ela exista, o que nem sempre acontece na busca em profundidade (Pearl, 1984). Além disso a solução encontrada será aquela de menor profundidade, ou seja, a que requer um menor número de operações para ser obtida.

Vale lembrar que no problema estudado neste trabalho as possíveis soluções requerem o mesmo número de operações, ou seja, estão na mesma profundidade da árvore de busca. Portanto a característica de encontrar soluções na menor profundidade possível (uma boa característica das buscas em largura) não traz vantagem para este caso.

3.3.2 Métodos de Busca Informada

Os métodos de busca desinformada acima descritos possuem a capacidade de fornecer um estado final, ou seja, uma configuração de rede radial, mas neste trabalho a finalidade não é simplesmente encontrar um estado final, e sim determinar qual dos inúmeros estados finais existentes é o ótimo.

Problemas deste tipo são normalmente muito mais difícies que problemas onde apenas se procura estados que satisfaçam uma determinada condição – por exemplo, radialidade para redes de distribuição de energia elétrica. Para resolvê-los é necessário introduzir nos métodos de busca a capacidade de obter e utilizar informações para guiar a busca e reduzir o esforço necessário. Os métodos de busca heurísticos e o algoritmo A* são procedimentos clássicos de busca informada (Nilson, 1980).

A seguir são apresentados dois métodos de busca informada, ambos baseados no procedimento backtracking.

3.3.2.1 Backtracking Informado

Para se introduzir informações na busca backtracking, pode-se armazenar o valor da função objetivo da melhor solução factível (estado final) encontrada até o momento (o valor inicial de S será o valor das perdas por Efeito Joule para a configuração da rede em operação antes de iniciada a otimização), que denominaremos de S; este valor é comparado com o valor associado a cada nova configuração de rede, encontrada no decorrer da busca, que representaremos por f(n). Caso encontremos outra solução factível que possua f(n) inferior a S, este assume o valor de f(n). Desta forma, após concluída a busca podemos garantir que S nos fornece o valor da melhor solução factível existente (Cavellucci e Lyra, 1997).

Sabemos também que à medida que nos aprofundamos na árvore de busca, o valor de f(n) é sempre incrementado, pois nos afastamos cada vez mais da solução otimista. Portanto, caso encontremos alguma configuração de rede que possua f(n) superior a S, isto significa que seus sucessores não podem fornecer uma solução melhor do que aquela já conhecida, e portanto este caminho de busca pode ser abandonado. Isto é denominado *poda por dominância* (Pearl, 1984).

Na descrição dos passos do procedimento backtracking informado também é incluída a pilha CAMINHO, que contém as operações realizadas para atingir um determinado estado.

Considerando o valor inicial de S conhecido, a seqüência de passos é seguinte:

1° Passo: Armazene o valor da função objetivo da configuração de rede correspondente ao nó origem; identifique os arcos que formam ciclo e coloque-os em ABERTO por ordem dos valores de seus gradientes, de forma que o arco que fique no topo da pilha seja aquele que possua o maior gradiente;
2° Passo: Se ABERTO estiver vazio, pare; a pilha CAMINHO já contém a solução; caso contrário continue;
3° Passo: Gere um novo nó n extraindo o arco do topo da pilha ABERTO da configuração de rede correspondente ao nó origem, e resolva o problema P_r para a configuração de rede resultante (a solução de P_r fornece a nova distribuição ótima de fluxo e o novo valor de f(n)); adicione n à pilha CAMINHO;
4° Passo: Compare o novo valor de f(n) com S. Se $f(n) \geq S$, faça um backtracking, ou seja, descarte o novo nó gerado removendo-o da pilha CAMINHO, e volte ao 2° passo; caso contrário, se $f(n) < S$, continue;
5° Passo: Identifique os arcos que formam ciclo no nó n e coloque-os no topo de ABERTO, novamente observando a ordem dos gradientes; se não houver ciclos S assume o valor de f(n), isto é, $S \leftarrow f(n)$; extraia o nó n do topo da pilha CAMINHO, ou seja, faça um backtracking, pois n é o estado final que possui o melhor valor para as perdas encontrado até o momento.
6° Passo: Volte ao 2° passo.

Como a árvore de busca referente a este problema é finita e de profundidade conhecida, e não se perde caminhos para uma solução ótima no processo de poda, podemos afirmar que este método sempre nos fornecerá uma resposta, que corresponderá à distribuição radial de fluxo que minimiza as perdas por Efeito Joule.

3.3.2.2 Backtracking Heurístico

O método backtracking informado utiliza somente informações já conhecidas para processar o método de busca. O backtracking heurístico, por sua vez, utiliza não somente estes dados, mas informações obtidas através de métodos heurísticos, que nos fornecem uma estimativa sobre o potencial que cada caminho de busca possui de nos fornecer uma solução melhor do que aquela já conhecida. Quando esta estimativa supera o valor de S, o caminho de busca é podado; portanto o número de nós podados é normalmente bem superior ao mesmo número para a busca backtracking informada. Consequentemente, o tempo de processamento é reduzido.

A estimativa é expressa matematicamente da seguinte forma:

$$f_h(n) = f(n) + h(n) \tag{3.1}$$

onde h(n) é a previsão feita através de uma heurística do quanto a função f(n) será incrementada para se obter uma configuração radial a partir da configuração atual. A função heurística h(n) é definida da seguinte forma (Cavellucci e Lyra, 1997) :

$$h(n) = c(n) \,.\, \underline{w} \tag{3.2}$$

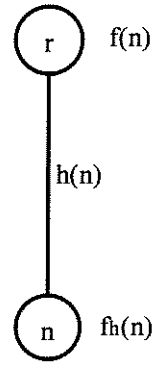


Fig. 3.2 Obtenção de $f_h(n)$ através da projeção de h(n) sobre f(n).

Nesta equação, c(n) corresponde ao número de ciclos existentes no nó *n* e $\underline{w}$ corresponde à estimação média do incremento nas perdas que cada ciclo provocará quando estiver eliminado. Para utilizarmos esta nova informação, basta substituirmos f(n) por $f_h(n)$ na última seqüência de passos apresentada. Quando h(n) fornece uma boa estimativa sobre o acréscimo das perdas no caminho sob expansão, obtemos mais rapidamente o ponto a partir do qual o caminho pode ser podado, e conseqüentemente reduzimos de forma considerável o espaço de busca.

É possível mostrar que o método backtracking heurístico fornece o ótimo global do problema, a menos que a função h(n) superestime as perdas, pois neste caso poderia ocorrer um backtracking no caminho que nos levaria ao ótimo global. Nos resultados computacionais apresentados no capítulo 5 pode-se observar a diferença de desempenho entre o backtracking informado e o backtracking heurístico.

3.4 Motivação para Uso de Processamento Paralelo

A utilidade do processamento paralelo se torna evidente, a partir do momento que observando o grafo do espaço de estados da Fig. 3.1, verificamos que o nível de profundidade *um* deste grafo é composto pelas *m* configurações de rede que podemos obter a partir da configuração de rede inicial. Se utilizarmos processamento paralelo, gerando um novo processo computacional para a resolução da sub-árvore de busca para cada um destes estados, pode-se obter uma economia de tempo substancial na busca da solução ótima.

Antes de uma exposição detalhada da implementação dos métodos de busca aqui expostos com processamento paralelo, é interessante abordarmos alguns aspectos genéricos sobre processamento paralelo e apresentarmos o software de paralelização PVM (Geist et al., 1994) utilizado neste trabalho.

CAPÍTULO 4

O PROCESSAMENTO PARALELO

4.1 Introdução

Processamento paralelo é atualmente uma área de pesquisa de intensa atividade, motivada por uma série de fatores. Um deles, é que sempre existiu a necessidade de se obter soluções de problemas computacionais de grande porte, mas apenas recentemente os avanços tecnológicos, como o surgimento do processamento paralelo, fizeram com que a possibilidade de se obter as soluções de alguns destes problemas aumentasse consideravelmente. Além disso, a disponibilidade de poderosos computadores paralelos também está gerando interesse na solução de novos problemas, que anteriormente não eram sequer discutidos.

A necessidade de resolver problemas cada vez maiores e mais complexos, sempre esteve à frente das capacidades de processamento e tem providenciado uma força motivadora para o desenvolvimento de computadores cada vez mais velozes, e processamento paralelo.

Problemas envolvendo equações diferenciais parciais (EDPs), como dinâmica de fluidos, meteorologia e processamento de imagens exigem um grande esforço computacional, devido ao volume de processamento numérico a ser realizado. Estes problemas podem ser facilmente decompostos, e portanto têm sido os primeiros candidatos à paralelização.

Alguns exemplos de outros tipos de problemas cujo interesse foi despertado mais recentemente são a análise, simulação e otimização de sistemas de grande porte interconectados.

Em ambas as classes de aplicações, as principais razões para aplicarmos processamento paralelo são custo e velocidade, pois o hardware não deve ser proibitivamente caro, e o processamento

deve terminar dentro de um tempo que seja aceitável para a aplicação em questão.

Os dois próximos itens foram baseados em Bertsekas e Tsitsiklis (1989).

4.2 Características Principais do Processamento Paralelo

Quando trabalhamos com processamento paralelo, devemos prestar atenção em algumas características próprias deste tipo de processamento para que possamos obter o melhor desempenho possível. Façamos então uma pequena exposição de algumas destas características.

4.2.1 Tipos de Memórias

A primeira dessas características é o tipo de memória do computador utilizado. Existem dois tipos de memórias mais utilizados: a memória compartilhada e a memória distribuída. Existem também uma grande variedade de outras arquiteturas de processamento paralelo que mesclam as propriedades destes dois tipos de memórias.

A memória compartilhada utiliza uma memória global que pode ser acessada por todos os processadores. Um processador se comunica com outro escrevendo a mensagem em uma determinada posição da memória global, que é lida por outro processador quando este acessar esta mesma posição da memória. Isto resolve o problema de comunicação entre os processadores, mas introduz o problema da concorrência de acesso de diferentes processadores ao mesmo endereço de memória. Para se contornar este problema, faz-se uso de sistemas de chaveamento, como o mostrado na Fig. 4.1. Naturalmente, a complexidade deste sistema aumenta proporcionalmente com o número de processadores, refletindo no aumento dos tempos de acesso à memória.

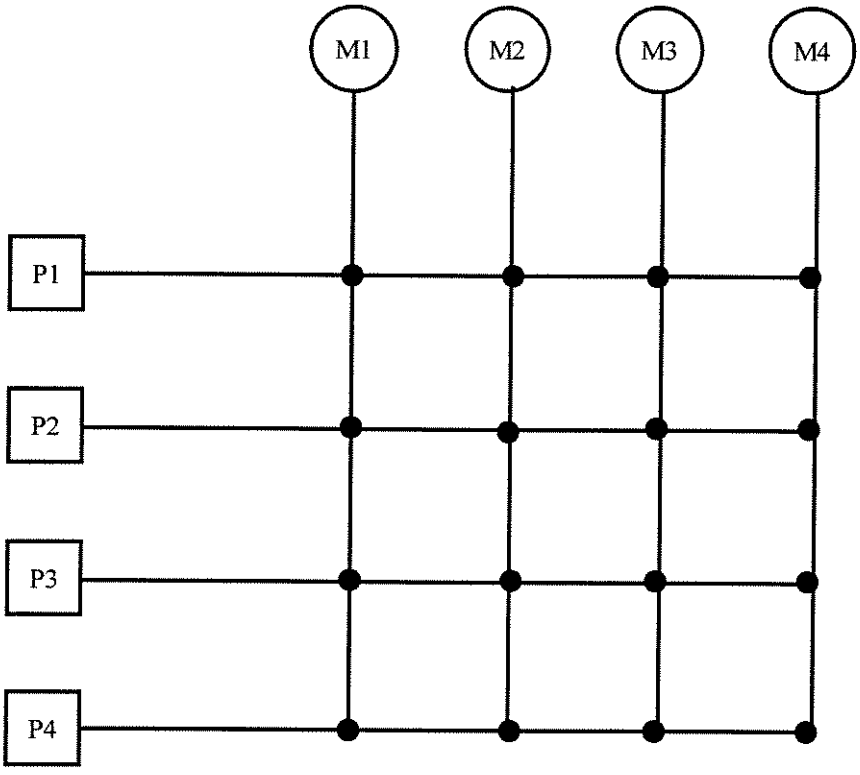


Fig. 4.1 Sistema de interligamento de memórias em uma arquitetura de memória global

Na memória distribuída, não existe uma memória global, mas sim uma memória local para cada processador – mesmo nas arquiteturas de memória compartilhada cada processador pode ter sua memória local. Neste caso, os processadores não necessitam estar necessariamente na mesma máquina. Atualmente, é possível interligar estações de trabalho e supercomputadores através de um sistema de comunicação, de forma a obtermos um único ambiente de trabalho, que passa a ser denominado **ambiente de computação distribuída** ou **máquina paralela virtual**. Este ambiente proporciona uma flexibilidade de trabalhar simultaneamente com diferentes configurações de hardware e de formas de interligação. Isto não acontece quando trabalhamos com computadores paralelos de memória compartilhada, que exigem arquiteturas mais rígidas, com processadores arquiteturas e formas de intercomunicação previamente definidas.

A Fig. 4.2 mostra a configuração básica para memória distribuída.

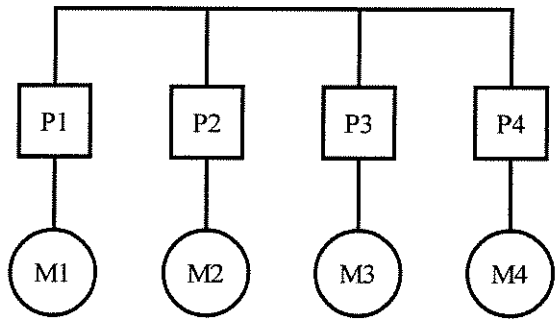


Fig. 4.2 Esquema básico de uma arquitetura de memória distribuída

4.2.2 Comunicação entre processadores

Outra componente importante do processamento paralelo é a comunicação entre os processadores. Em muitos algoritmos o tempo requerido para tal comunicação normalmente não é uma fração desprezível do tempo de processamento total. Neste caso, dizemos que o algoritmo possui uma penalidade de comunicação P_C, que pode ser expressa pela seguinte relação:

$$P_c = \frac{T_{total}}{T_{comp}} \tag{4.1}$$

Nesta expressão, T_{total} é o tempo requerido para executar o algoritmo de um dado problema e T_{comp} é o tempo atribuído apenas ao processamento, ou seja, o tempo que seria necessário se toda a comunicação fosse instantânea. Discutiremos um pouco sobre como acontece esta comunicação, e quais são os fatores que influenciam na penalidade de comunicação.

Consideraremos o ambiente de computação distribuída como uma rede de processadores conectados por "links" de comunicação. Cada processador utiliza sua memória local para armazenar dados do problema e resultados intermediários. A troca de informações com os outros processadores é realizada em grupos de bits, denominados pacotes, propagados através dos "links" de comunicação da rede. O comprimento do pacote pode variar de algumas dezenas de bits até alguns milhares de bits. Um pacote percorre um "link" de comunicação sem interrupção, isto é, todos os bits do pacote são transmitidos consecutivamente – a memória compartilhada também pode ser vista como uma rede de comunicação, onde cada processador pode enviar informações aos demais armazenando-as nesta

memória.

O tempo requerido pela comunicação pode ser dividido em quatro partes:

1ª) Tempo de processamento da comunicação: este é o tempo requerido para preparar a informação para a transmissão. Por exemplo, a montagem da informação em pacotes, introdução das informações de controle, como o endereço do destino, seleção do "link" a ser utilizado, a transferência dos pacotes para os devidos "buffers", etc.

2ª) Tempo de fila: a partir do momento que a informação foi preparada para a comunicação, é preciso esperar em uma fila por ordem de prioridade. Esta espera têm várias razões. Por exemplo, o "link" encontrar-se momentaneamente indisponível porque outros pacotes de informação ou pacotes do sistema de controle estão utilizando-o, ou estão programando-o para o uso do pacote dado. Outra razão é que pode se tornar necessário retardar a transmissão do pacote para garantirmos a disponibilidade de todos os recursos necessários, como por exemplo, espaço no "buffer" do destino.

3ª) Tempo de transmissão: é o tempo requerido para transmissão de todos os bits do pacote.

4ª) Tempo de propagação: é o tempo entre o fim da transmissão do último bit do pacote pelo processador transmissor, e a recepção do último bit do pacote pelo processador receptor.

Dependendo do algoritmo, um ou mais dos tempos citados podem ser desprezados. Por exemplo, em alguns casos a informação é gerada com suficiente regularidade e os recursos de transmissão são suficientes, de modo que nunca existe a necessidade do tempo de fila. Em outros casos a distância física entre o processador transmissor e o receptor é tão pequena que o tempo de propagação pode ser desprezado. Para a maioria dos casos, podemos considerar que o tempo de propagação e o tempo de preparação de um dado link é constante para todos os pacotes, sendo o tempo de transmissão proporcional ao número de bits, ou seja, ao comprimento do pacote. Podemos então expressar o tempo de comunicação que um pacote necessita para percorrer um link como:

$$T = P + R.C + F \tag{4.2}$$

onde P representa os tempos de preparação e propagação, R o tempo de transmissão requerido por um único bit, C o comprimento do pacote, e F o tempo de fila. Em alguns casos, o tempo de transmissão R.C é muito maior que a soma dos tempos de preparação e propagação, principalmente quando o pacote possui um grande comprimento; em outros casos, ocorre o contrário. De qualquer forma, na grande maioria dos casos, a soma P + R.C representa um tempo muito maior que aquele normalmente requerido para executar uma simples operação numérica, como uma multiplicação de variáveis expressas em ponto flutuante. Portanto, se um algoritmo paralelo requeresse transmissão de um pacote para toda pequena operação numérica, o tempo de comunicação seria superior ao tempo de processamento, e sua performance deixaria a desejar.

4.2.3 Sincronização de Processos

Em alguns algoritmos paralelos, faz-se necessário coordenar as atividades dos diferentes processadores. Esta coordenação é freqüentemente implementada através da divisão do algoritmo em fases. Durante cada fase, todos os processadores precisam executar um número de operações que depende dos resultados das operações dos outros processadores na fase anterior – entretanto, o tempo de processamento de qualquer processador durante uma fase é independente do tempo de processamento dos demais processadores dentro desta fase. Em cada fase, os processadores não interagem entre si (pois a comunicação é realizada ao final da fase). Uma fase se completa quando todos os processadores finalizam suas tarefas pré-estabelecidas para a mesma – portanto, aqueles processadores que terminam primeiro suas tarefas precisam esperar pela finalização das tarefas atribuídas aos demais processadores para que uma nova fase seja iniciada. Esses algoritmos são denominados ***síncronos.***

Em algoritmos síncronos devemos, sempre que possível, procurar o equilíbrio na distribuição de tarefas entre os processadores, pois aqueles com menor carga (menor volume de processamento a realizar), terminarão suas tarefas antes e ficarão à espera dos demais. Como o tempo de espera é indesejável (o processador fica inativo neste período), ele deve ser minimizado, através de distribuição eqüitativa das tarefas. Idealmente, todos os processadores terminarão suas tarefas em tempos aproximadamente iguais.

Algoritmos que não necessitam de comunicação entre processadores durante a execução, e consequentemente não necessitam de divisão em fases, são denominados algoritmos ***assíncronos***.

Neste caso, a coordenação dos processadores torna-se mais fácil, pois eles se comunicam apenas ao fim da execução.

4.3 Análise de Desempenho

Existem essencialmente dois parâmetros que utilizados na análise do desempenho de algoritmos paralelos: ***speedup*** (S) e ***eficiência*** (E).

O speedup estabelece a relação entre o tempo requerido pelo melhor algoritmo serial existente para a resolução de um dado problema (T_{serial}) e o tempo requerido pelo algoritmo paralelo que utiliza p processadores para a resolução do mesmos problema ($T_{paralelo}$). Portanto, o speedup é expresso conforme abaixo:

$$S = \frac{T_{serial}}{T_{paralelo}} \tag{4.3}$$

A eficiência calcula a média da fração de tempo que cada processador está sendo realmente utilizado. Portanto podemos expressar a eficiência como:

$$E = \frac{T_{serial}}{p.T_{paralelo}} = \frac{S}{p} \tag{4.4}$$

Devemos observar que o ideal seria obtermos $S = p$ e $E = 1$. Note que para isto ocorrer o tempo de comunicação e o tempo de espera para sincronismo necessitam ser nulos, tornando esta situação ideal praticamente inatingível. Além disso, devemos lembrar que a grande maioria dos algoritmos não são totalmente paralelizáveis, pois possuem uma parte do processamento que deve ser efetuado serialmente, impondo assim um limitante superior ao speedup.

Se a parte do processamento a ser realizado serialmente é uma fração f do processamento total, o speedup passa a ser expresso como:

$$S \leq \frac{1}{f + \frac{1-f}{p}} \leq \frac{1}{f}, \forall p \tag{4.5}$$

A expressão acima é conhecida como **Lei de Amdahl**. Ela mostra que, mesmo quando temos à disposição um grande número de processadores (o que permite executar a fração paralelizável do algoritmo em um tempo muito pequeno), a fração serial do processamento nos leva a um gargalo, ou seja, estabelece um limite inferior para o tempo total de processamento.

4.4 O Software de Paralelização

O software de paralelização utilizado neste trabalho denomina-se PVM (Parallel Virtual Machine). O PVM é um software que permite uma rede de computadores heterogêneos ser utilizada como um único computador paralelo de grande porte, ou seja, uma máquina paralela virtual. Conseqüentemente, problemas grandes podem ser solucionados através da capacidade agregada de vários computadores.

O desenvolvimento do PVM começou em 1989 no Oak Ridge National Laboratory (ORNL), Tennessee, EUA. Atualmente é utilizado em pesquisas e aplicações computacionais em todo do mundo.

O PVM é responsável por inicializar e gerenciar tarefas na máquina virtual, permitindo que as tarefas se comuniquem e se sincronizem umas com as outras. Uma tarefa é definida como uma unidade computacional para o PVM (análogo a um processo UNIX); freqüentemente as tarefas do PVM são implementadas como processos UNIX, mas não necessariamente.

Códigos escritos em FORTRAN 77 ou C, (neste trabalho utilizou-se a linguagem C), podem ser paralelizados utilizando instruções "message-passing", comuns à maioria dos computadores de memória distribuída. Enviando e recebendo mensagens, múltiplas tarefas de uma aplicação podem cooperar para solucionar um problema em paralelo. Em outras palavras, o PVM consiste de um conjunto de rotinas que quando inseridas em um programa originalmente serial permite a sua execução em uma máquina paralela virtual. A Tab. 4.1 ilustra algumas funções do PVM.

Pvm_mytid	Retorna o número de identificação do processo
Pvm_parent	Retorna o número de identificação do processo pai
Pvm_spawn	Gera processos filhos
Pvm_initsend	Prepara buffer para envio de dados
Pvm_pk	Empacota dados para a comunicação
Pvm_send	Envia dados
Pvm_recv	Recebe dados
Pvm_unpk	Desempacota dados

Tab. 4.1 Exemplos de funções do PVM

A filosofia do PVM baseia-se no conceito de **processo pai** e **processos filhos.** O processo pai é o responsável por iniciar o processamento, ativar os processos filhos, preparar o "buffer", empacotar e enviar os dados aos processos filhos, receber o resultado obtido por cada um deles e finalizar a execução. Aos processos filhos cabem o recebimento e desempacotamento dos dados, e a efetuação do processamento numérico desejado; cabe também o empacotamento e envio do resultado obtido ao processo pai.

Em algumas situações, como na aplicação deste trabalho, o próprio processo pai pode ser utilizado para o processamento numérico. Este recurso permite economizar a geração de um processo filho.

A geração do processo pai e dos processos filhos pode ser realizada através de dois modelos de paralelização: **mestre-escravo** e **SPMD**. No modelo mestre-escravo, existem dois códigos fonte, que possuem objetivos bem definidos. O primeiro, o mestre, é responsável pela execução do processo pai; o segundo, o escravo, é responsável pela execução dos processos filhos.

No modelo SPMD existe um único código fonte que executa o processo pai e também os processos filhos. Isto é possível devido à existência de uma rotina do PVM que, quando solicitada, retorna o número de identificação do processo pai (se esta rotina for solicitada dentro do processo pai, retorna número negativo). A resposta obtida (número positivo ou negativo), é utilizada como variável de controle para execução do código como processo pai ou como processo filho.

Este modelo só se aplica aos problemas que possuem a característica da recursividade, pois o processo pai, ao gerar os processos filhos, está somente ativando a si próprio várias vezes.

A aplicação estudada neste trabalho tem a característica de recursividade. Conseqüentemente, pode ser implementada com o modelo mestre-escravo ou com o SPMD.

Eventualmente, uma dessas alternativas poderia levar a melhores resultados computacionais, no entanto, como não se observou diferença significativa entre os resultados obtidos com os dois modelos, adotou-se apenas o mestre-escravo para apresentação dos resultados computacionais relatados no próximo capítulo. O modelo mestre-escravo tem a vantagem conceitual de uma lógica mais simples.

CAPÍTULO 5

IMPLEMENTAÇÃO, RESULTADOS E CONCLUSÕES

5.1 Adaptação do Algoritmo ao Processamento Paralelo

Expostas algumas características do processamento paralelo de modo genérico, o software de paralelização, as definições de processo pai e processos filhos e os modelos de paralelização, passemos ao nosso caso específico. O que precisa ser modificado no algoritmo para processarmos em paralelo?

5.1.1 Particionamento do Espaço de Estados e Geração dos Processos Filhos

As modificações são necessárias após a solução otimista ter sido obtida, quando os m arcos que formam ciclos nesta solução são conhecidos e armazenados em uma pilha. Isto significa que o estado inicial gera m estados filhos no primeiro nível da árvore de busca. A partir deste momento, particiona-se o espaço de estados da Fig. 3.1 em m sub-árvores, onde cada sub-árvore consiste no conjunto dos estados sucessores de cada um dos m estados filhos.

No processamento serial, processa-se o método de busca backtracking informado ou backtracking heurístico em uma sub-árvore por vez, portanto, somente após todos os sucessores do primeiro estado filho terem sido processados, é que se inicia a busca a partir do segundo estado filho.

No processamento paralelo, a busca da configuração radial ótima pelos métodos backtracking informado ou backtracking heurístico a partir de cada um dos m estados filhos do nível

de profundidade *um* é executada por um processo filho. Como o processo mestre pode ficar responsável pelo processamento do método de busca a partir de um dos m estados filhos, o número de processos filhos a serem gerados é calculado como l:

$$l = m - 1 \tag{5.1}$$

5.1.2 Inserção do Vetor Lista Negra

Quando não processamos em paralelo, ao gerarmos uma configuração de rede, é possível verificar se esta configuração de rede já foi processada por algum estado anterior (durante a busca, vários estados possuirão a mesma configuração de rede). Se isto já ocorreu, não é preciso fazê-lo novamente. Porém, ao processarmos em paralelo, cada processo pode verificar somente as configurações de rede processadas por ele mesmo, não tendo como verificar se outro processo já processou esta configuração de rede. Como o número de estados com mesma configuração de rede é elevado, isto acarreta em enorme aumento no volume de processamento a ser realizado.

Para contornar este problema, foi criado um vetor denominado lista negra. Quando o processo pai informa, ao primeiro processo filho, o arco k que este deve retirar da configuração de rede da solução otimista para iniciar o método de busca, este arco k também é armazenado na lista negra. Como todas as configurações de rede processadas pelo primeiro processo filho não possuirão o arco k, podemos dizer que o primeiro processo filho processa todas as possíveis configurações de rede em que o arco k está ausente. Portanto, todos os demais processos filhos não precisam processar as configurações de rede que não possuam o arco k. A função da lista negra é informar a cada processo filho, quando este é gerado, quais os arcos que já foram encaminhados aos processos filhos anteriores. Assim, este processo filho não necessita processar qualquer configuração de rede que não possua algum dos arcos contidos na lista negra recebida.

Outra vantagem da utilização da lista negra, é a possibilidade de reduzirmos o número de processos filhos a serem gerados. Como o número mínimo de arcos a serem retirados da rede para a obtenção de uma configuração de rede radial é dado por c_a (1.1), os processos filhos que são gerados de acordo com (5.1), mas cuja lista negra é tão grande que o número de arcos que podem ser retirados é menor que c_a, jamais poderão fornecer uma configuração de rede radial, e portanto não existe a necessidade de gerá-los. Logo, o número de processos filhos a serem gerados passa a ser de n:

$$n = m - c_a \tag{5.2}$$

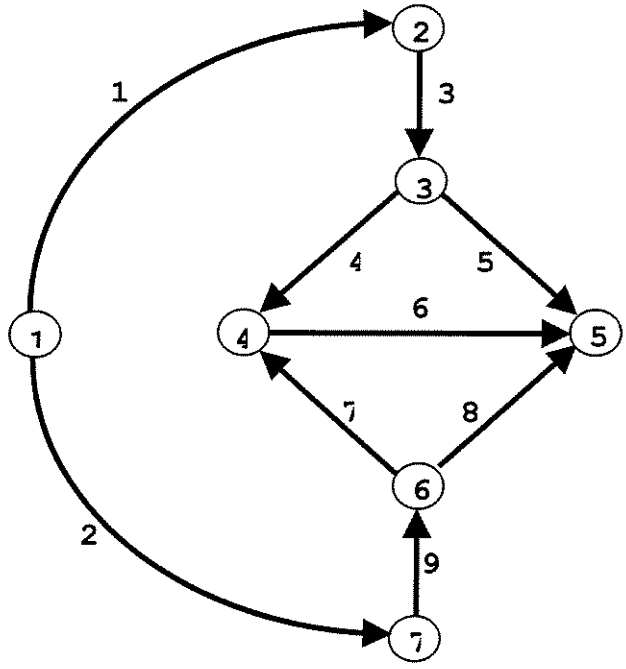


Fig. 5.1a Solução otimista para a Fig. 1.3

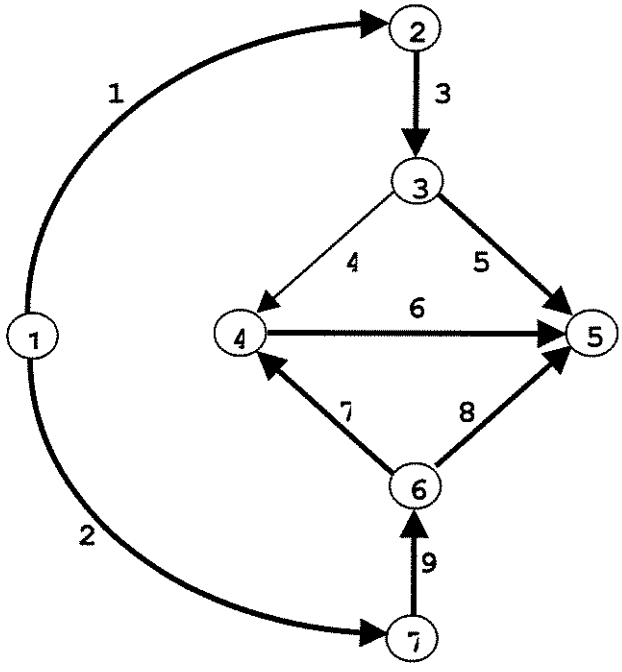


Fig. 5.1b Abertura da chave 4 na Fig. 5.1a.

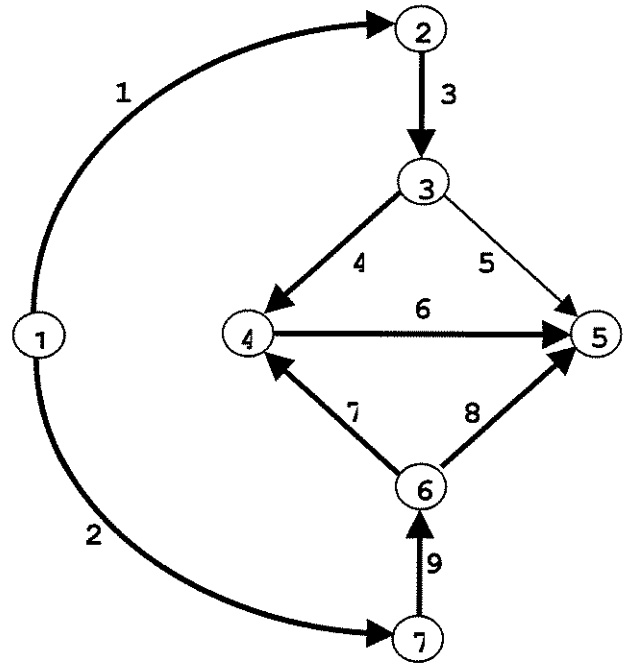


Fig. 5.1c Abertura da chave 5 na Fig. 5.1a.

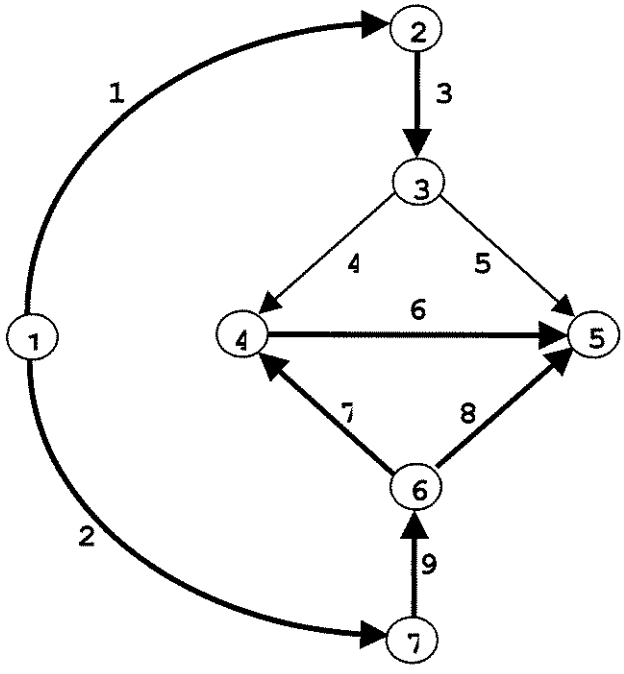


Fig. 5.1d Abertura da chave 5 na Fig. 5.1b.

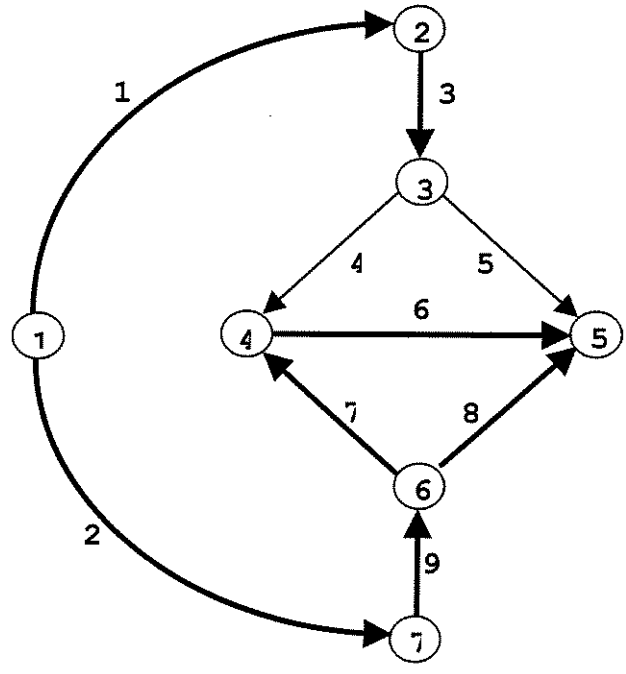


Fig. 5.1e Abertura da chave 4 na Fig. 5.1c.

Fig. 5.1 Exemplo de utilização da lista negra

A Fig. 5.1 ilustra a utilidade da lista negra. A Fig. 5.1a apresenta a configuração otimista. Esta rede possui nove arcos, o que implica na geração de nove possíveis estados no nível de profundidade *um*. Duas destas nove possibilidades são apresentadas na Fig. 5.1b e na Fig. 5.1c. Considerando que o estado correspondente à Fig. 5.1b seja gerado primeiro, isto significa que ao gerarmos o estado correspondente à Fig. 5.1c, o arco 4 está inserido na lista negra que é passada ao segundo processo filho, que será responsável pelo processo de busca a partir desta configuração de rede.

Ao iniciarmos a busca do primeiro processo filho, a partir da Fig. 5.1b, que possui oito arcos fechados, e todos formando ciclo, podemos gerar oito estados no nível de profundidade *dois*. Um destes oito estados é apresentado na Fig. 5.1d, onde retira-se o arco 5 da Fig. 5.1b. Da mesma forma, a busca a partir da Fig. 5.1c, feita pelo segundo processo filho, também pode gerar oito estados no nível de profundidade *dois*. Um destes oito estados correspondente à retirada da arco 4 na Fig. 5.1b, obtendo-se assim a Fig. 5.1e.

Como pode-se observar, a configuração de rede da Fig. 5.1e é idêntica à da Fig. 5.1d. E como cada uma foi obtida por um processo filho, que não enxerga as configurações já processadas pelos demais, isto significa que os dois processos filhos processariam a mesma configuração de rede, caso a lista negra não existisse. No entanto, o segundo processo filho possui o arco 4 na sua lista negra; ao identificar que este arco forma ciclo na Fig. 5.1c, e que deveria ser inserido na sua pilha ABERTOS, antes de inseri-lo na pilha, consulta a lista negra; ao constatar que o arco 4 está nela, retira o arco 4 da relação a ser acrescida à pilha ABERTOS. Consequentemente, o estado correspondente à Fig. 5.1e não é gerado pelo segundo processo filho.

5.1.3 Estabelecimento de Critérios para Comunicação

Todos os processos filhos recebem o valor da solução factível inicial para executar a poda por dominância. Porém, quando um dos processos encontra uma nova solução ótima factível, e esta não é comunicada aos demais processos, a poda por dominância dos demais se torna menos eficiente, ou seja, algum tempo será perdido na exploração de configurações de rede que não podem fornecer resultados melhores do que a nova solução ótima factível encontrada.

No entanto, sabemos que fazer esta comunicação também tem o seu preço. Não é possível executarmos processamento e comunicação simultaneamente, logo necessitamos interromper momentaneamente o processamento para fazermos a comunicação, o que não é muito interessante para o desempenho do algoritmo.

Resta então a seguinte pergunta: é melhor fazermos a comunicação para ganharmos tempo com a redução do volume de processamento numérico, ou é melhor não perdermos tempo com a comunicação e realizarmos um maior volume de processamento numérico? Isto possibilitou diversas opções de implementação. A primeira implementação não executa a comunicação durante o processamento, apenas no início e no fim dos processos filhos, enquanto a segunda e a terceira a executam. Na segunda implementação, cada processo finaliza uma fase quando detecta uma solução factível, e na terceira cada processo finaliza uma fase após ter explorado um número pré-determinado de nós na sua árvore de busca. Esta última, teoricamente, proporciona um menor tempo de espera para sincronização dos processos, pois atribui a estes um volume de processamento numérico aproximadamente iguais entre si.

A comunicação entre os processos ocorre da seguinte forma: ao final de cada fase, cada processo filho interrompe o seu processamento e informa ao processo pai o seu valor de S, ou seja, o

valor da função objetivo da melhor solução factível conhecida por ele até o momento. Após receber os valores de S de todos os processos filhos, o processo pai detecta qual a melhor de todas as soluções factíveis existentes até o momento e devolve esta informação aos processos filhos, dando continuidade ao processamento destes.

As seqüências de passos para o processo pai, e para os processos filhos, são apresentados a seguir:

Processo Pai:

1º Passo: determinar a solução otimista e armazenar em uma pilha os arcos que formam ciclo nesta solução;
2º Passo: gerar os processos filhos;
3º Passo: enviar a cada processo filho um arco armazenado na pilha, o valor inicial de S e a lista negra;
4º Passo: receber dos processos filhos, ao final de cada fase ou ao final da execução, os valores da melhor solução factível conhecida por cada um deles, e a informação se o processo filho finalizou ou não; caso todos os processo filhos tenham finalizado, vá para o 6º passo; caso contrário, continue no 5º passo;
5º Passo: retornar aos processos filhos que não finalizaram, o valor da melhor solução factível S já existente; retornar ao 4º Passo;
6º Passo: Imprimir a solução do problema e finalizar a execução.

Processos Filhos:

1º Passo: receber do processo pai o arco a ser retirado, o valor inicial de S e a lista negra;
2º Passo: processar o método de busca até finalizar uma fase ou finalizar a execução;
3º Passo: enviar ao processo pai o valor da melhor solução factível detectada e a informação se o processo foi finalizado ou não; caso não tenha finalizado, vá para o 4º passo.
4º Passo: receber do processo pai o valor da melhor solução existente até o momento; retornar ao 2º passo.

5.2 Descrição do Ambiente Utilizado

A implementação foi realizada nas máquinas do Centro Nacional de Processamento de Alto Desempenho em São Paulo (CENAPAD-SP), situado no Centro de Computação da UNICAMP, e no CENAPAD-NE, situado no Núcleo de Processamento de Dados da Universidade Federal do Ceará.

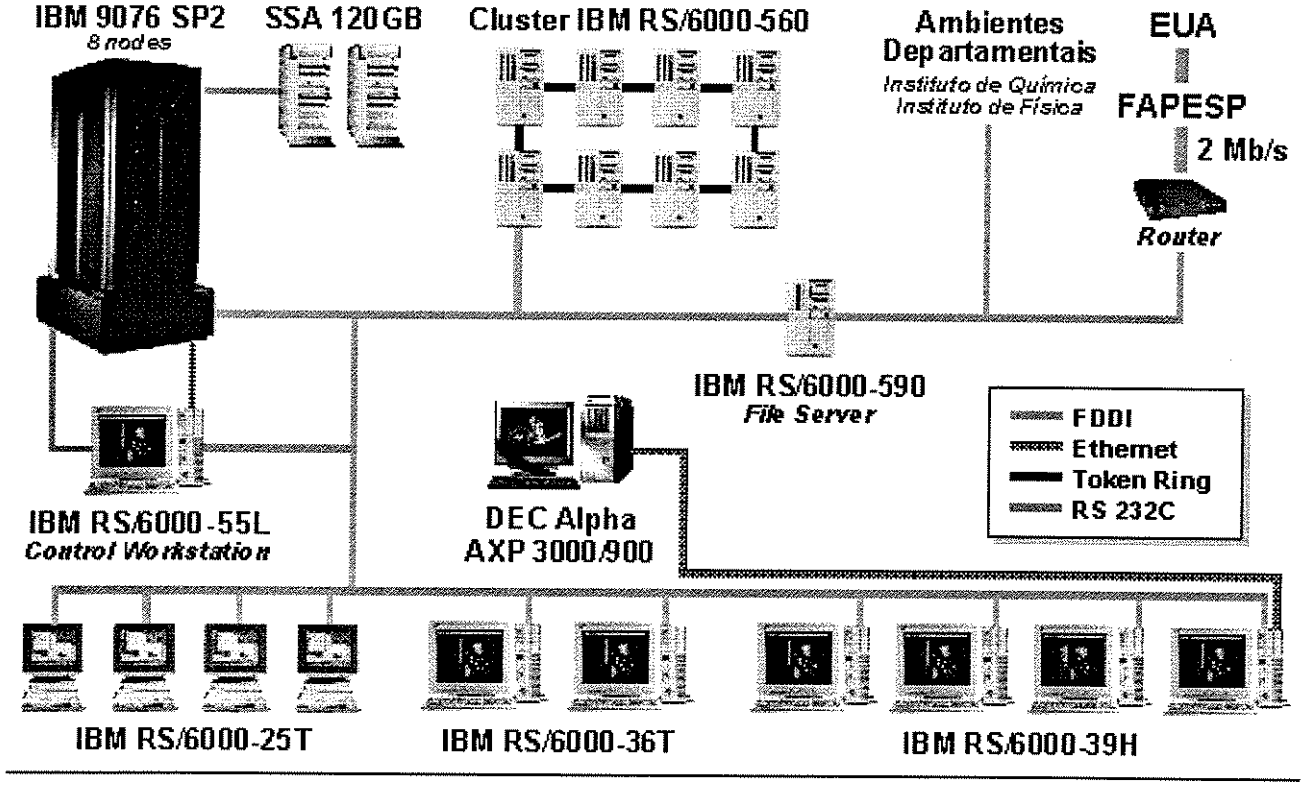


Fig. 5.1 Diagrama do ambiente do CENAPAD-SP

A Fig. 5.1 mostra as máquinas que compõem o ambiente do CENAPAD-SP. Esta figura pode ser visualisada no seguinte "site":
http://www.cenapad.unicamp.br/CENAPAD/Ambiente/Diagrama/diagrama.html.

Através desta página, pode-se obter uma descrição completa sobre as máquinas do ambiente, bastando para isso clicar sobre a máquina desejada. Portanto, façamos apenas uma rápida descrição das máquinas de maior interesse que compõem o ambiente. Como se pode notar pela Fig. 5.1, trata-se de um ambiente de computação distribuída. As máquinas são as seguintes:

IBM 9076 SP2 (Scalable Power Parallel System 2): supercomputador com 8 processadores IBM POWER RISC 6000 modelo 370. Cada processador possui 256 Mbytes de memória RAM, e pode atingir uma performance de pico igual a 200 Mflops (o que totaliza 1,6 Gflop). A velocidade de clock é de 66 MHz, e os processadores são interconectados através de um High Performance Switch com capacidade de 48,3 MB/s.

Cluster IBM RS/6000-560: consiste de 8 estações de trabalho, possuindo cada estação um processador IBM Power Risc 6000 modelo 560. Cada processador possui 128 Mbytes de memória RAM e velocidade de clock de 50 MHz. Os processadores são interconectados por uma rede de cabos coaxiais Ethernet token-ring LAN com uma banda nominal de passagem de 16Mbps.

IBM RS/6000-590: é uma das máquinas servidoras de arquivos do ambiente, e possui um processador IBM Power Risc 6000 modelo 590, com 128 Mbytes de memória RAM e velocidade de clock de 66,7 MHz.

IBM RS/6000 - OU 55L: é a outra servidora de arquivos de ambiente, e estação de controle do SP2. Possui um processador IBM Power Risc 6000 modelo 55L, com 128 Mbytes de memória RAM e velocidade de clock de 41 MHz.

IBM RS/6000-25T: 4 máquinas com processadores IBM Power Risc modelo 25T. Cada uma possui 32 Mbytes de memória RAM e velocidade de clock de 66,7 MHz.

IBM RS/6000-36T: 2 máquinas com processadores IBM Power Risc modelo 36T. Cada uma possui 64 Mbytes de memória RAM e velocidade de clock de 50MHz.

IBM RS/6000-39h: 4 máquinas com processadores IBM Power Risc modelo 39H. Cada uma possui 256 Mbytes de memória RAM e as demais 128 Mbytes. A velocidade de clock é de 67 MHz.

5.3 Resultados

Os resultados apresentados a seguir foram obtidos utilizando-se as máquinas IBM RS/6000-25T e IBM RS/6000-36T, pois nestas máquinas existem horários estabelecidos para que se possa dispor do processador com exclusividade, fator importante quando se deseja cronometrar o tempo de execução do programa.

Foram obtidos resultados para as três redes de testes apresentadas abaixo:

	NÓS	ARCOS	CHAVES ABERTAS	PERDAS INICIAIS(kW)	PERDAS ÓTIMAS(kW)	ECONOMIA
Rede 48	48	51	4	2,525	2,117	16,16%
Rede Wu	34	38	5	131,269	83,849	36,12%
Rede Bauru 9	56	64	9	101,922	89,030	12,65%

Tab. 5.1 Redes de testes utilizadas.

A rede 48 é uma rede hipotética, a rede Wu é a mesma que foi utilizada por Baran e Wu (Baran and Wu, 1989) e a rede Bauru 9 corresponde a uma área de aproximadamente 20% da rede de distribuição da cidade de Bauru, de acordo com a rede apresentada por Carneiro da Silva (1990).

Podemos notar que a rede 48 possui profundidade quatro na sua árvore de busca, enquanto a rede Wu possui profundidade cinco, e a rede Bauru9 possui profundidade nove. Como foi exposto no primeiro capítulo, quanto maior o número de chaves abertas, maior deve ser o tempo de processamento.

Antes de apresentar os tempos totais de processamento para cada uma destas redes, é apresentado o tempo necessário (em segundos) para a geração dos processos filhos, importantes para uma boa interpretação dos resultados dos tempos totais de processamento.

	Número de Processadores				
REDES	2	3	4	5	6
Rede 48	2,256	1,545	1,345	1,228	1,029
Rede Wu	37,886	14,229	11,038	7,560	8,043
Rede Bauru 9	4,899	2,659	2,205	1,994	1,942

Tab. 5.2 Tempos de processamento para geração dos processos filhos.

5.3.1 Backtracking Informado

5.3.1.1 Rede 48

A configuração inicial para a rede 48 possui as chaves dos arcos 10, 24, 27 e 29 no estado aberto. Com esta configuração as perdas por Efeito Joule são de 2,525kW, conforme a Tab. 5.1.

Após a obtenção da solução otimista, a pilha ABERTOS possui o seguinte conteúdo:

ÍNDICE	*ARCOS*	*ÍNDICE*	*ARCOS*	*ÍNDICE*	*ARCOS*	*ÍNDICE*	*ARCOS*
20	**2**	15	**21**	10	**9**	5	**40**
19	**32**	14	**24**	9	**39**	4	**27**
18	**19**	13	**7**	8	**25**	3	**10**
17	**18**	12	**35**	7	**8**	2	**26**
16	**22**	11	**23**	6	**11**	1	**29**

Tab. 5.3 Pilha ABERTOS após a obtenção da solução otimista da rede 48.

Conforme pode ser observado na Tab. 5.3, existem 20 arcos formando ciclos na configuração de rede da solução otimista. Isto significa que o estado inicial gera 20 estados filhos no primeiro nível da árvore de busca, e portanto, existem 20 sub-árvores a serem processadas.

De acordo com (5.2), é necessário gerar 16 processos filhos neste caso. Portanto o processo mestre envia aos 16 processos filhos os 16 arcos que estão no topo da pilha ABERTOS, desempilha os três arcos seguintes *(27, 10 e 26)* sem acrescentá-los à lista negra, e inicia o método de

busca na sub-árvore que parte do estado filho que não possui o arco *29* na sua configuração de rede. Como os três arcos anteriores não foram acrescentados à lista negra, isto significa que os arcos *27, 10, 26* e *29* são os únicos que o processo mestre tem permissão para retirar da rede no seu processo de busca. Conseqüentemente o processo mestre só pode detectar uma configuração radial. Vejamos então o número de nós processados em cada uma das 20 sub-árvores de busca no processamento serial e nas três versões do processamento paralelo.

O que foi denominado de 1ª, 2ª e 3ª implementação paralela, refere-se respectivamente a:

- processamento paralelo sem interrupção para comunicação entre os processos durante o processamento;

- processamento paralelo com interrupção para comunicação entre os processos a cada nova solução ótima detectada;

- processamento paralelo com interrupção para comunicação entre os processos a cada x nós explorados na árvore de busca, sendo:

$x = 10$, para a rede 48;

$x = 50$, para a rede Wu;

$x = 50$, para a rede Bauru 9.

SUB-ÁRVORE	ARCO RETIRADO	IMPLEMENT. SERIAL	1ª IMPLEM. PARALELA	2ª IMPLEM. PARALELA	3ª IMPLEM. PARALELA
1	2	1	1	1	1
2	32	1	1	1	1
3	19	1	1	1	1
4	18	17	17	17	15
5	22	15	15	15	15
6	21	23	23	19	18
7	24	28	28	25	19
8	7	11	11	11	11
9	35	25	25	16	16
10	23	30	46	33	19
11	9	11	15	15	13
12	39	22	27	21	18
13	25	16	16	16	16
14	8	11	14	14	14

15	11	7	8	8	8
16	40	9	12	12	12
17	27	6	4	4	4
18	10	1	0	0	0
19	26	1	0	0	0
20	29	1	0	0	0
TOTAL		237	264	229	201

Tab. 5.4 Número de nós processados em cada sub-árvore de busca da rede 48.

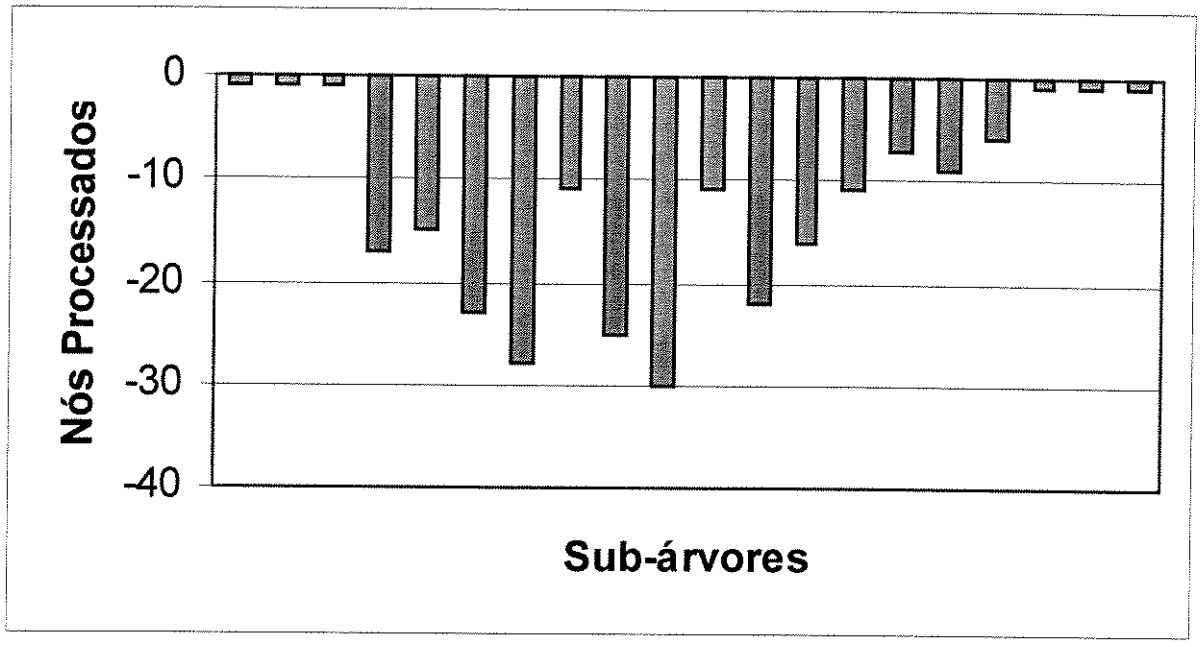
Após concluída a otimização, todas as versões de implementação apontam como solução a mudança da configuração inicial com as chaves 10, 24, 27 e 29 abertas para a configuração com as chaves 40, 10, 27 e 29 abertas. Portanto, em relação à configuração inicial deve-se fechar a chave 24 e abrir a chave 40.

Quando a solução é apresentada como 40, 10, 27 e 29, isto significa que este é o conteúdo da pilha CAMINHO ao final do processamento. Como a pilha CAMINHO é preenchida de acordo com a ordem de retirada dos arcos, isto quer dizer que o primeiro arco retirado foi o 40, o segundo foi o 10, o terceiro foi o 27 e o último foi o 29. E se o primeiro arco retirado foi o 40, de acordo com a Tab. 5.4, sabemos que a solução ótima foi detectada pela 16ª sub-árvore, embora esta esteja entre as que processaram um pequeno número de nós.

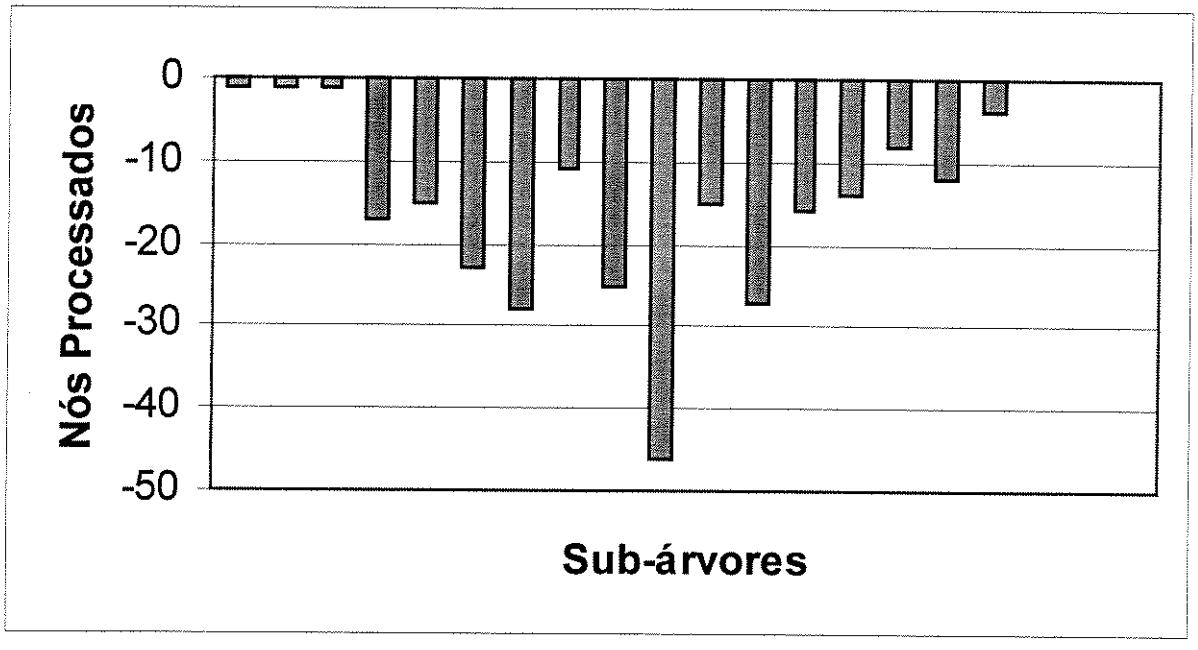
Vejamos o perfil da árvore de busca para os resultados acima. Nos gráficos a seguir, o eixo y apresenta o número de nós processados em cada sub-árvore (eixo x). Para os gráficos correspondentes ao processamento paralelo, o eixo x contém da esquerda para a direita os processos filhos por ordem de geração. A última barra correspondente ao processo mestre.

Os tempos (em segundos) para o processamento das quatro versões de implementação variando o número de processadores até 6 são apresentados a seguir. Em cada valor de tempo para as implementações paralelas está embutido o tempo necessário para a geração dos processos filhos, apresentados na Tab. 5.2.

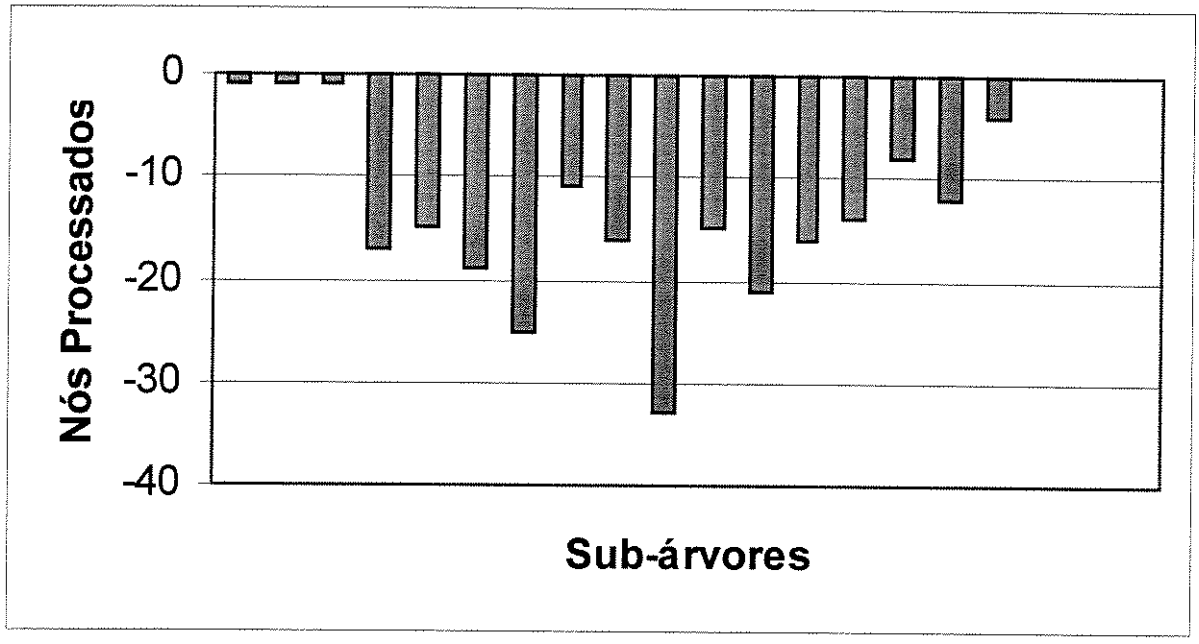
Através dos tempos da Tab. 5.5, podemos determinar o *speed-up* e a *eficiência* do processamento paralelo de acordo com (4.3) e (4.4).



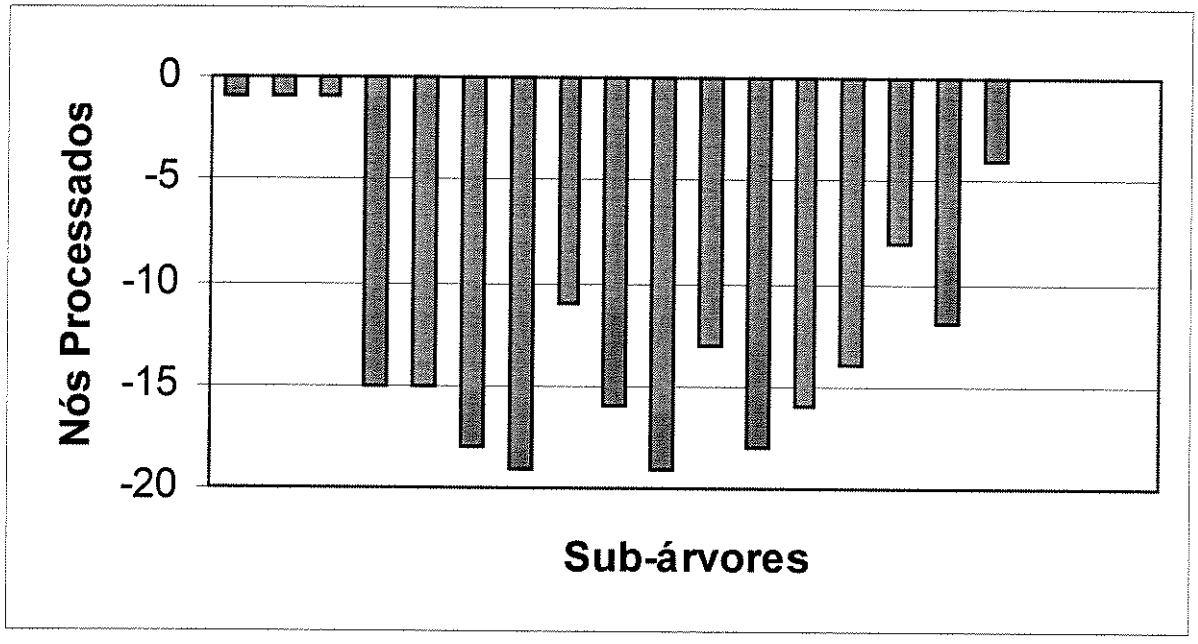
Gráf. 5.1 Perfil da árvore de busca da implementação serial da rede 48.



Gráf. 5.2 Perfil da árvore de busca da primeira implementação paralela da rede 48.



Gráf. 5.3 Perfil da árvore de busca da segunda implementação paralela da rede 48.



Gráf. 5.4 Perfil da árvore de busca da terceira implementação paralela da rede 48.

	REDE 48	IMPLEMENT. SERIAL	1ª IMPLEM. PARALELA	2ª IMPLEM. PARALELA	3ª IMPLEM. PARALELA
	NÓS PROC.	237	264	229	201
Número de Processadores	1	2,924			
	2		3,792	4,359	3,905
	3		2,818	3,162	2,840
	4		2,289	2,676	2,492
	5		2,160	2,439	2,178
	6		1,815	2,109	1,898

Tab. 5.5 Tempos de processamento para a rede 48.

	REDE 48	1ª IMPLEM. PARALELA	2ª IMPLEM. PARALELA	3ª IMPLEM. PARALELA
Número de Processadores	2	0,771	0,671	0,749
	3	1,038	0,925	1,030
	4	1,277	1,093	1,173
	5	1,354	1,199	1,343
	6	1,611	1,386	1,541

Tab. 5.6 Valores de *speed-up* para a rede 48.

	REDE 48	1ª IMPLEM. PARALELA	2ª IMPLEM. PARALELA	3ª IMPLEM. PARALELA
Número de Processadores	2	0,386	0,335	0,374
	3	0,346	0,308	0,343
	4	0,319	0,273	0,293
	5	0,271	0,240	0,269
	6	0,269	0,231	0,257

Tab. 5.7 Valores de *eficiência* para a rede 48.

5.3.1.2 Rede Wu

A configuração inicial para a rede Wu possui as chaves dos arcos 20, 34, 35, 36 e 37 no estado aberto. Vejamos o conteúdo da pilha ABERTOS após a obtenção da solução otimista

ÍNDICE	*ARCOS*	*ÍNDICE*	*ARCOS*	*ÍNDICE*	*ARCOS*	*ÍNDICE*	*ARCOS*
36	**19**	27	**27**	18	**28**	9	**13**
35	**2**	26	**3**	17	**16**	8	**31**
34	**23**	25	**20**	16	**15**	7	**6**
33	**35**	24	**4**	15	**18**	6	**14**
32	**33**	23	**12**	14	**37**	5	**11**
31	**5**	22	**30**	13	**26**	4	**36**
30	**24**	21	**8**	12	**17**	3	**9**
29	**22**	20	**21**	11	**7**	2	**10**
28	**34**	19	**29**	10	**25**	1	**32**

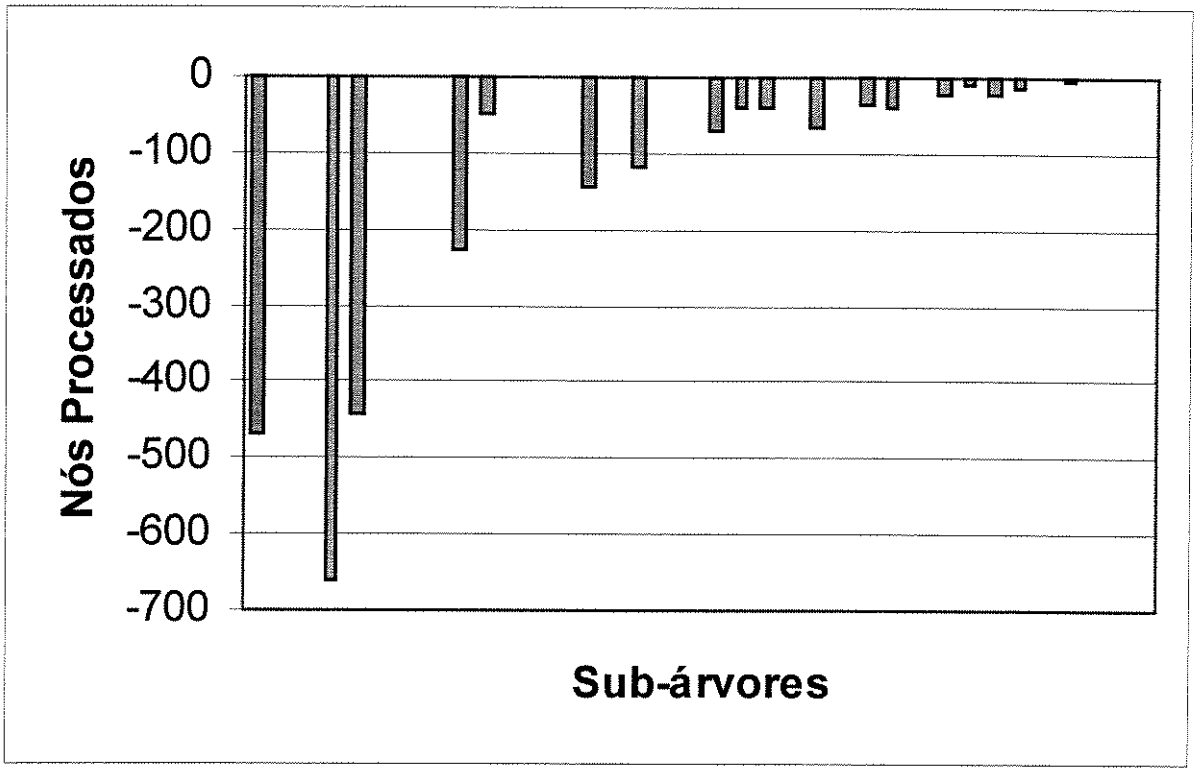
Tab. 5.8 Pilha ABERTOS após a obtenção da solução otimista da rede Wu.

SUB-ÁRVORE	ARCO RETIRADO	IMPLEMENT. SERIAL	1ª IMPLEM. PARALELA	2ª IMPLEM. PARALELA	3ª IMPLEM. PARALELA
1	19	470	470	70	84
2	2	1	1	1	1
3	23	1	77	48	53
4	35	660	688	256	124
5	33	443	966	388	164
6	5	1	493	81	103
7	24	1	270	86	101
8	22	1	1	1	1
9	34	229	434	255	195
10	27	46	305	166	111

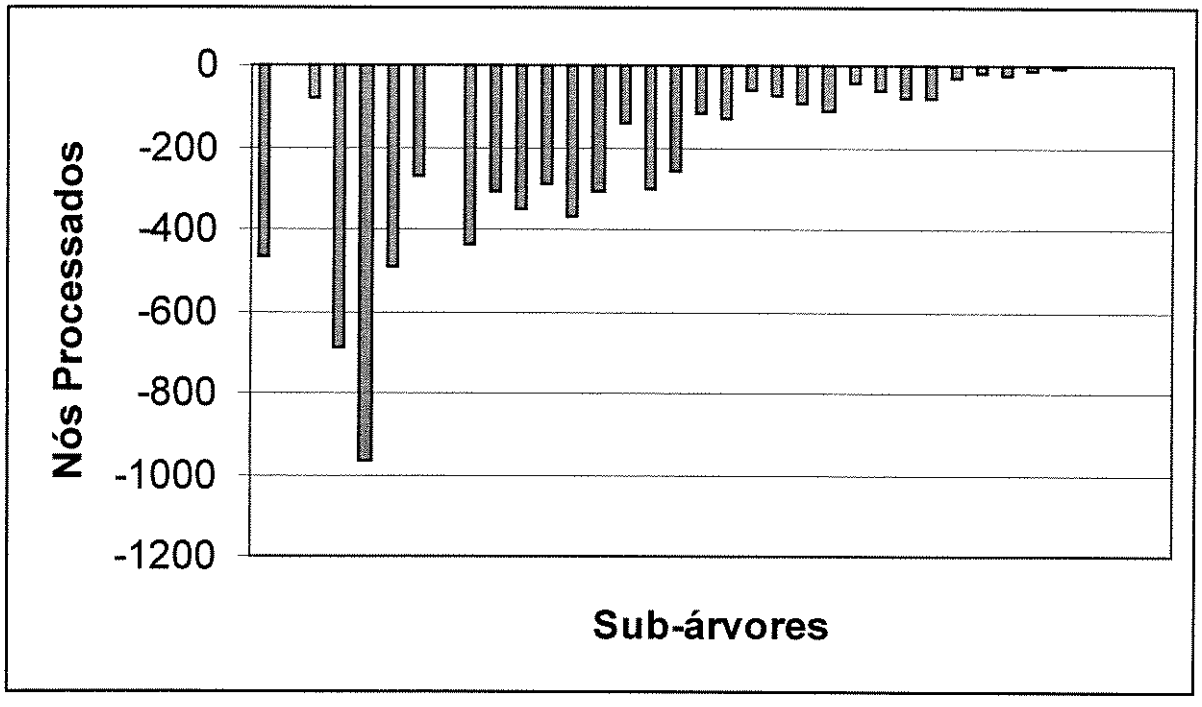
11	3	1	350	63	88
12	20	1	291	34	91
13	4	1	370	65	89
14	12	144	305	226	164
15	30	1	144	56	71
16	8	117	303	225	145
17	21	1	260	90	80
18	29	1	115	33	74
19	28	72	128	105	119
20	16	40	63	57	63
21	15	40	73	56	73
22	18	1	90	36	61
23	37	67	113	106	113
24	26	1	44	20	44
25	17	37	61	52	61
26	7	40	78	66	78
27	25	1	83	37	63
28	13	21	30	30	30
29	31	10	16	16	16
30	6	24	24	24	24
31	14	12	12	12	12
32	11	1	4	4	4
33	36	3	0	0	0
34	9	2	0	0	0
35	10	1	0	0	0
36	32	1	0	0	0
TOTAL		2494	6662	2765	2500

Tab. 5.9 Número de nós processados em cada sub-árvore de busca da rede Wu.

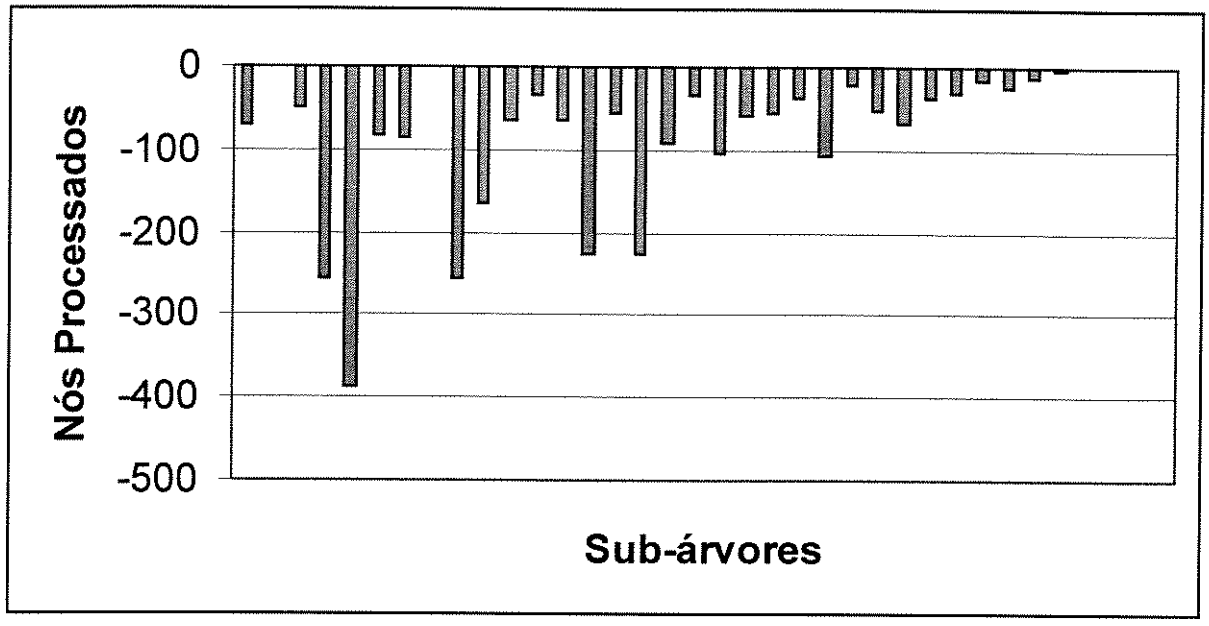
Após a otimização, o conteúdo da pilha CAMINHO é: 37, 7, 14, 9 e 32. Portanto a solução foi detectada na 23ª sub-árvore.



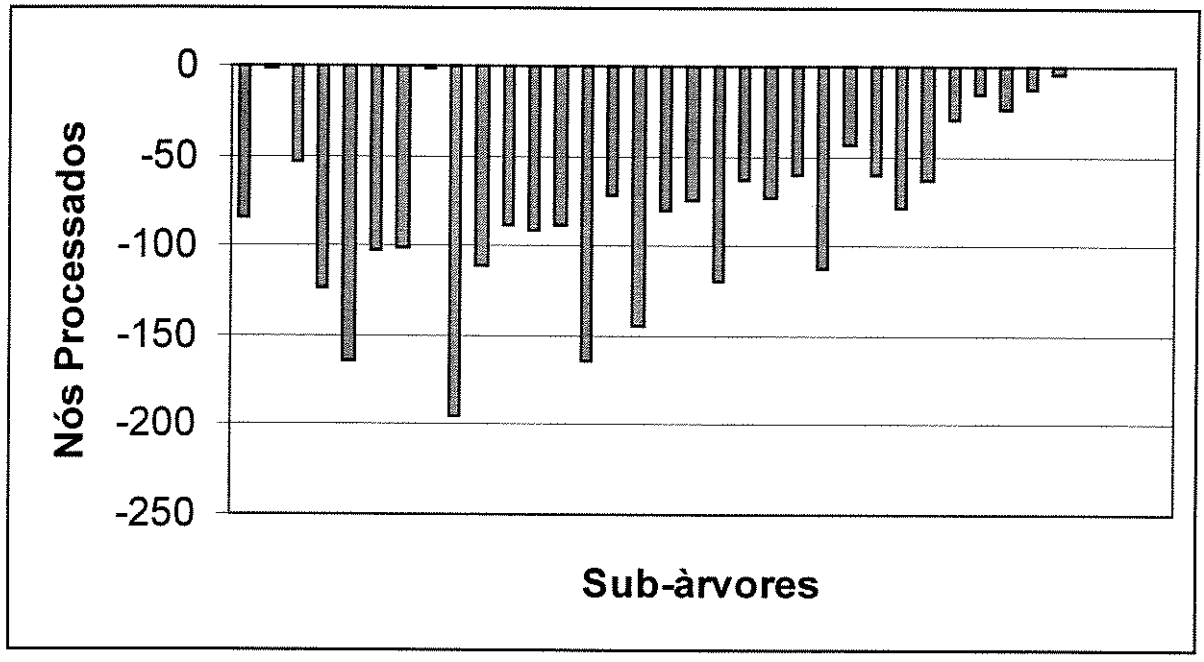
Gráf. 5.5 Perfil da árvore de busca implementação serial da rede Wu.



Gráf. 5.6 Perfil da árvore de busca da primeira implementação paralela da rede Wu.



Gráf. 5.7 Perfil da árvore de busca da segunda implementação paralela da rede Wu.



Gráf. 5.8 Perfil da árvore de busca da terceira implementação paralela da rede Wu.

	REDE WU	IMPLEMENT. SERIAL	1ª IMPLEM. PARALELA	2ª IMPLEM. PARALELA	3ª IMPLEM. PARALELA
	NÓS PROC.	2494	6662	2765	2500
Número de Processadores	1	48,325			
	2		100,071	69,503	57,464
	3		71,744	43,372	32,311
	4		56,554	35,790	29,310
	5		47,203	36,454	23,868
	6		47,152	32,376	22,332

Tab. 5.10 Tempos de processamento para a rede Wu.

	REDE WU	1ª IMPLEM. PARALELA	2ª IMPLEM. PARALELA	3ª IMPLEM. PARALELA
Número de Processadores	2	0,483	0,695	0,841
	3	0,674	1,114	1,496
	4	0,854	1,350	1,649
	5	1,024	1,326	2,025
	6	1,025	1,493	2,164

Tab. 5.11 Valores de *speed-up* para a rede Wu.

	REDE WU	1ª IMPLEM. PARALELA	2ª IMPLEM. PARALELA	3ª IMPLEM. PARALELA
Número de Processadores	2	0,241	0,348	0,420
	3	0,225	0,371	0,499
	4	0,214	0,338	0,412
	5	0,205	0,265	0,405
	6	0,171	0,249	0,361

Tab. 5.12 Valores de *eficiência* para a rede Wu.

5.3.1.3 Rede Bauru 9

A configuração inicial da rede Bauru 9 possui as chaves dos arcos 60, 63, 64, 65, 71, 72, 74, 80 e 81 no estado aberto. O conteúdo da pilha ABERTOS após a obtenção da solução otimista é o seguinte:

ÍNDICE	*ARCOS*	*ÍNDICE*	*ARCOS*	*ÍNDICE*	*ARCOS*	*ÍNDICE*	*ARCOS*
31	**2**	23	**75**	15	**57**	7	**59**
30	**1**	22	**6**	14	**73**	6	**61**
29	**3**	21	**7**	13	**10**	5	**52**
28	**5**	20	**65**	12	**54**	4	**80**
27	**77**	19	**72**	11	**47**	3	**62**
26	**53**	18	**55**	10	**81**	2	**78**
25	**76**	17	**58**	9	**63**	1	**64**
24	**71**	16	**74**	8	**60**		

Tab. 5.13 Pilha ABERTOS após a obtenção da solução otimista da rede Bauru 9.

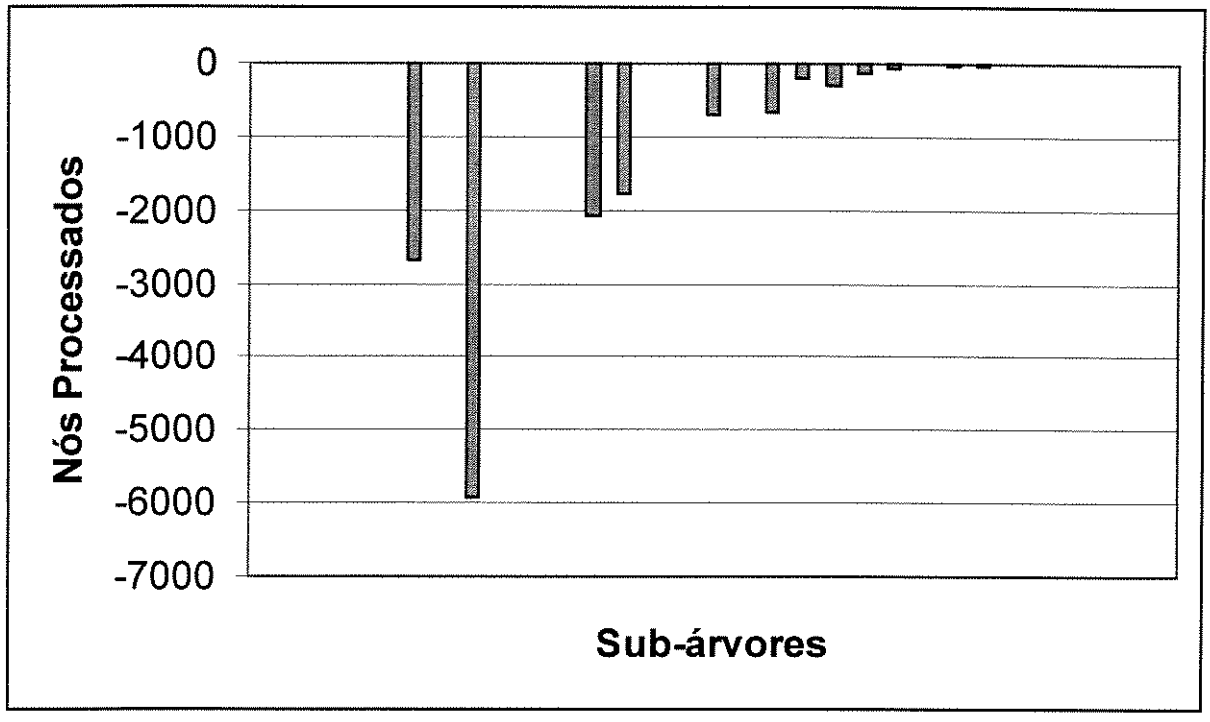
SUB-ÁRVORE	ARCO RETIRADO	IMPLEMENT. SERIAL	1ª IMPLEM. PARALELA	2ª IMPLEM. PARALELA	3ª IMPLEM. PARALELA
1	2	1	1	1	1
2	1	1	1	1	1
3	3	1	1	1	1
4	5	1	1	1	1
5	77	1	1	1	1
6	53	2684	2683	2683	158
7	76	1	1	1	1
8	71	5926	5926	5547	4007
9	75	1	1	1	1
10	6	1	1	1	1

11	7	1	1659	1659	143
12	65	2065	2148	2148	2148
13	72	1787	2587	2587	1905
14	55	1	1	1	1
15	58	1	1816	1816	136
16	74	705	1722	1722	815
17	57	1	1	1	1
18	73	666	1695	1695	775
19	10	214	512	512	269
20	54	286	609	609	338
21	47	141	190	190	168
22	81	76	167	167	147
23	63	9	89	89	89
24	60	47	0	0	0
25	59	17	0	0	0
26	61	1	0	0	0
27	52	16	0	0	0
28	80	8	0	0	0
29	62	4	0	0	0
30	78	2	0	0	0
31	64	1	0	0	0
TOTAL		14667	21813	21434	11108

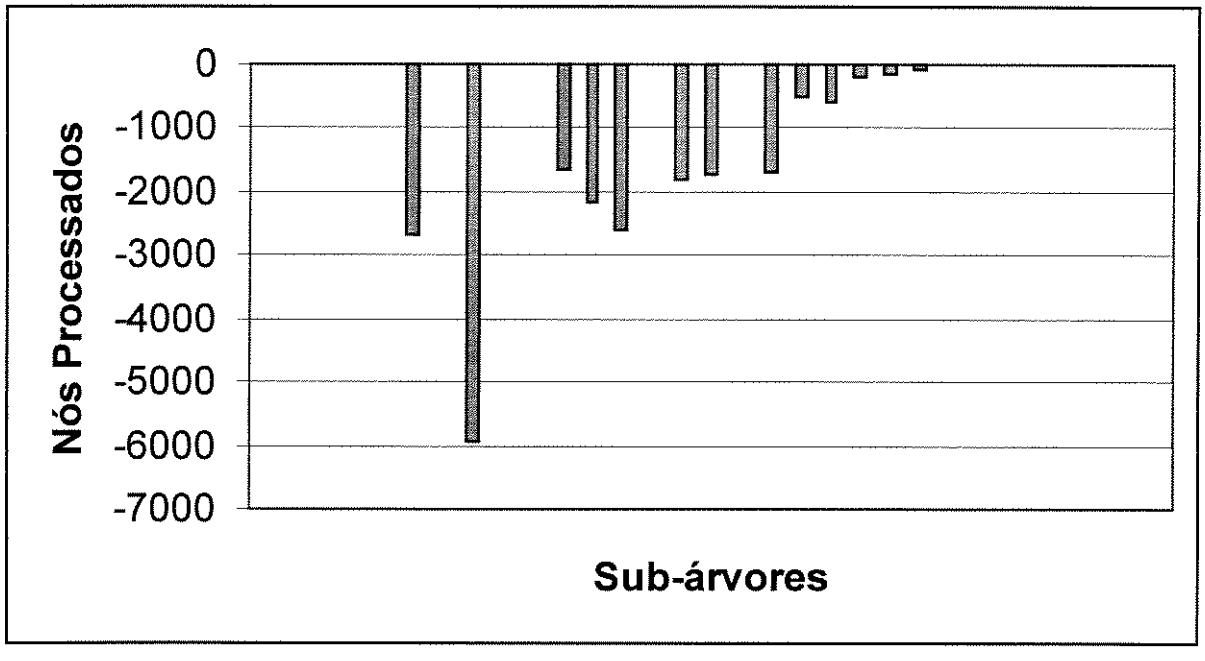
Tab. 5.14 Número de nós processados em cada sub-árvore de busca da rede Bauru 9.

A solução obtida para a rede Bauru 9 é a abertura das chaves 65, 47, 72, 74, 81, 60, 78, 54, 64, ou seja, em relação à configuração inicial, devemos fechas as chaves 63, 71 e 80, e abrir 47, 78 e 54.

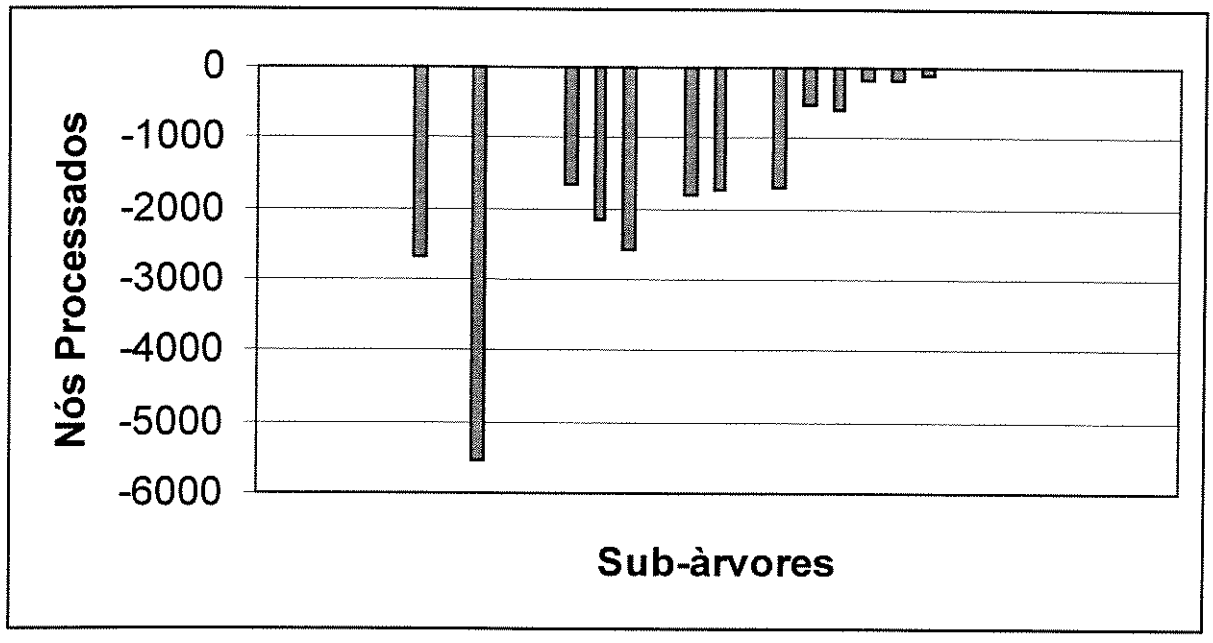
A solução é encontrada pela 12ª sub-árvore.



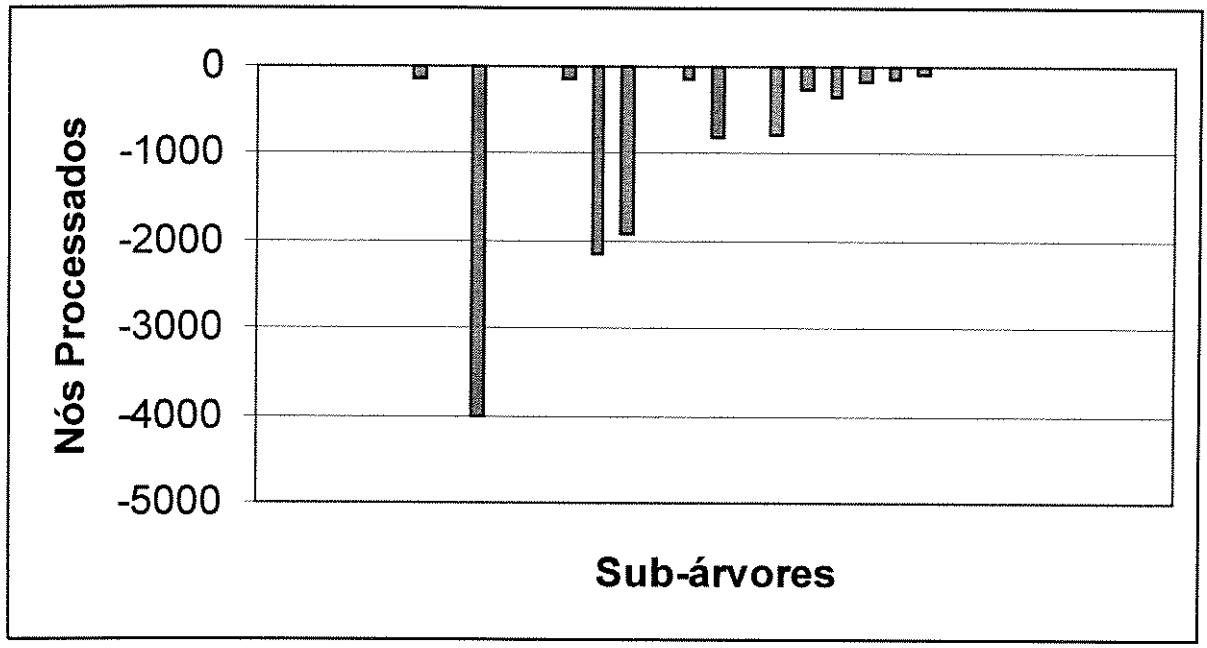
Gráf. 5.9 Perfil da árvore de busca da implementação serial da rede Bauru 9.



Gráf. 5.10 Perfil da árvore de busca da primeira implementação paralela da rede Bauru 9.



Gráf. 5.11 Perfil da árvore de busca da segunda implementação paralela da rede Bauru 9.



Gráf. 5.12 Perfil da árvore de busca da terceira implementação paralela da rede Bauru 9.

	REDE BAURU 9	IMPLEMENT. SERIAL	1ª IMPLEM. PARALELA	2ª IMPLEM. PARALELA	3ª IMPLEM. PARALELA
	NÓS PROC.	14667	21813	21434	11108
Número de Processadores	1	1218,735			
	2		1082,158	1193,689	763,664
	3		594,018	867,516	532,189
	4		721,918	968,031	655,360
	5		686,280	971,511	618,895
	6		507,068	753,339	458,230

Tab. 5.15 Tempos de processamento para a rede Bauru 9.

	REDE BAURU 9	1ª IMPLEM. PARALELA	2ª IMPLEM. PARALELA	3ª IMPLEM. PARALELA
Número de Processadores	2	1,126	1,021	1,596
	3	2,052	1,405	2,290
	4	1,688	1,259	1,860
	5	1,776	1,254	1,969
	6	2,403	1,618	2,660

Tab. 5.16 Valores de *speed-up* para a rede Bauru 9.

	REDE BAURU 9	1ª IMPLEM. PARALELA	2ª IMPLEM. PARALELA	3ª IMPLEM. PARALELA
Número de Processadores	2	0,563	0,510	0,798
	3	0,684	0,468	0,763
	4	0,422	0,315	0,465
	5	0,355	0,251	0,394
	6	0,401	0,270	0,443

Tab. 5.17 Valores de *eficiência* para a rede Bauru 9.

5.3.2 Backtracking Heurístico

Conforme foi exposto no item 3.3.2.2, no backtracking heurístico a função f(n) é incrementada de h(n), onde h(n) é calculado de acordo com (3.2). Torna-se necessário então detectar qual o maior valor de $\underline{w}$ que mantém a obtenção da solução ótima. Os valores de $\underline{w}$ são seguintes:

$\underline{w} = 0{,}09$ para a rede 48;
$\underline{w} = 0{,}89$ para a rede Wu;
$\underline{w} = 0{,}088$ para a rede Bauru 9.

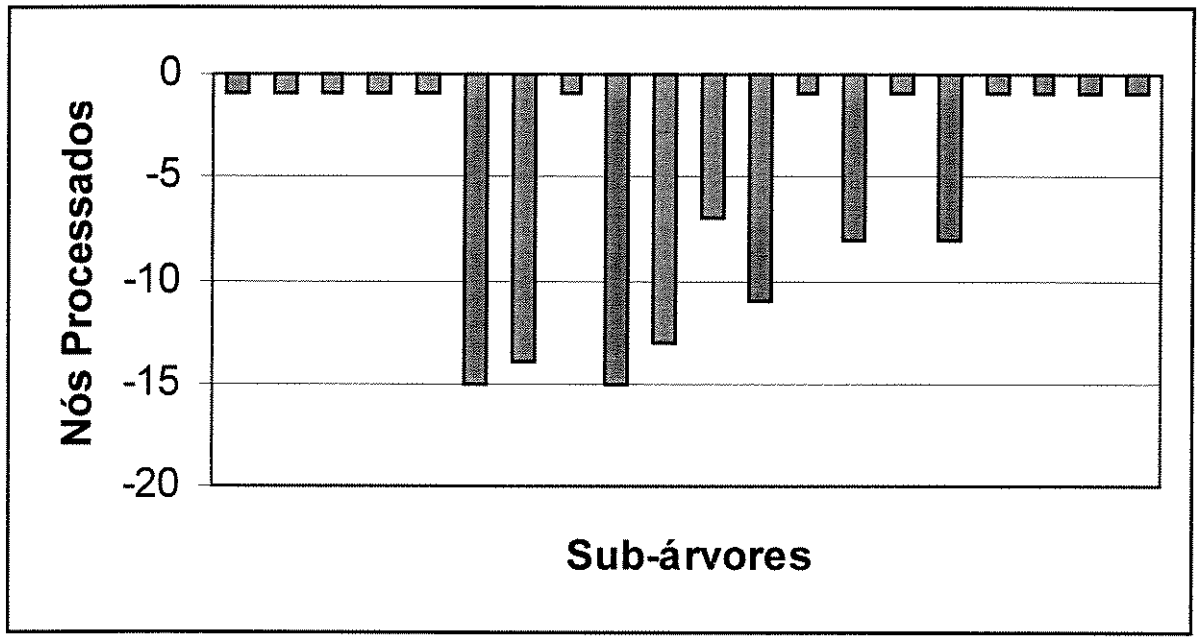
No backtracking heurístico, a pilha ABERTOS possuirá o mesmo conteúdo do backtracking informado após a solução da solução otimista. Conseqüentemente, existem os mesmos números de sub-árvores. Mas o número de nós processados em cada sub-árvore sofre alterações. Vejamos então como isto acontece nas três redes de dados.

5.3.2.1 Rede 48

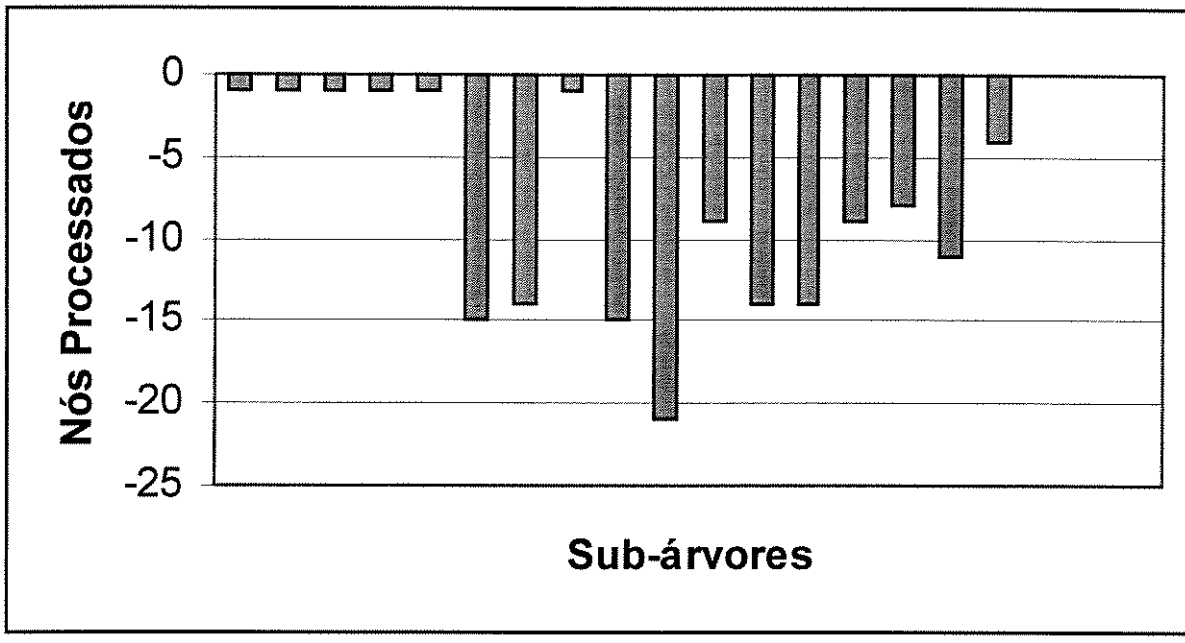
SUB-ÁRVORE	ARCO RETIRADO	IMPLEMENT. SERIAL	1ª IMPLEM. PARALELA	2ª IMPLEM. PARALELA	3ª IMPLEM. PARALELA
1	2	1	1	1	1
2	32	1	1	1	1
3	19	1	1	1	1
4	18	1	1	1	1
5	22	1	1	1	1
6	21	15	15	15	14
7	24	14	14	14	14
8	7	1	1	1	1
9	35	15	15	14	15
10	23	13	21	17	17
11	9	7	9	9	9
12	39	11	14	13	14
13	25	1	14	14	13

14	8	8	9	9	9
15	11	1	8	8	8
16	40	8	11	11	11
17	27	1	4	4	4
18	10	1	0	0	0
19	26	1	0	0	0
20	29	1	0	0	0
TOTAL		103	140	134	134

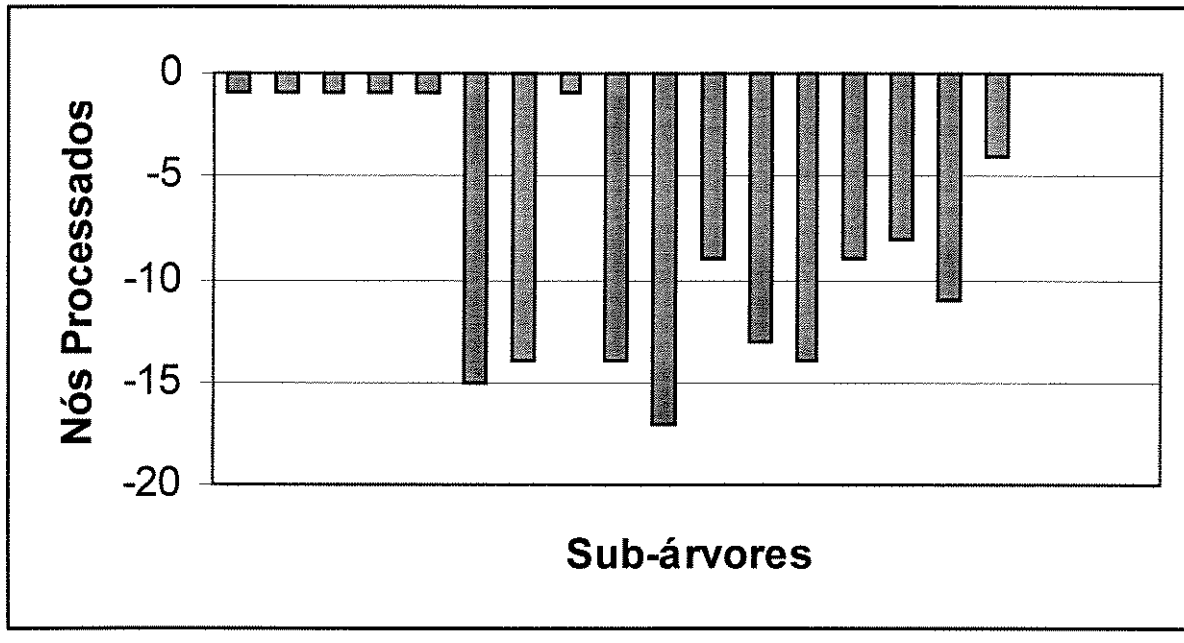
Tab. 5.18 Número de nós processados em cada sub-árvore de busca da rede 48.



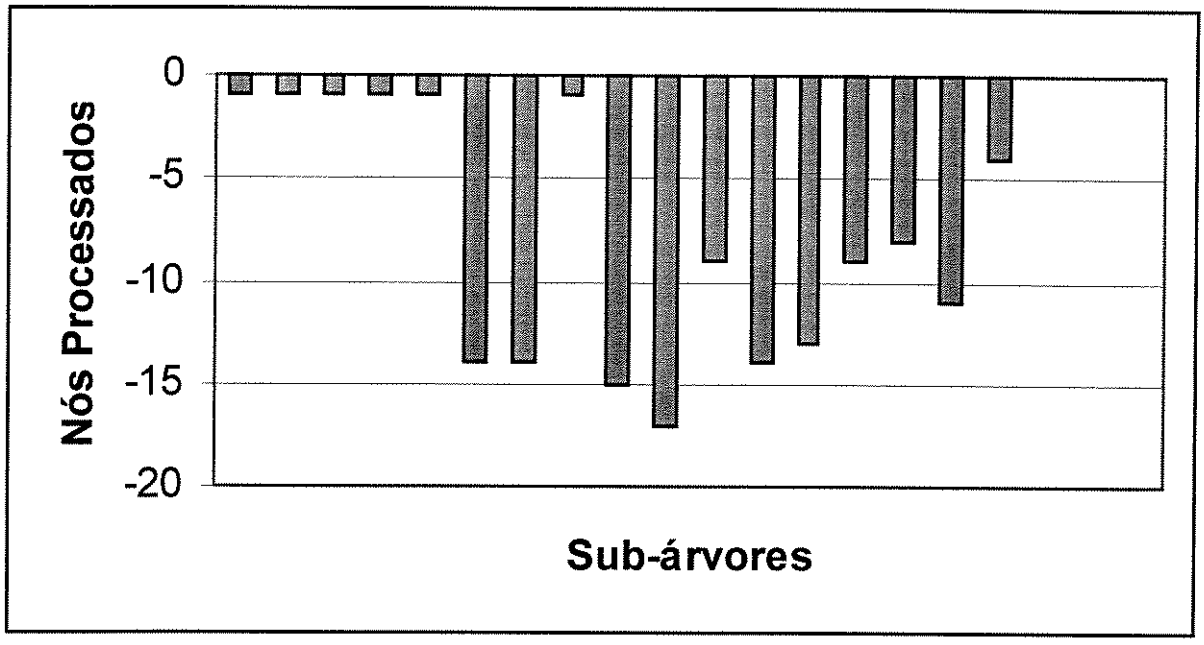
Gráf. 5.13 Perfil da árvore de busca da implementação serial da rede 48.



Gráf. 5.14 Perfil da árvore de busca da primeira implementação paralela da rede 48.



Gráf. 5.15 Perfil da árvore de busca da segunda implementação paralela da rede 48.



Gráf. 5.16 Perfil da árvore de busca da terceira implementação paralela da rede 48.

	REDE 48	IMPLEMENT. SERIAL	1ª IMPLEM. PARALELA	2ª IMPLEM. PARALELA	3ª IMPLEM. PARALELA
	NÓS PROC.	103	140	134	134
Número de Processadores	1	1,536			
	2		3,105	3,768	3,905
	3		2,397	2,789	2,840
	4		1,910	2,478	2,492
	5		1,741	2,190	2,178
	6		1,550	1,909	1,898

Tab. 5.19 Tempos de processamento para a rede 48.

	REDE 48	1ª IMPLEM. PARALELA	2ª IMPLEM. PARALELA	3ª IMPLEM. PARALELA
Número de Processadores	2	0,495	0,408	0,393
	3	0,641	0,551	0,541
	4	0,804	0,620	0,616
	5	0,882	0,701	0,705
	6	0,991	0,805	0,809

Tab. 5.20 Valores de *speed-up* para a rede 48.

	REDE 48	1ª IMPLEM. PARALELA	2ª IMPLEM. PARALELA	3ª IMPLEM. PARALELA
Número de Processadores	2	0,247	0,204	0,197
	3	0,214	0,184	0,180
	4	0,201	0,155	0,154
	5	0,176	0,140	0,141
	6	0,165	0,134	0,135

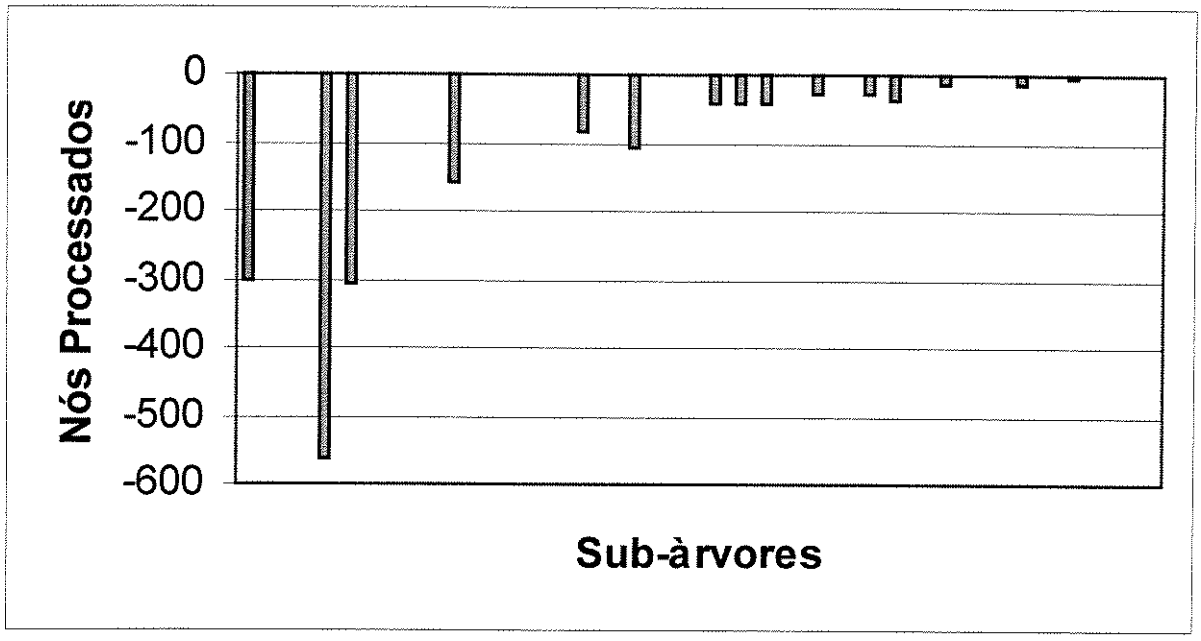
Tab. 5.21 Valores de *eficiência* para a rede 48

5.3.2.2 Rede Wu

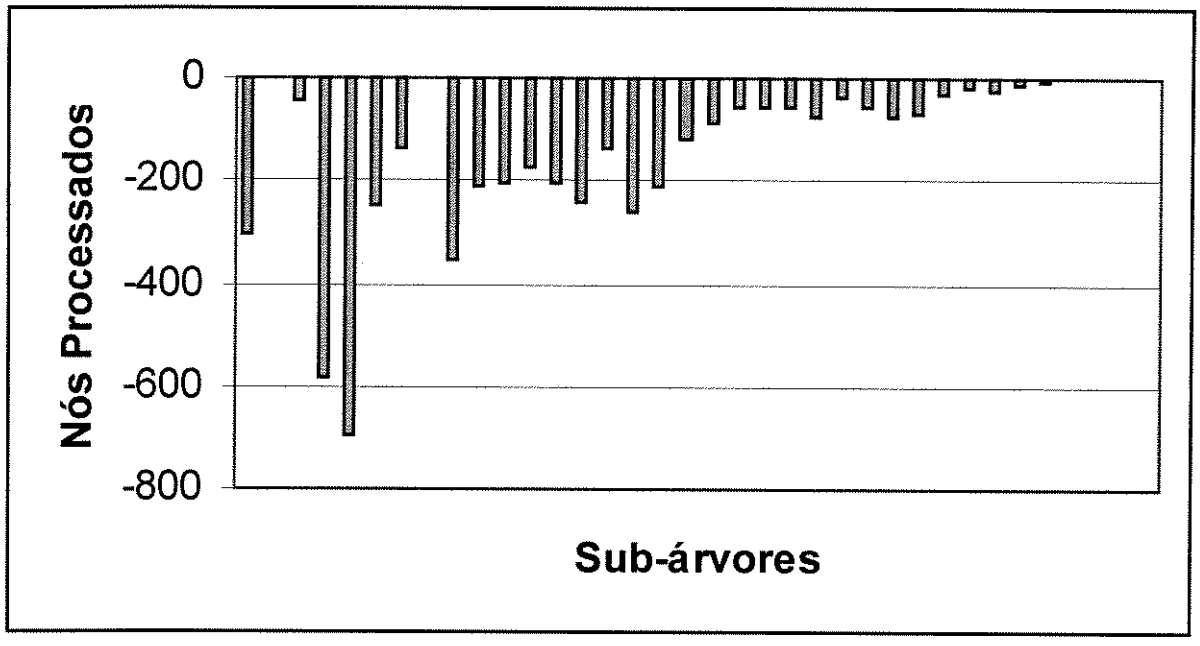
SUB-ÁRVORE	ARCO RETIRADO	IMPLEMENT. SERIAL	1ª IMPLEM. PARALELA	2ª IMPLEM. PARALELA	3ª IMPLEM. PARALELA
1	19	301	301	70	77
2	2	1	1	1	1
3	23	1	46	40	46
4	35	562	586	170	101
5	33	306	694	324	111
6	5	1	251	81	93
7	24	1	137	74	101
8	22	1	1	1	1

9	34	159	352	192	145
10	27	1	211	106	98
11	3	1	202	63	88
12	20	1	175	43	74
13	4	1	206	65	85
14	12	85	240	176	109
15	30	1	139	32	68
16	8	105	263	172	109
17	21	1	209	58	81
18	29	1	115	33	74
19	28	44	87	59	87
20	16	40	56	56	56
21	15	40	56	56	56
22	18	1	55	55	53
23	37	30	77	77	77
24	26	1	39	20	39
25	17	27	56	52	56
26	7	38	75	64	75
27	25	1	66	24	58
28	13	14	30	30	30
29	31	1	16	16	16
30	6	1	24	24	24
31	14	12	12	12	12
32	11	1	4	4	4
33	36	3	0	0	0
34	9	2	0	0	0
35	10	1	0	0	0
36	32	1	0	0	0
TOTAL		1788	4782	2250	2105

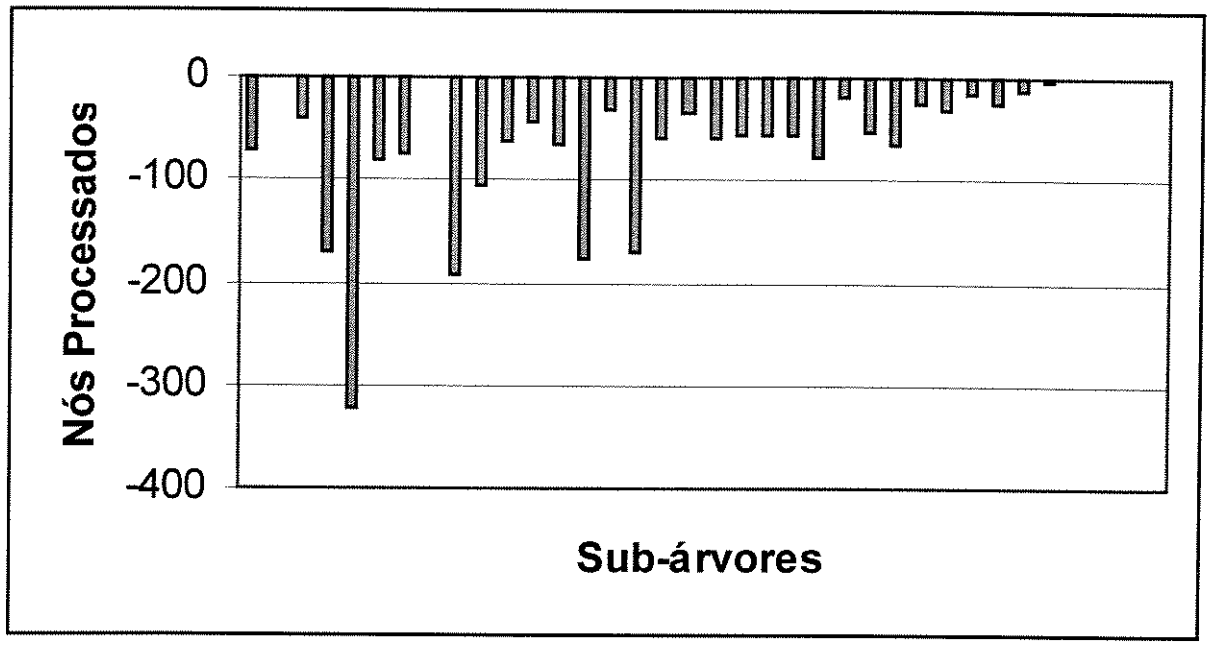
Tab. 5.22 Número de nós processados em cada sub-árvore de busca da rede Wu.



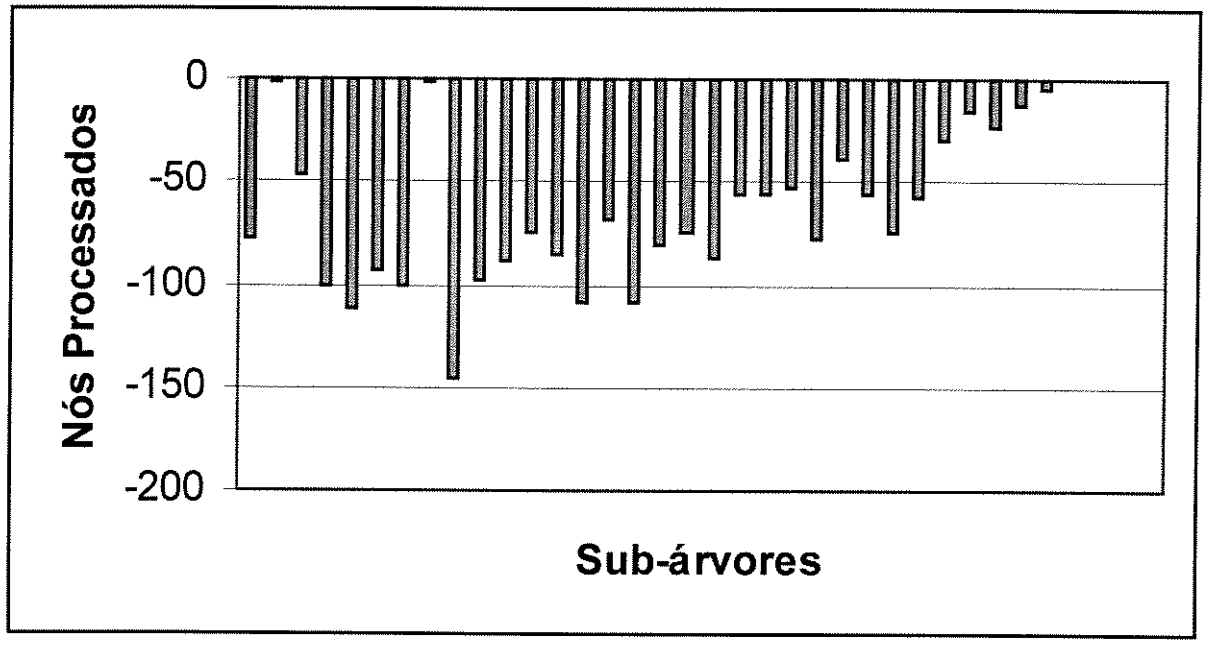
Gráf. 5.17 Perfil da árvore de busca da implementação serial da rede Wu.



Gráf. 5.18 Perfil da árvore de busca da primeira implementação paralela da rede Wu.



Gráf. 5.19 Perfil da árvore de busca da segunda implementação paralela da rede Wu



Gráf. 5.20 Perfil da árvore de busca da terceira implementação paralela da rede Wu

	REDE WU	IMPLEMENT. SERIAL	1ª IMPLEM. PARALELA	2ª IMPLEM. PARALELA	3ª IMPLEM. PARALELA
	NÓS PROC.	1788	4782	2250	2105
Número de Processadores	1	32,463			
	2		70,259	67,631	55,452
	3		44,705	40,832	30,034
	4		34,737	29,108	18,351
	5		32,335	28,166	16,521
	6		30,971	26,592	17,496

Tab. 5.23 Tempos de processamento para a rede Wu.

	REDE WU	1ª IMPLEM. PARALELA	2ª IMPLEM. PARALELA	3ª IMPLEM. PARALELA
Número de Processadores	2	0,462	0,480	0,585
	3	0,726	0,795	1,081
	4	0,935	1,115	1,769
	5	1,004	1,153	1,965
	6	1,048	1,221	1,855

Tab. 5.24 Valores de *speed-up* para a rede Wu.

	REDE WU	1ª IMPLEM. PARALELA	2ª IMPLEM. PARALELA	3ª IMPLEM. PARALELA
Número de Processadores	2	0,231	0,240	0,293
	3	0,242	0,265	0,360
	4	0,234	0,279	0,442
	5	0,201	0,231	0,393
	6	0,175	0,203	0,309

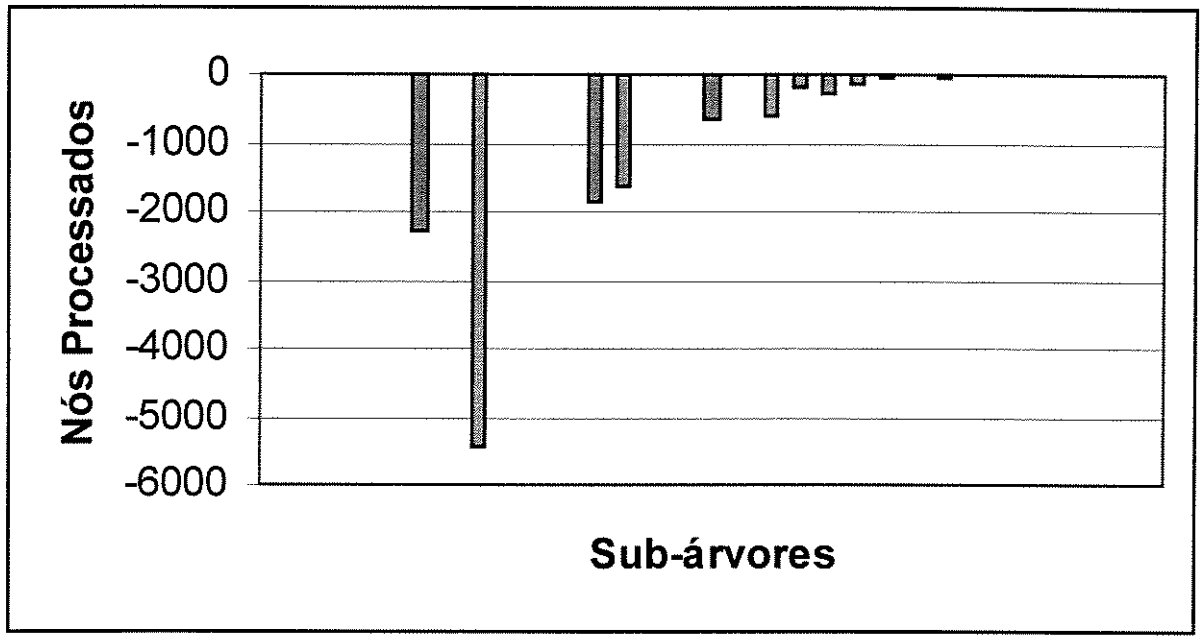
Tab. 5.25 Valores de *eficiência* para a rede Wu.

5.3.2.3 Rede Bauru 9

SUB-ÁRVORE	ARCO RETIRADO	IMPLEMENT. SERIAL	1ª IMPLEM. PARALELA	2ª IMPLEM. PARALELA	3ª IMPLEM. PARALELA
1	2	1	1	1	1
2	1	1	1	1	1
3	3	1	1	1	1
4	5	1	1	1	1
5	77	1	1	1	1
6	53	2259	2259	2259	155
7	76	1	1	1	1
8	71	5445	5445	5051	3477
9	75	1	1	1	1
10	6	1	1	1	1
11	7	1	1627	1627	143
12	65	1876	1954	1954	1954
13	72	1636	2436	2436	1756
14	55	1	1	1	1
15	58	1	1722	1722	145
16	74	668	1704	1704	776
17	57	1	1	1	1
18	73	625	1686	1686	733
19	10	193	507	507	259
20	54	277	607	607	335
21	47	134	190	190	168
22	81	62	167	167	145
23	63	1	88	88	88
24	60	43	0	0	0
25	59	17	0	0	0
26	61	1	0	0	0
27	52	16	0	0	0
28	80	8	0	0	0
29	62	4	0	0	0

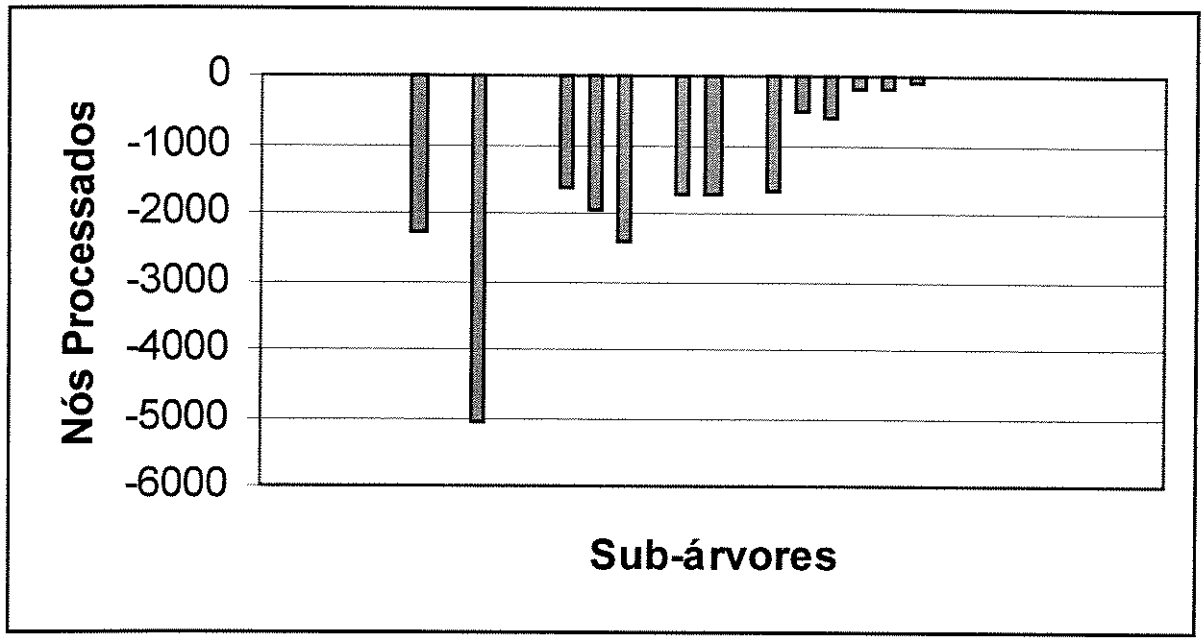
30	78	2	0	0	0
31	64	1	0	0	0
TOTAL		13280	20402	20008	10144

Tab. 5.26 Número de nós processados em cada sub-árvore de busca da rede Bauru 9.

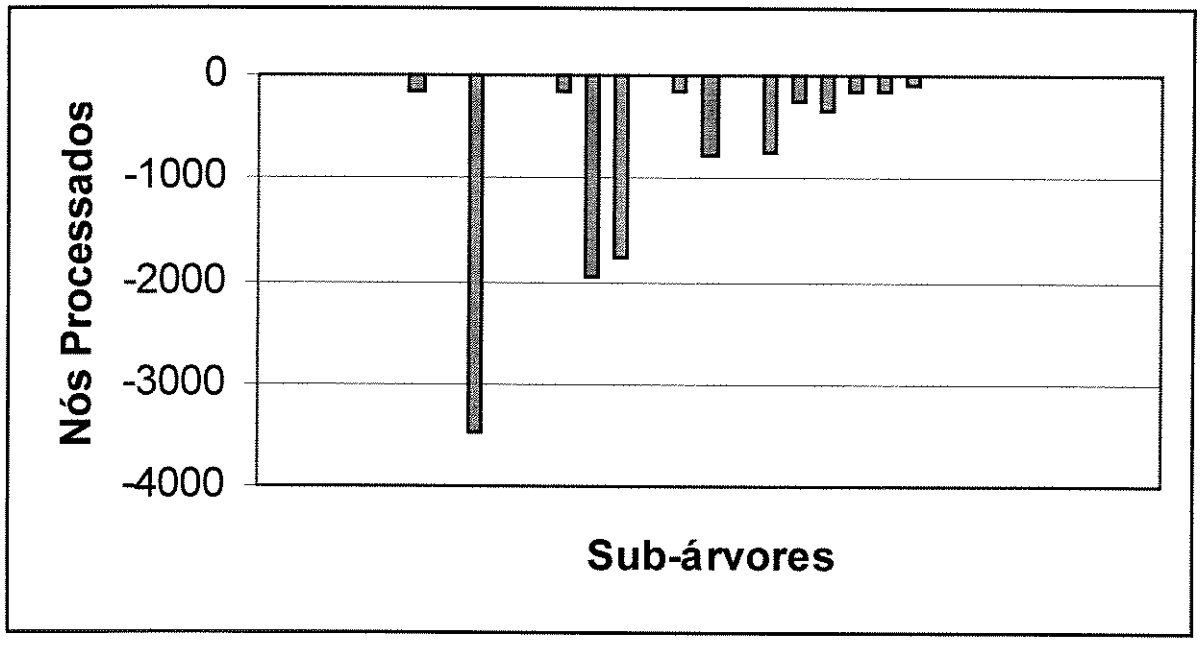


Gráf. 5.21 Perfil da árvore de busca da implementação serial da rede Bauru 9.

Gráf. 5.22 Perfil da árvore de busca da primeira implementação paralela da rede Bauru 9.



Gráf. 5.23 Perfil da árvore de busca da segunda implementação paralela da rede Bauru 9



Gráf. 5.24 Perfil da árvore de busca da terceira implementação paralela da rede Bauru 9

	REDE BAURU 9	IMPLEMENT. SERIAL	1ª IMPLEM. PARALELA	2ª IMPLEM. PARALELA	3ª IMPLEM. PARALELA
	NÓS PROC.	13280	20402	20008	10144
Número de Processadores	1	1070,941			
	2		1085,026	1068,760	687,532
	3		678,632	853,126	465,558
	4		584,798	850,643	579,786
	5		693,377	853,915	551,335
	6		490,942	680,169	408,493

Tab. 5.27 Tempos de processamento para a rede Bauru 9.

	REDE BAURU 9	1ª IMPLEM. PARALELA	2ª IMPLEM. PARALELA	3ª IMPLEM. PARALELA
Número de Processadores	2	0,987	1,002	1,558
	3	1,578	1,255	2,300
	4	1,831	1,259	1,847
	5	1,545	1,254	1,942
	6	2,181	1,575	2,622

Tab. 5.28 Valores de *speed-up* para a rede Bauru 9.

	REDE BAURU 9	1ª IMPLEM. PARALELA	2ª IMPLEM. PARALELA	3ª IMPLEM. PARALELA
Número de Processadores	2	0,494	0,501	0,779
	3	0,526	0,418	0,767
	4	0,458	0,315	0,462
	5	0,309	0,251	0,388
	6	0,364	0,262	0,437

Tab. 5.29 Valores de *eficiência* para a rede Bauru 9.

5.3.3 Análise de Variação de Parâmetros

Expostos os resultados para o backtracking informado e o backtrracking heurístico, é interessante observar como a variação de parâmetros pode afetar estes resultados.

Estes parâmetros são os valores de x (número de nós processados entre duas comunicações em cada sub-árvore de busca na terceira implementação paralela) e $\underline{w}$ (estimativa do quanto a eliminação de cada ciclo incrementa a função objetivo no backtracking heurístico).

5.3.3.1 Variação de x

Sendo x o número de nós processados em cada sub-árvore de busca entre duas comunicações, isto significa que um valor baixo de x resultará em um grande número de comunicações a ser realizada, o que por um lado é bom, pois mantém todos os processos melhor informados sobre a melhor solução ótima existente até o momento, o que melhora a eficiência da poda por dominância, por outro resultará em grande perda de tempo para realizar estas comunicações. Um número muito elevado para x produzirá o efeito inverso. A poda por dominância fica menos inteligente, o que resulta em um maior número de nós a serem processados, mas um pequeno tempo é gasto com comunicação.

Para exemplificar, utilizaremos uma nova rede, a rede Bauru 10. Esta rede consiste da rede Bauru 9 acrescida de um arco conectando dois blocos de carga já existentes. Com um arco a mais e o mesmo número de blocos de carga, torna-se necessário abrir uma chave a mais para obtermos uma configuração radial, portanto passamos a ter uma árvore de busca com profundidade 10. Assim também é possível verificar como varia o tempo de processamento necessário quando aumentamos a profundidade da árvore de busca em uma rede específica.

Os tempos a seguir foram obtidos utilizando-se o backtracking informado e 6 processadores.

x	NÓS PROCESSADOS	Tempo (s)
100	38138	2013,978
500	41116	1981,653
1000	44371	1988,769

Tab. 5.30 Análise de variação de x na rede Bauru 10.

Como pode ser observado na Tab. 5.30, a variação de x afeta consideravelmente o número de nós processados. Com relação aos tempos, embora a variação tenha sido pequena, pode-se notar que o melhor desempenho ocorreu para x = 500, pois enquanto x = 100 gasta mais tempo com comunicação, x = 1000 gasta um pouco mais de tempo por ter maior volume de processamento a realizar. Portanto, existe um número ideal para x em cada caso. Abaixo deste número, o ganho obtido com a eficiência da poda por dominância não compensa o esforço requerido pela comunicação, levando a um aumento no tempo total de processamento; acima deste número, o ganho obtido com a redução do esforço para comunicação não compensa a perda de eficiência da poda por dominância.

5.3.3.2 Variação de w

O valor ótimo de w é aquele que permite processar o menor número possível de nós sem perder a solução ótima. A Tab. 5.31 contém o número de nós processados para alguns valores de w para a rede Bauru 9, e a informação se o ótimo foi mantido ou não.

w	NÓS PROCESSADOS	ÓTIMO
0,08	13577	SIM
0,085	13338	SIM
0,086	13332	SIM
0,087	13315	SIM
0,088	13280	SIM
0,089	13490	NÃO
0,09	13395	NÃO

Tab. 5.31 Análise da variação de w.

Como pode ser observado na Tab. 5.31, quanto maior o valor de w, menor o número de nós processados, pois aumenta o número de nós cuja projeção da função objetivo $f_h(n)$ excede S.

No entanto, ao passarmos w de 0,088 para 0,089 o número de nós processados aumentou. Isto se deve ao fato do ótimo ter sido perdido, pois conseqüentemente passou-se a ter um pior valor de referência para a poda por dominância.

Com o valor ótimo (w = 0,088), é possível obter a mesma solução ótima encontrada por w = 0,08 processando 297 nós a menos, portanto é interessante ajustar w corretamente, e não apenas encontrar um valor para w que mantenha o ótimo. No entanto, esse valor de w é função da distribuição de carga, o que impossibilita a sua reutilização quando esta distribuição de carga tiver sido alterada.

5.4 Comentários, Conclusões e Sugestões para Trabalhos Futuros

A primeira observação é que dentre as versões de implementação utilizadas, a terceira implementação paralela foi destacadamente a melhor. Isto se deve à melhor distribuição de esforços entre os processos, que é conseqüência da existência de comunicação, e ao bom sincronismo obtido com este critério de comunicação. A segunda implementação paralela, apesar de também realizar comunicação obteve resultados piores devido ao critério de comunicação possuir péssimo sincronismo.

Como pode ser observado em todos os gráficos de perfil da árvore de busca, as primeiras sub-árvores são as que processam um maior número de nós, devido ao fato de que ainda não existem muitas configurações de rede repetidas (o que nas implementações em paralelo equivale a uma lista negra pequena). Por sua vez, os processos filhos que são gerados por último sempre processam poucos nós porque já não é possível gerar muitas configurações que não tenham sido processadas (lista negra grande).

Nas versões em paralelo, o processo pai, por ser o que possui a maior lista negra, é sempre um dos primeiros a finalizar, ficando apenas na espera da resposta dos processos filhos. Portanto, o tempo total de processamento fica diretamente relacionado com o tempo de execução dos processos filhos mais demorados. Como, normalmente são poucos os processos filhos que demoram, quando utilizamos números maiores de processadores, alguns executam rapidamente seus processos e ficam sem mais tarefas a realizar enquanto os outros processadores se ocupam com os processos filhos demorados. Isto explica o baixo *speed-up* a má *eficiência* obtida em quase todos os casos. Na rede 48, que propiciou os valores mais baixos de *speed-up*, isto foi conseqüência do tempo necessário para gerar os processos filhos (Tab. 5.2), que é da mesma ordem do tempo do processamento serial, e no processamento paralelo o tempo de processamento acaba ficando menor que o tempo de geração dos processos filhos.

Quando acrescentamos um processador, alteramos a distribuição dos processos filhos entre os processadores e, se por coincidência, os processos mais demorados forem atribuídos a um mesmo processador, este pode ter mais serviço a fazer do que quando tínhamos um processador a menos. Foi o que ocorreu com a rede Bauru 9 ao passarmos de três processadores para quatro, e depois para cinco. Com quatro ou cinco processadores, um deles tinha mais cálculo para fazer do que o processador mais atarefado quando eram três processadores. Isto poderia ser evitado se soubéssemos a priori quais processos são mais demorados. Sabemos que eles estão entre os primeiros processos a serem gerados, mas não sabemos exatamente quais são eles.

Outro detalhe interessante é que os arcos que formam a solução ótima estão sempre no fundo da pilha ABERTOS. Na rede 48, por exemplo, os quatro arcos que formam a solução estão entre os cinco últimos arcos da pilha. Por que isto ocorre é algo que precisa ser melhor investigado, assim como outros critérios para ordenação da pilha, como menor fluxo, por exemplo.

A principal conclusão é que através da comparação dos tempos de processamento de Bauru 9 e Bauru 10 podemos dizer que este método é capaz de solucionar problemas com nível de profundidade até 9, com os recursos computacionais disponíveis neste trabalho, dentro do prazo de

10 minutos considerados adequados para a implementação em tempo real.

Portanto, ainda não é viável resolver uma rede como a de Bauru por completo (a rede completa possui profundidade em torno de 100). Mas isto pode ser realizado através de técnicas de particionamento de redes, pois particionando a rede completa em várias sub-redes, com níveis de profundidade dentro do que se consegue resolver, pode-se processar estas sub-redes em paralelo simultaneamente. Se é possível que a união da solução ótima de cada uma das sub-redes produzirá a solução ótima da rede inteira também é algo que precisa ser melhor investigado.

Outro recurso capaz de aumentar o tamanho da rede que se pode resolver, é aplicar o paralelismo em mais de um nível. Neste trabalho paralelizou-se o primeiro nível da árvore de busca. Com a existência de mais recursos computacionais, pode-se pensar em paralelizar o segundo nível, ou seja, em gerar os processos netos. Isto complica um pouco o algoritmo do ponto de vista da comunicação entre os processos, pois os processos filhos precisariam se comunicar com seus processos netos e com o processo mestre, mas não seria necessário gerar processos netos em todos os processos filhos, e sim apenas nos primeiros processos filhos, onde estão os processos mais demorados, e que justificam a necessidade de geração dos processos netos.

Se na resolução de Bauru 9 o processamento paralelo permitiu converter uma busca de profundidade 9 em 23 buscas de profundidade 8, que demoram cerca de 10 minutos para serem processadas, pode-se pensar em resolver Bauru 10 convertendo uma busca de profundidade 10 em várias buscas de profundidade 9, processadas pêlos processos filhos, e em seguida converter algumas destas buscas de profundidade 9 em várias buscas de profundidade 8 através da geração dos processos netos. A perspectiva é obter a solução também dentro de 10 minutos.

Isto também pode ser estendido para a paralelização do *n-ésimo* nível, ou até onde a complexidade da comunicação permitir.

Referências Bibliográficas

Ahuja, R. K., Magnanti, T. L. & Orlin, J. B. (1993). Network Flows: Theory, Algorithms, and Applications. Prentice Hall, Englewood Cliffs, NJ.

Baran, M. E. & Wu, F. F. (1989). Network reconfiguration in distribution systems for loss reduction and load balancing. IEEE Transactions on Power Delivery 4, 1401–1407.

Bazaraa, M. S., Jarvis, J. J. & Sherali, H. D. (1990). Linear Programming and Network Flows. John Wiley & Sons.

Bertsekas, D. P. & Tsitsiklis, J. N. (1989). Parallel and Distribution Computation: numerical methods, Prentice Hall.

Borozan, V., Rajičić, D. & Ačkovski, R. (1995). Improved method for loss minimization in distribution networks. IEEE Transactions on Power Systems 10, 1420–1425.

Bunch, J. B., Miller, R. D. & Wheeler, J. E. (1982). Distribution system integrated voltage and reactive power control. IEEE Transactions on Power Apparatus and Systems 101, 284–289.

Carneiro da Silva, M. (1990). "Planejamento a longo prazo em sistemas de distribuição de energia elétrica". Tese de Doutorado, Faculdade de Engenharia Elétrica, UNICAMP.

Cavellucci, C. & Lyra, C. (1997). Minimization of energy losses in electric power distribution systems by intelligent search strategies. International Transactions in Operational Research, Vol. 4 N° 1, pp. 23-33.

Civanlar, S., Grainger, J. J., Yin, H. & Lee, S. S. H. (1988). Distribution feeder reconfiguration for loss reduction. IEEE Transactions on Power Delivery 3, 1217–1223.

Geist, A., Beguelim, A., Dongarra, J., Jiang, W., Manchek, R.& Sunderam, V. (1994). PVM 3 User's Guide and Reference Manual.

Glamocanin, V. (1990). Optimal loss reduction of distribution networks. IEEE Transactions on Power Systems 5, 774–782.

Goswami, S. K., Basu, S. K. (1992). A new algorithm for the reconfiguration for distribution feeders for loss minimization. IEEE Transactions on Power Delivery 7, 1484–1491.

Jung, K. H., Kim, H. & Ko, Y. (1993). Network reconfiguration algorithm for automated distribution systems based on artificial intelligence approach. IEEE Transactions on Power Delivery 8, 1933–1941.

Lyra Filho, C. (1984). Contribuição ao Planejamento de Produção de Energia Elétrica em Sistemas de Potência. Tese de Doutorado. Faculdade de Engenharia de Campinas, UNICAMP.

Luemberger, D. G. (1984). Linear and Nonlinear Programming. Addison–Wesley, Reading, MA.

Nara, K., Shiose, A., Kitagawa, M. & Ishihara, T. (1992). Implementation of genetic algorithm for distribution systems loss minimum re-configuration. IEEE Transactions on Power Systems 7, 1044–1051.

Nilson, N. J. (1980). Principles of Artificial Intelligence. Tioga Pub. Co., Palo Alto, CA.

Pearl, J. (1984). HEURISTICS: Intelligent Search Strategies for Computer Problem Solving. Addison–Wesley, Reading, MA.

Sárfi, R. J., Salama, M. M. A. & Chikhani, A. Y. (1996). Distribution system reconfiguration for loss reduction: An algorithm based on network partitioning theory. IEEE Transactions on Power Systems 11, 504–510.

Shirmohammadi, D. & Hong, H. W. (1989). Reconfiguration of electric distribution networks for resistive line losses reduction. IEEE Transactions on Power Delivery 4, 1492–1498.

Taleski, R. & Rajičić, D. (1997). Distribution network reconfiguration for energy loss reduction. IEEE Transactions on Power Systems 12, 398–406.

Taylor, T. & Lubkeman, D. (1990). Implementation of heuristic search strategies for distribution feeder reconfiguration. IEEE Transactions on Power Delivery 5, 239–246.

Wagner, T. P., Chikhani, A. Y. & Hackam, R. (1991). Feeder reconfiguration for loss reduction: an application of distribution automation. IEEE Transactions on Power Delivery 6, 1922–1931.