Universidade Federal Fluminense

MARCELO COSTA PINTO E SANTOS

Algoritmo Dual Ascent Distribuído Aplicado ao Problema de Steiner em Grafos

NITERÓI

2008

MARCELO COSTA PINTO E SANTOS

Algoritmo Dual Ascent Distribuído Aplicado ao Problema de Steiner em Grafos

Tese de Doutorado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Doutor. Área de concentração: Computação Distribuída e Paralela.

Orientadora:

Lúcia Maria de Assumpção Drummond

Universidade Federal Fluminense

NITERÓI

2008

Algoritmo Dual Ascent Distribuído Aplicado ao Problema de Steiner em Grafos

Marcelo Costa Pinto e Santos

Tese de Doutorado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Doutor.

Aprovada por:

Prof. D.Sc. Lúcia Maria de Assumpção Drummond / IC-UFF (Presidente)

Prof. Ph.D. Valmir Carneiro Barbosa / COPPE-UFRJ

Prof. Ph.D. Virgílio Augusto Fernandes de Almeida / DCC-UFMG

Prof. D.Sc. Anna Dolejsi Santos / IC-UFF

Prof. D.Sc. Eduardo Uchoa Barboza / Eng.Produção-UFF

Niterói, novembro de 2008.

Aos nossos pais, imortais em nossos corações.

Agradecimentos

Agradeço a meu pai, Raul, cujo incentivo foi importante na decisão de iniciar este trabalho e infelizmente saiu de nosso convívio durante seu desenvolvimento. A minha mãe, Erika, que teve de se contentar com menos apoio meu do que merecia, durante um período de difíceis e dolorosas mudanças.

A minha esposa e filhos, Beatriz, Bruno e Daniel, que suportaram bravamente meu mau humor quando as coisas demoravam a se acertar.

A minha orientadora, Lúcia Drummond, pela dedicação e tempo dispendidos neste trabalho e ao Professor Eduardo Uchoa, cuja colaboração foi decisiva no sucesso desta empreitada.

A meus amigos da UFF, sempre prontos a ajudar, meus colegas do Departamento de Informática do CTU/UFJF, que sempre me apoiaram, muitas vezes com sacrifício pessoal e a Professora Elisabeth, do departamento de Códigos e Linguagens, pela ajuda no correto uso de nossa língua.

A CAPES, pelo apoio financeiro.

Resumo

Apresentamos neste trabalho uma versão distribuída para o algoritmo Dual Ascent, cuja versão seqüencial foi proposta por [Wong 1984] e tem se mostrado uma eficiente heurística para solução do Problema de Steiner em Grafos. Estamos interessados no desenvolvimento de boas heurísticas para este problema porque o roteamento *multicast* pode ser modelado como um problema desta classe. Várias aplicações como jogos *on-line*, bate-papo em tempo real, vídeo conferência e banco de dados distribuído, que dependem de roteamento *multicast* eficiente estão sendo mais utilizados com a expansão da internet. O algoritmo apresentado provê meios para obtenção de uma boa solução para o problema com grafos orientados e não orientados, além de fornecer um limite inferior de excelente qualidade, tipicamente 3% abaixo do ótimo, que pode ser utilizado para avaliação de soluções obtidas com o Dual Ascent ou quaisquer outras heurísticas. Apresentamos também adaptações para a variação do problema com limitação de saltos, utilizado para modelar situações em que limitações de QoS sejam importantes e para a versão on-line, quando devemos adaptar a solução à inclusão e exclusão de nós no grupo de terminais.

Abstract

In this work we present a distributed version for Dual Ascent algorithm whose sequential version was proposed by [Wong 1984] and has been recognized as an efficient heuristic for the solution for the Steiner Problem in Graphs. We are interested in the development of good heuristics for this problem because multicast routing can be modeled as a problem of this class. Several emerging Internet applications such as on-line games, real-time chat programs, video conferencing, distributed databases, depend on efficient multicast routing. The algorithm gives good solutions for both directed and non-directed versions of the problem, and presents a good lower bound, typically less than 3% below optimum. This lower bound can be used to evaluate the solution obtained with Dual Ascent and other heuristics. We also present adaptations to the problem with hop constraint, important for situations where QoS is required and for the on-line problem, where the solution must be adapted to inclusions and exclusions in the terminals group.

Palavras-chave

1. Algoritmos distribuídos
2. *Multicast*
3. Problema de Steiner em Grafos

Glossário

ANSI	:	*American National Standards Institute*
IC	:	Instituto de Computação
MPI	:	*Message Passing Interface*
MPICH	:	Implementação do MPI com código aberto do *Argonne National Laboratory*
QoS	:	Qualidade de Serviço (*Quality of Service*)
UFF	:	Universidade Federal Fluminense

Problemas:

SPG	:	Problema de Steiner em Grafos (*Steiner Problem in Graphs*)
SPDG	:	Problema de Steiner em Digrafos (*Steiner Problem in Directed Graphs*)
cSPG	:	Problema de Steiner em Grafos com Restrições (*Constrained Steiner Problem in Graphs*)
cSPDG	:	Problema de Steiner em Digrafos com Restrições (*Constrained Directed Steiner Problem in Directed Graphs*)
hcSPG	:	Problema de Steiner em Grafos com Restrição ao Número de Saltos (*Hop-Constrained Steiner Problem in Graphs*)
oSPDG	:	Problema de Steiner em Grafos *on-line* (*On-Line Steiner Problem in Graphs*)
MST	:	Árvore Geradora Mínima (*Minimun Spanning Tree*)
hcMST	:	Problema da Árvore Geradora Mínima com Restrição ao Número de Saltos (*Hop-Constrained Minimum Spanning Tree*)

Símbolos:

V	:	Conjunto de vértices de um grafo
E	:	Conjunto de arestas de um grafo não orientado
A	:	Conjunto de arcos de um grafo orientado
D	:	Diâmetro de um grafo
$G = (V, E)$	:	Grafo de uma instância para o SPG
$G_d = (V, A)$	:	Grafo de uma instância para o SPDG
$G' = (V', E')$	:	Grafo solução para um SPG
$G'_d = (V', A')$	:	Grafo solução para um SPDG
S	:	Solução parcial para um problema de Steiner
T	:	Conjunto de terminais para um problema de Steiner
r	:	Nó raiz para um SPDG
t	:	Um terminal de um problema de Steiner
v	:	Um vértice qualquer de um grafo
e	:	Uma aresta de um grafo não orientado
a	:	Um arco de um grafo orientado
c	:	Custo de uma aresta ou arco de um grafo
$\bar{c}$	:	Custo reduzido de uma aresta ou arco de um grafo
LI, gb	:	Limite inferior, *global bound*
pb	:	Limite inferior parcial, *partial bound*
LS	:	Limite superior
Otm	:	Solução ótima para um problema de Steiner
G_s	:	Grafo de Saturação
R	:	Conjunto de nós de uma componente fortemente conexa de um grafo de saturação
W(R)	:	Conjunto de nós que atingem R por um arco saturado
$\delta^-(W)$	:	Conjunto de arcos entrantes a W
Δ	:	Menor custo reduzido entre arcos entrantes a um grafo de saturação
δ	:	Menor custo reduzido entre arcos entrantes a parte de um grafo de saturação
Q	:	Conjunto de nós atingidos pela raiz em um grafo de saturação

Algoritmos:

3BASIC	:	Algoritmo seqüencial exato, específica para o caso de 3 terminais
ADH	:	*Average Distance Heuristic*
ARIES	:	*A Rearrangeable Inexpensive Edge-based on-line Steiner Algorithm*
BSMA	:	*Bounded Shortest Multicast Algorithm*
CHINS	:	*Cheapest Insertion*
CHINS-3	:	Heurística seqüencial baseada em caminhos mínimos que utiliza o 3BASIC
DA	:	Dual Ascent
DAH	:	DA, algoritmo para distância mínima e PrimSPH no grafo em G_s
DAHs	:	DAH com limite para o numero de saltos
DCSP	:	*Delay Constrained Shortest Path Algorithm*
DDMC	:	*Destination Driven Multicast Routing*
EBA	:	*Edge Bounded Algorithm*
Kruskal-SPH	:	Heurística distribuída para o SPG baseada no algoritmo de [Kruskal 1956] para MST
OSPF	:	*Open Shortest Path First*
Prim-SPH	:	Heurística distribuídoa para o SPG baseado no algoritmo de [Prim 1957] para MST
QDMR	:	*QoS Dependent Multicast Routing*
RIP	:	*Routing Information Protocol*
SPH	:	Algoritmo para cálculo da distância mínima e Prim-SPH
SPHs	:	SPH com limite para o numero de saltos
VTDM	:	*Virtual Trunk Dynamic Algorithm*

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

O Problema de Steiner em Grafos (SPG - *Steiner Problem in Graphs* [Hakimi 1971, Voss 1992, Winter 1987]) pode ser definido como segue:

> Dado um grafo não orientado $G = (V, E)$, onde V é o conjunto de seus vértices e E é o conjunto de suas arestas, com pesos positivos associados às arestas e um conjunto de terminais $T \subseteq V$, encontrar um subgrafo conexo de G de custo mínimo que contenha todos os vértices de T.

Em outras palavras, achar uma árvore de custo mínimo contendo todos os terminais e possivelmente alguns nós não terminais, chamados "nós de Steiner". A versão do mesmo problema para grafos direcionados (SPDG - *Steiner Problem in Directed Graphs*) é o caso em que $G_d = (V, A)$ é um grafo direcionado onde V representa o conjunto de seus vértices e A o conjunto de seus arcos, e existe um terminal especial $r \in T$ chamado *raiz*. O problema consiste em encontrar uma arborescência de custo mínimo que contenha caminhos de r para cada um dos demais terminais. Ambos os problemas (SPG e SPDG) são NP-Completos [Garey e Johnson 1979], o que nos motiva a pesquisar heurísticas que encontrem boas soluções, consumindo recursos computacionais que viabilizem sua utilização prática.

Várias aplicações de importância crescente nos dias atuais como: teleconferência, vídeo sob demanda, jogos em tempo real e bancos de dados distribuídos, dependem de transmissões entre um subconjunto de nós de uma rede. Isto é chamado de *multicast* ou *broadcast seletivo*. O roteamento das mensagens para uma sessão *multicast* consiste em encontrar caminhos entre os participantes desses subconjuntos. Esse problema é freqüentemente modelado como um SPG de acordo com [Novak et al. 2001b] e [Oliveira e Pardalos 2005].

Quando a fonte das informações a serem transmitidas encontra-se em um único nó, como no vídeo sob demanda, por exemplo, e/ou os custos associados aos canais de comunicação são assimétricos, o problema é melhor modelado como SPDG.

Algoritmos seqüenciais possuem a grande desvantagem de necessitar que todos os dados do grafo sejam concentrados em um único nó e, se imaginarmos a Internet como sendo o grafo do problema, tal desvantagem se torna importante. Principalmente neste contexto, quando tratamos de redes com muitos nós espalhados sobre uma grande extensão territorial, é de grande interesse o estudo de algoritmos distribuídos, em que o processo de roteamento possa ser realizado com cada participante tomando decisões baseadas em suas informações locais e, no máximo, em informações simples recebidas de seus vizinhos imediatos, sem a necessidade de concentrarmos todas as informações sobre a rede em um único ponto.

O algoritmo apresentado neste trabalho é uma versão distribuída do seqüencial chamado Dual Ascent (DA), introduzido por [Wong 1984], e que apresenta as seguintes características:

- Exaustivos experimentos computacionais sobre as principais classes do problema apresentadas na literatura mostram que o DA leva a resultados melhores que os principais algoritmos conhecidos [Poggi de Aragão et al. 2001, Voss 1992, Werneck 2001, Polzin e Vahdati 2001, Poggi de Aragão e Werneck 2002];

- O Dual Ascent é um exemplo do que mais tarde seria conhecido como *algoritmo primal-dual* [Goemans e Williamson 1996]. Isto pode ser interpretado como trabalharmos com a versão dual de um problema de programação linear. Na prática, isto significa que o Dual Ascent não retorna somente uma solução, mas também uma garantia de qualidade desta solução, na forma de um *Limite Dual* ou *Limite Inferior*, que é um valor menor ou igual ao ótimo do problema. Por exemplo, podemos obter uma solução de valor 1.000 e um limite inferior de valor 980. Isto significa que a solução encontrada não é pior do que 2% em relação ao ótimo, pois sabemos, com certeza, que o ótimo possui custo igual ou superior a 980. A qualidade dos limites inferiores fornecidos pelo Dual Ascent é notável, normalmente não mais do que 3% abaixo do ótimo. Por esta razão, o Dual Ascent é uma peça chave para os melhores algoritmos exatos para o SPG [Polzin e Vahdati 2001, Poggi de Aragão et al. 2001]. O limite inferior pode ter grande importância também para qualquer heurística. Se um usuário não ficar satisfeito com a garantia de qualidade de uma determinada solução, ele pode re-executar o algoritmo utilizado, caso este não seja determinís-

tico, ou tentar outra heurística, possivelmente melhor adaptada àquela instância em particular;

- Utilizável para o SPDG e o SPG. O algoritmo apresentado trabalha sobre o problema direcionado (SPDG). No entanto, com operações simples podemos transformar um SPG em SPDG e depois fazer a conversão inversa, para derivarmos uma solução obtida no SPDG para o SPG. Podemos transformar o SPG em SPDG tomando um terminal aleatoriamente como raiz e criando arcos simétricos de igual valor em ambos os sentidos de uma aresta. A transformação inversa consiste em criar arestas onde houver arcos, independente do sentido;

- Vários algoritmos para o roteamento *multicast* baseiam-se no conhecimento prévio do caminho de menor custo de um nó até todos os demais do grafo. Alguns autores [Bauer e Varma 1996, Novak et al. 2001a, Gatani et al. 2005] argumentam que algoritmos de camadas inferiores (RIP, OSPF) já mantêm tabelas de roteamento com essas informações, que podem ser utilizadas sem custo por seus algoritmos. Não consideramos essa argumentação completamente satisfatória, pois se, no futuro, for desejada a troca do protocolo que mantém a tabela utilizada e a nova versão não necessite dessa tabela, o algoritmo de roteamento *multicast* terá de ser substituído também ou, no mínimo, construir sua própria tabela de distâncias mínimas. Além disso, os custos das conexões utilizadas pelos protocolos de rede podem não ser os mais adequados para se considerar no *multicast*. Na maioria dos casos práticos, o custo de um caminho para o RIP ou OSPF está associado ao número de saltos intermediários até atingir a rede de destino, como se o custo de cada conexão fosse sempre unitário. É verdade que alguns algoritmos possibilitam ao administrador da rede escolher uma métrica diferente, mas nada garante que mesmo essa métrica menos usual atenderá satisfatoriamente ao roteamento *multicast*, e estaríamos limitando a abrangência do *multicast* a redes sobre as quais possuímos alguma ingerência administrativa. Em outras palavras, provavelmente não seremos capazes de, na prática, convencer o administrador de um *backbone* estrangeiro a alterar a métrica utilizada em sua rede para que o *multicast* funcione satisfatoriamente. O Dual Ascent apresentado não pressupõe o conhecimento prévio de caminhos mínimos.

A complexidade de pior caso do algoritmo distribuído proposto é $O(|T|.|V|^2)$ para o tempo global de execução. Por "tempo global de execução", entendemos o tamanho da maior cadeia de envio de mensagens com relação de causalidade e que, portanto, não poderiam ser enviadas em paralelo. $O(|T|.|V|^2)$ é a complexidade para o número

de mensagens trocadas, e $O(|V|)$ para o processamento local em um nó arbitrário. Por "tempo local" entendemos o número de operações executadas entre o recebimento de uma mensagem (ou a inicialização espontânea), a realização do processamento decorrente desse evento, até que o processador entre em estado de espera pela próxima mensagem (ou termine a execução do algoritmo).

Uma alternativa à execução do Dual Ascent distribuído, seria a eleição de um nó líder que resolvesse localmente o problema e então enviasse a solução para todos os demais nós. Seriam gastos $O(|V|.|E|)$ mensagens e $O(|V|)$ tempo para concentrar todas as informações relevantes sobre o grafo no líder, a complexidade de tempo para resolver localmente o problema utilizando do Dual Ascent seqüencial é $O(|E|.min(|E|, |T|.|V|))$ [Hwang et al. 1992]. Considerando estas complexidades e as necessidades de memória no nó líder, a abordagem distribuída se mostra como uma alternativa atraente.

Resultados computacionais de um protótipo implementado em linguagem C e MPI são apresentados para instâncias direcionadas e não direcionadas. Implementamos também a melhor heurística distribuída conhecida para o problema, o Prim-SPH [Bauer e Varma 1996, Rugelj e Klavzar 1997]. O Dual Ascent apresentou resultados médios melhores que o Prim-SPH e esforço computacional de mesma ordem de grandeza.

Uma variação importante do SPDG, é o problema com **limitação de saltos**. Nesse caso, o número de nós intermediários nos caminhos entre o nó raiz e cada um dos terminais não pode ser superior a um determinado limite. Essa variação é freqüentemente utilizada na modelagem de problemas que envolvem a distribuição de multimídia. Isto se deve ao fato de o número de saltos intermediários ser uma boa aproximação para o atraso (*delay*) associado a um caminho, e essa medida ter grande importância nas transmissões de multimídia.

Outro caso importante sob o ponto de vista prático, é a versão *on-line* do problema. Nesse caso, nós comuns podem se transformar em terminais e terminais podem se tornar nós comuns com o passar do tempo. O algoritmo deve adaptar os caminhos utilizados para atingir cada novo conjunto de terminais, provavelmente utilizando estruturas que permaneçam válidas apesar das mudanças, diminuindo o custo das adaptações.

O restante deste trabalho está organizado da seguinte forma: o Capítulo 2 contextualiza o problema, abordando as principais heurísticas seqüenciais e distribuídas para o SPG/SPDG. O Capítulo 3 descreve o algoritmo original, seqüencial, no qual se baseia nossa versão distribuída. No Capítulo 4 aparece a principal contribuição deste trabalho, a descrição do Dual Ascent distribuído. Uma análise teórica da complexidade pode ser

encontrada no Capítulo 5 e, no Capítulo 6, encontramos resultados experimentais. O Capítulo 7 discute a adaptação para o caso com limitação de saltos. O Capítulo 8 mostra como o algoritmo pode ser utilizado para o caso *on-line*. Conclusões podem ser apreciadas no último Capítulo, 9.

Capítulo 2

Problema de Steiner em Grafos

O Problema de Steiner em Grafos (SPG - *Steiner Problem in Graphs*) consiste em dados:

- o grafo $G = (V, E)$ onde V é o conjunto de vértices e E, o conjunto de arestas;
- um custo positivo c_e associado a cada aresta;
- um conjunto $T \subseteq V$, de terminais;

encontrar

$G' = (V', E')$

tal que

- $T \subseteq V'$;
- $\sum_{e \in E'} c_e$ seja mínimo.

A versão direcionada do problema (SPDG) é definida de forma semelhante. Dados:

- o grafo direcionado $G_d = (V, A)$ onde V é o conjunto vértices e A o conjunto de arcos;
- um terminal raiz $r \in V$;
- um custo positivo c_a associado a cada aresta;
- um conjunto $T \subseteq V$, de terminais;

encontrar

$$G'_d = (V', A')$$

tal que

- existam caminhos de r a todo $t \in T$;
- $\sum_{a \in A'} c_a$ seja mínimo.

O SPG é, na realidade, uma variação do "Problema de Steiner Euclidiano" que trata da conexão de um conjunto de pontos de um plano. Um caso especial desse problema foi estudado por Fermat, em 1640:

> Encontrar um ponto que minimize a soma das distâncias entre três outros pontos de um plano.

A generalização do problema, extrapolando o plano cartesiano e considerando um número arbitrário de pontos a conectar (conjunto de terminais), foi estudada por **Jacob Steiner** (1796-1863).

O primeiro algoritmo de que se tem notícia para o problema foi formulado por [Melzak 1961].

Apesar de NP-Completo [Karp 1972], portanto, sem possibilidade de solução exata em tempo polinomial, soluções com complexidade de pior caso exponencial mais eficientes vêm sendo pesquisadas. Algoritmos de enumeração implícita do tipo *Branch-and-Bound* como o *Branch-and-Cut* e o *Branch-and-Ascent* descritos em [Werneck 2001] são exemplos dessas iniciativas.

Neste trabalho, estamos mais interessados em heurísticas eficientes para o problema. Neste capítulo, serão abordadas as principais heurísticas seqüenciais e distribuídas para o SPG/SPDG encontradas na literatura.

2.1 Algumas Heurísticas Seqüenciais

Muito já se sabe sobre heurísticas seqüenciais para o SPG/SPDG. Excelentes resumos podem ser encontrados em [Voss 1992, Oliveira et al. 2005]. Heurísticas para solução do SPG, normalmente baseiam-se em duas idéias básicas:

- inserção sucessiva de arestas a uma solução inicial até que se conectem todos os terminais;
- criação de múltiplas soluções parciais, que vão sendo unidas até que todos os terminais encontrem-se em um mesmo componente único.

A adaptação da primeira classe de algoritmos para o SPDG é geralmente possível, tomando-se o terminal raiz como ponto inicial de construção da solução. Já a segunda classe de heurísticas, que tenta unir componentes parciais entre si, é de difícil adaptação para esse caso mais genérico, pois grupos que não contenham o nó raiz não têm como decidir o sentido mais adequado dos arcos a serem utilizados em sua conexão pois não sabem, de antemão, qual dos dois será conectado primeiro ao terminal raiz.

Uma idéia simples para se conseguir uma solução para o problema seria a construção de uma árvore geradora mínima, posteriormente passando-se a uma fase de "poda", onde seriam retirados os nós não terminais de grau um. Assim seriam excluídos os nós que não fossem necessários para a conexão de terminais. Essa abordagem apresenta qualidade de solução ruim [Voss 1992], além de envolver na construção da árvore geradora mínima todos os nós do grafo, o que pode ser intolerável, caso o número de terminais seja muito menor do que o número de nós do grafo, o que nos parece uma situação prática bastante freqüente se lembrarmos que o grafo pode ser uma representação da Internet.

Outra abordagem simples é a escolha de um terminal inicial aleatório para o caso SPG e o terminal raiz para o caso SPDG, e a inclusão na solução de todos os caminhos mínimos entre o terminal inicial e os demais. Infelizmente esse algoritmo também apresenta soluções pobres [Voss 1992], no entanto, serve de base para outras idéias com melhores resultados práticos.

A idéia de redução do grafo original a um grafo transformado, onde são representados apenas os terminais como vértices e os caminhos mínimos entre eles como arestas foi explorada por diversos autores [Choukhmane 1978, Kou et al. 1981, Plesnik 1981]. Aplica-se um algoritmo para construção de árvore geradora mínima sobre o grafo transformado. A volta ao grafo original é trivialmente conseguida com a substituição das arestas no grafo transformado pelo caminho mínimo. No entanto a solução resultante pode conter ciclos com arestas desnecessárias, portanto, uma fase posterior de limpeza é necessária. Para o caso específico do problema com 3 terminais, existe uma variação enumerativa dessa idéia que obtém soluções exatas em tempo polinomial [Chen 1983]. Esse algoritmo é chamado 3BASIC e é utilizado em diversas outras heurísticas como, por exemplo, na CHINS-3,

descrita por [Voss 1992]:

1. aplica-se o 3BASIC a todos os possíveis grupos de três terminais do grafo, tomando-se como solução inicial S o grupo que apresente menor custo;
2. descobre-se o terminal t com menor distância ao conjunto S. Suponhamos, sem perda de generalidade, que o terminal de S mais próximo de t seja t_1;
3. desconecte t_1 dos demais terminais de S e utilize o 3BASIC para conectar os os dois conjuntos resultantes a t, produzindo uma nova solução parcial S;
4. retorne ao passo dois enquanto S não contiver todos os terminais.

[Takahashi e Matsuyama 1980] sugerem o algoritmo CHINS (*Cheapest Insertion*). De forma semelhante ao algoritmo de Prim [Prim 1957] para MST, um terminal é escolhido como ponto de partida e é conectado, utilizando-se o caminho mínimo, ao terminal que estiver mais próximo. Estes dois terminais conectados formam uma solução parcial que cresce durante a execução do algoritmo. A cada rodada de crescimento um terminal é conectado à solução parcial através de seu caminho mínimo. Os terminais são conectados um a um, em ordem crescente de distância para a solução parcial, até que não restem terminais a conectar. Para o SPG, a escolha do terminal pelo qual se iniciará o processo tem influência na qualidade da solução final. Portanto, pode-se imaginar a execução do algoritmo para cada uma das possibilidades de início, tomando-se a melhor solução encontrada. Para o SPDG, o terminal inicial será sempre o nó raiz.

O algoritmo ADH (*Average Distance Heuristic*) [Rayward-Smith 1983] considera inicialmente cada terminal como uma solução parcial vazia de arestas. Repete-se os passos abaixo, até que todos os terminais estejam conectados:

1. descubra o nó v (não necessariamente terminal) mais central em relação grafo, ou seja, que possui menor distância média para todas as soluções parciais;
2. inclua os caminhos mínimos entre v e as duas soluções parciais mais próximas de v, fundindo-as em uma única.

A idéia básica pode ser utilizada em conjunto com o 3BASIC tomando os 3 componentes S_1, S_2 e S_3 mais próximos do ponto central v e verificando qual das possibilidades de combinação com 3 elementos entre S_1, S_2, S_3 e v produz a melhor solução, esta se transformando em um componente único para a próxima iteração do algoritmo.

Uma heurística baseada em uma espécie de "contração" do grafo, sob alguns aspectos semelhante ao Dual Ascent, é descrita em [Plesnik 1981]. Abaixo, resumimos a idéia apresentada no artigo.

1. descubra c_t, custo da menor aresta incidente a cada terminal;
2. reduza os custos $c_{i,j}$ das arestas adjacentes a terminais para:
   - $c'_{i,j} = \max\{0, c_{i,j} - 2 * c_t\}$ se $i \wedge j$ forem terminais;
   - $c'_{i,j} = c_{i,j} - c_t$, caso contrário;
3. resolva o SPG nos subgrafos formados pelos conjuntos conexos C_i, formados pelos arcos com custos reduzidos a zero, utilizando CHINS. Os arcos incluídos nas soluções desses subproblemas devem ser incluídos na solução do SPG original;
4. se ainda existirem terminais a conectar, reduza os conjuntos C_i a terminais e retorne ao passo 1.

O algoritmo apresentado por [Shaikh e Shin 1997], chamado DDMC (*Destination Driven for Low-Cost Multicast*), é inspirado nos algoritmos [Prim 1957] e [Dijkstra 1959] que se baseiam na medida do custo acumulado para se atingir um determinado nó. Cada nó informa os custos dos caminhos que dispõe para todos os demais nós aos seus vizinhos. Os vizinhos somam o custo para atingir o remetente da mensagem a cada rota recebida, guarda a rota se esta for a melhor conhecida até o momento. Ao descobrir uma nova rota, um nó a informa aos seus vizinhos, de tal forma que as informações sobre rotas vão se propagando pela rede, sempre se acumulando os custos. O algoritmo proposto é semelhante, no entanto, os terminais sempre informam custo zero para todas as rotas de que dispõe. Portanto, caminhos que passem por terminais são preferidos a outros, pois parecerão mais baratos, o que poderia resultar em uma boa solução para o SPG/SPDG.

O Algoritmo Dual Ascent original, descrito em [Wong 1984], poderia ser encaixado nesta seção, mas como constitui a base para a principal contribuição deste trabalho, optamos por destacá-lo no Capítulo 3.

2.2 Heurísticas Distribuídas

Algumas heurísticas para o problema de Steiner [Chen et al.1993, Kompella et al. 1993a] baseiam-se na execução de algoritmos distribuídos para a construção de árvores

geradoras mínimas [Gallager et al. 1983]. Esses algoritmos usualmente constroem uma árvore geradora mínima envolvendo todos os vértices do grafo e depois eliminam as arestas e nós desnecessários para que todos os terminais sejam atingidos, como no caso seqüencial. Essa abordagem simples pode levar a resultados ruins [Doar e Leslie 1993], além de apresentarem um problema grave: as técnicas para construção de árvores geradoras mínimas envolvem todo o grafo e, portanto, mesmo que o número de elementos do conjunto de terminais seja pequeno em relação ao número de vértices da instância, todo o grafo será envolvido na operação de roteamento *multicast*. É altamente desejável que o esforço computacional em algoritmos para rede concentre-se nos nós diretamente envolvidos no problema ou no seu entorno, e que esse trabalho diminua quando os nós relevantes se encontrem topologicamente concentrados. Portanto, técnicas mais sofisticadas, que apresentem melhores resultados, com menores custos computacionais, são desejadas.

O algoritmo Prim-SPH (SPH-*Shortest Path Heuristic*) [Bauer e Varma 1996, Rugelj e Klavzar 1997] baseia-se no CHINS e, como este, inicia a construção da árvore de distribuição por um terminal arbitrário para o SPG e pela raiz para o SPDG. A cada ciclo, o terminal mais próximo é conectado à árvore parcial, utilizando-se o caminho de custo mínimo entre o terminal e a árvore parcial. Desta forma, uma árvore única vai sendo construída até que todos os terminais sejam atingidos. O algoritmo trabalha em ciclos de *convergecast/broadcast*. Cada nó informa em *convergecast* qual o terminal mais próximo ao ramo da árvore nele enraizado. Quando o *convergecast* atinge o terminal inicial, ele sabe qual o terminal mais próximo de toda a solução parcial, e dispara um *broadcast* informando o próximo terminal a ser conectado e a partir de qual nó. Mais recentemente [Novak et al. 2001a] apresenta melhoria nos resultados práticos, sem conseguir alterar as complexidades analíticas desse algoritmo. Ele evita os ciclos de *convergecast/broadcast* utilizando uma tabela de distâncias mínimas que é transmitida a cada nó incluído na árvore. Ao receber a tabela, um nó a atualiza caso conheça rotas mais curtas para algum terminal. Desta forma, o terminal recém conectado recebe, no instante da conexão, uma tabela com todas os caminhos mínimos atualizados e tem condições de decidir qual o próximo terminal a ser conectado e a partir de qual nó. Se a origem da próxima conexão for ele mesmo, o processo pode ser iniciado imediatamente; caso contrário, a tabela tem de ser transmitida para o nó a partir do qual se inicia o caminho mínimo. As complexidades desses algoritmos, tanto para o tempo quanto para o número de mensagens enviadas é $O(|T|.|V|)$. Entendemos "tempo" como o comprimento da maior cadeia de mensagens com relação de causalidade, ou seja, maior cadeia onde uma mensagem é enviada em decorrência do recebimento de outra.

Outro algoritmo distribuído, chamado Kruskal-SPH [Bauer e Varma 1996], é baseado no algoritmo para árvores geradoras mínimas de Kruskal [Kruskal 1956]. Nessa abordagem, pares de terminais se unem através de caminhos mínimos, formando-se várias árvores parciais em paralelo. Essas árvores parciais vão se unindo progressivamente, também com a utilização dos caminhos de menor custo, até que uma árvore única, conectando todos os terminais, tenha sido construída. A complexidade de tempo deste processo é $O(D.|V|)$ e de mensagens é $O(|T|.|V|)$, sendo D o diâmetro do grafo. [Singh e Vellanki 1998] conseguiram melhorar a complexidade de mensagens para $O(|V| \log |T|)$.

Outra abordagem que apresenta resultados semelhantes é o ADH (*Average Distance Heuristic*) [Gatani et al. 2005, Gatani et al. 2006]. Como o Kruskal-SPH, esse algoritmo cria várias árvores parciais em paralelo, cada terminal dando origem a uma árvore no início da execução. O ADH descobre nós que possuam menor distância média a um grupo de árvores parciais, podendo esses nós centrais serem ou não terminais, participarem ou não de alguma árvore parcial. Descoberto o nó com menor distância média, o algoritmo conecta as duas árvores parciais mais próximas a esse nó, utilizando caminhos mínimos. A complexidade em tempo do ADH é $O(D.|T|)$ e em mensagens é $O(|E| + |T|.|V|)$.

Todos esses algoritmos (Prim-SPH, Kruskal-SPH e ADH) assumem que cada nó conhece o caminho mínimo para cada um dos demais nós do grafo. Se não for esse o caso, a computação distribuída destas distâncias poderá acrescentar uma complexidade de mensagens de $O(|E|.|V|)$ e uma complexidade em tempo de $O(|V|)$ [Segall 1983].

Devemos observar também que algoritmos que criam várias árvores parciais em paralelo (Kruskal-SPH e ADH) não são facilmente adaptáveis ao SPDG e, portanto, não são os mais adequados para comparações com o DA. No caso direcionado, o problema não é conectar todos os terminais entre si, e sim, conectar todos os terminais não raiz, ao terminal raiz. Portanto, não temos qualquer pista sobre o melhor sentido de conexão entre dois terminais t_1 e t_2, se $t_1 \rightarrow t_2$ ou inversamente $t_2 \rightarrow t_1$ para construirmos árvores parciais. Por outro lado, a adaptação dos algoritmos que constroem árvores únicas, como o Prim-SPH, são mais facilmente adaptáveis ao SPDG, pois ao iniciarmos o processo pelo terminal raiz, sempre saberemos o sentido dos arcos a utilizar na conexão de um terminal não raiz.

Apesar de [Shaikh e Shin 1997] não apresentarem detalhadamente uma versão distribuída para seu algoritmo, a tarefa não é difícil pois o processo se baseia fortemente no algoritmo de Dijkstra, que é distribuído por natureza. As complexidades do DDMC são $O(|E|.|V|)$ para número de mensagens e $O(|V|)$ para tempo.

A complexidade dos principais algoritmos citados encontra-se resumida na Tabela 2.1:

Algoritmo	Complexidade de Tempo	Complexidade de Mensagens
Prim-SPH	$O(\|T\|.\|V\|)$	$O(\|T\|.\|V\|)$
Kruskal-SPH	$O(D.\|V\|)$	$O(\|V\| \log \|T\|)$
ADH	$O(D.\|T\|)$	$O(\|E\| + \|T\|.\|V\|)$
DDMC	$O(\|V\|)$	$O(\|E\|.\|V\|)$

Tabela 2.1: Complexidades dos principais algoritmos distribuídos da literatura

2.3 Conclusão

Neste capítulo definimos o problema de Steiner em suas versões direcionada e não direcionada e posicionamos o leitor historicamente neste problema que ocupa a mente de pesquisadores há mais de 350 anos. Como o problema de Steiner é comprovadamente NP-Completo [Karp 1972], é de grande interesse a pesquisa de heurísticas que busquem soluções de boa qualidade em tempo razoável. Mostramos diferentes abordagens para a obtenção de boas soluções, presentes nos principais algoritmos heurísticos conhecidos até o momento.

Capítulo 3

Algoritmo Dual Ascent Seqüencial

O algoritmo Dual Ascent [Wong 1984] foi proposto para resolver o SPDG mas pode ser adaptado para o SPG. Neste algoritmo associa-se a cada arco um **custo reduzido** não negativo $\bar{c}_a$, inicialmente igual ao custo original do arco. Custos reduzidos vão sofrendo subtrações durante a execução do algoritmo até que eventualmente atinjam zero. Denominamos arcos com custos reduzidos iguais a zero de **arcos saturados**. Esses arcos saturados formam um **grafo de saturação** $G_S = (V, A(\bar{c}))$, onde $A(\bar{c}) = \{a \in A : \bar{c}_a = 0\}$.

Por definição, todos os custos originais são maiores que zero, logo, o grafo de saturação inicialmente não contém arcos.

Seja R um conjunto de nós de uma componente fortemente conexa de G_S, ou seja, um conjunto maximal que contenha um ciclo passando por todos os nós em R. Chamaremos este conjunto de **componente raiz** se:

1. R contém pelo menos um terminal;
2. R não contém r (raiz);
3. não existe terminal $t \notin R$ que atinja R por um caminho em G_S;

Dado um componente raiz R, definimos:

- $W(R) \supseteq R$ como o conjunto de nós que atinjam R por um caminho em G_S, incluindo os próprios vértices de R;
- $\delta^-(W)$ conjunto de arcos entrantes a $W(R)$ (arcos cujo nó de origem não pertença a $W(R)$ e nó de destino pertença a $W(R)$).

Feitas essas definições, passamos ao algoritmo:

Algoritmo: Dual Ascent
$\bar{c}_a \leftarrow c_a$, para todo $a \in A$;
$LI \leftarrow 0$;
Enquanto (existem componentes raízes em G_S) {
Escolha um componente raiz R;
$W \leftarrow W(R)$;
$\Delta \leftarrow \min_{a \in \delta^-(W)} \bar{c}_a$;
$\bar{c}_a \leftarrow \bar{c}_a - \Delta$, para todo $a \in \delta^-(W)$;
$LI \leftarrow LI + \Delta$;
}
Saída: G_S e LI

Inicialmente cada terminal, exceto o nó raiz, corresponde a um componente raiz. A cada iteração, o algoritmo escolhe um componente raiz R, determina o custo do menor arco entrante à $W(R)$, que chamaremos Δ, subtrai esse valor de todos os arcos entrantes a $W(R)$ e acrescenta Δ ao limite inferior parcial para o problema (LI). Pelo menos um arco, o de menor custo, será saturado a cada iteração. Algumas saturações reduzem o número de componentes raízes, criando caminhos em G_S da raiz para o componente raiz (violando a segunda condição para existência do componente raiz), ou criando caminhos em G_S entre componentes raízes, neste caso, diminui o número de componentes com a união entre dois componentes raízes previamente existentes.

A ordem de escolha dos componentes raízes durante a execução do algoritmo influi no seu resultado. [Werneck 2001] estuda diversos critérios para a ordem de escolha dos componentes raízes:

- **random**: o componente é escolhido aleatoriamente. Todos os componentes têm igual probabilidade de escolha a cada iteração;
- **first**: uma lista de vértices é mantida e percorrida sempre na mesma ordem, partindo-se do início. O primeiro vértice encontrado na lista que faça parte de um componente, determina seu componente como o escolhido para a iteração;
- **circular**: semelhante ao anterior, porém iniciando-se a busca a partir do vértice posterior ao escolhido na iteração anterior. Quando o fim da lista é atingido, retorna-se ao início;

- **minarcs/maxarcs**: seleciona-se o componente que possua o menor/maior número de arcos entrantes;
- **minvalue/maxvalue**: seleciona-se o componente que possua o menor/maior arco entrante mínimo, ou seja, o componente que possibilite o menor/maior acréscimo ao limite inferior parcial;
- **minsaturated/maxsaturated**: o componente com menor/maior número de arcos saturados é selecionado.

O critério para escolha do componente raiz a cada iteração que apresentou melhor qualidade de solução é o **minarcs** [Werneck 2001], seguido pelo **circular** e o **minsaturated**. A presença de dois métodos que minimizam a velocidade de crescimento dos componentes entre os melhores (**minarcs** e **minsaturated**) sugere que quanto mais lentamente o algoritmo saturar os arcos do grafo, melhor resultado obterá, obviamente maior vai ser também o tempo de execução. As melhores soluções alcançadas pelo método **circular** em relação ao **first** confirmam a hipótese. O **first** também tende a terminar o algoritmo mais rapidamente, já que privilegia o crescimento de um componente, tornando-o progressivamente maior que as demais, provavelmente com mais arestas entrantes que terão seus custos reduzidos em uma mesma iteração. O método **circular**, que tende a balancear o tamanho das componentes, apresenta melhores resultados.

Supondo o grafo finito, fatalmente chegaremos a um ponto em que não existirão mais componentes raízes. Neste momento, a soma de todos os valores Δ computados em cada iteração será um limite inferior (LI) para o problema e G_S conterá pelo menos um caminho direcionado de r para todos os demais terminais. Portanto, conterá pelo menos uma solução para o SPDG. G_S conterá também muitos arcos redundantes, desnecessários na solução do SPDG. Wong propõe o seguinte procedimento para a limpeza de G_S e a obtenção da solução final para o SPDG:

1. Seja Q o conjunto de nós que podem ser atingidos pela raiz. Construa uma arborescência de custo mínimo no subgrafo G_S induzido por Q;
2. Remova dessa arborescência todas as folhas que não sejam terminais. Repita esse passo enquanto existirem folhas não terminais.

[Polzin e Vahdati 2001, Poggi de Aragão et al. 2001] descobriram que a execução do CHINS [Takahashi e Matsuyama 1980] sobre G_S é muito rápida (pois G_S é usualmente

esparso) e leva à soluções que são até mesmo melhores do que as obtidas pelo método de Wong. De qualquer forma, a saída do algoritmo será a solução obtida dessa limpeza, juntamente com o limite inferior que dará uma garantia de qualidade para essa solução.

Os custos reduzidos obtidos pelo Dual Ascent podem ser ainda bastante úteis. Seja LS (Limite Superior) o custo da solução encontrada. Pode-se provar [Werneck 2001] que todos os arcos a tais que $LI + \bar{c}_a \geq LS$ não podem pertencer a qualquer solução com custo menor que LS e, conseqüentemente, não podem pertencer à solução ótima. Isso significa que todos os arcos que possuam custo reduzido maior ou igual a $LS - LI$ podem ser eliminados de qualquer busca por soluções de custo mais baixo. Esse processo de redução da instância é chamado de **eliminação por custos reduzidos**.

Exaustivas avaliações de desempenho foram realizadas utilizando as principais instâncias para referência (*benchmarks*) da literatura [Koch et al. 2000], com número de arestas variando entre 625 e 100.000. A Tabela 3.1 traz um resumo dos resultados. A coluna *Garantia* é a diferença percentual média entre a solução encontrada e o correspondente limite inferior. Em 66 das 214 instâncias testadas, esta diferença foi igual a zero, provando ter sido encontrada a solução ótima. A coluna *Redução* é o número médio percentual de arcos eliminados por custos reduzidos.

Classe	Número de instâncias	Garantia (média %)	Redução (média %)	Nr. de ótimos encontrados
OR-Library	57	0.82	83.8	37
VLSI	77	1.19	47.9	15
Incidence	80	2.77	51.0	14
Total	214	1.68	58.7	66

Tabela 3.1: Avaliação experimental do Dual Ascent Seqüencial [Poggi de Aragão et al. 2001].

Apesar de o algoritmo alcançar boas garantias na prática (cada garantia é válida somente para aquela instância em particular), não se consegue provar que essa garantia esteja afastada do ótimo de um fator constante [Williamson 2002]. Isso é interessante, porque se prova [Goemans e Williamson 1996] que uma variação desse algoritmo para grafos não direcionados é 2-aproximada, ou seja, existe uma garantia analítica de que a solução não apresentará custo superior ao dobro do ótimo. Infelizmente essa variação apresenta resultados muito inferiores na prática.

Podemos acompanhar nas figuras 3.1 a 3.11 um pequeno exemplo de execução passo a passo do Dual Ascent seqüencial. Em todas as figuras, representamos os terminais por

quadrados e nós comuns por círculos; o terminal raiz encontra-se hachurado. Os arcos não saturados são representados por linhas descontínuas, com a indicação na figura de seu custo reduzido. Os arcos entrantes ao componente raiz escolhido para crescimento em uma determinada iteração são destacados traçando-se a semi-reta que os representa de forma contínua. Arcos saturados também são representados por semi-retas contínuas, sem a indicação de seu custo reduzido já que, por definição, o custo reduzido de arcos saturados é sempre igual a zero.

3.1 Exemplo

Na Figura 3.1 (cortesia da apresentação *Recent Advances on the Practical Solution of the Steiner Problem in Graphs* - Eduardo Uchoa, 2002), observamos o grafo inicial com um terminal raiz, três outros terminais (t_1, t_2 e t_3) e quatro nós comuns. O componente raiz formado pelo terminal t_1 é selecionado e o custo do menor de seus arcos entrantes (2) é acrescido ao limite inferior.

Na Figura 3.2, podemos observar que o valor somado ao limite inferior foi subtraído dos arcos entrantes ao primeiro componente raiz escolhido, incluindo nele um nó comum. A operação se repete para o componente agora composto por dois nós, o custo do menor arco entrante (3) é subtraído de todos os demais, saturando mais um arco, como pode ser observado na Figura 3.3.

O processo se repete nas figuras seguintes com a escolha dos componentes raízes iniciados pelos terminais t_2 (Figura 3.3), t_3 (Figura 3.4), t_2 (Figura 3.5) e novamente t_1 (Figura 3.6).

Na Figura 3.7, existem vários arcos entrantes de mesmo custo, todos foram saturados na mesma iteração. A saturação da Figura 3.8 cria um caminho de arcos saturados entre os terminais t_2 e t_3. Por essa razão, os componentes iniciados por esses terminais não mais podem existir separadamente, se fundindo em um único, utilizado no ciclo da Figura 3.9, que cria um caminho de arcos saturados entre o terminal raiz e todos os demais. Portanto, não existem mais componentes raízes e o algoritmo termina, com o limite inferior final em 20. Na Figura 3.10, podemos observar o conjunto de arcos saturados, cuja limpeza gera uma solução para o problema (Figura 3.11) de custo 23.

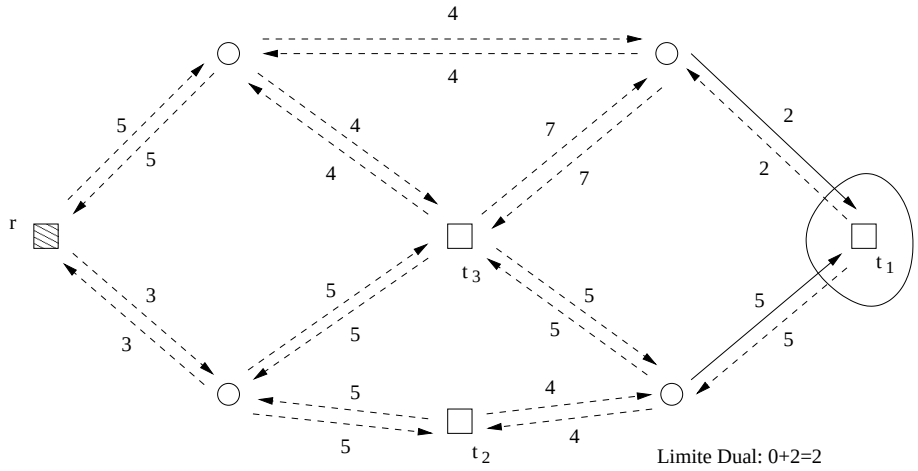

Figura 3.1: Exemplo de execução do DA seqüencial. O componente raiz do terminal t_1 é escolhido na primeira iteração.

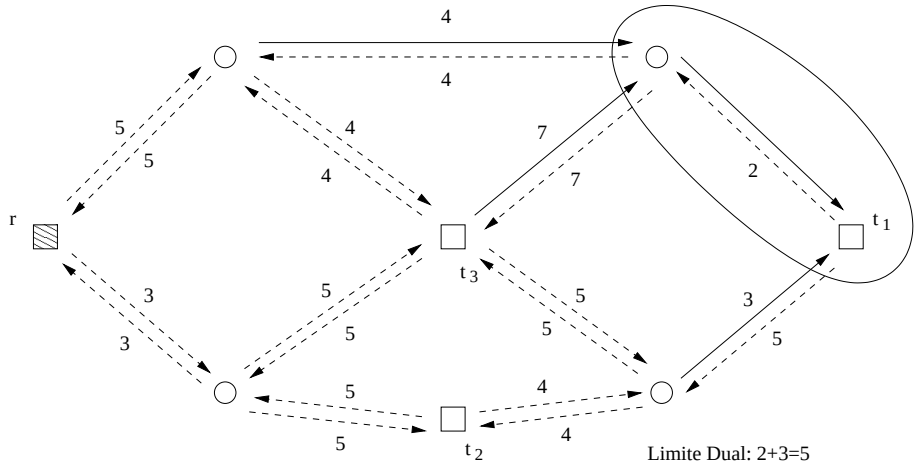

Figura 3.2: Exemplo de execução do DA seqüencial. Arco saturado causa a expansão do componente raiz associado ao terminal t_1.

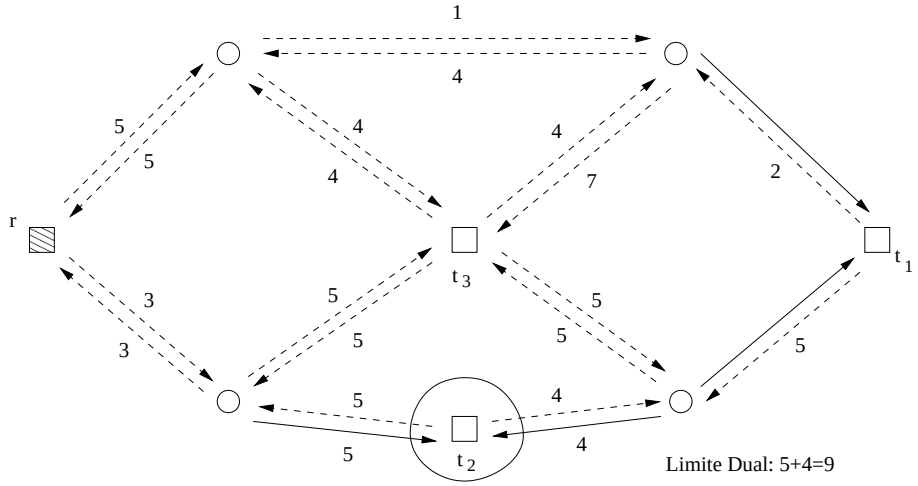

Figura 3.3: Exemplo de execução do DA seqüencial. Componente referente ao terminal t_2 é selecionado.

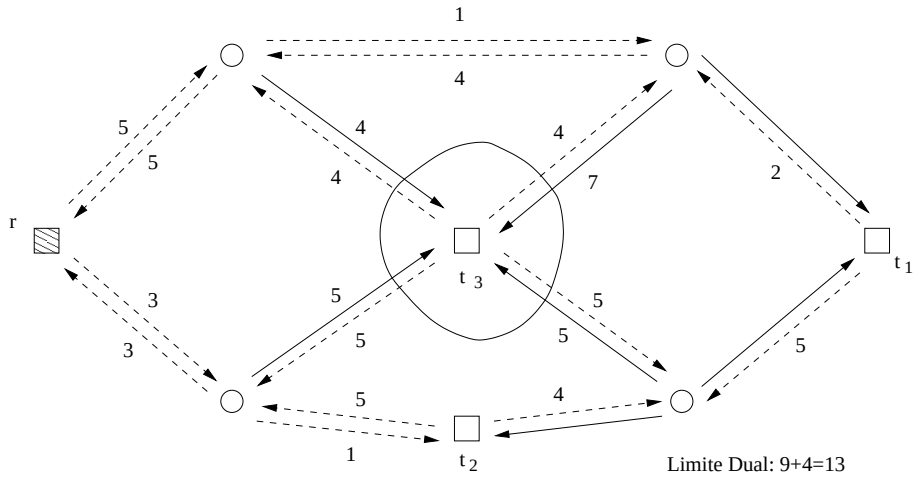

Figura 3.4: Exemplo de execução do DA seqüencial. Componente referente ao terminal t_3 é selecionado.

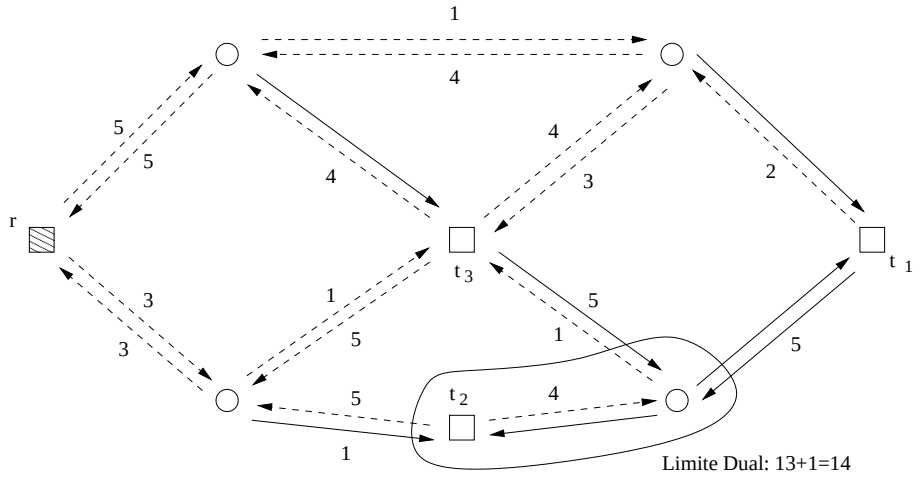

Figura 3.5: Exemplo de execução do DA seqüencial. Componente t_2 é selecionado.

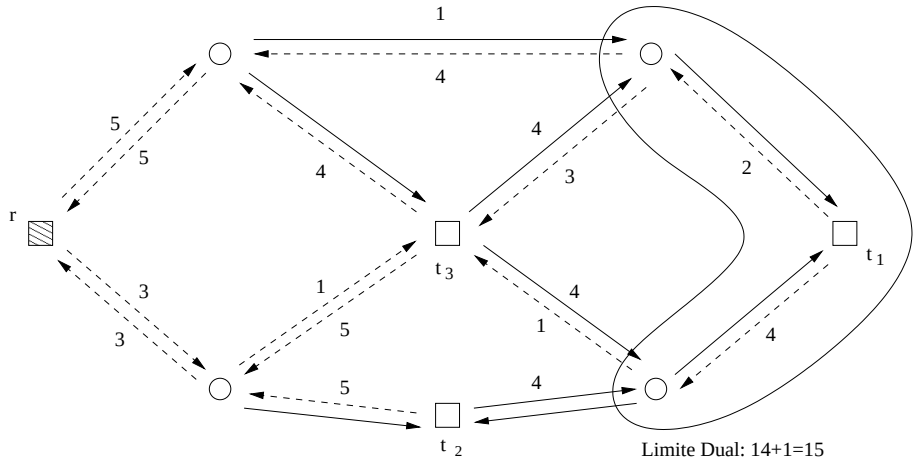

Figura 3.6: Exemplo de execução do DA seqüencial. Componente t_1 volta a ser selecionado.

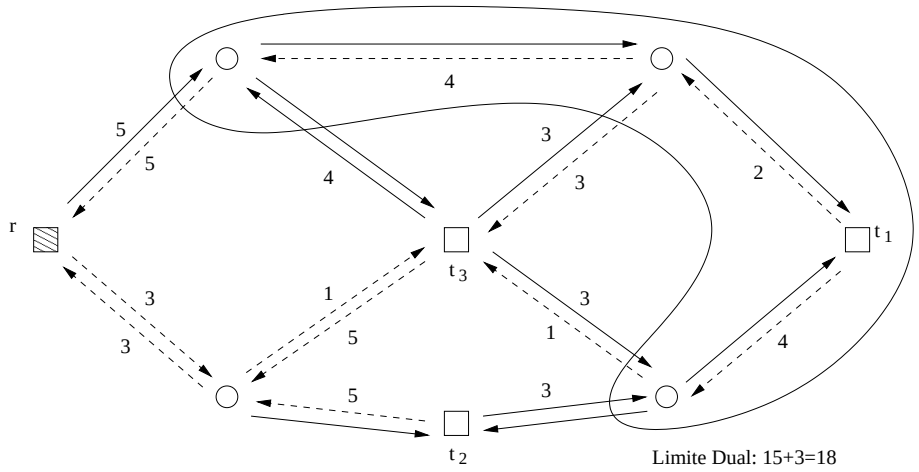

Figura 3.7: Exemplo de execução do DA seqüencial. Vários arcos podem ser saturados em uma mesma iteração.

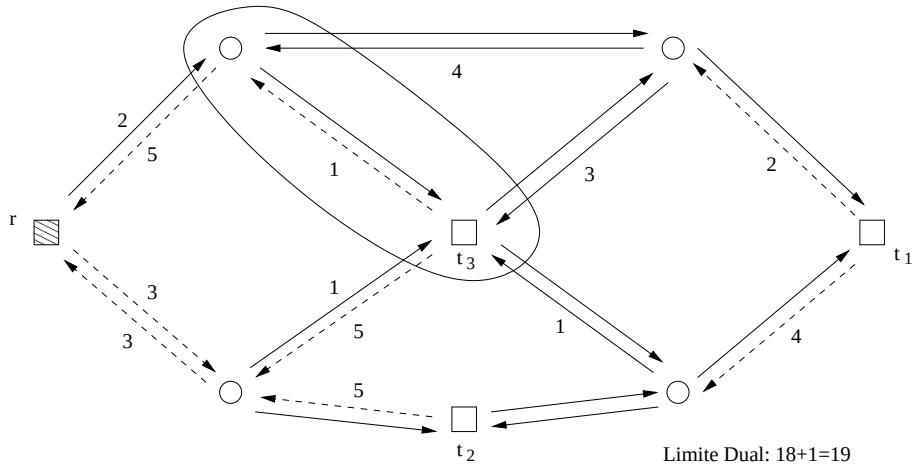

Figura 3.8: Exemplo de execução do DA seqüencial. Dois componentes raízes são unidos.

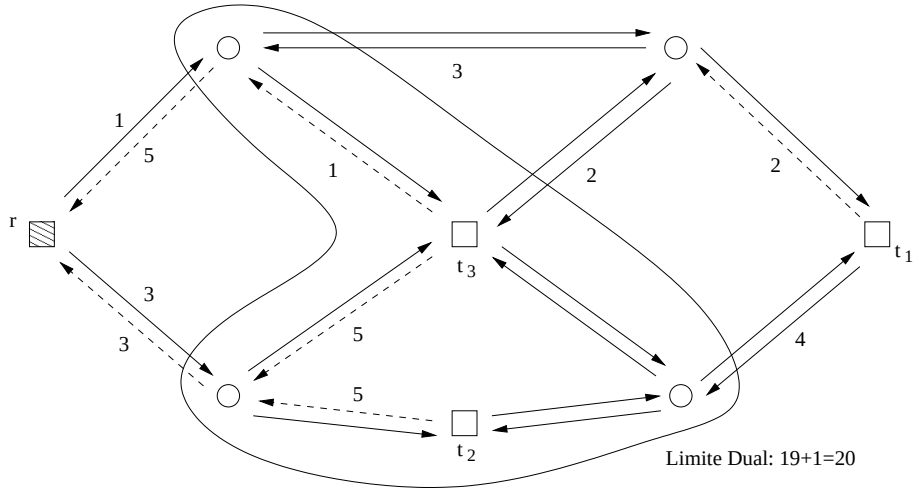

Figura 3.9: Exemplo de execução do DA seqüencial. Saturação inclui a raiz.

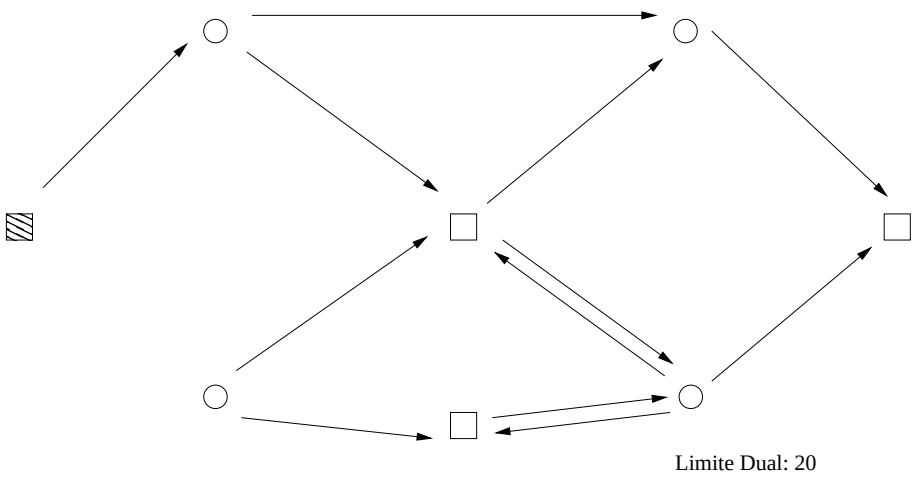

Figura 3.10: Grafo de saturação.

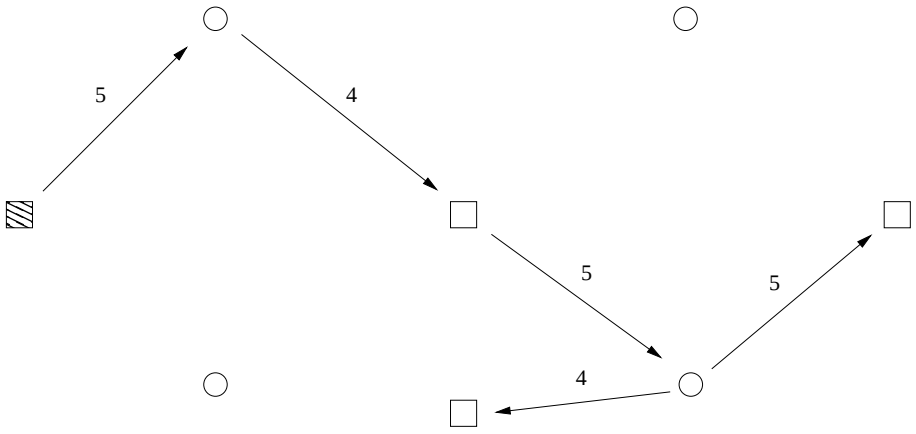

Figura 3.11: Solução final, custo 23.

3.2 Conclusão

Nesse capítulo descrevemos a versão seqüencial do algoritmo Dual Ascent, base para a construção da versão distribuída. Mostramos além da descrição formal da heurística, alguns dos bons resultados práticos presentes na literatura e um exemplo de sua execução em um grafo simples, com o intuito de que o leitor ganhe alguma familiaridade com suas rotinas, facilitando o entendimento da versão distribuída apresentada a seguir.

Capítulo 4

Dual Ascent Assíncrono Distribuído

4.1 Terminologia e Premissas

Assumimos que cada nó conhece o custo de cada um de seus arcos locais. Durante a execução, todos os dados sobre o estado de um arco são mantidos por seus nós de destino. Cada nó executa o mesmo algoritmo, que consiste em enviar mensagens por arcos locais e aguardar pelo recebimento de outras que, quando recebidas, disparam processamento que pode conter o envio de novas mensagens. Mensagens podem ser enviadas independentemente e chegam depois de um tempo indeterminado, mas finito, sem erros e em seqüência. Os nós do grafo encontram-se inicialmente em estado latente. Todos os terminais, exceto o nó raiz, ativam-se espontaneamente; outros nós ativam-se ao receber mensagens de vizinhos ativos. Assumimos que os nós do grafo possuem identificações distintas que podem ser totalmente ordenadas.

A execução do algoritmo cria **fragmentos**. Um fragmento é definido como um conjunto constituído por todos os nós que atingem um terminal t por um caminho de arcos saturados (conjunto $W(t)$ da versão seqüencial). O terminal t, criador de um fragmento, será designado como seu *líder*. Um nó pode pertencer a diversos fragmentos simultaneamente.

Por definição, todos os nós em um fragmento estão conectados ao seu líder por um caminho de arcos saturados. Sobre esses arcos, o algoritmo mantém, em cada fragmento, uma árvore utilizada para envio de mensagens internas ao fragmento, que podem ser de dois tipos: mensagens em *convergecast* dos nós do fragmento para o líder e mensagens em *broadcast* do líder do fragmento para seus nós. Uma *rodada de crescimento* consiste na descoberta do menor custo reduzido entre os arcos entrantes ao fragmento e da subtração desse valor do custo reduzido de todos os arcos entrantes. A primeira operação é executada

utilizando um *convergecast*, e seu resultado é então enviado em *broadcast* para que a segunda parte da rodada de crescimento possa ocorrer.

Com a diminuição dos custos reduzidos, novos arcos se saturam, seus nós de origem são incluídos no fragmento, e a árvore ganha novas folhas. A cada rodada, o líder do fragmento atualiza o limite inferior parcial acrescentando a ele o valor reduzido dos arcos entrantes. Com o crescimento dos fragmentos eles tendem a se sobrepor, com nós participando de mais de um fragmento. Quando fragmentos possuem arcos entrantes comuns, temos um problema de exclusão mútua que causa a suspensão temporária do crescimento de fragmentos. Um fragmento termina seu crescimento quando uma rodada satura um arco que inclua o nó raiz em sua árvore. Quando o nó raiz for incluído em todos os fragmentos, ele inicia um *broadcast* que termina o algoritmo.

4.2 Crescimento dos Fragmentos

Inicialmente, um fragmento composto somente por um terminal t seleciona o arco entrante de custo reduzido mínimo, neste ponto, o custo original. O limite inferior parcial iniciado com zero é acrescido desse valor, que também é subtraído do custo reduzido de cada um dos arcos entrantes. Mensagens *Include(t)* são transmitidas pelos arcos saturados. O nó que envia o *Include(t)* marca aquele arco como *To_leaf(t)*. Esses arcos definem uma árvore direcionada do líder para todos os outros nós do fragmento, utilizada para operações de *broadcast*. Ao receber essa mensagem, um nó se considera incluído no fragmento e marca o arco de recebimento como *To_leader(t)*. Os arcos marcados como *To_leader(t)* definem uma árvore direcionada ao líder do fragmento, utilizada para operações de *convergecast*. Esse nó também envia uma mensagem *Check(t)* a todos os demais vizinhos, comunicando que ele pertence ao fragmento t e perguntando aos vizinhos se eles também fazem parte de t. A mensagem *Check(t)* deve ser respondida com uma mensagem *Ack(t, true)*, se o vizinho também fizer parte de t, ou *Ack(t, false)* caso não faça parte.

O próximo passo é calcular o menor custo reduzido entre os arcos entrantes ao fragmento. Os nós folhas do fragmento enviam em seus arcos *To_leader(t)* uma mensagem *Conv(t,Smallest,δ)* com o menor custo reduzido entrante a ele, δ, cujo vizinho correspondente não pertença ao fragmento t. Os nós não folhas, ao receberem as mensagens *Conv(t,Smallest,δ)* comparam os custos recebidos com os custos reduzidos de arcos entrantes locais de vizinhos que não pertençam a t. O menor desses valores é enviado por seu arco marcado como *To_leader(t)*, utilizando outra mensagem *Conv(t,Smallest,δ)*.

Quando o líder receber uma mensagem *Conv(t,Smallest,δ)* para cada arco local saturado, poderá calcular o menor custo reduzido associado a arcos entrantes ao fragmento como um todo, chamado Δ. Nesse ponto o líder pode acrescentar Δ ao limite inferior parcial e iniciar um *broadcast* de volta aos membros do fragmento, utilizando uma mensagem *Broad(t,Smallest,Δ)*. Ao receber essa mensagem, o nó diminui os custos reduzidos associados a seus arcos entrantes locais, o que pode ocasionar saturações com envio de mensagens *Include(t)*, com o conseqüente crescimento do fragmento. O nó recém-incluído "checa" seus vizinho e inicia um novo *convergecast*, assim como folhas que não tenham incluído novos nós. Veja a Figura 4.1.

Caso um nó folha não possua arcos entrantes passíveis de inclusão no fragmento, ele deve informar $\delta = \infty$, excluindo seus arcos locais do cálculo do menor custo reduzido entre arcos entrantes ao fragmento. Se, ao final do *convergecast*, o líder determinar $\Delta = \infty$, temos uma instância inviável do problema, pois não existe caminho entre o nó raiz e o terminal líder desse fragmento.

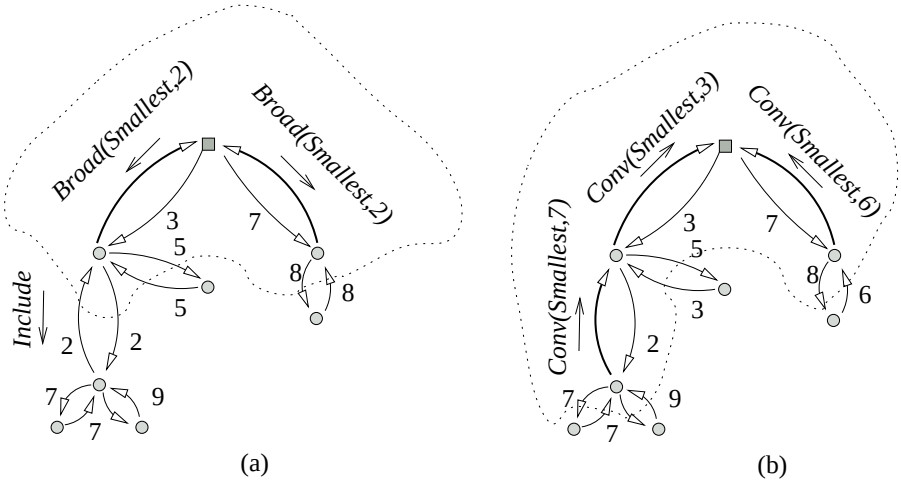

Figura 4.1: Crescimento de fragmentos

Quando vários nós são incluídos em um fragmento em uma mesma rodada de crescimento, é possível que um nó recém-incluído v_1 receba uma mensagem *Ack(t,false)* de um nó v_2 que ainda receberá um *Include(t)* naquela mesma rodada, como ilustrado pela Figura 4.2. Neste caso, v_1 irá considerar o arco (v_2, v_1) como entrante ao fragmento e seu custo reduzido poderá ser equivocadamente considerado o menor e determinar o valor de Δ. Quando a mensagem *Broad(t,Smallest,Δ)* com esse valor atingir v_1, este já terá sido informado (pela mensagem *Check(t)*) que o nó v_2, na realidade, faz parte do fragmento t. O custo reduzido do arco (v_2, v_1) não será reduzido. Portanto, é possível que ocorra uma *rodada de crescimento degenerada*, uma rodada de crescimento em que nenhum nó é

incluído no fragmento. No entanto, toda rodada de crescimento degenerada será seguida por uma não degenerada, pois em rodadas onde nenhum nó é incluído, não pode ocorrer que nós participantes do fragmento não sejam reconhecidos como tais por seus vizinhos. No caso extremo, a cada dois ciclos de crescimento um nó será necessariamente incluído no fragmento.

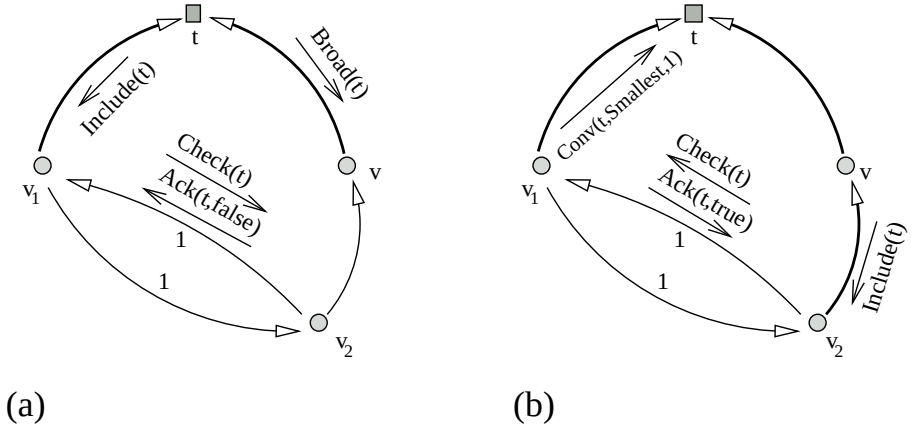

Figura 4.2: Possível ciclo de crescimento degenerado

Outra situação ocorre quando dois nós v_1 e v_2 enviam mensagens *Include(t)* para o mesmo nó v_3 na mesma rodada de crescimento. Sem perda de generalidade, suponha que a mensagem enviada por v_1 chegue primeiro. O segundo *Include(t)* será respondido por uma mensagem *AlreadyIncluded(t)*, e o nó v_2 saberá que o arco (v_3, v_2) não deve ser marcado como *To_leaf(t)*, mantendo a integridade da árvore associada ao fragmento t. Observe que esse caso gera arcos saturados que não pertencem a árvore de *broadcast/convergecast* do fragmento.

Finalmente, é possível que um nó recém-incluído também inclua outros nós na mesma rodada de crescimento. Suponha que o nó v_1 seja incluído no fragmento t. Ele envia mensagens *Check(t)* e aguarda por mensagens *Ack(t)* em resposta. Então, é levantado o menor custo reduzido entre os arcos entrantes com origem em nós que não pertençam a t. Se este valor for positivo, v_1 saberá que é uma folha e enviará uma mensagem *Conv(t,Smallest,δ)* como usual. No entanto, como outros fragmentos crescem em paralelo, possivelmente reduzindo custos de arcos locais de v_1, é possível que o valor de δ encontrado seja igual a zero. Existem arcos saturados (v_2, v_1) tais que v_2 não pertence a t. Neste caso, v_1 envia *Include(t)* a v_2, que continuará a rodada de crescimento.

4.3 Suspensão de Fragmentos

Fragmentos com pelo menos um arco entrante comum não podem crescer simultaneamente, visto que o custo reduzido desse arco seria alterado concorrentemente. Essa situação corresponde a um problema de exclusão mútua em que o recurso compartilhado é o custo reduzido do arco comum. Do ponto de vista do crescimento de fragmentos da seção anterior, esse problema de exclusão mútua ocorre quando dois ou mais fragmentos possuem um nó comum. A fim de resolver esse conflito, somente um dos fragmentos, o de maior identificação, irá continuar o crescimento enquanto os demais ficarão suspensos até que o fragmento de maior identificação termine seu crescimento.

Por exemplo, na Figura 4.3, partes "a" e "b", os fragmentos t_1 e t_2 possuem um nó em comum v e t_1 é suspenso. Quando um nó que já participa de um fragmento recebe uma mensagem *Include* de outro, existe um conflito potencial como ilustrado na parte "a" da Figura 4.3. A fim de simplificar o algoritmo, sem sacrificar sua corretude e complexidade, essa condição é suficiente para suspender todos os fragmentos que incluíram v, exceto o de maior identificação. O algoritmo garante que, em um dado momento, cada nó participe de apenas um fragmento ativo.

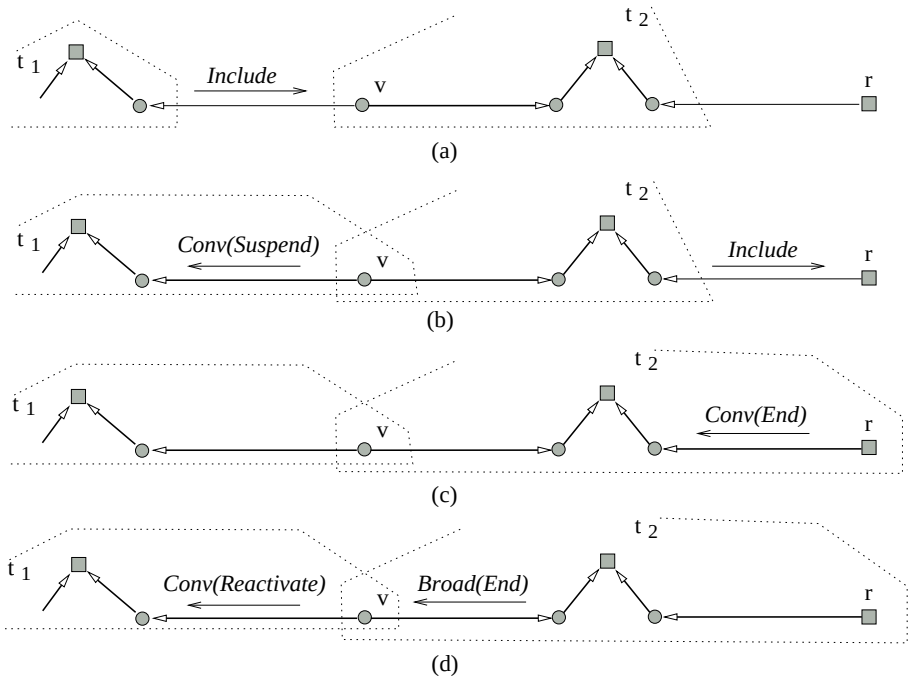

Figura 4.3: Suspendendo e reativando fragmentos

Dois casos devem ser considerados a fim de manter a garantia de exclusão mútua. Quando o nó recebe o *Include*(t_1) de um fragmento com identificação menor do que o fragmento t_2 ativo, do qual ele atualmente faz parte, ele responde imediatamente a

t_1 com a mensagem *Conv(t_1,Suspend)*. No segundo caso, onde a identificação de t_1 é maior do que a de t_2, o nó precisa esperar a próxima rodada de crescimento de t_2, processar o *Broad(t_2,Smallest,Δ)* referente a essa rodada e responder com uma mensagem *Conv(t_2,Suspend)*. Somente depois da resposta a t_2, o nó pode enviar o *Conv(t_1,Smallest,δ)* aguardado por t_1.

Quando um fragmento é suspenso, seu líder envia uma mensagem *Broad(t,Suspend)* comunicando a todos os seus membros a suspensão. Este procedimento é necessário pois membros de t podem ter recebido mensagens *Include* de fragmentos cujos líderes possuam identificações maiores que t. Esses membros aguardam o próximo *broadcast* de t. A mensagem *Broad(t,Suspend)* habilita os componentes de t a participarem de outros fragmentos ativos, mesmo de identificações menores que t.

No exemplo da Figura 4.3, partes "c" e "d", podemos observar que depois de t_1 ser suspenso, t_2 cresce e inclui o nó raiz. A finalização do crescimento de t_2 causa a reativação de t_1.

4.4 Terminação

Um fragmento termina seu crescimento definitivamente quando ele for atingido pelo nó raiz por um caminho de arcos saturados. Quando o nó raiz recebe um *Include(t)*, ele responde com uma mensagem *Conv(t,End)*. O líder do fragmento, então, envia *Broad(t,End,pb)* contendo seu limite inferior parcial "pb" (*partial bound*) a todos os seus membros e termina.

Ao receber a mensagem *Broad(t,End,pb)* indicando que o fragmento t terminou, um nó tenta reativar o fragmento de maior identificação t_1, que tenha sido suspenso por ele. Isso é feito com o envio da mensagem *Conv(t_1,Reactivate)*. Ao receber o *Conv(t_1,Reactivate)*, o líder de t_1 enviará *Broad(t_1,Reactivate)* para reiniciar seu crescimento. Observe que a tentativa de reativação pode falhar caso t_1 possua outro conflito em outra parte de sua árvore de saturação. Nesse caso, o nó que detecte o novo conflito responde à mensagem *Broad(t_1,Reactivate)* com uma mensagem *Conv(t_1,Suspend)*.

O nó raiz, ao receber a mensagem *Broad(t,End,pb)*, acumula o limite inferior parcial referente a t a fim de obter o limite inferior global. Ao detectar que todos os fragmentos terminaram e que atinge todos os terminais por caminhos de arcos saturados, o nó raiz inicia um *broadcast* para a terminação global do algoritmo.

4.5 Pseudo Código

O algoritmo seguinte é executado por cada nó e consiste em respostas para cada uma das mensagens que podem ser geradas. Adicionalmente, a rotina a ser executada na ativação espontânea de um terminal é exibida. Cada nó enfileira mensagens recebidas e as processa em ordem de chegada.

Exceto o nó raiz, cada nó pode ser *leader* ou *non-leader* para cada um dos fragmentos. Assumimos que para cada arco $a = (v_1, v_2)$ do nó v_1 para o nó v_2 existe um arco $a' = (v_2, v_1)$ do nó v_2 para o nó v_1, e os nós v_1 e v_2 são ditos "vizinhos".

A variáveis a seguir devem ser mantidas em cada nó para cada fragmento t de que ele faça parte:

- **fragment_state(t)**: [*unknown, active, finished, suspended*]. Estado do fragmento t. Inicializado com *unknown*;
- **state(t, a_1)**: [*internal, entering, outgoing, to_leaf, to_leader*]. Estado do arco a_1 para o fragmento t. Inicializado com *entering* ou *outgoing*, dependendo da direção do arco;
- **subtree_cost(t, v)**: Menor custo entre todos os arcos entrantes ao fragmento t na subárvore enraizada no nó v;
- **conflict(t, v)**: Lógico. Se verdadeiro, indica que o fragmento t tem um conflito na subárvore enraizada no nó v;
- **end(t, v)**: Lógico. Se verdadeiro, indica que o nó raiz foi incluído pelo fragmento t na subárvore enraizada no nó v;
- **waited_ack(t)** e **waited_conv(t)**: Número de mensagens *Ack* ou *Conv* aguardadas para iniciar um *convergecast* ou *broadcast* em um fragmento t. Inicializado com zero;
- **wait(t)**: Lógico. Se verdadeiro, indica que o fragmento t está aguardando pela suspensão de outro fragmento de menor identificação para continuar sua execução. Inicializado com falso;
- **δ**: Menor custo reduzido de arco entrante em uma parte do fragmento.

Cada terminal mantém as variáveis:

- **pb** : limite inferior parcial. Inicializado com zero;
- **Δ** : menor custo reduzido de arco entrante no fragmento.

O nó raiz mantém as variáveis:

- **terminals** : número de terminais finalizados;
- **gb** : limite inferior global;
- **terminated** : Lógico. Verdadeiro quando todos os terminais finalizaram. Indica terminação global.

Algoritmo:

(A.1) Ativação espontânea, executada em todo t, exceto o nó raiz:

```
1  t ← id do terminal;
2  Δ ← menor custo entre os arcos entrantes;
3  pb ← Δ;
4  fragment_state(t) ← active;
5  para cada arco a tal que state(t,a)=entering :
6      reduced_cost(a) ← reduced_cost(a) - Δ;
7      se reduced_cost(a) = 0
8          waited_conv(t) ← waited_conv(t) + 1;
9          state(t,a') ← to_leaf;
10         envie INCLUDE(t) pelo arco a';
```

(A.2) Ao receber INCLUDE(t) pelo arco $a = (v_1, v_2)$:

```
1  se fragment_state(t) = active
2      state(t,a) ← internal;
3      envie ALREADYINCLUDED(t) pelo arco a';
4  senão
5      se nó é raiz
6          envie CONV(t, END) pelo arco a';
7      senão
8          fragment_state(t) ← active;
9          state(t,a') ← to_leader;
10         para cada arco a_1 com state(t,a_1)=outgoing :
11             waited_ack(t) ← waited_ack(t) + 1;
12             envie CHECK(t) pelo arco a_1;
13     se waited_ack(t) = 0 execute procedimento convergecast(t);
```

(A.3) Ao receber ALREADYINCLUDED(t) pelo arco $a = (v_1, v_2)$:

```
1 state(t,a) ← internal;
2 waited_conv(t) ← waited_conv(t) - 1;
3 se waited_conv(t) = 0 execute procedimento convergecast(t);
```

(A.4) Ao receber CHECK(t) pelo arco $a = (v_1, v_2)$:

```
1 se fragment_state(t) = active
2     state(t, a) ← internal;
3     envie ACK(t, true) pelo arco a';
4 senão
5     envie ACK(t, false) pelo arco a';
```

(A.5) Ao receber ACK(t, answer) pelo arco $a = (v_1, v_2)$:

```
1 se answer
2     state(t, a) ← internal;
3 senão
4     se reduced_cost(a)=0
5         waited_conv(t) ← waited_conv(t) + 1;
6         state(t, a') ← to_leaf;
7         envie INCLUDE(t) pelo arco a';
8 waited_ack(t) ← waited_ack(t) - 1;
9 se waited_ack(t) = 0 and waited_conv(t) = 0 execute procedimento convergecast(t);
```

(A.6) Ao receber CONV(t, type, δ) pelo arco $a = (v_1, v_2)$:

```
1  se type = REACTIVATE
2     conflict(t, v1) ← FALSE;
3     se ∄t1 > t tal que fragment_state(t1)=active
4        se nó é o líder de t
5           para cada arco a1 com state(t, a1)= to_leaf :
6              waited_conv(t) ← waited_conv(t) + 1;
7              envie BROAD(t, REACTIVATE) pelo arco a1;
8        senão
9           envie CONV(t, REACTIVATE) pelo arco a1 tal que state(t, a1)= to_leader;
10 senão se type = SMALLEST
11    subtree_cost(t, v1) ← δ;
12 senão se type = SUSPEND
13    conflict(t, v1) ← TRUE;
14 senão se type = END
15    end(t, v1) ← TRUE;
16 waited_conv(t) ← waited_conv(t) - 1;
17 se waited_conv(t) = 0 e waited_ack(t) = 0 execute procedimento convergecast(t);
```

(A.7) procedimento convergecast(t):

```
1  se ∃t1 < t tal que fragment_state(t1)=active
2     wait(t) ← TRUE;
3  senão
4     se ∃t1 > t tal que fragment_state(t1)=active
5        local_conflict ← TRUE;
6     se nó é o líder de t
7        execute procedimento leader-broadcast(t);
8     senão
9        execute procedimento non-leader-convergecast(t);
10    se ∃t1 > t tal que wait(t1)=TRUE
11       wait(t1) ← FALSE;
12       execute procedimento convergecast(t1);
```

(A.8) procedimento leader-broadcast(t):

1 **se** end(t,v) = TRUE para pelo menos um vizinho v
2 **envie** BROAD(t, END, pb) em cada arco a tal que state(t, a)=to_leaf ;
3 **senão se** local_conflict **ou** conflict(t, v) = TRUE para pelo menos um vizinho v
4 **envie** BROAD(t, SUSPEND) em cada arco a tal que state(t, a)=to_leaf;
5 **senão**
6 $\Delta \leftarrow$ menor custo entre
todo arco a tal que state(t, a)=entering **e**
todo subtree_cost(t, v) tal que state(t, a)=to_leaf, sendo a arco
local com origem em v;
7 pb $\leftarrow$ pb + Δ;
8 **para** cada arco a com state(t, a)=to_leaf :
9 waited_conv(t) $\leftarrow$ waited_conv(t) + 1;
10 **envie** BROAD(t, SMALLEST, Δ) pelo arco a;
11 **para** cada arco a com state(t, a)=entering :
12 reduced_cost(a) $\leftarrow$ reduced_cost(a) - Δ;
13 **se** reduced_cost(a) = 0
14 state(t, a') $\leftarrow$ to_leaf;
15 waited_conv(t) $\leftarrow$ waited_conv(t) + 1;
16 **envie** INCLUDE(t) por a';

(A.9) procedimento non-leader-convergecast(t):

1 **se** end(t, v) = TRUE para pelo menos um vizinho v
2 **envie** CONV(t, END) pelo arco a tal que state(t, a)=to_leader;
3 fragment_state(t)$\leftarrow$ finished;
4 **senão se** local_conflict **ou** conflict(t, v) = TRUE para pelo menos um vizinho v
5 **envie** CONV(t, SUSPEND) pelo arco a tal que state(t, a)=to_leader;
6 fragment_state(t)$\leftarrow$ suspended;
7 **senão**
8 $\delta \leftarrow$ menor custo entre
todo arco a tal que state(t,a)=entering **e**
todo subtree_cost(t, v) tal que state(t, a)=to_leaf, sendo a arco
local com origem em v;
9 **envie** CONV(t, SMALLEST, δ) pelo arco a tal que state(t, a)=to_leader;

(A.10) Ao receber BROAD(t, SMALLEST, Δ) pelo arco $a = (v_1, v_2)$;

1 **para** cada arco a_1 com state(t, a_1)= to_leaf :
2 waited_conv$(t) \leftarrow$ waited_conv(t) + 1;
3 **envie** BROAD(t, SMALLEST, Δ) pelo arco a_1;
4 **para** cada arco a_1 com state(t, a_1)=entering :
5 reduced_cost$(a_1) \leftarrow$ reduced_cost(a_1) - Δ;
6 **se** reduced_cost(a_1)=0
7 state$(t, a_1') \leftarrow$ to_leaf;
8 waited_conv$(t) \leftarrow$ waited_conv(t) + 1;
9 **envie** INCLUDE(t) por a_1';
10 **se** waited_conv$(t) = 0$ **execute** procedimento convergecast(t);

(A.11) Ao receber BROAD(t, SUSPEND) pelo arco $a = (v_1, v_2)$;

1 fragment_state(t)=suspended;
2 **envie** BROAD(t, SUSPEND) por cada arco a_1 tal que state(t, a_1)=to_leaf;
3 **se** $\exists\ t_1 > t$ tal que wait(t_1) = TRUE
4 wait$(t_1) \leftarrow$ FALSE;
5 **execute** procedimento convergecast(t_1);

(A.12) Ao receber BROAD(t, REACTIVATE) pelo arco $a = (v_1, v_2)$;

1 fragment_state(t)=active;
2 **para** cada arco a_1 com state(t, a_1)= to_leaf :
3 waited_conv$(t) \leftarrow$ waited_conv(t) + 1;
4 **envie** BROAD(t, REACTIVATE) pelo arco a_1;
5 **para** cada arco a_1 com state(t, a_1)=entering **e** com reduced_cost(a_1)=0 :
6 state$(t, a_1') \leftarrow$ to_leaf;
7 waited_conv$(t) \leftarrow$ waited_conv(t) + 1;
8 **envie** INCLUDE(t) pelo arco a_1';
9 **se** waited_conv$(t) = 0$ **execute** procedimento convergecast(t);

(A.13) Ao receber BROAD(t, END, pb) pelo arco $a = (v_1, v_2)$;

```
1  fragment_state(t)=finished;
2  se nó não é raiz
3     envie BROAD(t, END, pb) por cada arco a_1 tal que state(t, a_1)= to_leaf;
4     se ∃ t_1 > t tal que wait(t_1) = TRUE
5        wait(t_1) ← FALSE;
6        execute procedimento convergecast(t_1);
7     senão se ∃ t_1 tal que t_1 é a maior
               identificação de fragmento com fragment_state(t_1)=suspended
8        envie CONV(t_1, REACTIVATE) pelo arco a_1 tal que state(t_1, a_1)= to_leader;
9  senão
10    gb ← gb + pb;
11    terminals ← terminals + 1;
12    se terminals = total de terminais
13       terminated ← TRUE;
```

4.6 Exemplo

As figuras a seguir ilustram uma possível execução do algoritmo considerando uma instância pequena, com apenas quatro nós: três terminais e um nó não-terminal.

A Figura 4.4 apresenta o grafo da instância de nosso exemplo, em que arcos são representados por setas pontilhadas com seus custos indicados pelos números próximos. Terminais são quadrados e o nó não-terminal é um círculo. Ao lado de cada terminal exibimos também o limite inferior, inicializado com zero. No raiz, o limite é referente a todo o grafo (gb) e, nos demais terminais, é referente apenas aos respectivos fragmentos (pb).

A Figura 4.5 apresenta o grafo após a ação A.1 do algoritmo, em que todos os terminais acordam espontaneamente, subtraem o valor do menor custo reduzido de todos os arcos entrantes e enviam mensagens *Include* pelos arcos saturados, representados por setas com linhas cheias.

A Figura 4.6 mostra o resultado da execução da rotina A.2 pelo nó v, que envia mensagens *Check* para seus vizinhos. A Figura 4.7 mostra as mensagens *Ack* enviadas em resposta quando t_1, t_2 e r executam A.4.

Ao receber a mensagem *Ack*, o nó v executa A.5. Ao receber o último *Ack* aguardado, ele executa o procedimento A.7 "convergecast". Como o nó v não é líder, o procedimento "non-leader-convergecast" A.9 é executado. Ele detecta o conflito e envia mensagens *Conv* apropriadas a cada um dos líderes envolvidos. O líder do fragmento t_1 recebe uma mensagem *Conv(Suspend)*, enquanto t_2 recebe uma mensagem *Conv* contendo o menor custo reduzido entre os arcos entrantes, como mostra a Figura 4.8.

Na Figura 4.9, t_1 informa a seu fragmento através da mensagem *Broad(Suspend)* a suspensão (A.8, linhas 3 e 4), e t_2 inicia uma nova rodada de crescimento enviando a mensagem *Broad(Smallest)* (A.8, linhas 6 a 10) e uma mensagem *Include* (A.8, linhas 11 a 16).

Nas figuras 4.9 e 4.10, podemos verificar que o nó raiz recebe duas mensagens *Include* na mesma rodada de crescimento. Ele responde ao primeiro com uma mensagem *Conv(End)* (A.2, linha 6) e ao segundo com uma mensagem *AlreadyIncluded* (A.2, linhas 2 e 3) como pode ser observado nas figuras 4.10 e 4.11.

Na Figura 4.12, t_2 inicia o *broadcast*, que finalizará seu fragmento (A.8 linhas 1 e 2). O nó raiz executa a rotina A.13, linha 10, atualizando o limite inferior global e o nó v

informa a t_1 que não existe motivo para que permaneça suspenso através da mensagem *Conv(Reactivate)* (A.13, linhas 7 e 8) como pode ser observado na Figura 4.13.

A Figura 4.14 mostra o nó v recebendo a mensagem *Broad(Reactivate)* de t_1. Ele verifica que existem arcos entrantes ao fragmento t_1 sem a marcação *To_leaf(t_1)*, pois o arco foi saturado por t_2, portanto, o nó v envia uma mensagem *Include* por esse arco e o marca como *To_leaf(t_1)* (A.12 linhas 5 a 8).

Na Figura 4.15 podemos observar as trocas de mensagens que finalizam o fragmento t_1 e na Figura 4.16 o *broadcast* do limite parcial de t_1. Neste ponto, o nó raiz detecta que todos os fragmentos finalizaram (A.13 linhas 12 e 13).

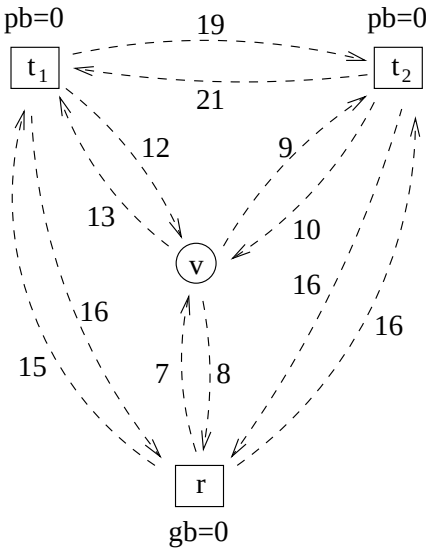

Figura 4.4: Grafo Original

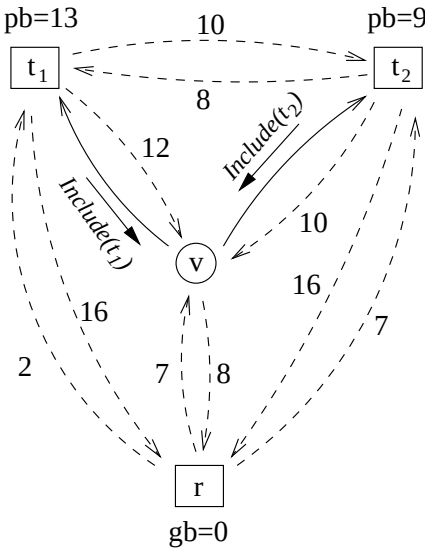

Figura 4.5: Inclusão do nó v

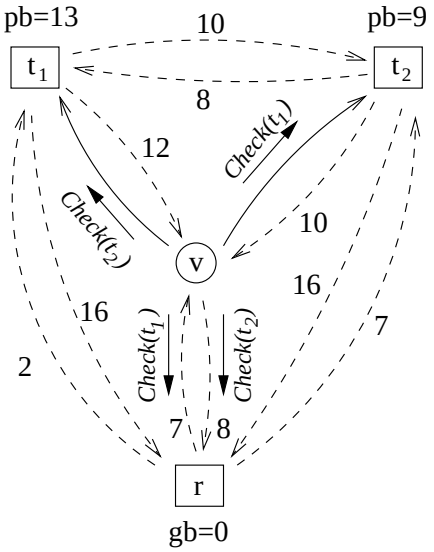

Figura 4.6: Envio de mensagens *Check* aos vizinhos

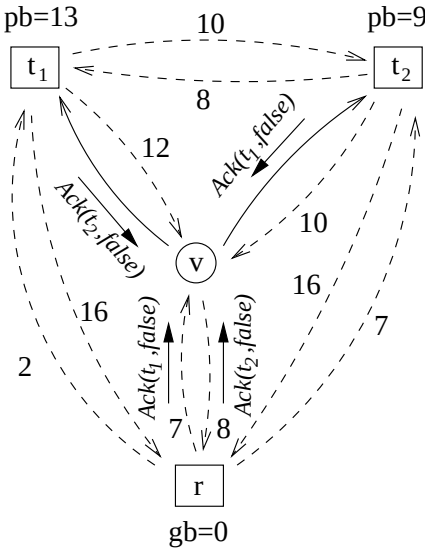

Figura 4.7: Respondendo ao *Check* com mensagens *Ack*

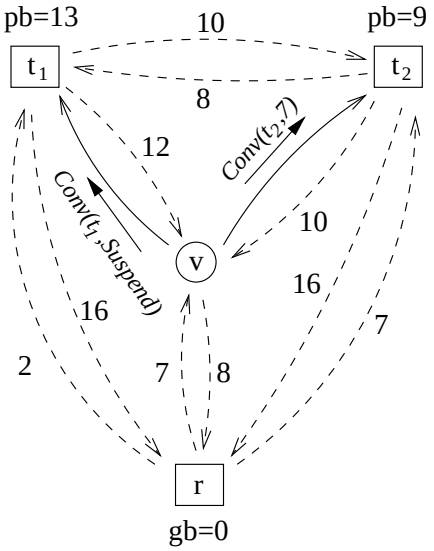

Figura 4.8: *Convergecast*'s diferentes para t_1 e t_2

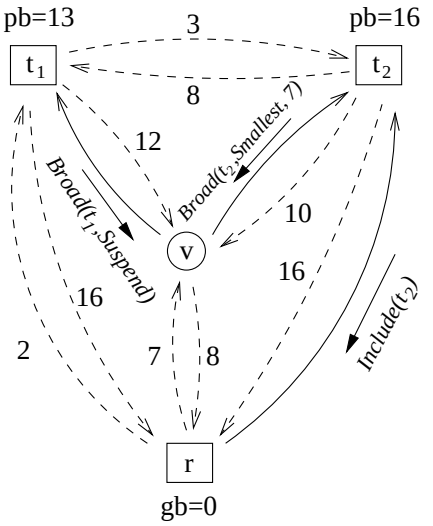

Figura 4.9: t_1 é suspenso e t_2 cresce.

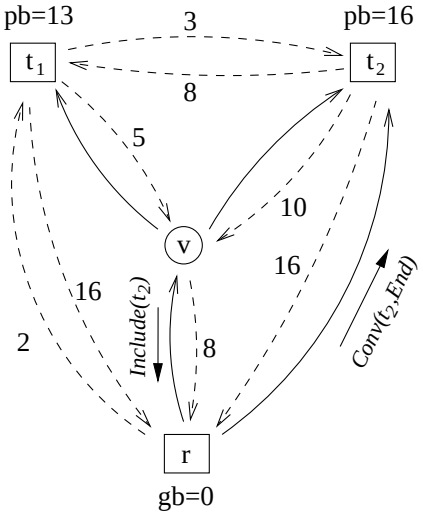

Figura 4.10: Raiz é alcançada por t_2

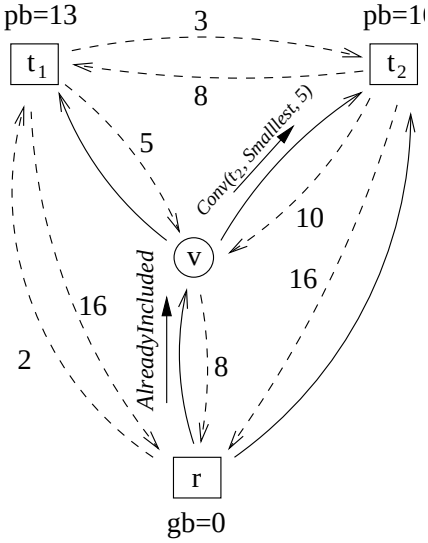

Figura 4.11: *Convergecast* para finalizar o fragmento t_2

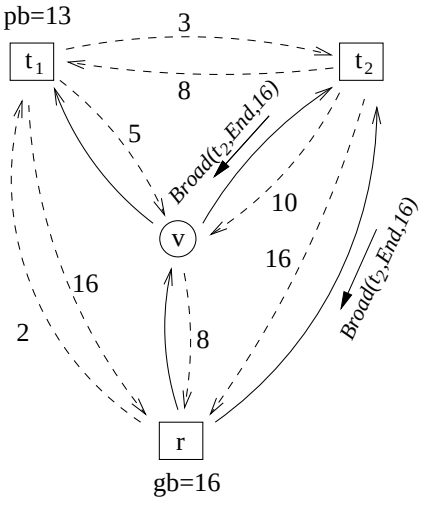

Figura 4.12: Finalização do fragmento t_2

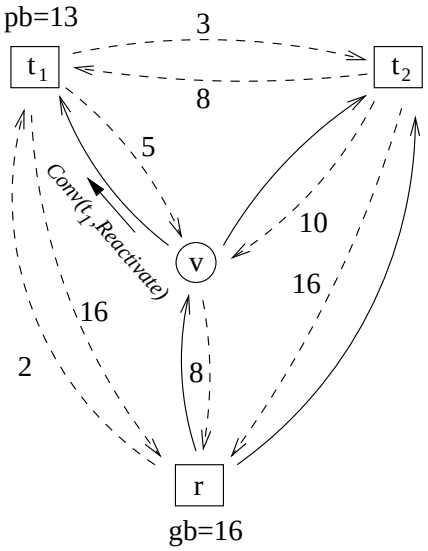

Figura 4.13: Reativação do fragmento t_1

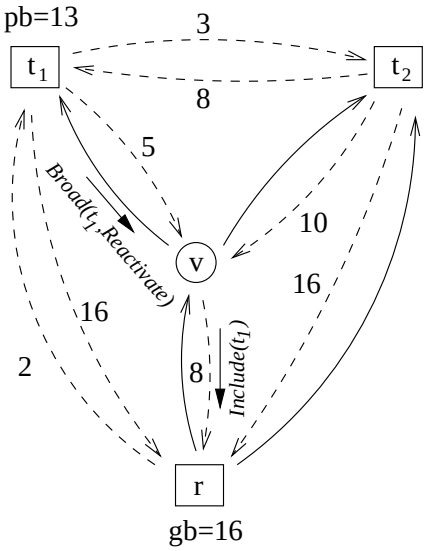

Figura 4.14: Raiz é incluída pelo fragmento t_1

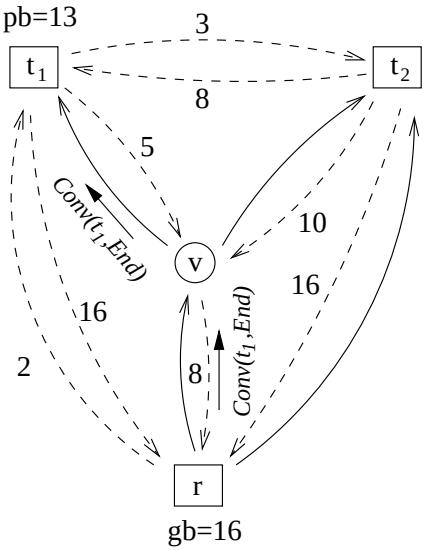

Figura 4.15: t_1 pode finalizar

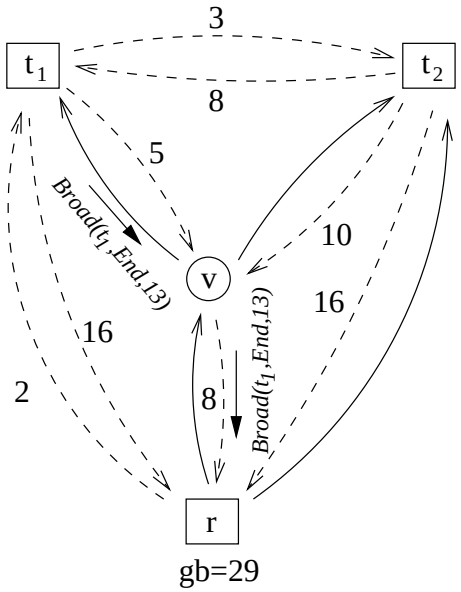

Figura 4.16: O Fragmento t_1 termina. Raiz detecta fim global.

4.7 Conclusão

Apresentamos a versão distribuída para o Dual Ascent, principal contribuição deste trabalho. Adicionalmente foi apresentado um exemplo da execução do algoritmo em um grafo pequeno a fim de facilitar a compreensão da heurística pelo leitor. A descrição do algoritmo, assim como a discussão e os resultados práticos apresentados nos dois próximos capítulos foram submetidos à apreciação da comunidade científica quando apresentados no SBRC'2007 [Santos et al.] e no WEA'2007 [Santos et al. 2007]. Também tivemos um trabalho aceito para publicação na *Networks*, da editora John Wiley & Sons [Drummond et al. 2008].

Capítulo 5

Análise do Algoritmo

5.1 Correção

Teorema: *O algoritmo garante que o nó raiz atinge todos os terminais em um tempo finito.*

Prova:

Pela descrição da Seção 4.2, para cada duas rodadas de crescimento consecutivas, pelo menos um nó é incluído em um fragmento. Um fragmento ativo realiza rodadas de crescimento até que seja suspenso ou termine. Pela descrição da Seção 4.3, segue que a exclusão mútua é garantida para o acesso a arcos entrantes de nós compartilhados e que, em casos de conflito, somente o fragmento com maior número de identificação não será suspenso. Portanto, pelo menos o fragmento ativo com maior número de identificação, t_n, está sempre realizando rodadas de crescimento e é alcançado pelo nó raiz em no máximo $2.|V|$ dessas rodadas. Seja t_{n-1} o fragmento não terminado com maior número de identificação após o término de t_n. Ou t_{n-1} está ativo, ou foi suspenso por t_n. No segundo caso, t_{n-1} será reativado durante a terminação de t_n. ■

5.2 Custo de Comunicação

Desejamos um limite superior para o número de mensagens trocadas durante a execução do algoritmo. Quando um nó é incluído por um fragmento, ele envia mensagens *Check* para todos os seus vizinhos, exceto o remetente do *Include*. Podem ser enviadas no máximo $|V| - 2$ mensagens *Check* durante cada inclusão. Esse processo pode se repetir para cada um dos $|V|$ nós do grafo, para cada um dos $|T| - 1$ fragmentos. Como para cada

mensagem *Check* é enviada exatamente uma mensagem *Ack*, o número de mensagens *Check* e *Ack* enviadas é, no máximo, $O(|T|.|V|^2)$.

Mostraremos agora que o número de envios de nenhum outro tipo de mensagem pode ser maior do que esse limite.

Cada rodada de crescimento requer, no máximo, $O(|V|)$ mensagens *Conv*, *Broad* e *Include*. Podem existir no máximo $O(|V|)$ rodadas de crescimento para cada fragmento, logo, $O(|T|.|V|^2)$ é um limite superior válido para esses tipos de mensagens.

Considerando agora os procedimentos de suspensão e reativação: a reativação de um fragmento é disparada somente quando um fragmento de número de identificação maior termina. Logo, cada fragmento pode ser reativado somente uma vez por cada outro fragmento e o número total de reativações é, no máximo, $O(|T|^2)$. Para cada reativação, o terminal realiza um *broadcast* em sua árvore, que nunca demandará mais do que $O(|V|)$ mensagens. Como não são enviadas mensagens especiais no processo de suspensão, $O(|T|^2.|V|)$ é um limite superior válido para o número de mensagens enviadas em decorrência de suspensões e reativações de fragmentos.

Esse limite é assintoticamente ajustado. O pior caso ocorre quando temos um grafo completo em que todos os terminais (incluído o nó raiz) são conectados por arcos de custo muito alto, de forma que só sejam saturados quando todos os nós não-terminais forem incluídos, para cada um dos fragmentos. Nesse caso serão enviadas $\Theta(|T|.|V|^2)$ mensagens *Check* e *Ack*.

5.3 Custo de Tempo Global

A complexidade em tempo global para o modelo assíncrono adotado assume que computação local não gasta tempo, e que o tempo de envio de uma mensagem de um nó a cada um dos nós em seu conjunto de vizinhos é $O(1)$. A complexidade é, então, o número de mensagens na maior cadeia causal da forma "receber uma mensagem e enviar uma mensagem em conseqüência" que ocorra em qualquer execução do algoritmo, sobre todas as variações possíveis na estrutura do grafo de entrada [Barbosa 1996]. Essa complexidade não pode ser maior do que a complexidade em mensagens. Mostraremos a seguir que existe um caso que o algoritmo requer uma cadeia com $\Theta(|T|.|V|^2)$ mensagens.

O pior caso para o tempo com o Dual Ascent ocorre quando não existe paralelismo no crescimento dos fragmentos. Quando um fragmento está em crescimento, todos os outros

de identificação maior já terminaram e todos com identificação menor estão suspensos.

O crescimento de um fragmento cria cadeias de mensagens de comprimento $O(|V|^2)$. Cada rodada de crescimento requer uma cadeia causal de mensagens *Broad* e *Conv*, cujo tamanho é proporcional ao maior caminho de arcos saturados em sua árvore de *broadcast/convergecast*. O mais longo desses caminhos pode ter comprimento $O(|V|)$ e o crescimento completo de um fragmento pode gastar $O(|V|)$ rodadas. Mesmo nessa situação, o tempo total para o crescimento de todos os $|T|$ fragmentos poderia ainda ser quadrático porque vários nós podem ser incluídos em um fragmento em uma única rodada de crescimento, pois os fragmentos utilizam arcos saturados em rodadas de crescimento prévias de outros fragmentos. No entanto, no caso seguinte, isso não é suficiente para evitar a complexidade cúbica.

Considere um grafo direcionado completo com $m+2n+2$ nós, $n \geq m$, composto pelos conjuntos de vértices $T = \{t_1, \ldots, t_{m=|T|}\}$, $U = \{u_1, \ldots, u_n\}$, $W = \{w_1, \ldots, w_n\}$, mais os vértices r e s. O vértice r é o nó raiz, vértices em T são terminais e o restante dos vértices são não-terminais. Todos os fragmentos incluem s no início da execução do algoritmo, de forma a ocorrer um conflito envolvendo todos os fragmentos. O fragmento t_m continuará seu crescimento e todos os demais serão suspensos. Suponha que esse fragmento realiza n rodadas de crescimento, incluindo os vértices de w_1 a w_n e formando um caminho de arcos saturados de comprimento n. O fragmento t_m realiza, então, mais n rodadas de crescimento, desta vez incluindo nós de U, um por rodada de crescimento, como ilustrado na Figura 5.1. Cada uma dessas rodadas de crescimento toma $\Theta(n)$ tempo. Finalmente, t_m inclui o nó raiz, saturando o arco (r, t_m). O crescimento completo tomou $\Theta(n^2)$ em tempo. Neste ponto o fragmento t_{m-1} é reativado e inclui o vértice w_1. Todos os vértices em W são incluídos nessa mesma rodada de crescimento. Isso significa que todos os n vértices foram incluídos em tempo $O(n)$, mas agora o fragmento t_{m-1} tem um caminho em sua árvore de *broadcast/convergecast* de comprimento n. O fragmento t_{m-1} fará mais n rodadas de crescimento, incluindo nós de U um por um. Cada uma dessas rodadas de crescimento tomará $\Theta(n)$ tempo. Esse fragmento terminará após incluir o nó raiz, com seu crescimento completo consumindo tempo $\Theta(n^2)$. Todos os demais fragmentos comportam-se de maneira semelhante, e o custo em tempo da execução completa do algoritmo será $\Theta(m.n^2) = \Theta(|T|.|V|^2)$.

A Figura 5.1 ilustra a situação descrita no parágrafo anterior. t_m, o terminal de maior identificação, criou uma longa cadeia de arcos saturados, incluindo os nós de w_1 a w_n e terminou, reativando t_{m-1}, o terminal com a segunda maior identificação. t_{m-1} inclui

w_1, que causa inclusão de todos os demais elementos de W em uma única rodada de crescimento, pois são utilizados os arcos saturados por t_m. Após isto, em cada saturação, como a destacada com uma seta reforçada na figura, uma longa cadeia de mensagens percorrerá todos os nós de w_1 a w_n, elevando a complexidade para o crescimento de t_{m-1} a mesma ordem de grandeza de t_m, apesar de aproveitar as saturações feitas por esse terminal.

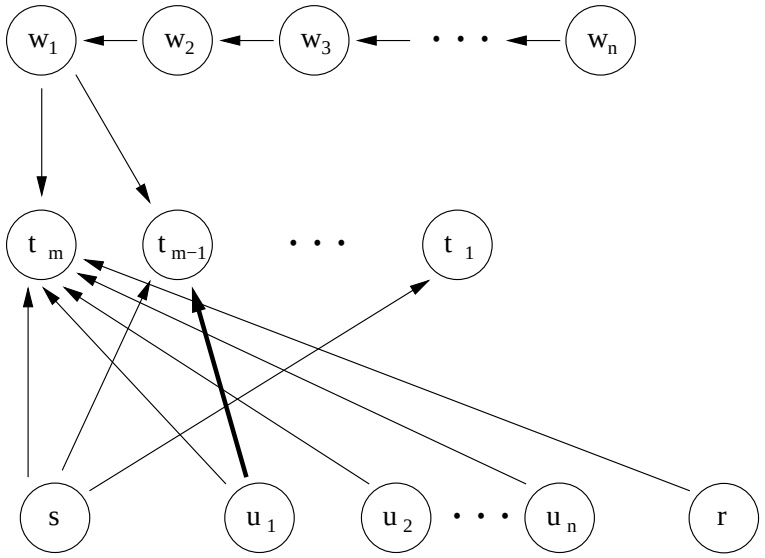

Figura 5.1: Pior caso para a complexidade em tempo.

5.4 Custo de Tempo Local

A complexidade de processamento local do algoritmo, dada pelo número de ciclos de CPU decorrentes da ativação espontânea ou do processamento decorrente do recebimento de uma mensagem é $O(|V|)$.

Uma alternativa à execução do DA seria a eleição de um líder para resolver localmente o problema e depois enviar a solução a todos os membros do grafo. Gastaríamos $O(|V|.|E|)$ mensagens e $O(|V|)$ tempo para concentrar todas as informações relevantes sobre G em um líder, a complexidade do DA seqüencial é $O(|E|.\min\{|E|, |T|.|V|\})$ [Hwang et al. 1992]. Considerando estas complexidades e as necessidades de memória no nó líder, a abordagem totalmente distribuída se mostra como uma alternativa atraente.

5.5 Conclusão

Neste capítulo provamos que o algoritmo termina em tempo finito fornecendo um grafo que contenha pelo menos uma boa solução para o problema de Steiner. Fornecemos limites analíticos de pior caso para as complexidades de tempo global, número de mensagens trocadas e tempo local. Como poderemos observar no próximo capítulo, o custo apresentado em situações práticas de execução do algoritmo é mais baixo que estes limites teóricos.

Capítulo 6

Resultados Experimentais

O algoritmo distribuído foi implementado em C e MPICH2-1.0.3 e executado em um *cluster* com 15 microcomputadores com processador Athlon com *clock* de 1.8 GHz. Utilizamos para os testes instâncias retiradas da *Steinlib* [Koch et al. 2000] e instâncias que tentam reproduzir as características da Internet.

Com referência às instâncias da *Steinlib*, metade dos testes foi executado sobre instâncias não direcionadas das séries B, I080 e P4Z. Instâncias da série B [Aneja 1980, Beasley 1984] são compostas por grafos aleatórios de densidades variadas e pesos de arestas entre 1 e 10. Instâncias da série I080 [Duin 1993] são grafos também aleatórios de diferentes densidades, mas com os pesos das arestas escolhidos para tornar as instâncias de resolução mais difícil. As instâncias da série P4Z [Chopra et al. 1992] são compostas por grafos completos.

Como os grafos da *Steinlib* são todos não direcionados e o algoritmo proposto foi projetado para trabalhar com o caso mais geral, dos grafos direcionados, criamos instâncias com pesos diferentes para os arcos entre o mesmo par de nós. Cada aresta do grafo original foi substituída por um par de arcos de sentidos opostos e custos iguais ao da aresta original multiplicado por um fator aleatório uniformemente distribuídos na faixa [0.5,1.5] e arredondados. Um terminal foi escolhido aleatoriamente para ser o nó raiz.

O restante dos testes foi realizado sobre grafos que tentam reproduzir as características da Internet. Estudos têm mostrado que o grau dos nós na rede induzida pela WWW (*World Wide Web*) e pela topologia física da Internet no nível dos roteadores e dos sistemas autônomos é regido por leis de potências [M. Faloutsos 1999]. E que a rede apresenta as características ditas *Small World*: o comprimento do caminho entre dois nós quaisquer é relativamente curto, e os nós encontram-se organizados em agrupamentos (*clusters*) muito

bem definidos [Watts e Strogatz 1998].

Geramos instâncias utilizando do sistema BRITE (*Boston University Representative Internet Topology Generator*) [Medina et al. 2001]. Dentre os vários modelos para geração de grafos disponíveis no BRITE, escolhemos o "Generalized Linear Preference" [Bu e Towsley 2002] para gerar os grafos da série GLP. Esse modelo utiliza leis de potência para determinação do grau dos nós e também características *Small World*. Utilizamos os parâmetros padrões para a geração de redes para simulação da Internet em nível de roteador. As instâncias geradas pelo BRITE não são orientadas.

As instâncias da série SW foram geradas segundo o modelo sugerido para grafos simuladores da Internet no nível de Sistemas Autônomos (*AS - Autonomous Systems*) em [Jin e Bestavros 2006]. Nessa classe, foram gerados grafos direcionados. Logo, é a única classe em que não temos necessariamente o arco no sentido inverso quando dois nós são conectados, apesar de sempre existir o caminho orientado entre quaisquer pares de nós.

Também implementamos versões distribuídas do DDMC [Shaikh e Shin 1997] e Prim-SPH [Novak et al. 2001a] (Kruskal-SPH e ADH não podem ser adaptados para grafos direcionados), incluindo o algoritmo para determinação das distâncias mínimas entre cada par de nós. O Prim-SPH foi aplicado sozinho, sobre o grafo original e para encontrar a solução contida no grafo de saturação conseguido com o Dual Ascent distribuído. O DDMC foi aplicado sobre as instâncias da *Steinlib* mas, por ter apresentado resultados muito pobres, foi abandonado nos testes seguintes com instâncias da Internet.

As tabelas 6.1, 6.2 e 6.3 apresentam os resultados para cada instância individual. As colunas têm o seguinte significado:

- $|V|$, ($|E|$ ou $|A|$), $|T|$ mostram o tamanho da instância (números de vértices, arestas ou arcos e terminais);
- **Otm** é o custo da solução ótima (calculado com o algoritmo *branch-and-bound* descrito em [Poggi de Aragão et al. 2001]);
- **DDMC** é o custo da solução obtida com o algoritmo DDMC, acrescido do custo do cálculo das distâncias mínimas entre cada par de nós do grafo;
- **SPH** é o custo da solução obtida com o algoritmo Prim-SPH utilizado sozinho, sobre o grafo original, acrescido do custo do cálculo das distâncias mínimas entre cada par de nós do grafo;
- **LI** é o limite inferior obtido com o Dual Ascent;

- $\frac{|Er|}{|E|}\%$ **ou** $\frac{|Ar|}{|A|}\%$ é a proporção de arestas/arcos saturados ao final da execução do Dual Ascent;

- **DAH** é o custo da solução obtida pela execução do Prim-SPH sobre o grafo de saturação obtido com o Dual Ascent.

Observe que os resultados obtidos pelo Dual Ascent representam valores médios de 5 execuções. A carga dos processadores e os tempos de comunicação podem alterar-se em cada execução, alterando o ritmo de crescimento dos fragmentos, levando o Dual Ascent a resultados ligeiramente diferentes. O desvio padrão médio nos resultados de todas as execuções foi de 0,4%.

A tabela 6.4 permite uma comparação entre os limites inferiores e os custos de solução obtidos pela versão distribuída apresentada e a versão seqüencial para as instâncias da *Steinlib*. Na implementação seqüencial dispomos de uma visão global do sistema e utilizamos este conhecimento na escolha da ordem de crescimento dos fragmentos. Isto melhora a qualidade dos limites inferiores e custos, portanto, podemos esperar da versão distribuída limites inferiores ligeiramente mais baixos e custos de solução ligeiramente mais altos do que com a versão seqüencial. De fato foi a situação que predominou nos testes práticos. O limite inferior da versão seqüencial ficou em média 2,9% abaixo do ótimo, enquanto a seqüencial 0,8%. O custo da solução na versão seqüencial ficou em média 5,1% acima do ótimo, enquanto o valor correspondente para a implementação seqüencial manteve-se em média 3,4% acima do ótimo. O desvio padrão para todas as medidas mostrou-se baixo (todas abaixo de 5,6). A fim de facilitar a visualização, organizamos a tabela com as colunas em ordem crescente de valores esperados.

Utilizamos *competitividade*, a razão entre o custo da árvore obtida pela heurística e o custo da árvore ótima para comparar a qualidade das soluções obtidas com cada método. Os gráficos das figuras 6.1, 6.2 e 6.3 mostram a percentagem cumulativa de casos em que a competitividade é menor ou igual ao custo ótimo acrescido de um fator que varia no eixo das abscissas. Está claro que com o DAH obtêm-se soluções de melhor qualidade do que com o SPH, e a diferença é muito maior quando comparamos com o DDMC.

Como pode ser visto nas tabelas 6.1, 6.2 e 6.3, em uma minoria dos casos o SPH provê resultados de melhor qualidade do que o DAH. Portanto, se executarmos os dois algoritmos e tomarmos a solução de melhor qualidade, certamente conseguiremos resultados melhores. Os valores assim obtidos aparecem nos gráficos sob o rótulo DAH+SPH. Entretanto, existe uma abordagem mais interessante para se obter resultados semelhan-

Instância					DDMC	SPH	Dual Ascent		DAH
Nome	$\|V\|$	$\|E\|$	$\|T\|$	Otm			LI	$\frac{\|Er\|}{\|E\|}\%$	
b01	50	63	9	82	88	85	82,0	27,8	**84,6**
b02	50	63	13	83	100	84	82,4	26,9	**83,8**
b03	50	63	25	138	149	**138**	136,4	25,3	139,2
b04	50	100	9	59	110	64	58,8	27,3	**59,6**
b05	50	100	13	61	87	**62**	60,8	27,3	**62,0**
b06	50	100	25	122	168	**127**	121,6	27,1	132,2
b07	75	94	13	111	159	**111**	110,8	27,5	112,0
b08	75	94	19	104	157	**104**	104,0	27,7	106,0
b09	75	94	38	220	259	**220**	216,0	23,4	221,2
b10	75	150	13	86	141	91	86,0	27,7	**89,8**
b11	75	150	19	88	140	**90**	87,8	27,6	90,8
b12	75	150	38	174	232	**174**	174,0	27,7	175,0
b13	100	125	17	165	223	**174**	155,2	19,0	180,0
b14	100	125	25	235	265	**238**	230,6	20,1	238,4
b15	100	125	50	318	365	**318**	317,6	27,3	320,4
b16	100	200	17	127	195	136	124,6	26,3	**132,4**
b17	100	200	25	131	177	**132**	128,2	26,1	133,0
b18	100	200	50	218	315	225	215,0	26,0	**218,0**
i080-001	80	120	6	1.787	2.187	2.164	1.770,6	39,7	**1.787,0**
i080-011	80	350	6	1.479	3.202	1.671	1.462,4	44,7	**1.596,0**
i080-021	80	3.160	6	1.175	5.760	**1.471**	1.159,8	32,1	1.476,0
i080-031	80	160	6	1.570	6.571	**1.570**	1.570,0	34,1	**1.570,0**
i080-041	80	632	6	1.276	2.251	1.600	1.192,8	38,2	**1.279,0**
i080-101	80	120	8	2.608	2.662	3.009	2.608,0	34,6	**2.772,0**
i080-111	80	350	8	2.051	6.071	**2.142**	1.792,6	31,1	2.262,8
i080-121	80	3.160	8	1.561	6.160	2.054	1.539,4	34,1	**1.844,0**
i080-131	80	160	8	2.284	2.879	2.561	2.238,0	39,1	**2.555,6**
i080-141	80	632	8	1.788	2.648	2.058	1.662,4	45,9	**1.865,2**
i080-201	80	120	16	4.760	5.073	5.435	4.555,2	41,2	**5.129,0**
i080-211	80	350	16	3.631	6.268	4.132	3.259,6	33,4	**3.839,0**
i080-221	80	3.160	16	3.158	3.600	4.386	3.063,6	29,2	**3.499,0**
i080-231	80	160	16	4.354	5.291	**4.834**	3.961,0	44,5	4.917,0
i080-241	80	632	16	3.538	4.566	4.463	3.247,6	40,6	**3.805,0**
i080-301	80	120	20	5.519	5.855	**5.628**	5.223,4	30,4	5.630,0
i080-311	80	350	20	4.554	5.482	5.853	4.215,6	38,5	**5.115,8**
i080-321	80	3.160	20	3.932	4.065	5.527	3.925,0	22,2	**4.461,0**
i080-331	80	160	20	5.226	5.407	5.947	4.831,2	43,2	**5.589,0**
i080-341	80	632	20	4.236	5.567	5.728	3.997,2	34,6	**4.529,0**
P401	100	4.950	5	155	193	170	149,8	0,8	**157,0**
P402	100	4.950	5	116	206	**116**	116,0	0,4	**116,0**
P403	100	4.950	5	179	398	**184**	176,0	1,0	199,0
P404	100	4.950	10	270	943	**270**	243,2	1,0	**270,0**
P405	100	4.950	10	270	912	291	268,6	0,7	**270,0**
P406	100	4.950	10	290	629	319	286,0	1,0	**290,0**
P407	100	4.950	20	590	2261	**601**	576,8	0,7	609,0
P408	100	4.950	20	542	1.170	559	520,4	0,7	**551,0**

Tabela 6.1: Resultados para instâncias da SteinLib não direcionadas.

Instância					DDMC	SPH	Dual Ascent		DAH
Nome	$\|V\|$	$\|A\|$	$\|T\|$	Otm			LI	$\frac{\|Ar\|}{\|A\|}\%$	
b01d	50	126	9	77	107	82	77,0	13,9	**77,0**
b02d	50	126	13	101	145	107	100,4	13,4	**104,0**
b03d	50	126	25	170	175	176	165,4	10,3	**173,4**
b04d	50	200	9	58	75	61	57,4	13,5	**59,6**
b05d	50	200	13	61	85	**64**	60,6	13,6	65,4
b06d	50	200	25	128	179	134	126,8	13,2	**129,6**
b07d	75	188	13	122	201	**122**	121,6	13,6	125,0
b08d	75	188	19	115	172	126	114,8	13,7	**116,0**
b09d	75	188	38	240	282	244	239,2	13,4	**242,4**
b10d	75	300	13	90	119	**91**	89,0	13,5	92,6
b11d	75	300	19	103	143	**104**	100,8	13,1	105,2
b12d	75	300	38	168	269	**171**	161,0	11,5	174,0
b13d	100	250	17	165	**168**	202	163,8	13,3	179,4
b14d	100	250	25	235	**250**	261	234,0	13,4	250,4
b15d	100	250	50	342	370	356	341,0	13,4	**345,6**
b16d	100	400	17	133	170	**133**	131,6	13,4	141,6
b17d	100	400	25	139	180	**142**	137,0	13,3	145,2
b18d	100	400	50	258	266	271	258,0	13,8	**259,8**
i080-001d	80	240	6	1.751	2.276	1.815	1.729,4	16,3	**1.780,8**
i080-011d	80	700	6	1.220	1.585	1.296	1.220,0	17,4	**1.227,6**
i080-021d	80	6.320	6	741	961	847	673,0	5,8	**768,4**
i080-031d	80	320	6	1.514	1.970	1.706	1.455,0	16,5	**1.552,8**
i080-041d	80	1264	6	946	1.230	1.026	900,0	5,9	**983,0**
i080-101d	80	240	8	2.322	3.009	2.706	2.322,0	10,5	**2.322,0**
i080-111d	80	700	8	1.580	2.054	1.600	1.499,2	13,2	**1.580,2**
i080-121d	80	6.320	8	977	1.250	1.097	910,0	5,3	**1.050,0**
i080-131d	80	320	8	1.875	2.508	2.061	1.844,0	17,9	**2.016,0**
i080-141d	80	1.264	8	1.404	1.825	1.596	1.265,6	10,1	**1.404,0**
i080-201d	80	240	16	4.321	4.335	4.502	4.228,4	22,7	**4.322,8**
i080-211d	80	700	16	2.951	3.825	3.174	2.808,6	12,8	**3.141,0**
i080-221d	80	6.320	16	1.985	2.589	2.293	1.799,0	5,1	**2.082,0**
i080-231d	80	320	16	4.156	5.403	4.623	4.036,0	14,0	**4.415,2**
i080-241d	80	1.264	16	2.492	3.250	**2.617**	2.351,2	8,7	2.746,8
i080-301d	80	240	20	5.714	7.402	6.365	5.390,8	24,8	**5.916,0**
i080-311d	80	700	20	3.471	4.553	**3.813**	3.266,6	12,9	4.065,0
i080-321d	80	6.320	20	2.453	3.187	3.137	2.374,0	4,1	**2.755,4**
i080-331d	80	320	20	4.506	5.800	4.911	4.258,6	19,3	**4.877,0**
i080-341d	80	1.264	20	3.132	4.072	**3.361**	3.025,0	15,4	3.412,0
P401d	100	9.900	5	145	295	165	145,0	0,4	**145,0**
P402d	100	9.900	5	102	152	**102**	102,0	0,2	**102,0**
P403d	100	9.900	5	169	189	186	166,0	0,5	**180,0**
P404d	100	9.900	10	270	305	**279**	214,8	0,3	285,0
P405d	100	9.900	10	248	305	250	243,2	0,5	**248,0**
P406d	100	9.900	10	281	350	303	226,8	0,3	**296,0**
P407d	100	9.900	20	546	658	590	523,0	0,4	**575,0**
P408d	100	9.900	20	502	821	520	480,2	0,3	**502,0**

Tabela 6.2: Resultados para instâncias da SteinLib direcionadas.

Instância					SPH	Dual Ascent		DAH
Nome	$\|V\|$	$\|A\|$	$\|T\|$	Otm		LI	$\frac{\|Ar\|}{\|A\|}\%$	
glp1	100	334	5	3.901	**3.931**	3.901,0	7,9	**3.931,0**
glp2	100	358	5	4.071	4.430	4.071,0	7,7	**4.169,0**
glp3	100	374	5	3.952	4.437	3.873,0	7,2	**4.245,0**
glp4	100	356	5	4.197	**4.576**	4.145,0	7,8	**4.576,0**
glp5	100	336	5	4.025	4.805	4.016,0	7,8	**4.122,0**
glp1	100	334	10	7.005	**7.155**	6.947,8	7,7	7.181,6
glp2	100	358	10	6.325	**6.400**	6.325,0	7,8	**6.400,0**
glp3	100	374	10	6.981	7.435	6.978,2	7,9	**7.353,8**
glp4	100	356	10	6.654	7.299	6.616,0	6,7	**7.258,0**
glp5	100	336	10	5.593	**5.651**	5.359,0	7,9	5.682,0
sw1	100	1.243	5	117.665	150.633	115.184,0	6,1	**138.950,0**
sw2	100	1.026	5	108.151	165.396	108.126,8	6,1	**108.496,0**
sw3	100	1.125	5	655.510	**665.396**	642.330,0	4,2	678.153,0
sw4	100	1.555	5	62.341	65.991	60.710,4	4,5	**64.698,4**
sw5	100	860	5	135.675	141.099	131.494,0	6,1	**137.928,0**
sw1	100	1.243	10	124.776	**140.012**	124.105,0	5,8	150.685,0
sw2	100	1.026	10	161.999	163.077	161.465,2	5,8	**162.299,2**
sw3	100	1.125	10	186.062	**191.000**	17.990,0	5,8	**191.000,0**
sw4	100	1.555	10	128.074	191.733	106.268,0	8,8	**168.972,0**
sw5	100	860	10	169.200	**171.503**	167.012,0	9,9	171.530,0

Tabela 6.3: Resultados para instâncias da Internet.

tes. Utilizando o limite inferior, podemos avaliar a solução obtida com o DAH e somente quando a qualidade estiver abaixo de um determinado limite, executar o SPH. Aplicamos essa estratégia e executamos o SPH somente quando DAH/LI ultrapassa 1,05. Isso aconteceu em 49% dos casos e os resultados foram praticamente idênticos aos já conseguidos em DAH+SPH, tornando as curvas DAH+SPH e "DAH+SPH dependendo de LI" indistinguíveis nas instâncias da *Steinlib*.

Também comparamos o desempenho prático em relação ao tempo, dado pelo tamanho da maior cadeia de mensagens com relação de causalidade e o número total de mensagens enviadas.

Essas medidas podem ser observadas nos gráficos das figuras 6.4 a 6.9. Pode ser visto que o DAH apresenta custos superiores ao SPH na maioria das instâncias, gastando mais tempo e enviando maior número de mensagens. No entanto, o aumento não é muito grande, ficando as diferenças de desempenho sempre abaixo de um pequeno fator 4.

Nas instâncias maiores da série I080 e em toda a série P4Z, o DAH apresentou custos inferiores ao SPH. Isso se deve ao grafo de saturação obtido com o DA ser de tamanho muito inferior ao grafo original, como pode ser observado nas tabelas 6.1 e 6.2. Como

Instância				Limite Inferior		Ótimo	Custo da Solução	
Nome	$\|V\|$	$\|E\|$	$\|T\|$	Distr.	Seq.		Seq.	Distr.
b01	50	63	9	82,0	82,0	82,0	82,0	84,6
b02	50	63	13	82,4	83,0	83,0	83,0	83,8
b03	50	63	25	136,4	138,0	138,0	139,0	139,2
b04	50	100	9	58,8	59,0	59,0	59,0	59,6
b05	50	100	13	60,8	61,0	61,0	61,0	62,0
b06	50	100	25	121,6	121,0	122,0	122,0	132,2
b07	75	94	13	110,8	111,0	111,0	111,0	112,0
b08	75	94	19	104,0	104,0	104,0	104,0	106,0
b09	75	94	38	216,0	220,0	220,0	220,0	221,2
b10	75	150	13	86,0	86,0	86,0	86,0	89,8
b11	75	150	19	87,8	88,0	88,0	88,0	90,8
b12	75	150	38	174,0	174,0	174,0	174,0	175,0
b13	100	125	17	155,2	165,0	165,0	165,0	180,0
b14	100	125	25	230,6	235,0	235,0	235,0	238,4
b15	100	125	50	317,6	318,0	318,0	318,0	320,4
b16	100	200	17	124,6	127,0	127,0	127,0	132,4
b17	100	200	25	128,2	131,0	131,0	131,0	133,0
b18	100	200	50	215,0	218,0	218,0	218,0	218,0
i080-001	80	120	6	1.770,6	1.787,0	1.787,0	1.787,0	1.787,0
i080-011	80	350	6	1.462,4	1.401,0	1.479,0	1.488,0	1.596,0
i080-021	80	3.160	6	1.159,8	1.175,0	1.175,0	1.175,0	1.476,0
i080-031	80	160	6	1.570,0	1.570,0	1.570,0	1.570,0	1.570,0
i080-041	80	632	6	1.192,8	1.259,0	1.276,0	1.375,0	1.279,0
i080-101	80	120	8	2.608,0	2.594,0	2.608,0	2.693,0	2.772,0
i080-111	80	350	8	1.792,6	1.901,0	2.051,0	2.056,0	2.262,8
i080-121	80	3.160	8	1.539,4	1.559,0	1.561,0	1.737,0	1.844,0
i080-131	80	160	8	2.238,0	2.185,0	2.284,0	2.285,0	2.555,6
i080-141	80	632	8	1.662,4	1.713,0	1.788,0	1.971,0	1.865,2
i080-201	80	120	16	4.555,2	4.760,0	4.760,0	4.760,0	5.129,0
i080-211	80	350	16	3.259,6	3.573,0	3.631,0	3.822,0	3.839,0
i080-221	80	3.160	16	3.063,6	3.150,0	3.158,0	3.304,0	3.499,0
i080-231	80	160	16	3.961,0	4.277,0	4.354,0	4.462,0	4.917,0
i080-241	80	632	16	3.247,6	3.408,0	3.538,0	3.642,0	3.805,0
i080-301	80	120	20	5.223,4	5.519,0	5.519,0	5.519,0	5.630,0
i080-311	80	350	20	4.215,6	4.534,0	4.554,0	5.213,0	5.115,8
i080-321	80	3.160	20	3.925,0	3.932,0	3.932,0	3.932,0	4.461,0
i080-331	80	160	20	4.831,2	5.132,0	5.226,0	5.226,0	5.589,0
i080-341	80	632	20	3.997,2	4.189,0	4.236,0	4.517,0	4.529,0
P401	100	4.950	5	149.8	155,0	155,0	155,0	157,0
P402	100	4.950	5	116,0	116,0	116,0	116,0	116,0
P403	100	4.950	5	176,0	179,0	179,0	179,0	199,0
P404	100	4.950	10	243,2	270,0	270,0	270,0	270,0
P405	100	4.950	10	268,6	270,0	270,0	270,0	270,0
P406	100	4.950	10	286,0	288,0	290,0	290,0	290,0
P407	100	4.950	20	576,8	590,0	590,0	590,0	609,0
P408	100	4.950	20	520,4	542,0	542,0	542,0	551,0

Tabela 6.4: Resultados obtidos com as versões paralela e distribuída do Dual Ascent.

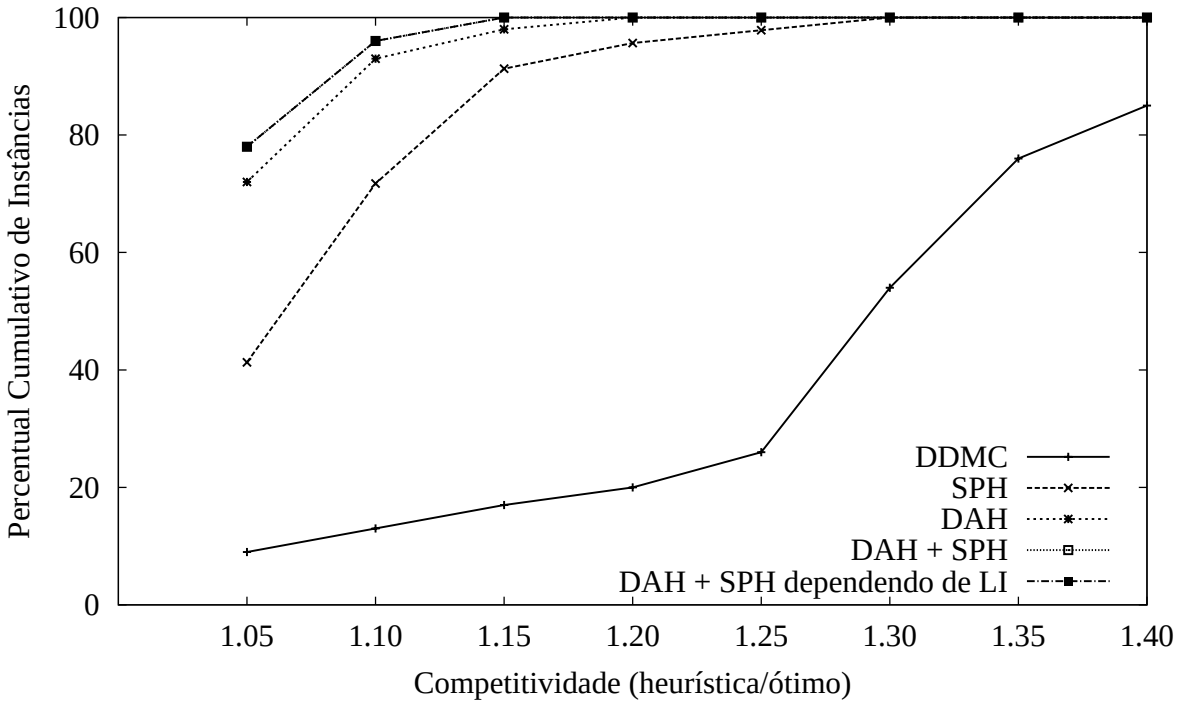

Figura 6.1: Qualidade da solução – Instâncias da SteinLib direcionadas.

o custo de execução do DA depende pouco do número de arcos do grafo, seu custo de execução foi compensado pela economia obtida na execução do algoritmo de distância mínima e o Prim-SPH sobre uma instância menor.

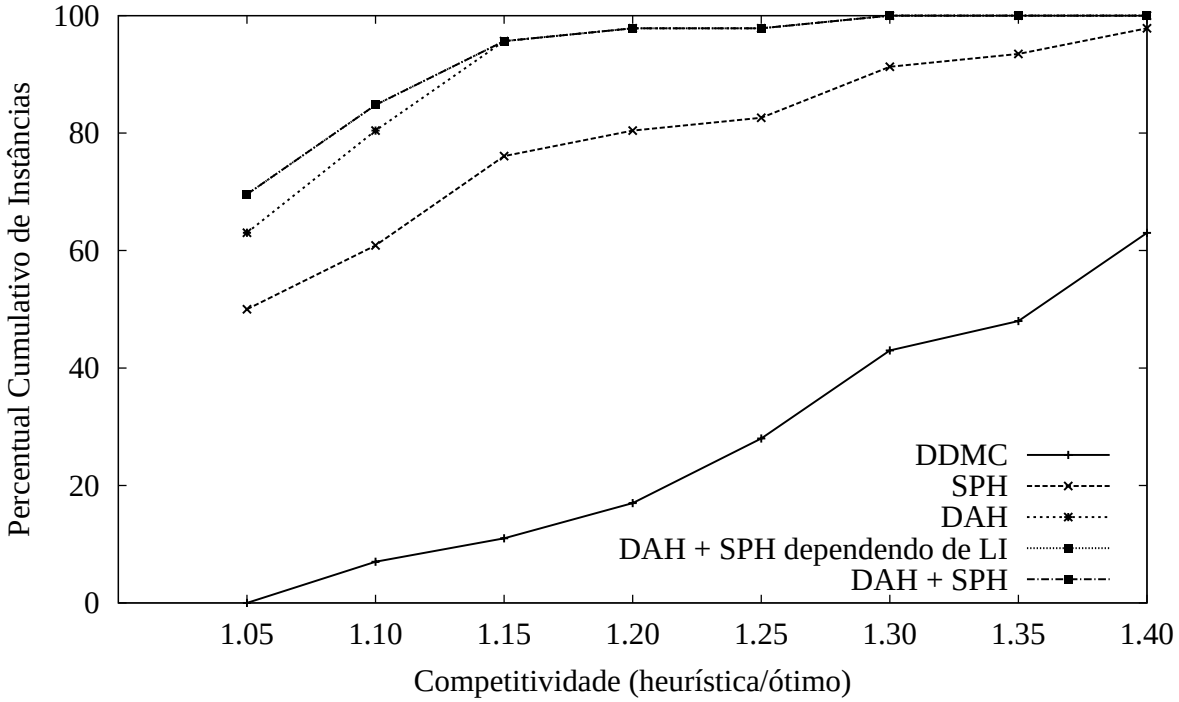

Figura 6.2: Qualidade da solução – Instâncias da SteinLib não direcionadas.

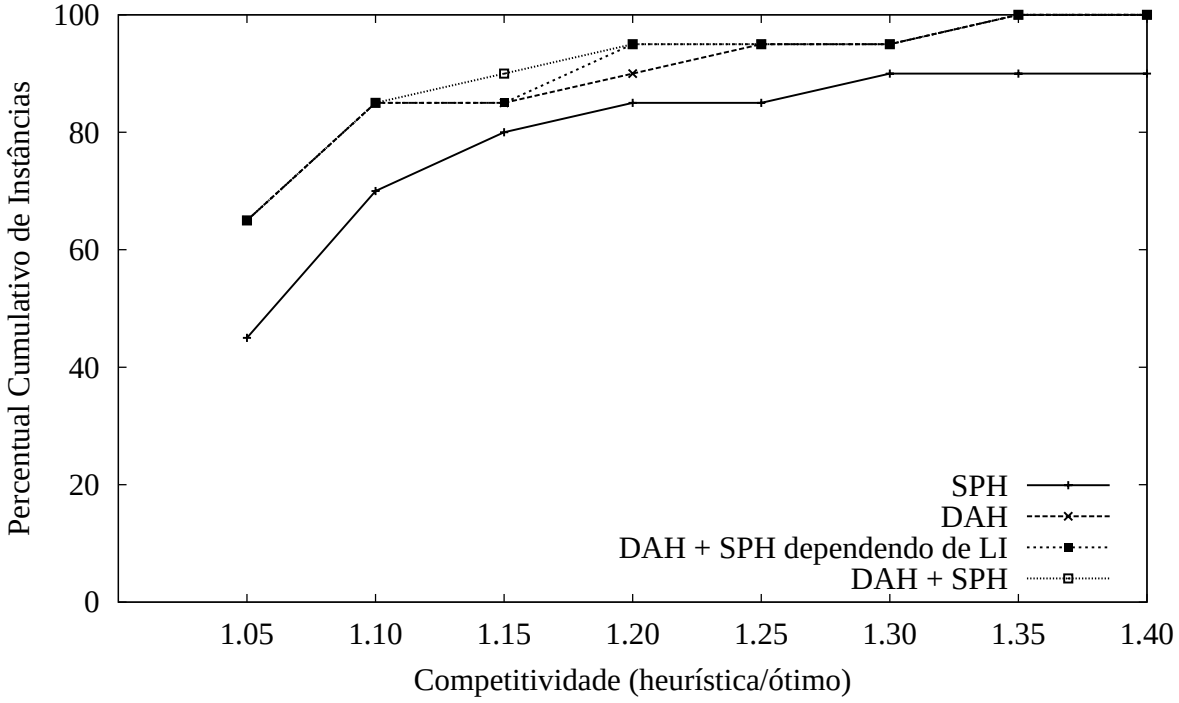

Figura 6.3: Qualidade da solução – Instâncias da Internet.

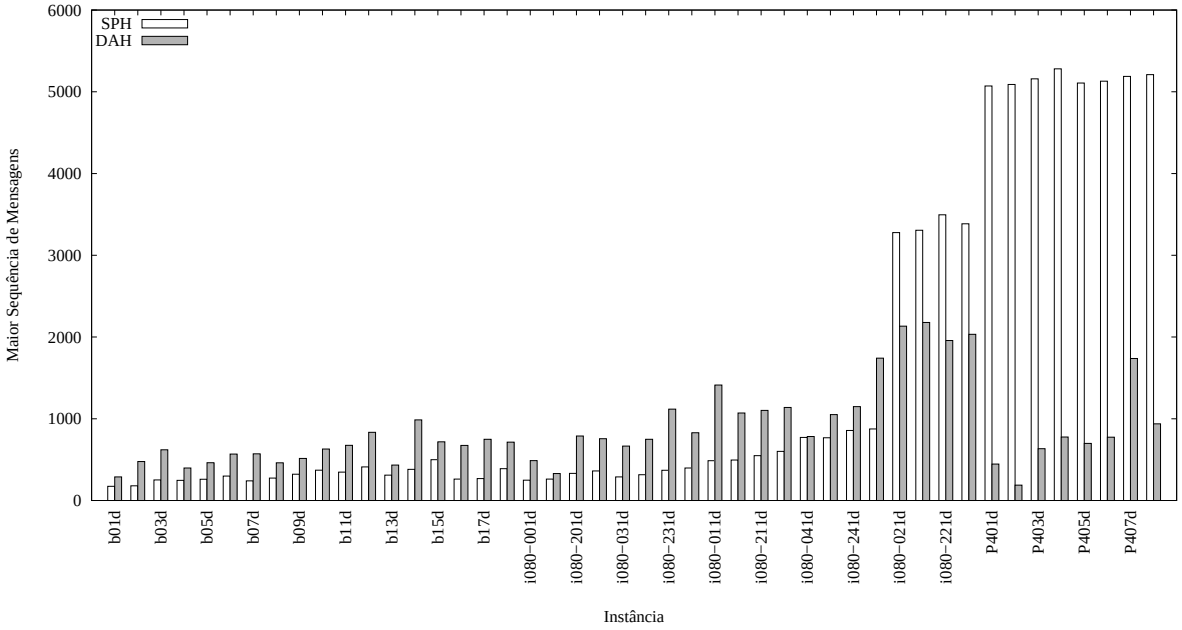

Figura 6.4: Tempo – Instâncias da SteinLib direcionadas.

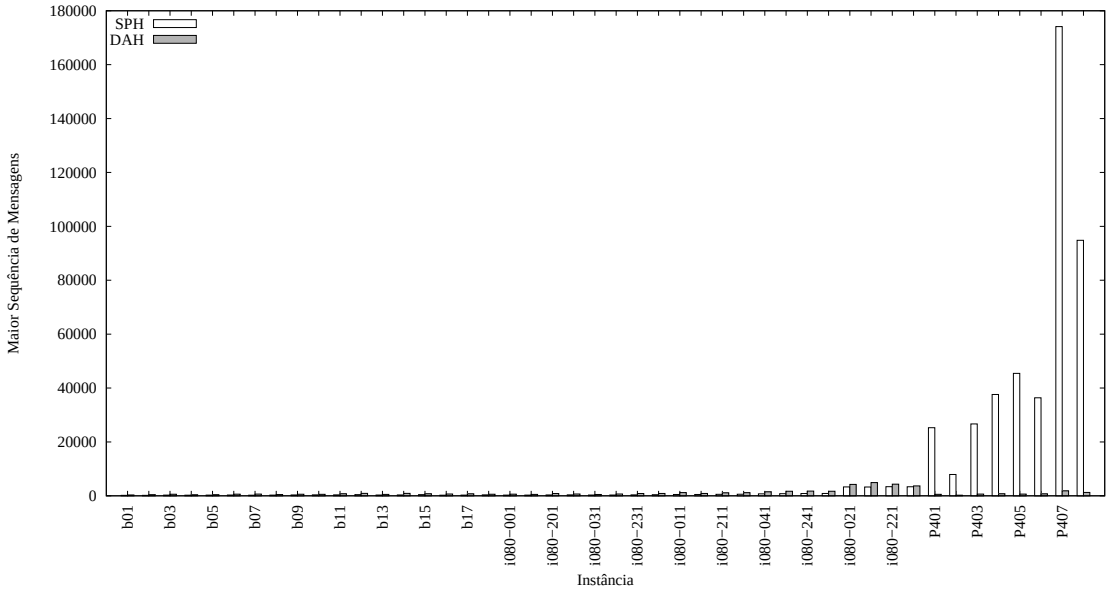

Figura 6.5: Tempo – Instâncias da SteinLib não direcionadas.

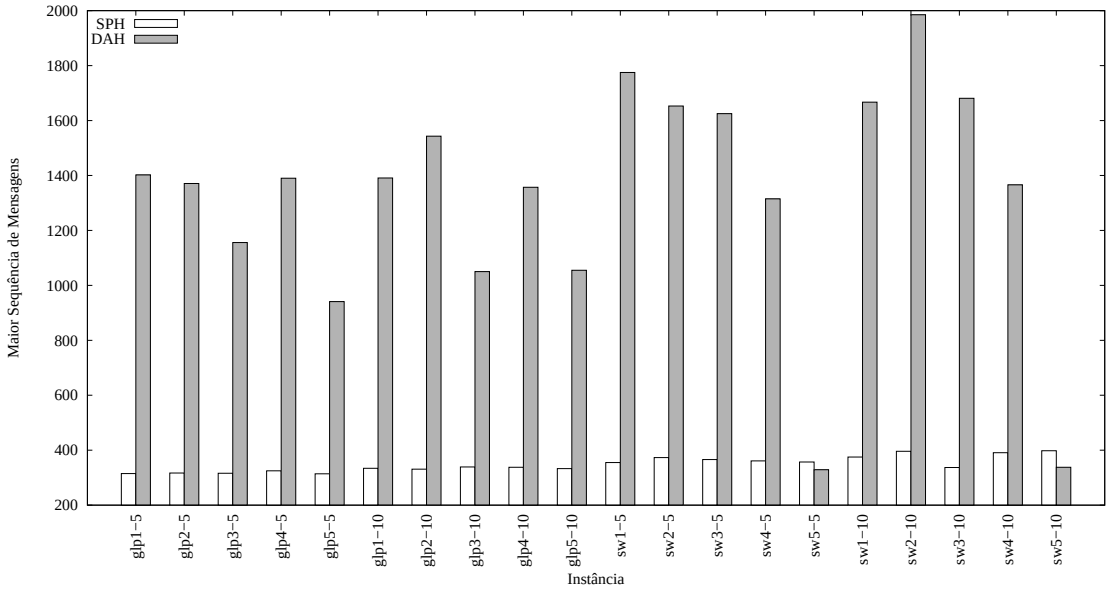

Figura 6.6: Tempo – Instâncias da Internet.

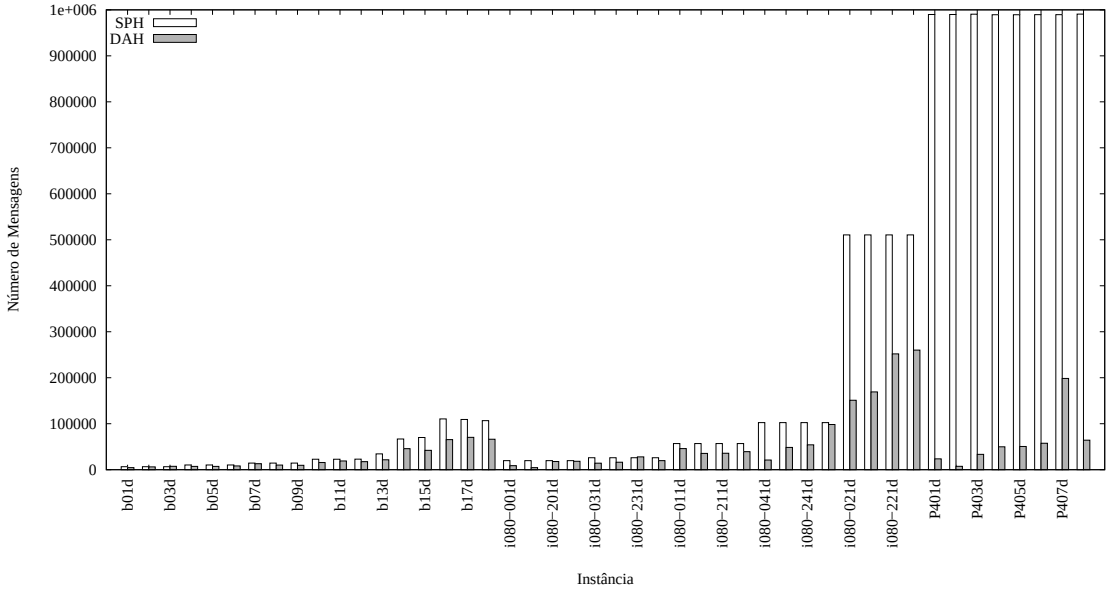

Figura 6.7: Mensagens – Instâncias da SteinLib direcionadas.

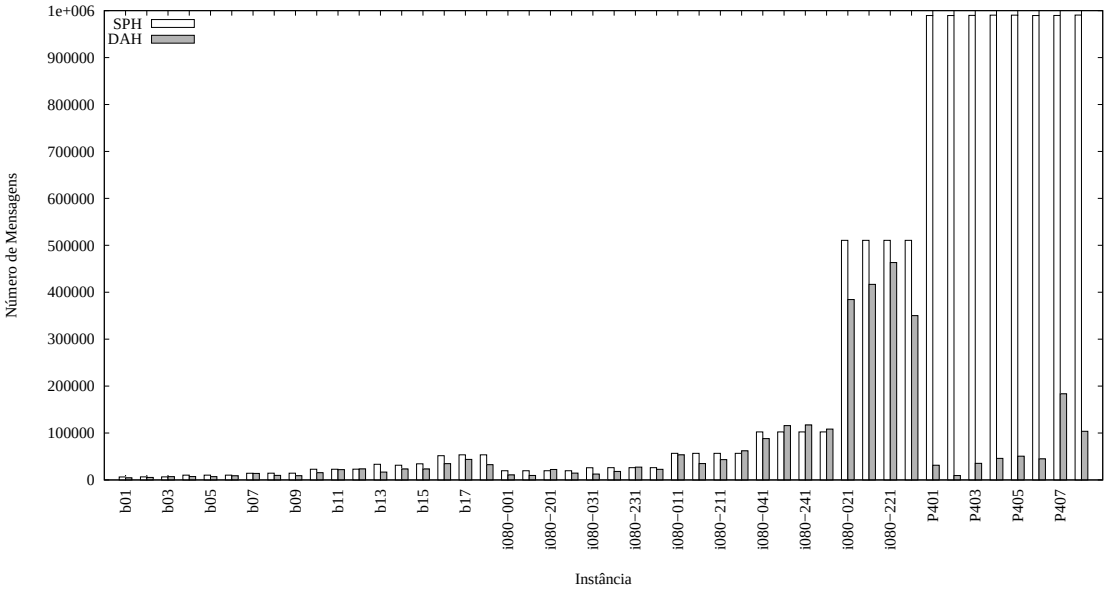

Figura 6.8: Mensagens – Instâncias da SteinLib não direcionadas.

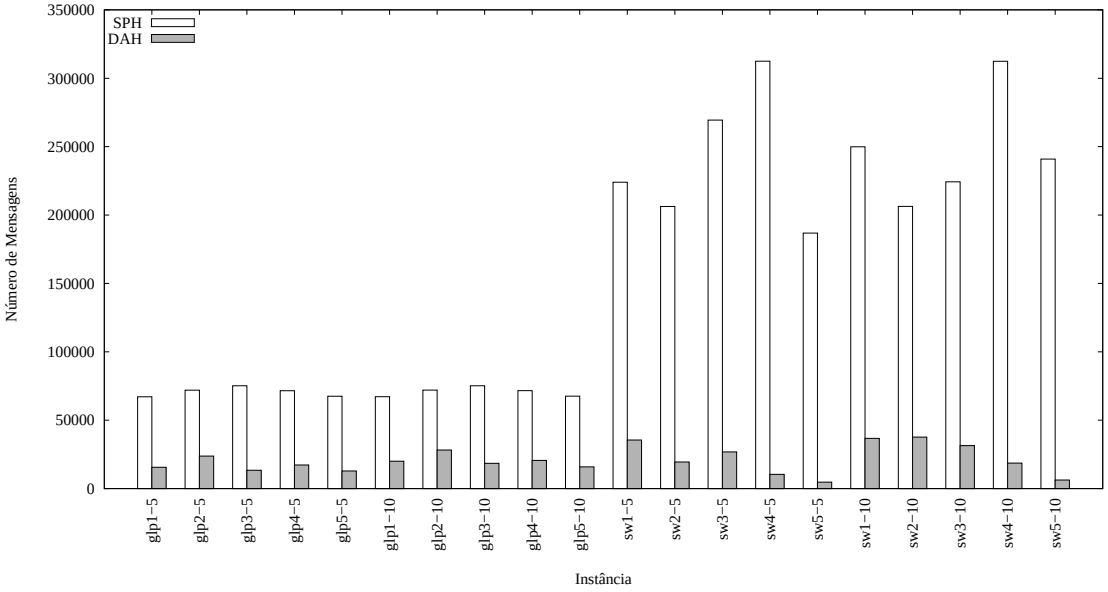

Figura 6.9: Mensagens – Instâncias da Internet.

6.1 Conclusão

Apresentamos resultados computacionais da execução do Dual Ascent distribuído sobre um número razoavelmente grande e variado de instâncias, mostrando que a versão distribuída apresenta resultados comparáveis à seqüencial, em média de melhor qualidade que os principais algoritmos distribuídos conhecidos para o problema. Os custos computacionais do Dual Ascent não apresentam ritmo de crescimento diferente do algoritmo Prim-SPH, utilizado como base de comparação, sendo até mesmo menor para grafos muito densos, desde que levemos em consideração o custo do cálculo da distância mínima entre cada par de nós, necessário para o Prim-SPH.

Capítulo 7

Problema de Steiner com Limitação de Saltos

A variação do problema de Steiner no qual devemos, além de minimizar o custo da solução, manter uma segunda métrica abaixo de um determinado limite, chamada "Problema de Steiner com Restrições", é de grande interesse prático, pois a distribuição de multimídia em *multicast* freqüentemente é melhor modelada desta forma. Desejamos minimizar o custo da arborescência, mas a aplicação não tolera que a latência fique acima de determinado valor, por exemplo.

Mais formalmente, o Problema de Steiner com Restrições pode ser definido da seguinte forma. Dados:

- o grafo $G_d = (V, A)$ onde V é o conjunto vértices e A o conjunto de arcos;
- um terminal raiz $r \in V$;
- dois números reais positivos c e d associados a cada aresta;
- um número real l;
- um conjunto $T \subseteq V$, de terminais;

encontrar

$G'_d = (V', A')$

tal que

- existam caminhos de r a todo $t \in T$, sendo $path(r, t)$ o conjunto de arcos componentes do caminho de r até t;

- $\sum_{a \in A'} c_a$ seja mínimo;
- para todo $t \in T, \sum_{p \in path(r,t)} d_p < l$.

Para a maioria das redes utilizadas atualmente, o processamento local nos roteadores domina o crescimento da latência de comunicação. Portanto, uma simplificação comumente realizada na prática é associar a latência de comunicação ao número de conexões ponto a ponto necessárias para a transmissão completa, da origem ao destino final. Abordamos o problema como se cada canal de comunicação intermediário acrescentasse um valor constante igual a um à latência de comunicação completa, origem-destino, ou seja, $d_a = 1$, $\forall\ a \in A$, e l igual ao número máximo de saltos permitidos, que chamamos de H. Denominaremos o "Problema de Steiner em Digrafos com Limitação de Saltos" de **hcSPDG**.

7.1 Algumas Heurísticas Seqüenciais para o Problema com Restrições

A seguir relacionamos as principais abordagens para o o problema de Steiner com restrições encontradas na literatura.

[Kompella et al. 1993a] sugerem a construção de um grafo completo cujo conjunto de nós é o conjunto de terminais e as arestas são representações dos caminhos de menor custo, que não violem a restrição. Computa a árvore geradora mínima deste grafo e depois retorna ao grafo original. Ao executar a operação de retorno, ciclos são removidos com o cuidado de não se violar a restrição para o problema.

Alguns autores [Raghavan et al. 1999, Jia 1998] abordam o problema de forma semelhante ao CHINS. Descobrem-se os caminhos mínimos entre os nós respeitadas as restrições, e eles vão sendo incluídos na solução, em ordem crescente de custos até que existam caminhos entre todos os terminais. Novamente, a cada caminho mínimo incluído, ciclos podem surgir, e devem ser removidos, com o cuidado para que as restrições não sejam violadas.

[Feng e Yum 1999] propõem a construção de duas árvores, cada uma tentando minimizar um dos critérios; unindo-se as árvores, obteremos uma solução com ciclos que devem ser removidos, minimizando o custo sem que as restrições sejam violadas.

[Narang et al. 1999] avaliam a procura da solução a partir de um nó central, achado

no ponto médio do caminho mínimo que não viole a restrição entre os dois terminais mais distantes. Uma solução parcial é composta por esse caminho. A cada nó componente dessa solução é associado um valor residual que ainda pode ser suportado pela solução com referência à restrição. A solução final é construída ligando-se cada terminal à solução parcial pelo caminho mínimo que não viole a restrição.

O algoritmo BSMA (*Bounded Shortest Multicast Algorithm*) [Alrabiah 1999] propõe a construção de uma árvore geradora minimizando a métrica associada à restrição e gradualmente substituir caminhos, de forma a reduzir o custo. Obviamente, antes de cada substituição deve-se verificar se não houve violação da restrição.

7.2 Algumas Heurísticas Distribuídas para o Problema com Restrições

Destacamos a seguir as principais heurísticas distribuídas conhecidas para o hcSPDG.

O DCSP (*Delay-Constrained Shortest Path*) [Mokbel et al. 1999] propaga, a partir do nó raiz, *tokens* em inundação, que carregam informações sobre as métricas relevantes para otimização durante seu trajeto até o nó terminal. De posse dos diversos caminhos possíveis, o terminal elege o de menor custo que não viole a restrição e o encaminha para a raiz para que possa ser unido ao eleito dos outros terminais para formar a solução do problema. Os autores sugerem um valor máximo para o diâmetro do grafo, K, para limitação da propagação dos *tokens*, e apresenta a complexidade de operações $O(d.K^2.|V|^2)$ sendo d o grau médio dos nós.

Tomando por base o DDMC [Shaikh e Shin 1997], abordado no Capítulo 2, [Guo e Matta 1999] introduzem o QDMR (*QoS Dependent Multicast Routing*). Esse algoritmo decide qual rota deve ser utilizada para se um atingir um terminal com base em uma função com dois componentes: um custo associado a métrica cuja minimização é desejada, obtido como no DDMC e um valor residual referente a métrica associada a restrição. Esses componentes recebem pesos diferentes na decisão sobre a escolha do próximo nó para uma rota. À medida que o valor residual diminui e nos aproximamos de violar a restrição, o peso do componente referente a esse resíduo é aumentado, diminuindo-se a probabilidade de violação da restrição. O algoritmo não garante que a restrição não seja violada, apenas diminui a probabilidade disso acontecer.

Versões do Prim-SPH adaptadas para trabalhar com restrições podem ser encontradas em [Jia 1998, Kompella et al. 1993b]. A idéia básica do algoritmo é muito semelhante ao

Prim-SPH. Inicia-se a solução parcial com um nó raiz. A cada iteração, o terminal mais próximo da solução parcial é incluído a ela utilizando o caminho mínimo entre a solução parcial e o terminal. Todos os nós do caminho entre o terminal e a solução parcial são também incluídos na solução. Esse processo se repete até que todos os terminais estejam contidos na solução que, neste ponto, deixará de ser parcial. A diferença principal para o Prim-SPH já apresentado, sem restrições, é que apenas caminhos que não violem o limite imposto pela restrição podem ser considerados.

A implementação sugerida em [Jia 1998] utiliza duas estruturas de dados básicas:

- uma tabela chamada T2D (*tree to destination*), com uma entrada para cada elemento do conjunto de terminais e as seguintes informações: o custo de conexão do terminal até a solução parcial, a partir de que nó da solução a conexão deve ser iniciada, e uma variável lógica indicando se o terminal ainda aguarda por ser conectado à solução parcial;
- cada nó que compõe a solução parcial armazena o valor acumulado dele mesmo até a raiz, referente à métrica associada à restrição.

O algoritmo assume que protocolos das camadas inferiores disponibilizam tabelas contendo valores de custo e da métrica associada à restrição de cada nó para todos os demais. Assim, ao ser inicializado com a solução parcial contendo apenas a raiz, os valores para a primeira versão da tabela T2D podem ser conseguidos diretamente consultando essas tabelas. O nó raiz seleciona o terminal mais próximo para incluir na solução e envia a tabela T2D para o primeiro nó do caminho entre ele e este terminal. A tabela será passada de nó em nó até atingir o terminal a ser conectado. Cada nó que recebe a tabela se considera incluído na solução parcial, computa o valor da métrica associada à restrição da raiz até ele e verifica se dispõe de rota com custo mais baixo para algum terminal que não viole a restrição. Se possuir, ele atualiza T2D antes de enviá-la ao próximo nó a ser incluído. Cada terminal recém-conectado possui todas as informações necessárias para decidir qual o próximo terminal a ser incluído na solução parcial, e a partir de qual nó da solução parcial atual devem iniciar as inclusões para incluir este terminal (ponto de derivação). Desta forma, o nó recém incluído pode transmitir a tabela para o ponto de derivação e o algoritmo prossegue até que todos os terminais tenham sido incluídos.

[Huang e Lee 2005] apontam um erro na proposta original do Prim-SPH com restrições. Ciclos poderiam ser formados na solução parcial pois, a fim de manter a restrição, caminhos sub-ótimos do ponto de vista do custo poderiam ser preferidos a caminhos mais

curtos utilizados anteriormente para a conexão de outro terminal. Para solução, Huang sugere que um nó que já possua um pai na solução inicial, ao receber um novo pedido de conexão de origem diferente, rompa a conexão mais antiga, mantendo a integridade da árvore. Esse procedimento pode gerar folhas na árvore que não sejam terminais. Portanto, uma fase de limpeza que exclua essas folhas desnecessárias deve ser incluída no algoritmo.

Não é de nosso conhecimento a existência de um algoritmo distribuído específico para o Problema de Steiner Direcionado com Limitação de Saltos e não constam na literatura comparações de desempenho entre as heurísticas citadas para o hcSPDG. Como o Prim-SPH é a heurística que apresentou melhores resultados para o SPDG, com a qualidade da solução suplantada apenas pelo Dual Ascent, adaptamos o algoritmo de [Jia 1998] para o hcSPH, com as correções posteriormente sugeridas por [Huang e Lee 2005], para fins de balizamento e comparações com o Dual Ascent com restrições de saltos.

[Gouveia et al. 2007] introduzem uma forma de utilizarmos algoritmos para o SPDG na solução do problema da Árvore Geradora Mínima com Limitação de Saltos por meio de uma transformação, que aplicada ao grafo original, não direcionado, gera um grafo transformado, direcionado, com uma estrutura em camadas que encerra a limitação de saltos, permitindo a utilização de algoritmos para o SDPG na solução do problema original (Árvore Geradora Mínima com Limitação de Saltos). Nesse capítulo, mostraremos uma adaptação do Dual Ascent, inspirada no trabalho de [Gouveia et al. 2007], que permite tratar o hcSPDG como um SPDG aplicado sobre um grafo em camadas.

7.3 Transformação para o Grafo em Camadas

Um problema similar ao hcSPDG é o da árvore geradora mínima com restrições de saltos (hcMST) que consiste em: dados o grafo $G = (V, E)$ onde V é o conjunto vértices e E o conjunto de arestas, um custo positivo c associado a cada aresta, um número natural H (número máximo de saltos permitido) e um nó raiz r, encontrar uma árvore geradora AG com custo total mínimo, tal que o caminho entre r e todos os demais nós não contenha mais do que H arestas.

[Gouveia et al. 2007] sugerem a solução do problema aplicando qualquer algoritmo para solução do SPDG sobre um grafo criado por uma transformação sobre o grafo original, que gera um grafo transformado em camadas que encerra a limitação de saltos em sua topologia. O grafo transformado é composto por uma camada inicial contendo a origem 0, e H camadas adicionais, contendo cada uma todos os demais nós diferentes de 0. Arcos

direcionados, correspondentes às arestas do grafo original, conectam as camadas sempre da de nível menor para a de nível maior. Desta forma o grafo transformado possui uma fonte (o nó 0) e vários sumidouros (os nós da última camada), como pode ser observado na Figura 7.1. A fim de evitarmos ambigüidades, acrescentamos o número da camada à identificação do nó no grafo transformado. Assim o nó (1,2) representa o nó 1 da camada 2, (1,3) o nó 1 na camada 3, e assim por diante.

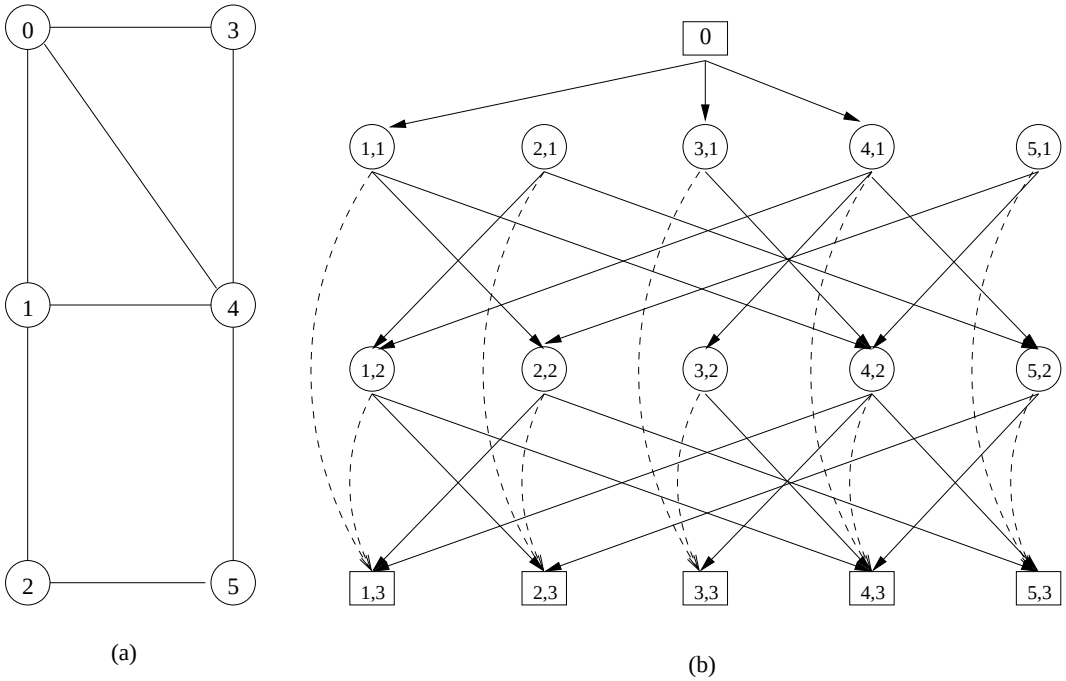

Figura 7.1: Transformação com limite de saltos (H=3). (a) grafo original (b) grafo em camadas

Supondo $G = (V, E)$ o grafo original, para o qual a árvore geradora mínima com limite de saltos H é desejada, geramos o grafo transformado $G_t = (V_t, A_t)$ como segue:

- $V_t = \{0\} \cup \{(i, h) : 1 \leqslant h \leqslant H, i \in V \setminus \{0\}\}$
- $A_t = A_0 \cup A_1 \cup A_2$ sendo:

$$A_0 = \{(0, (j, 1)) : (0, j) \in E\}$$
$$A_1 = \{((i, h), (j, h + 1)) : (i, j) \in E, i \neq 0, 1 \leqslant h \leqslant H - 1\}$$
$$A_2 = \{((i, h), (i, H)) : i \in V \setminus \{0\}, 1 \leqslant h \leqslant H - 1\}$$

Os custos dos arcos nos conjuntos A_0 e A_1 são iguais aos dos arcos correspondentes no grafo original. Os custos dos arcos do conjunto A_2 são iguais a zero.

Temos que a solução para o hcMST sobre o grafo G é equivalente à solução do SPDG sobre o grafo G_t desde que o terminal raiz do SPDG seja o nó raiz do hcMST e os terminais do SPDG sejam os nós da última camada de G_t, isto é, $T = \{(i, H) : i \in V\}$. O grande mérito dessa transformação é viabilizar a utilização de todos os algoritmos já conhecidos para o SPDG na resolução do hcMST, inclusive o Dual Ascent.

Uma estratégia semelhante pode ser adotada para resolvermos o hcSPDG com algoritmos originalmente desenvolvidos para o SPDG. Neste caso, o grafo original será orientado, composto por arcos e não arestas, e o sentido dos arcos deve ser levado em consideração quando da geração do grafo transformado. Mais formalmente, a transformação seria a seguinte: dados o grafo $G_d = (V, A)$, onde V é o conjunto de vértices e A o conjunto de arcos, um terminal raiz $r \in V$, um conjunto $T \subseteq V$ de terminais e um número natural H (número máximo de saltos permitido), construir $G_t = (V_t, A_t)$ tal que:

- $V_t = \{(i, h) : 0 \leqslant h \leqslant H, i \in V\}$
- $A_t = A_0 \cup A_1$ sendo:

$$A_0 = \{((i, h), (j, h + 1)) : (i, j) \in A, 0 \leqslant h \leqslant H - 1\}$$
$$A_1 = \{((i, h), (i, h + 1)) : i \in V, 0 \leqslant h \leqslant H - 1\}$$

Os custos dos arcos no conjunto A_0 são iguais aos dos arcos correspondentes no grafo original. Os custos dos arcos do conjunto A_1 são iguais a zero.

Dessa forma, podemos aplicar algoritmos para o SPDG sobre o grafo transformado. A solução do SPDG sobre o grafo transformado será uma solução para o hcSPDG sobre o grafo original.

7.4 Dual Ascent Distribuído sobre o Grafo em Camadas

Considerando o problema seqüencial, qualquer algoritmo para o SPDG pode ser utilizado para solução do hcSPDG, sem qualquer alteração, trocando-se apenas o grafo de entrada do problema para o grafo em camadas e convertendo-se novamente para o formato original a saída do algoritmo. Para o caso distribuído, o grafo, na realidade, é a representação de uma rede de computadores. A geração do grafo em camadas não pode ser feita de forma tão explícita, no entanto, a idéia básica ainda pode ser aplicada.

No caso distribuído, para cada nó i do grafo original existe um processo que faz o papel das instâncias $(i,0)$, $(i,1)$, ... ,(i,H). Todas as mensagens devem carregar a camada de destino w de forma a permitir que, ao receber uma mensagem, o processo possa reagir adequadamente, fazendo o papel do nó (i,w). No Dual Ascent sobre o grafo transformado, as mensagens em *broadcast* e de inclusão são sempre para um outro nó em camada igual à camada do emissor menos um. De forma análoga, as mensagens em *convergecast* serão sempre para a camada igual à camada do emissor mais um. Sob outro ponto de vista, podemos dizer que as mensagens "carregam" o número de saltos já utilizados durante sua propagação. Os terminais da última camada, nível H, sempre iniciam ciclos de crescimento.

A Figura 7.2 mostra o grafo em camadas implementado para utilização com Dual Ascent e a Figura 7.3 uma solução para o problema, vista no grafo original (parte "a") e no grafo transformado (parte "b"). Além de termos o grafo original orientado, outras diferenças topológicas em relação à proposta original de [Gouveia et al. 2007] da Figura 7.1 podem ser observadas:

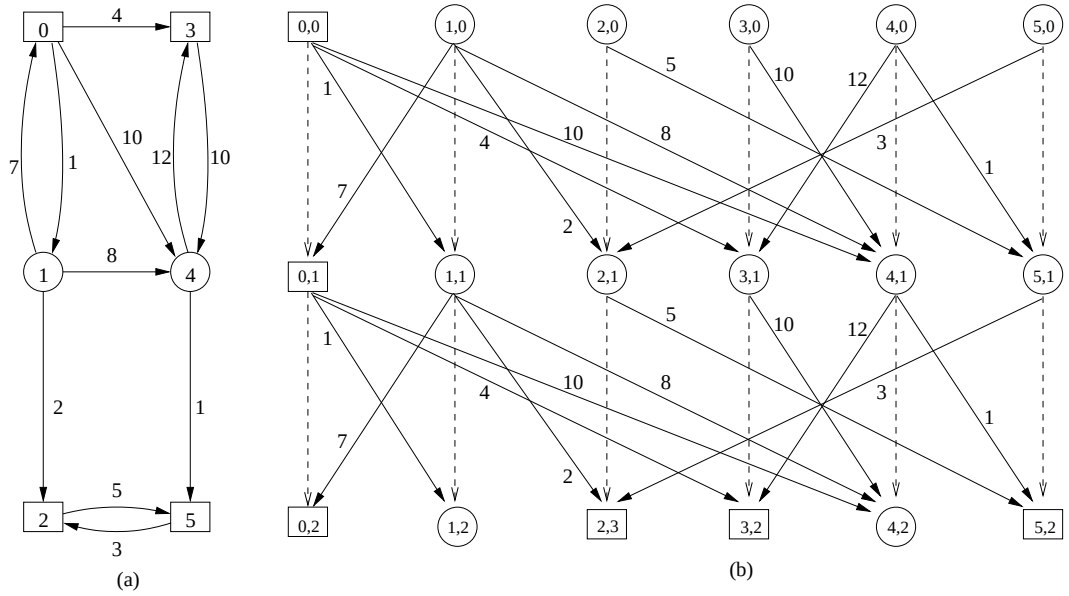

Figura 7.2: Transformação com limite de saltos (H=2) implementada com o Dual Ascent. (a) grafo original (b) grafo em camadas

- (i) os arcos de custo zero estão entre cada nó e seu correspondente na camada imediatamente inferior e não diretamente ao seu correspondente na última camada;
- (ii) a raiz aparece em todos os níveis, como os demais nós, e não apenas na primeira camada;

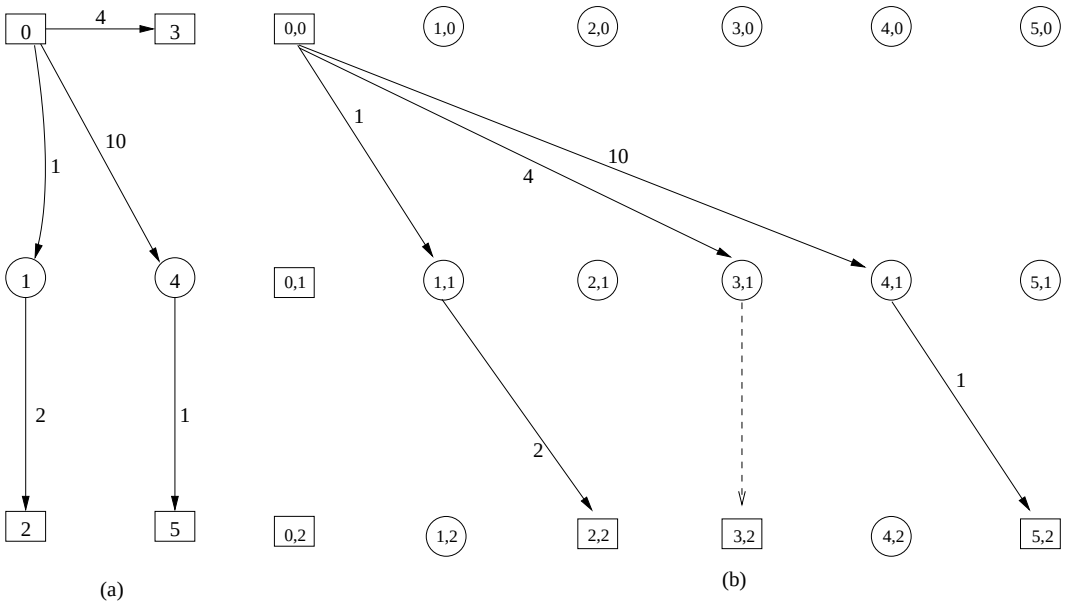

Figura 7.3: Uma solução para o problema (a) grafo original (b) grafo em camadas

- (iii) na primeira camada aparecem todos os nós e não apenas o nó raiz.

Essas alterações simplificam a implementação, pois os os arcos de entrada e saída das diversas instâncias de um nó são os mesmos do grafo original e deixam de existir diferenças entre os nós componentes das diversas camadas.

Apesar da diferença (i), cada nó continua a possuir um caminho de custo zero até sua instância na camada H. O possível aumento no número de saltos entre um nó e sua instância na última camada é irrelevante já que, na realidade, mensagens entre eles não são realmente enviadas, pois trata-se do mesmo "nó real".

Apesar de existirem diversas instâncias do nó raiz em nossa implementação (diferença ii), todas elas se comportam de maneira idêntica, independente da camada. Ao serem incluídas em um fragmento, respondem com uma mensagem que causa o término do crescimento do fragmento e a difusão do limite inferior parcial até então calculado por seu terminal líder para que seja calculado o limite inferior global. Pode ocorrer que um fragmento atinja mais de uma instância do nó raiz no último ciclo de crescimento. Nesse caso, o terminal líder do fragmento poderá receber mais de uma mensagem de terminação, o que não constitui problema. O nó raiz receberá múltiplas cópias do limite inferior parcial no último *broadcast* disparado no fragmento, uma em cada instância incluída na última rodada, devendo cuidar para não somar duas vezes a contribuição do fragmento ao limite inferior global. Isto é conseguido mantendo-se um vetor único, compartilhado por todas as instâncias da raiz, com as contribuições de cada fragmento ao limite inferior.

Uma alteração na semântica das mensagens *Ack* se torna necessária. Quando um nó receber uma mensagem *Check*, ele responde com uma mensagem *Ack(false)* se já participar do fragmento, ou se sua participação na árvore for impossível por violação da restrição de número de saltos. Sob o ponto de vista do grafo transformado, todos os nós da primeira camada, exceto o nó raiz, sempre respondem com *Ack(false)* a qualquer *Check*, tornando irrelevante a diferença topológica (iii).

7.5 Exemplo

Nas figuras 7.4 a 7.7, ilustramos o crescimento do fragmento associado ao terminal 5 na pequena instância exemplo da Figura 7.2. São mostrados apenas os arcos relevantes para o fragmento analisado, a fim de simplificar a ilustração, e facilitar seu entendimento. Ao acordar espontaneamente, o terminal $(5, 2)$ inclui suas instâncias dos níveis seguintes, como exibido na Figura 7.4. Os arcos que conectam as instâncias referentes a um mesmo nó têm custos iguais a zero, ou seja, já são saturados. Por estes arcos são enviadas mensagens *Include* já no primeiro ciclo de crescimento.

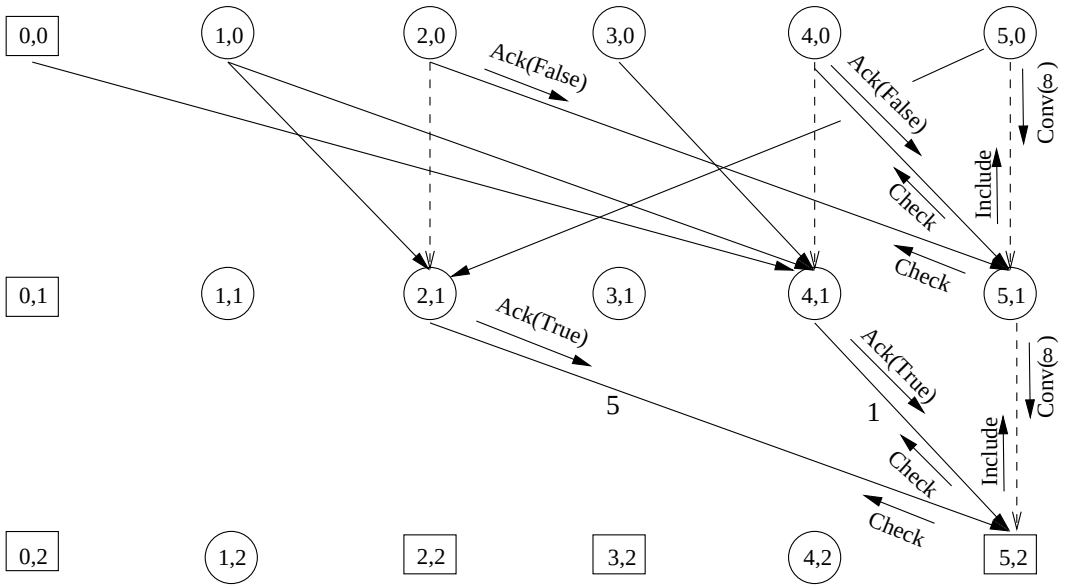

Figura 7.4: Crescimento do fragmento associado ao terminal 5. Check/Ack dos arcos locais e inclusão de demais instâncias do terminal 5

O segundo ciclo de crescimento, mostrado na Figura 7.5, mostra a inclusão do nó $(4, 1)$, que envia mensagens *Check* aos seus vizinhos da camada seguinte, os nós $(0, 0)$, $(1, 0)$, $(3, 0)$ e $(4, 0)$. A mensagem *Include* recebida contém a indicação do nível do grafo que deve tratar a mensagem ou, em outras palavras, a quantos saltos está o terminal que iniciou esse *broadcast*. Ao enviar a mensagem *Check*, o nó $(4, 1)$ pode então informar o próximo nível,

igual a 0, que indica que o limite de saltos foi atingido, portanto, todos os nós diferentes do nó raiz sabem que caminhos que os tenham como nó intermediário não podem ser admitidos na solução e respondem com uma mensagem *Ack(False)*, indicando que aquele arco não pode ser saturado e deve ser desconsiderado de qualquer processamento futuro. Os arcos utilizados na transmissão de mensagens *Ack(False)* deixam de ser mostrados na figura após a transmissão dessa mensagem.

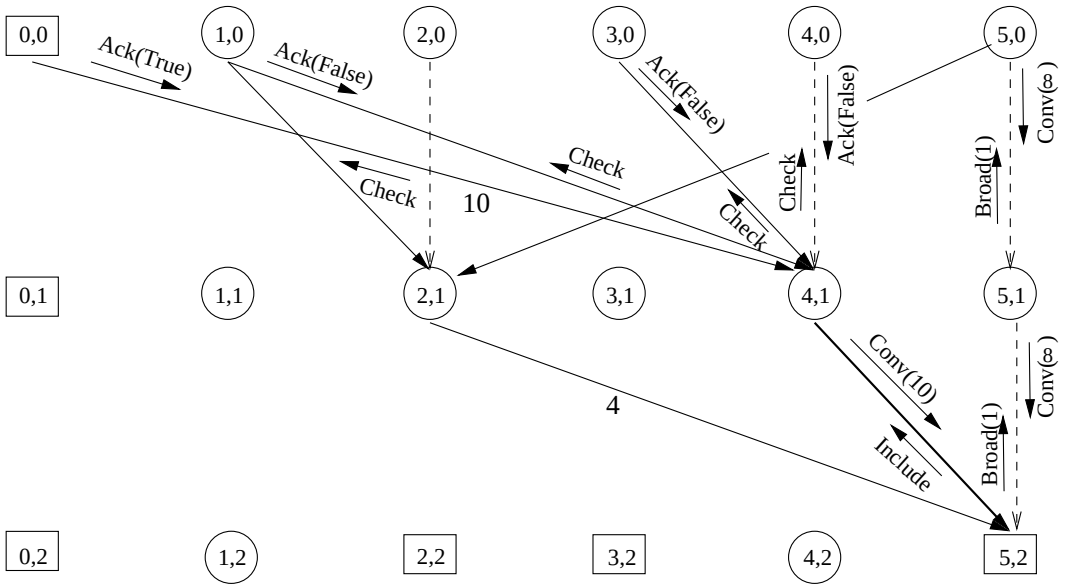

Figura 7.5: Crescimento do fragmento associado ao terminal 5. Inclusão do nó (4,1)

O processo se repete na Figura 7.6, onde o nó $(2,1)$ é incluído. Note que os nós $(5,0)$ e $(5,1)$ não possuem arcos entrantes ao fragmento e a mensagem que chega por este "ramo" da árvore de saturação ao nó $(5,2)$ transporta $\delta = \infty$. Na Figura 7.7 o fragmento inclui o nó raiz e termina.

7.6 Análise do Algoritmo

Como o grafo em camadas é sempre maior do que o grafo original, ($V_t = |V|.|H|$ e $E_t = |E|.|H|$), poderíamos esperar custos de execução do Dual Ascent iguais a $O(|T|.|V_t|^2)$ para número de mensagem e tempo, que são as complexidades do Dual Ascent distribuído. No entanto, como a topologia do grafo transformado é muito particular, alguma análise extra é necessária.

Quando um nó é incluído em um fragmento, ele envia mensagens *Check* para todos os seus vizinhos que, no caso do grafo em camadas, será igual ao número de nós do grafo original $|V|$. Portanto, o crescimento de um fragmento não enviará mais que $|V_t|.|V|$ men-

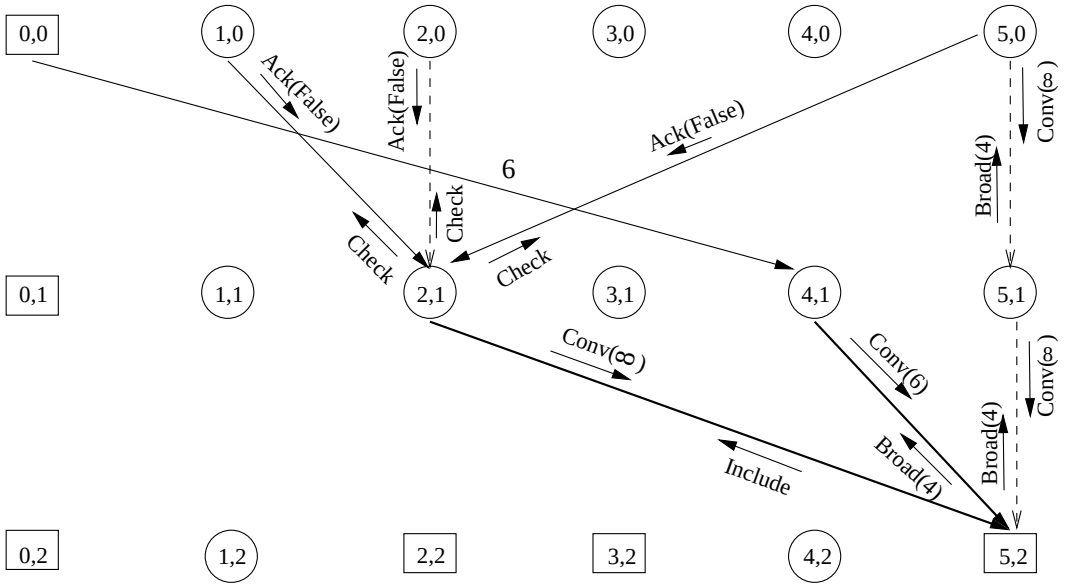

Figura 7.6: Crescimento do fragmento associado ao terminal 5. Inclusão do nó (2,1)

sagens *Check*. Como existirá um fragmento para cada terminal, o número de mensagens *Check* enviadas durante toda execução do algoritmo será da $O(|T|.|V_t|.|V|)$. Menor que $O(|T|.|V_t|^2)$ esperado, graças à topologia especial do grafo.

No entanto, o mesmo ganho não pode ser verificado quando analisamos as mensagens relativas ao crescimento dos fragmentos (*Broad, Include, Conv*). Cada rodada de crescimento demanda $O(|V_t|)$ mensagens. Como podem existir $O(|V_t|)$ rodadas por fragmento, $O(|T|.|V_t|^2)$ é um limite superior para o número de mensagens em *broadcast* enviadas. Portanto, este é um limite superior válido para o número total de mensagens enviadas. Não houve melhora no limite teórico em função da topologia especial do grafo transformado para o número de mensagens enviadas.

Quanto ao tempo, agora temos um limite H para o comprimento das cadeias de mensagens com relação de causalidade. Ainda podemos nos deparar com situações em que um único fragmento pode crescer de cada vez, porque todos os demais fragmentos que ainda não tenham incluído o nó raiz estão suspensos. A complexidade para o crescimento de um único fragmento é $O(|V_t|.H)$. Como existem $|T|$ fragmentos, a complexidade em tempo fica $O(|T|.|V_t|.H)$. A topologia especial do grafo em camadas torna possível um limite superior mais baixo quando comparado com a execução do algoritmo em um grafo qualquer de mesmo tamanho.

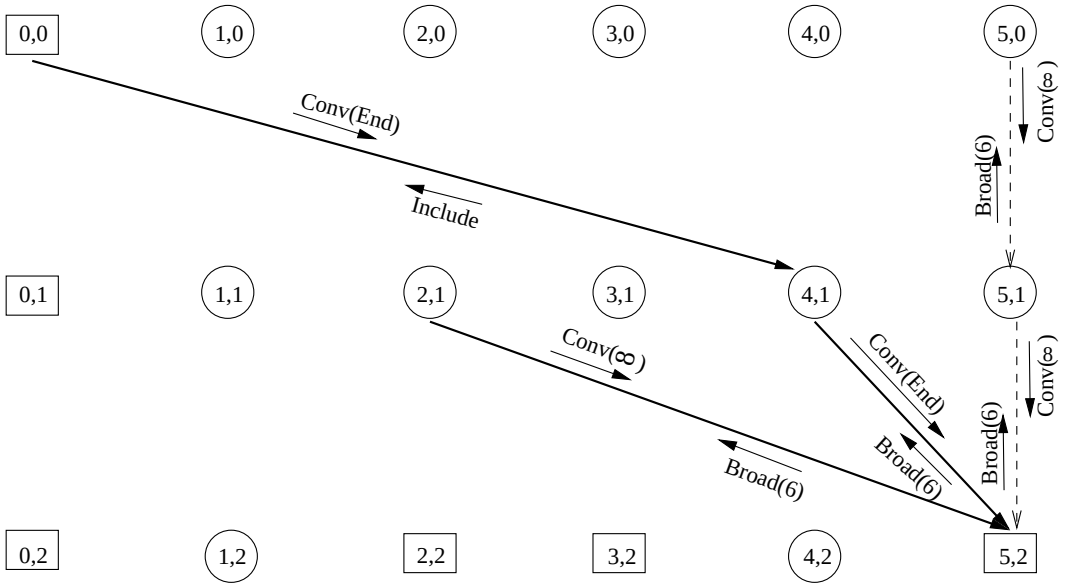

Figura 7.7: Crescimento do fragmento associado ao terminal 5. Inclusão do nó raiz com a finalização do crescimento

7.7 Resultados Experimentais

Utilizamos para testes o mesmo ambiente, as instâncias direcionadas geradas a partir da *SteinLib* e as simuladoras da Internet descritas no Capítulo 6. Nos grafos completos, I080-021, I080-121, I080-221 e I080-321 e toda a classe P4Z, realizamos testes com 2, 3, 4 e 5 para o limite de saltos (H). Para os demais grafos da *SteinLib*, foi calculada a menor distância entre a raiz e todos os terminais, e a maior entre todas as distâncias calculadas foi utilizada como limite de saltos, ou seja, utilizamos o maior limite de saltos possível para a instância. Para as classes da Internet, escolhemos os terminais aleatoriamente entre os nós que estivessem entre 3, 4 e 5 saltos de distância do terminal raiz, e utilizamos essas distâncias (3, 4 e 5) como limite de saltos para cada instância.

De forma análoga aos resultados exibidos no Capítulo 6, as colunas nas tabelas 7.1 e 7.2 possuem os seguintes significados:

- $|\boldsymbol{V}|$, $|\boldsymbol{A}|$ **e** $|\boldsymbol{T}|$ mostram o tamanho da instância;
- $\boldsymbol{H}$ é o limite de saltos utilizado;
- **Otm** é o valor da solução ótima calculada com o algoritmo "branch-and-bound" de [Poggi de Aragão et al. 2001] adaptado para limitação de saltos;
- **SPHs** é o valor da solução obtida na execução do Prim-SPH adaptado para limitação de saltos sobre o grafo original, anterior à transformação;

- **LI** é o limite inferior obtido com o Dual Ascent;
- $\frac{|Ar|}{|A|}\%$ é a proporção de arcos que são saturados;
- **DAHs** é o valor da solução obtida na execução do Prim-SPH adaptado para limitação de saltos sobre o grafo de arcos saturados gerado pelo Dual Ascent.

Os resultados referentes às execuções com 2, 3 e 4 para limite de saltos nas instâncias completas da *SteinLib* foram omitidas da Tabela 7.1, por não apresentarem informações relevantes, mas são considerados na geração dos gráficos relativos à bateria de testes. Executamos o Dual Ascent 5 vezes, e os resultados apresentados constituem a média aritmética das 5 execuções. Novamente, obtivemos um desvio padrão muito baixo: 0,6% para os valores das soluções.

Como nos testes sem limitação de saltos, utilizamos *competitividade*, a razão entre o valor da solução obtida com a heurística e o ótimo para comparar a qualidade das soluções obtidas com e sem a utilização do Dual Ascent. Os gráficos das figuras 7.8 e 7.9 mostram o percentual cumulativo dos casos em que a competitividade é menor ou igual ao valor do eixo das abscissas, respectivamente para as instâncias da *Steinlib* e representativas da Internet. Como esperado, a utilização do Dual Ascent leva novamente a resultados melhores, em média 4,1%. Os gráficos das figuras 7.8 e 7.9 mostram também uma linha referente ao melhor resultado entre os dois algoritmos testados e, como no Capítulo 6, os resultados com a execução do SPHs dependendo da diferença percentual entre o Limite Inferior e a solução obtida com o DAHs. O SPHs foi executado em apenas 50% dos casos, e os resultados obtidos foram praticamente idêntidos.

Também comparamos o desempenho prático do DAHs e do SPHs com respeito ao tempo, dado pelo tamanho da maior cadeia de mensagens com relação de causalidade e o número total de mensagens enviadas. Os resultados dessas medidas podem ser observados nos gráficos das figuras 7.10 a 7.13.

Instância						SPHs	Dual Ascent		DAHs
Nome	$\|V\|$	$\|A\|$	$\|T\|$	H	Otm	Solução	LI	$\frac{\|Ar\|}{\|A\|}\%$	Solução
b01d	50	126	9	5	79	**79**	79,0	65,0	**79,0**
b02d	50	126	13	5	136	155	136,0	83,0	**136,0**
b03d	50	126	25	8	175	179	171,8	86,2	**176,8**
b04d	50	200	9	4	63	73	63,0	44,0	**63,0**
b05d	50	200	13	3	82	84	82,0	46,0	**83,0**
b06d	50	200	25	6	144	155	129,6	63,1	**150,1**
b07d	75	188	13	6	126	132	126,0	71,0	**126,0**
b08d	75	188	19	6	140	163	139,2	77,0	**148,0**
b09d	75	188	38	7	240	246	239,4	83,0	**240,0**
b10d	75	300	13	5	109	139	105,0	51,0	**114,0**
b11d	75	300	19	5	138	155	122,8	60,2	**146,4**
b12d	75	300	38	5	207	307	197,6	63,0	**222,0**
b13d	100	250	17	6	201	219	196,0	76,0	**207,0**
b14d	100	250	25	7	288	296	284,4	88,0	**288,1**
b15d	100	250	50	9	370	391	364,0	96,0	**371,0**
b16d	100	400	17	6	167	218	142,0	58,0	**176,0**
b17d	100	400	25	6	160	167	154,0	57,0	**165,0**
b18d	100	400	50	5	287	327	271,0	62,0	**296,0**
i080-001d	80	240	6	3	2.320	**2.320**	2.320,0	55,0	**2.320,0**
i080-101d	80	240	8	6	2.322	2.574	2.309,0	48,0	**2.322,0**
i080-201d	80	240	16	6	4.408	4.744	4.337,0	63,0	**4.725,0**
i080-301d	80	240	20	5	6.231	**6.669**	5.975,0	73,0	6.724,0
i080-031d	80	320	6	4	1.646	**1.793**	1.628,0	43,0	1.795,0
i080-131d	80	320	8	4	1.998	2.356	1.894,6	45,4	**2.075,8**
i080-231d	80	320	16	4	4.686	5.186	4.382,0	54,0	**4.923,0**
i080-331d	80	320	20	4	4.715	5.224	4.666,0	53,0	**5.005,0**
i080-011d	80	700	6	2	1.744	**1.744**	1.744,0	20,0	**1.744,0**
i080-111d	80	700	8	3	2.185	**2.455**	2.073,2	25,0	2.653,2
i080-211d	80	700	16	3	3.411	4.143	3.128,0	28,0	**3.967,0**
i080-311d	80	700	20	3	3.942	4.508	3.672,0	27,0	**4.187,0**
i080-041d	80	1.264	6	2	953	1.026	940,2	11,0	**953,8**
i080-141d	80	1.264	8	2	2.382	**2.470**	2.382,0	13,0	**2.470,0**
i080-241d	80	1.264	16	2	3.769	**4.625**	3.670,0	16,0	4.911,0
i080-341d	80	1.264	20	2	4.064	4.621	3.910,0	15,0	**4.390,0**
i080-021d	80	6.320	6	5	741	847	673,0	8,0	**834,0**
i080-121d	80	6.320	8	5	977	**1.097**	927,4	5,4	2.695,2
i080-221d	80	6.320	16	5	1.982	2.293	1.815,2	8,1	**2.291,2**
i080-321d	80	6.320	20	5	2.449	3.137	2.235,8	7,6	**2.828,0**
P401d	100	9.900	5	5	158	**158**	158,0	2,0	**158,0**
P402d	100	9.900	5	5	104	**104**	104,0	1,0	**104,0**
P403d	100	9.900	5	5	181	**219**	181,0	2,0	220,0
P404d	100	9.900	10	5	334	**349**	300,0	2,0	368,0
P405d	100	9.900	10	5	276	325	274,0	2,0	**300,0**
P406d	100	9.900	10	5	346	454	331,4	2,2	**396,6**
P407d	100	9.900	20	5	673	836	595,7	2,3	**764,3**
P408d	100	9.900	20	5	576	633	551,8	2,9	**608,5**

Tabela 7.1: Resultados com limitação de saltos para instâncias da SteinLib.

Instância						SPHs	Dual Ascent		DAHs
Nome	$\|V\|$	$\|A\|$	$\|T\|$	H	Otm	Solução	LI	$\frac{\|Ar\|}{\|A\|}\%$	Solução
glp01	100	334	5	3	3.043	**3.043**	2.959,0	46,0	3.278,0
glp02	100	358	5	4	4.352	**4.496**	3.895,1	52,3	4.786,1
glp03	100	374	5	5	3.671	3.705	4.026,0	53,0	**3.671,0**
glp04	100	356	10	3	3.943	**3.943**	4.343,0	44,0	**3.943,0**
glp05	100	336	10	4	7.526	**7.526**	6.964,0	51,0	7.962,0
glp06	100	312	10	5	8.557	8.694	8.155,0	50,0	**8.557,0**
glp07	100	338	15	3	9.870	10.214	10.424,0	45,0	**9.870,0**
glp08	100	374	15	4	8.343	8.796	8.131,0	46,0	**8.663,0**
glp09	100	400	15	5	5.945	6.122	5.866,2	48,2	**6.069,4**
sw01	100	1.243	5	3	588.938	735.326	542.016,0	9,0	**684.718,0**
sw02	100	1.074	5	3	130.563	**130.563**	130.563,0	7,0	**130.563,0**
sw03	100	1.277	5	3	306.196	382.708	306.196,0	7,0	**306.196,0**
sw04	100	1.150	5	4	105.341	110.257	105.341,0	7,0	**105.341,0**
sw05	100	1.007	5	4	264.669	**269.626**	259.144,0	9,0	**269.626,0**
sw06	100	892	5	4	239.683	267.859	235.847,0	11,0	**239.683,0**
sw07	100	1.077	5	5	143.988	166.431	126.302,0	9,0	**163.169,0**
sw08	100	1.026	5	5	99.168	113.539	98.989,0	9,0	**100.477,0**
sw09	100	1.125	5	5	236.336	**262.207**	233.767,2	12,1	**262.207,0**
sw10	100	1.553	10	3	383.049	510.486	337.111,0	7,0	**487.943,0**
sw11	100	860	10	3	313.959	359.860	308.818,0	10,0	**350.411,0**
sw12	100	1.505	10	3	1.096.275	**1.178.259**	1.096.275,0	11,0	**1.178.259,0**
sw13	100	933	10	4	362.021	620.420	362.021,0	11,0	**503.299,0**
sw14	100	988	10	4	794.572	794.697	794.572,0	14,0	**794.572,0**
sw15	100	1.245	10	4	501.390	655.357	412.620,6	11,9	**501.407,1**
sw16	100	1.314	10	5	165.434	**183.654**	155.251,0	9,0	195.700,0
sw17	100	1.243	10	5	230.683	354.632	200.117,0	10,0	**266.769,0**
sw18	100	1.539	10	5	118.958	176.544	112.488,0	7,0	**145.887,0**
sw19	100	1.028	15	3	639.718	647.029	636.710,0	12,0	**644.513,0**
sw20	100	1.189	15	3	409.818	529.196	374.192,0	9,0	**442.350,0**
sw21	100	1.448	15	3	438.285	561.050	420.087,8	9,0	**465.200,0**
sw22	100	1.128	15	4	489.096	803.833	375.412,8	11,4	**580.640,7**
sw23	100	960	15	4	385.990	690.621	385.990,0	12,0	**445.032,0**
sw24	100	1.182	15	4	501.292	537.612	488.860,0	10,0	**536.724,0**
sw25	100	1.107	15	5	364.148	510.276	313.994,0	12,0	**384.652,0**
sw26	100	1.072	15	5	263.517	391.351	251.864,0	11,1	**307.810,9**
sw27	100	1.044	15	5	883.258	1.095.476	759.095,0	17,0	**1.044.499,0**

Tabela 7.2: Resultados com limitação de saltos para instâncias da Internet.

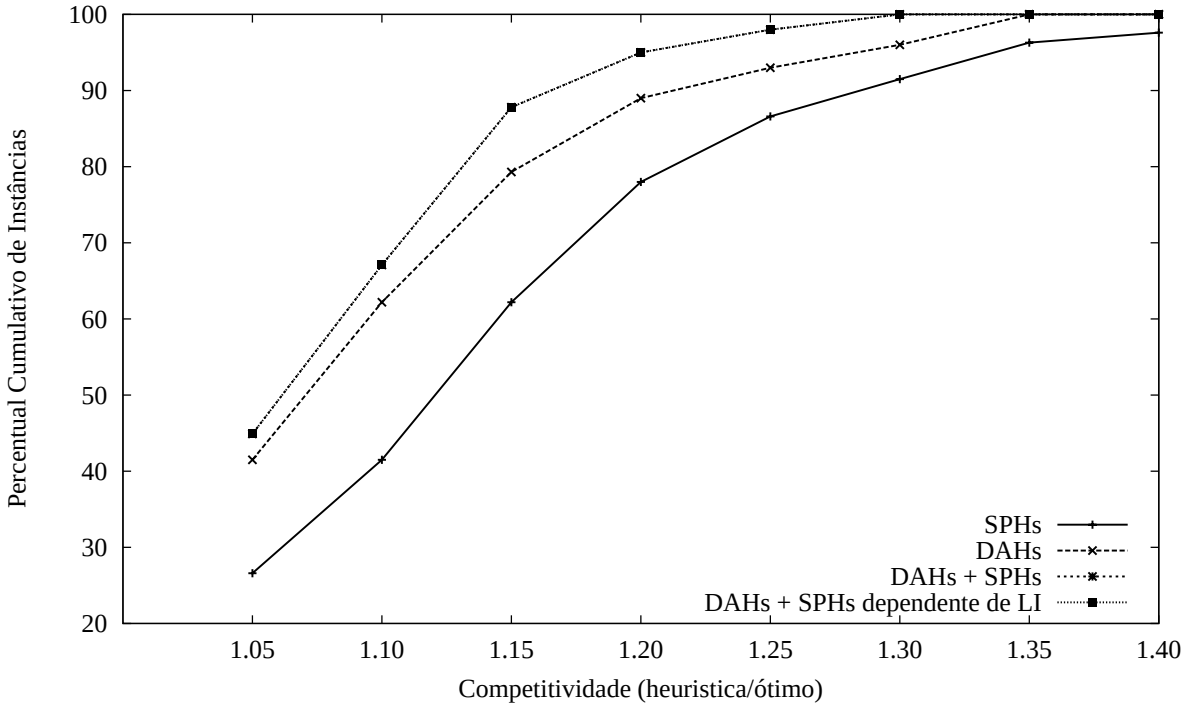

Figura 7.8: Qualidade da solução – Instâncias da SteinLib com limitação de saltos.

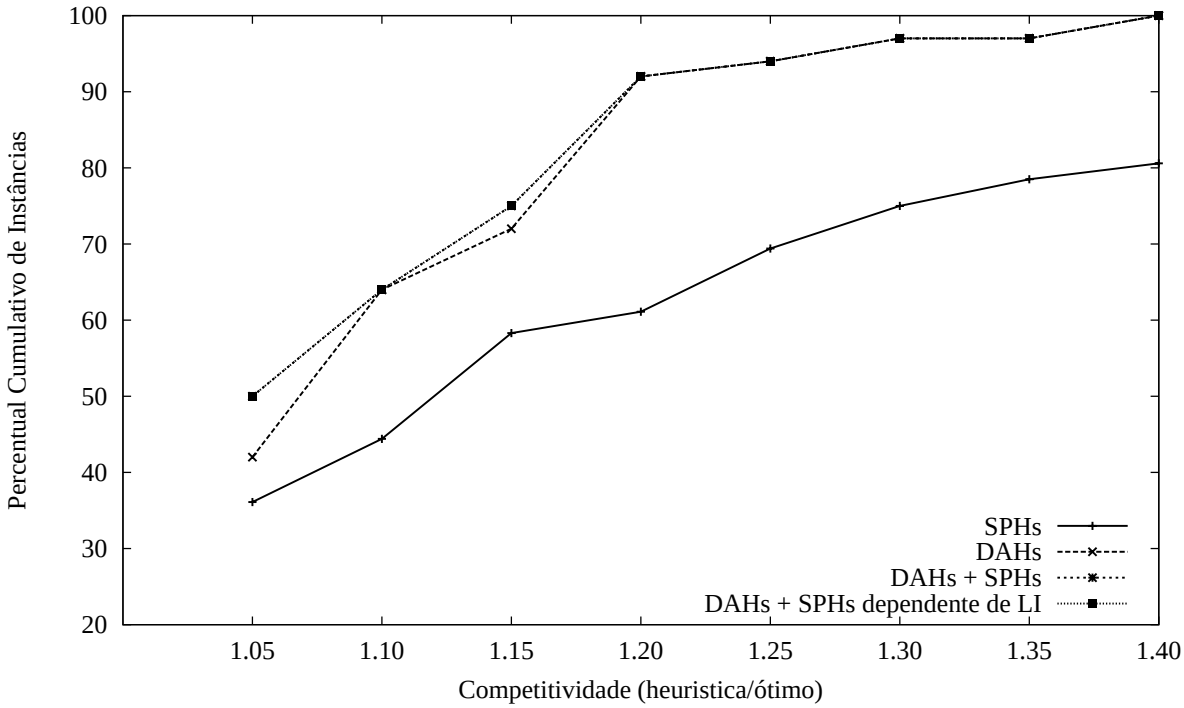

Figura 7.9: Qualidade da solução – Instâncias da Internet com limitação de saltos.

Figura 7.10: Tempo – Instâncias da SteinLib com limitação de saltos

Figura 7.11: Tempo – Instâncias da Internet com limitação de saltos

Figura 7.12: Mensagens – Instâncias da SteinLib com limitação de saltos

Figura 7.13: Mensagens – Instâncias da Internet com limitação de saltos

Observamos que o Dual Ascent com limitação de saltos apresenta maior sensibilidade ao limite de saltos do que o SPHs. O grafo transformado é sempre H vezes maior do que o grafo original e o Dual Ascent emprega mensagens *Check* para as diversas instâncias de um nó nas diversas camadas e explora as diversas instâncias de um mesmo arco, o que explica os custos maiores. Podemos observar a sensibilidade do Dual Ascent para o limite de saltos principalmente nas instância B10 à B12, que possuem maiores valores para H, e nas instâncias completas, quando variamos o limite de saltos (2, 3, 4 e 5). Nesses casos podemos observar claramente o rápido crescimento do custo do Dual Ascent em função do limite de saltos pela aparência de "serra" apresentada pelos gráficos. Apesar disso, para as instâncias de grande densidade, a aplicação do Prim-SPH sobre a instância reduzida compensa o aumento do custo decorrente do aumento do grafo, principalmente quanto ao número de mensagens enviadas.

O gráfico da Figura 7.14 foi construído utilizando-se apenas os grafos completos, últimas instâncias da série I080 e toda a série P4Z, ordenados por número de terminais. Nessas instâncias, executamos os algoritmos diversas vezes, com limites de saltos diferentes, de forma a poder estudar mais detalhadamente o comportamento do aumento do número de saltos sobre o paralelismo conseguido na execução do Dual Ascent. O SPHs apresentou baixa sensibilidade ao número de saltos. Portanto, traçamos uma linha única para a média dos valores obtidos com esse algoritmo para todos os limites utilizados (2, 3, 4 e 5). No caso do Dual Ascent, traçamos uma linha para cada limite de saltos. Podemos observar pela figura que para limites de saltos baixos, a execução do DAHs é comparável ao do SPHs e, mais importante, o ritmo de crescimento dos custos em função do crescimento da instância não é significativamente diferente para o DAHs e o SPHs.

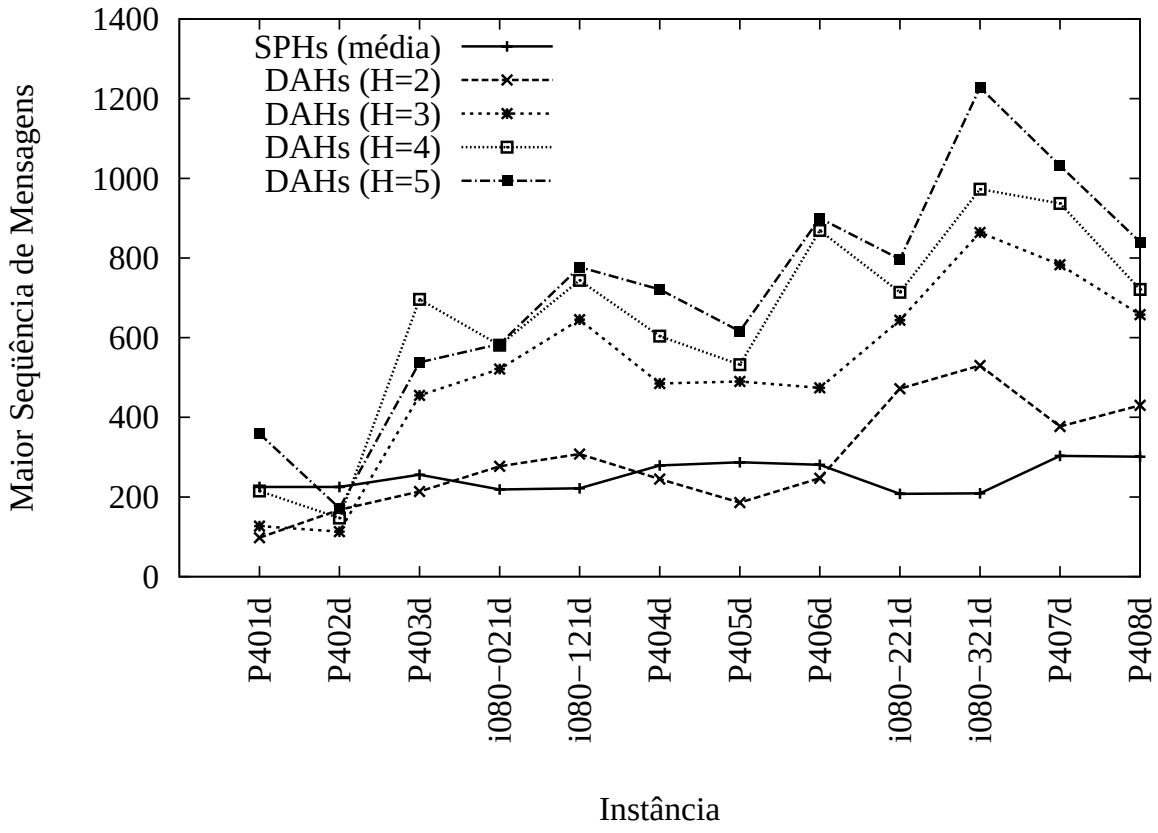

Figura 7.14: Tempo – Instâncias completas da SteinLib com limitação de saltos

7.8 Conclusão

Foi mostrado como a versão distribuída do Dual Ascent pode ser utilizada para o problema de Steiner com limitação de saltos. Essa variação do problema é de grande interesse prático, pois modela muito bem a transmissão de multimídia em multicast pela Internet, quando as aplicações não toleram uma latência acima de determinados limites.

A melhor qualidade da solução obtida com a utilização do novo algoritmo implica em aumento de custos, principalmente quando o limite de saltos tolerado for alto, mas não detectamos diferenças significativas no ritmo de crescimento desses custos.

O algoritmo foi testado em instâncias que mimetizam a Internet (BRITE-GLP, SW) com resultados muito semelhantes aos obtidos em instâncias geradas para avaliação do problema de um ponto de vista mais teórico (*SteinLib*), o que reforça nossa crença de que o Dual Ascent é uma alternativa a ser considerada para a distribuição de dados em *multicast* pela Internet.

Essa parte do trabalho foi apresentada durante no SBRC'2008 [Santos et al. 2008].

Capítulo 8

Problema de Steiner On-Line

O Problema de Steiner *on-line* em Grafos Direcionados pode ser definido como a seguir.

Dados:

- o grafo $G_d = (V, A)$ onde V é o conjunto de vértices e A o conjunto de arcos;
- um terminal raiz $r \in V$;
- um custo positivo c_a associado a cada arco $a \in A$;
- um conjunto $T_0 \subseteq V$, de terminais (conjunto inicial de terminais);
- uma seqüência de conjuntos de terminais $T = (T_1, T_2, ..., T_n)$ determinada por uma seqüência de operações $(o_1, o_2, ..., o_n)$ onde cada operação o_i pode ser:

    - uma inclusão de um terminal: $T_i = T_{(i-1)} \cup \{v\}, v \in V, v \notin T_{(i-1)}$;
    - uma exclusão de um terminal: $T_i = T_{(i-1)} - \{t\}, t \in T_{(i-1)}$.

Encontrar uma seqüência de grafos $S = (G'_0, G'_1, G'_2, ..., G'_n)$ tal que $G'_i = (V'_i, A'_i)$ onde:

- existam caminhos de r a todo $t \in T_i$, $T_i \subseteq V'_i$;
- $\sum_{a \in A'_i} c_a$ seja mínimo.

O problema assim definido é de grande interesse prático, pois pode ser utilizado na modelagem do problema de *multicast* dinâmico em redes, em que informações devem ser

roteadas a partir de uma fonte a um grupo de nós interessados minimizando uma métrica associada às conexões entre os nós e o conjunto de nós interessados pode ser alterado com o passar do tempo.

8.1 Algumas Heurísticas Distribuídas para o Problema de Steiner *On-Line*

O problema dinâmico é equivalente a uma seqüência de problemas estáticos, portanto, podemos utilizar, para sua solução, sucessivas aplicações de algoritmos para o problema estático. No entanto, essa abordagem pode apresentar custos desnecessariamente altos já que, na maioria dos casos, estruturas construídas no problema anterior continuam válidas e podem ser aproveitadas para a construção da solução após uma operação de inclusão ou exclusão de um nó do conjunto de terminais.

Não é de nosso conhecimento qualquer algoritmo que trate especificamente da versão direcionada desse problema, mas propostas existem para a versão não direcionada e algumas delas podem ser facilmente adaptadas para grafos direcionados.

A primeira e mais direta abordagem do tipo gulosa é proposta por [Waxman 1988]. Inclusões são feitas sempre com a utilização dos caminhos mínimos entre o nó incluído e a solução atual, e as exclusões são feitas apenas quando o terminal excluído possui grau um. Exclusões de terminais podem gerar nós não terminais de grau um, que são também excluídos da solução. Essa abordagem é bastante simples e pode ser adaptada facilmente para o caso direcionado. No entanto, vários trabalhos posteriores apresentaram soluções mais sofisticadas.

O EBA (*Edge-Bounded Algorithm*) [Imase e Waxman 1991] mantém as estratégias para inclusão e exclusão de nós do método guloso, mas após cada inclusão, verifica-se a possibilidade de baixar o custo da árvore de distribuição, verificando se o custo de conexão de algum nó antigo pode ser diminuído utilizando-se, mesmo que parcialmente, o caminho adicionado à árvore para a inclusão do novo nó.

Algumas abordagens apresentam bons resultados para redes com características específicas como VTDM (*Virtual Trunk Dynamic Multicast*) [Lin e Lai 1998]. Esse algoritmo avalia todos os nós da rede segundo uma função que estima a probabilidade do nó ser utilizado no caminho para a conexão de dois outros. Os nós com maior probabilidade são incluídos em um conjunto denominado *Virtual Trunk*, e uma árvore é mantida permanentemente com os componentes desse tronco de distribuição. Nós são incluídos no conjunto

de terminais por meio de seu caminho mínimo até o *Virtual Trunk* e exclusões são feitas de forma idêntica ao EBA. A abordagem pode ser interessante para redes de tamanho limitado, mas requer a concentração dos dados da rede para que o *Virtual Trunk* seja construído, o que pode inviabilizar sua utilização em redes grandes.

O ARIES [Bauer e Varma 1997] é provavelmente o trabalho mais influente na área, estando entre os mais freqüentemente citados e fornecendo base outros trabalhos publicados. O autor utiliza a mesma estratégia do EBA para inclusão e exclusão de nós, mas observa que o EBA tenta melhorar a solução somente quando ocorrem inclusões, o que pode baixar sua qualidade após exclusões sucessivas. A fim de contornar esse ponto fraco do EBA, o ARIES sugere monitorar o número de alterações processadas (inclusões e exclusões) e disparar "reorganizações" visando melhorar a qualidade da solução quando o número de alterações ultrapassa um determinado limite.

No ARIES quando um terminal t é incluído ou excluído, uma mensagem *counter_ update* é enviada aos terminais vizinhos. Considera-se um terminal t_1 vizinho a outro t_2, quando não existem outros terminais no caminho entre t_1 e t_2 na solução atual. O algoritmo define região como um subgrafo $G_i = (V_i, E_i)$ da instância $G = (V, E)$ para o problema, onde V_i possui pelo menos um nó v incluído ou excluído do conjunto de terminais que não tenha passado por reorganizações e todos os vizinhos de v, e E_i é o conjunto das arestas (v_j, v_k) tais que $v_j, v_k \in V_i$ e $(v_j, v_k) \in E$. Terminais mantêm contadores associados a cada região de que façam parte. Ao receber uma mensagem *counter_ update*, o terminal incrementa o contador correspondente à região associada. Quando o contador ultrapassa um determinado limite, a região passa por uma reorganização. A reorganização consiste em excluir todas as arestas da região a ser reorganizada, desconectando terminais da árvore e reconectando-os utilizando o algoritmo K-SPH [Bauer e Varma 1996], heurística não adaptável para a versão orientada do problema de Steiner. A versão original do ARIES não permite operações de inclusão, exclusão ou reorganização simultâneas. [Adelstein et al. 2003] aprimora o algoritmo para permitir operações simultâneas.

[Gatani et al. 2006] utiliza o ADH para construir uma solução inicial que é posteriormente mantida utilizando uma técnica semelhante a do ARIES, com contadores controlando o número de alterações ocorridas em determinadas regiões do grafo. Gatani utiliza a *stirring techinique* [di Fatta e Re 1998] para reorganização, técnica viável para grafos orientados e não orientados. Apesar de trabalhar com grafos não orientados, di Fatta assume que existe um terminal especial chamado *source* e define o conjunto de ancestrais de um terminal como sendo os nós da solução para o SPG no caminho entre este termi-

nal e o nó *source*. Isto posto, podemos definir *grafting point* de um terminal como seu ancestral mais próximo que possua pelo menos dois filhos. A *stirring techinique* consiste em reavaliar todas possibilidades para *grafting points* alternativos, que levem a árvores de menor custo. Para cada terminal t, é buscado um nó v_k tal que a distância entre t e v_k seja menor que a distância entre t e v_a sendo v_a o *grafting point* atual de t. Este procedimento é repetido até que não sejam conseguidas soluções de menor custo.

Apresentamos neste capítulo uma abordagem para o problema *on-line* baseada na versão distribuída do Dual Ascent. Nossa heurística dispara reorganizações quando o custo da solução para o problema de Steiner se afasta do limite inferior obtido com o DA além de um determinado limite. Desta forma, esperamos que as reorganizações sejam mais oportunas do que as realizadas com base em mecanismos que avaliem o número de alterações realizadas em regiões do grafo, como a maioria das soluções atualmente conhecidas.

8.2 Algoritmo Dual Ascent para o Problema de Steiner *On-Line*

A idéia geral do algoritmo é realizar operações de exclusão e inclusão de terminais como no EBA e no ARIES, mantendo atualizadas dinamicamente, de forma distribuída, as seguintes estruturas:

- árvores de saturação;
- grafo de saturação;
- limite inferior;
- solução para o problema de Steiner atual.

Quando a diferença entre o limite inferior e o custo da solução ultrapassar uma determinada tolerância, dispara-se uma reorganização que tenta baixar o custo da solução utilizando um algoritmo para o problema estático sobre o grafo de saturação.

O nó raiz é o nó onde o custo da solução e o limite inferior global são mantidos durante a execução do algoritmo, portanto, é o nó mais adequado para disparar as reorganizações quando necessário. No nó raiz pode-se manter com maior facilidade o estado do sistema,

que pode estar em reorganização, processando alterações no conjunto de terminais, ou estável.

Desejamos utilizar qualquer algoritmo distribuído para a solução do problema estático como rotina de reorganização, portanto, é nosso interesse fazer com que as rotinas de exclusão e inclusão de terminais sejam independentes da reorganização. Para isso, não serão permitidas alterações no conjunto de terminais durante reorganizações, nem reorganizações enquanto as estruturas mantidas atualizadas dinamicamente não estiverem estáveis.

As iniciativas de exclusão ou inclusão do conjunto de terminais partem espontaneamente dos terminais ou candidatos a terminal, respectivamente, mas a informação sobre o estado do sistema está no nó raiz, portanto, um terminal que deseja ser excluído do conjunto de terminais ou um nó que deseja ser incluído nesse conjunto, deve consultar o nó raiz sobre o estado do sistema antes de iniciar a rotina de exclusão ou inclusão. Este pedido é feito com o envio de mensagens *Leave* ou *Join*. Essa consulta funciona como um pedido de autorização para exclusão ou inclusão. Caso o sistema encontre-se em reorganização, o nó raiz envia a autorização apenas ao fim dessa rotina, evitando assim alterações no conjunto de terminais durante reorganizações. Assumimos que os nós que tenham solicitado exclusão ou inclusão no conjunto de terminais não farão novas solicitações enquanto a primeira não tenha sido completamente processada pelo sistema.

Serão permitidas que inclusões sejam processadas simultaneamente a outras inclusões e exclusões simultaneamente a outras exclusões. Não serão enviadas autorizações para inclusão enquanto exclusões estiverem sendo processadas. Essa última restrição evita *livelocks*, como descrito a seguir.

8.2.1 Inclusão de Terminais

Lidamos aqui com a situação onde um nó que não faça parte do conjunto de terminais é incluído nesse conjunto. Sob um ponto de vista prático, essa situação pode modelar o caso onde um nó deseja passar a fazer parte do conjunto de membros de *multicast*.

Um nó solicita autorização ao nó raiz para inclusão. Ao receber esta autorização, o terminal a ser incluído inicia o crescimento de um fragmento na maneira usual do Dual Ascent. Durante o último *convergecast* no fragmento, a mensagem *Conv(End)* carrega o custo acrescido à solução do problema de Steiner por motivo da inclusão do terminal. Para isto, durante a propagação do *Conv(End, Value)*, cada nó intermediário, caso ainda

não participe da solução, passa a se considerar como participante e soma o custo de seu arco local de saída a *Value*. Esse valor vai sendo carregado em *convergecast* até o líder do fragmento. Além da contribuição do terminal ao limite inferior como já ocorria no Dual Ascent estático, também é enviado ao nó raiz, com a mensagem *Broad(End)*, a contribuição do terminal ao custo da solução.

A situação é ilustrada pela Figura 8.1, onde linhas cheias indicam arcos pertencentes a solução do problema e linhas pontilhadas arcos ainda não participantes. Na parte "a" da figura, um terminal atinge o nó raiz em um ciclo de crescimento; a parte "b" mostra a propagação da mensagem *Conv(End, Value)*; na parte "c", o terminal informa esse acréscimo na mensagem *Broad(End)*. Note que na parte "b" da figura, a primeira mensagem *Conv(End, Value)* carrega zero para *Value* pois o arco de custo 11 já pertencia à solução do problema antes da inclusão do terminal.

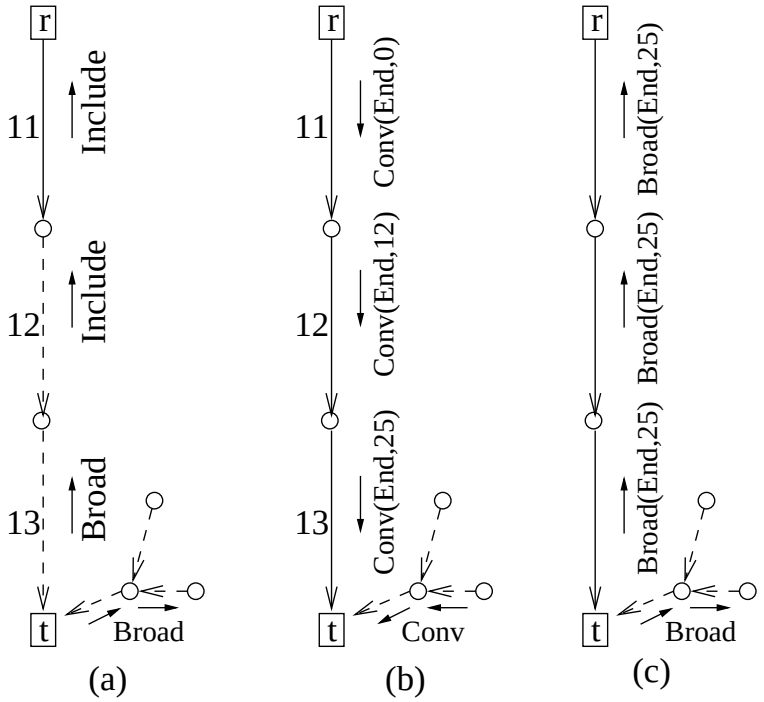

Figura 8.1: Inclusão de um terminal

8.2.2 Exclusão de Terminais

Lidamos aqui com a situação onde um nó que faça parte do conjunto de terminais é excluído desse conjunto, tornando-se um nó comum. Sob um ponto de vista prático, essa situação pode modelar o caso onde um membro deixa de ter interesse na participação em um grupo de *multicast*.

Quando um terminal sai do conjunto de terminais, devemos subtrair sua contribuição do limite inferior global e do custo da solução. Portanto, no nó raiz, devemos armazenar separadamente a contribuição de cada terminal a cada um desses valores.

As contribuições de cada terminal para o valor do custo reduzido dos arcos locais de todos os nós também devem ser canceladas por ocasião de uma exclusão, logo, esses valores também devem ser armazenados separadamente em cada nó. Devemos cancelar as saturações feitas por um terminal que esteja sendo excluído do conjunto de terminais. Essas saturações ocorreram nos arcos da árvore de saturação associada ao terminal, portanto, ao receber a autorização para exclusão, o terminal inicia um *broadcast* em sua árvore de saturação, informando ao seu fragmento a intenção de sair do conjunto de terminais.

A exclusão de um terminal não implica necessariamente em sua saída da solução. Um ex-terminal, pode estar sendo utilizado como nó intermediário no caminho entre o nó raiz e um nó atualmente componente do conjunto de terminais, portanto, a saída da solução só ocorre quando o terminal excluído for uma folha dessa árvore. Nesse caso, outros nós que se tornem folhas e não estejam no conjunto de terminais também podem ser excluídos da solução.

Cada nó mantém a informação de quais terminais o utilizam para a conexão com o nó raiz e se considera fora da solução somente quando não é utilizado por nenhum terminal. Um nó sabe que é necessário para a conexão de um terminal quando é utilizado para propagação da mensagem *Conv(End)* no último *convergecast* de crescimento do fragmento, quando o nó raiz é incluído. Ao receber uma mensagem de *broadcast* de exclusão, o nó passa a saber qual terminal foi excluído do conjunto de terminais e pode então avaliar se faz parte ou não da solução do problema de Steiner.

Cada nó, ao ser atingido pelo *broadcast* de exclusão, cancela os valores baixados pelo terminal em seus arcos locais. Um arco saturado pode então deixar esse estado. Nesse caso, o nó envia mensagens aos líderes dos fragmentos que utilizam o arco cujo estado foi alterado em suas árvores de saturação, informando que suas árvores e grafos de saturação foram danificados e devem ser reconstruídos, por meio da mensagem *React*. A reconstrução da árvore e grafos de saturação é feita com a exclusão do nó líder do fragmento do grupo de terminais seguida de sua reinclusão. Chamaremos este processo de exclusão e posterior inclusão de **reativação** do terminal. A inclusão e a exclusão decorrentes da reativação também necessitam de autorização da raiz.

Considere agora a seguinte seqüência de fatos ilustrada pela Figura 8.2 envolvendo os terminais t_1, t_2 e t_3, conectados como na parte "a" da figura. O custo total associado

ao arco (r, v_1) é 12. Os números entre parênteses indicam quanto cada terminal baixou do custo total do arco (r, v_1). Na parte "a", por exemplo, t_1 baixou 3, t_2 baixou 4, e t_3 baixou 5 do custo total de (r, v_1), portanto, o arco encontra-se saturado. Arcos saturados são mostrados em linhas contínuas, e arcos não saturados em linhas descontínuas.

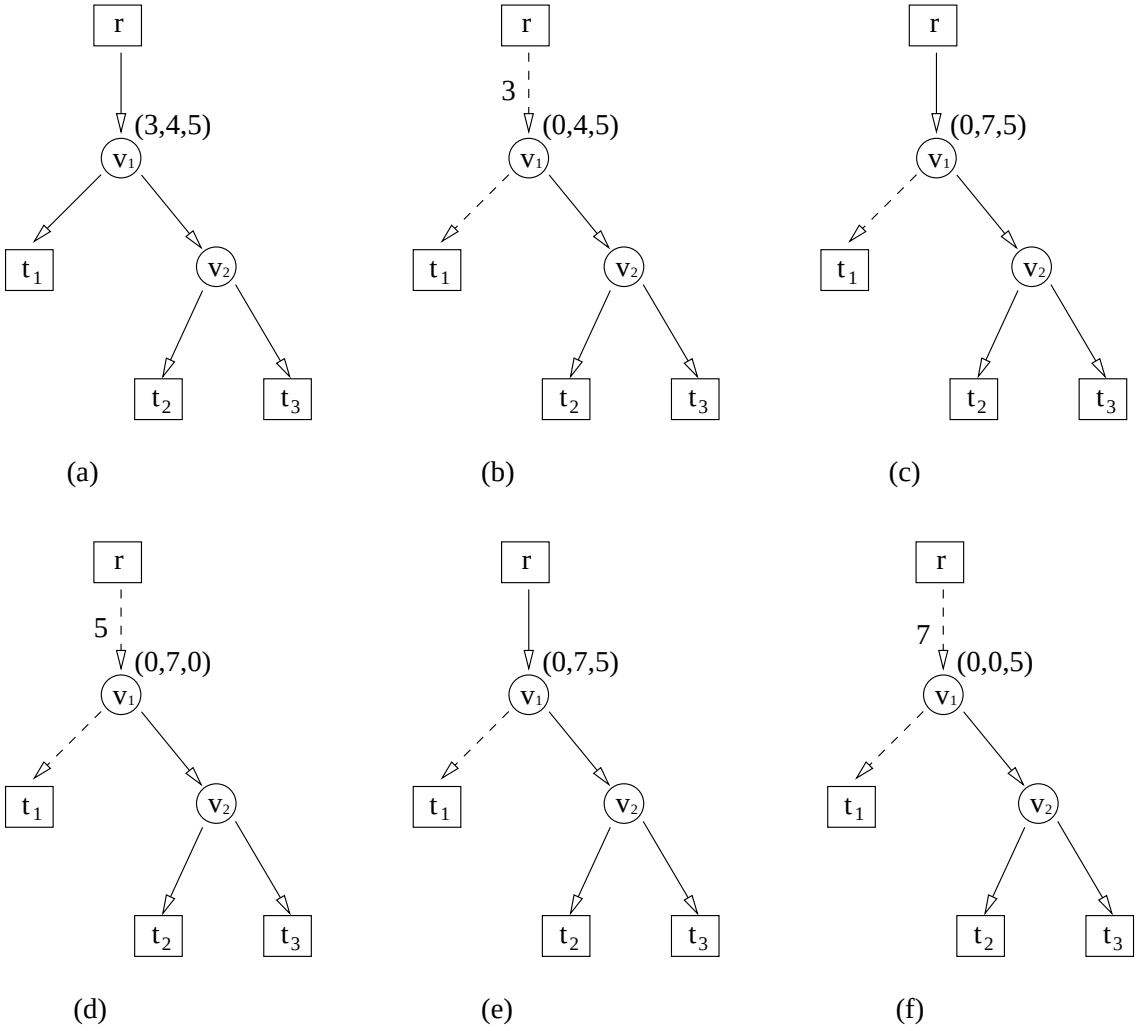

Figura 8.2: Situação de *livelock* potencial.

1. t_1 é excluído do conjunto de terminais e, por ser cancelada a saturação do arco (r, v_1) o nó v_1 solicita a reativação de t_2 e t_3 (parte "b" da figura);

2. Suponha que t_2 seja uma máquina muito rápida, e consiga se excluir e reincluir no sistema antes que t_3 reaja. A situação ficará como indicado na parte "c" da figura, com o terminal t_2 estável;

3. Agora o terminal t_3 executa a primeira fase da reativação, excluindo-se do conjunto de terminais, como indicado na parte "d" da figura. Durante o processo, v_1 detectará que t_2 também utiliza o arco (r, v_1) e solicitará sua reativação;

4. Suponhamos agora que, antes de t_2 reagir, t_3 seja reincluido no conjunto de terminais, como indicado na parte "e" da figura;

5. Suponha que agora t_2 inicie o processo de reativação. Durante a execução da primeira parte da rotina de reativação, t_2 causará a reativação de t_3 (parte "f" da figura).

Ocorre que, em decorrência de reativações, t_2 é excluído reativando t_3 que por sua vez é excluído reativando t_2. Configura-se um ciclo que pode constituir um *livelock*. Portanto, a raiz não autoriza inclusões, decorrentes ou não de reativações, enquanto estiverem sendo processadas exclusões ou existam operações de exclusão a serem processadas devido a reativações. Evitando que nós sejam incluídos, enquanto outros estão em processo de exclusão, prevenimos o problema. Ou seja, na etapa 4 do exemplo acima, a raiz não autorizaria a reinclusão de t_3 até que t_2 terminasse a exclusão.Para que o nó raiz saiba se existem exclusões em processamentos, nós que terminem a exclusão devem informar o fato ao nó raiz.

Juntamente com a informação de término de exclusão, o terminal enviará ao nó raiz o número de solicitações de reativação enviadas e recebidas durante o processo. Comparando os totais de solicitações enviadas e recebidas informados pelos terminais, o nó raiz saberá se existem ou não processamentos em curso e se pode ou não autorizar inclusões. Note que se estiver ocorrendo uma inclusão, exclusões serão autorizadas, pois inclusões não geram reativações, não podendo levar a situações de *livelock*.

Se um terminal decidir espontaneamente por sua exclusão durante uma reativação, ele simplesmente não solicita sua reinclusão ao final da exclusão. Se receber uma solicitação de reativação durante o processamento de exclusão, ignora o pedido pois não existe necessidade de reconstrução de estruturas que não serão mais utilizadas.

Pode ocorrer de um terminal receber uma mensagem de reativação atrasada, após já ter terminado o processamento de exclusão e encontrar-se em processo de reinclusão. Como o número de mensagens de reativação recebidas é informado pelo terminal excluído ao nó raiz no final da rotina de exclusão, torna-se necessária a informação ao nó raiz desse pedido de reativação atrasado em uma mensagem avulsa ($Late\_React$).

8.2.3 Reorganização

Com a estabilização do sistema, após a inclusão e/ou exclusão de terminais, o nó raiz pode comparar o limite inferior e o custo da solução para o problema de Steiner

atuais. Caso a diferença esteja acima de um determinado limite, uma reorganização é disparada. Uma reorganização consiste na alteração completa ou parcial da solução atual. Isso pode ser conseguido por qualquer algoritmo distribuído para o problema de Steiner. Independente do algoritmo escolhido para a reorganização, podemos limitar seu grafo de entrada ao grafo de saturação obtido com o DA, diminuindo custos de execução e podendo, mesmo assim, esperar qualidade superior para a solução.

À medida em que seqüências de inclusões vão sendo processadas, podemos esperar que os ganhos com o DA diminuam, pois terminais são incluídos a um sistema que já possui grandes grafos de saturação formados e essa situação leva a resultados piores segundo [Werneck 2001]. Entretanto, observe que as exclusões de terminais causam a exclusão e a reinclusão de outros terminais em uma região específica do grafo. As reinclusões não são autorizadas pelo nó raiz enquanto houver exclusões em processamento. Terminado o processamento de todas as exclusões, as autorizações para reinclusão serão enviadas pelo nó raiz. Os terminais reincluídos iniciam a recriação de seus grafos/árvores de saturação em paralelo, tendendo a formar grafos de saturação com tamanhos mais parecidos, como na versão estática do algoritmo. As exclusões funcionam como reorganizações dos grafos/árvores de saturação, diminuindo o efeito indesejado das inclusões de terminais a partir de um sistema com grandes grafos de saturação já formados.

8.3 Pseudo Código

Especificamos rotinas adicionais às do pseudo código da Seção 4.5 a fim de adaptá-lo para o funcionamento *on-line*. A rotina A.1 da Seção 4.5 não será mais de ativação espontânea sendo esse papel assumido pelas rotinas E.1 e I.1 descritas abaixo, que iniciam a exclusão e inclusão, respectivamente, de um elemento do conjunto de terminais.

Cada nó deverá manter:

- **reduc(t, a)** : Total reduzido do custo original do arco de entrada **a** por cada fragmento **t**;
- **in_tree**: Lógico. Se verdadeiro, indica que o nó já participa da solução;
- **sent_reacts(t)** : Número de solicitações de reativação enviadas pela exclusão do terminal **t**;
- **rec_reacts** : Número de solicitações de reativação recebidas.

A variável **fragment_state(t)** ganha três novas possibilidades. Além das [*unknown, active, finished, suspended*] já existentes, [*leaving , entering, reactivating*] passam a existir.

O nó raiz mantém as variáveis:

- **gt(t)** : Parcela do custo da solução atual referente ao terminal **t**;
- **gb(t)** : Parcela do limite inferior atual referente ao terminal **t**;
- **reorg** : Lógico. Se verdadeiro indica que o sistema está executando rotina de reorganização;
- **wait_leave** : Conjunto de nós que aguardam autorização para exclusão;
- **wait_join** : Conjunto de nós que aguardam autorização para inclusão;
- **leaving** : Número de terminais em processo de exclusão;
- **joining** : Número de terminais em processo de inclusão;
- **pending_reacts** : Número de mensagens *React* ainda não processadas pelo sistema. Se diferente de zero, indica sistema em estado transiente;
- **limit** : Limite tolerado para distância percentual entre o custo da árvore e o limite inferior.

As mensagens *Leave*, *Join* e *Late_React* das rotinas E.1, I.1 e E.7, respectivamente, são encaminhadas de um terminal ou candidato a terminal ao nó raiz, que não é necessariamente seu vizinho. Para o caso das mensagens *Leave* e *Late_React*, o sistema dispõe de uma árvore entre o nó raiz e o terminal que pode ser utilizada para o encaminhamento desta mensagem. Para o caso da mensagem *Join*, o sistema ainda não possui qualquer informação sobre o candidato a terminal, e não possui condições de dar suporte a esta comunicação. Para este encaminhamento, podemos utilizar qualquer algoritmo distribuído que descubra uma rota entre o candidato a terminal e o raiz. Do ponto de vista prático, podemos utilizar algum sistema de comunicação *unicast* oferecido pelo ambiente. Observe que podemos utilizar qualquer mecanismo de comunicação *unicast*, e o DA on-line não utiliza qualquer estrutura de dados interna do algoritmo de encaminhamento *unicast*, mantendo a independência entre os algoritmos *unicast* e *multicast*.

Algoritmo:

(E.1) Ativação espontânea, executada quando um terminal *t* deseja sair do conjunto de terminais:

1 **encaminhe** LEAVE(t, REQ) ao raiz;

(E.2) Ao receber LEAVE(t, REQ):

1 gb(t) ← 0;
2 gt(t) ← 0;
3 **se** reorg
4 wait_leave ← wait_leave ∪ {t}
5 **senão**
6 **encaminhe** LEAVE(t, OK) a t;
7 leaving ← leaving + 1;

(E.3) Ao receber LEAVE(t, Ok):

1 sent_reacts ← 0;
2 rec_reacts ← 0;
3 waited_conv ← 0;
4 fragment_state(t) ← leaving;
5 **para** cada arco local a tal que state(t,a)=entering :
6 **envie** BROAD(t, LEAVE) pelo arco a';
7 waited_conv ← waited_conv + 1;

(E.4) Ao receber BROAD(t, LEAVE):

```
1 sent_reacts(t) ← 0;
2 para cada arco local de entrada a :
3    se reduc(t, a) > 0
4        reduc(t, a) ← 0;
5        para cada terminal t' tal que state(t', a)=entering :
6            envie REACT(t) pelo arco a' tal que state(t, a')=to_leader;
7            sent_reacts(t) ← sent_reacts(t) + 1;
8 waited_conv ← 0;
9 para cada arco local a tal que state(t,a)=entering :
10     envie BROAD(t, LEAVE) por a;
11     waited_conv ← waited_conv + 1;
10 fragment_state(t) ← unknown;
12 se ∄t' tal que fragment_state(t') = finished
13    in_tree = false;
14 se waited_conv = 0 execute procedimento convergecast_leave(t);
```

(E.5) Ao receber CONV(t, LEAVE, s_reacts):

```
1 sent_reacts(t) ← sent_reacts(t) + s_reacts;
2 waited_conv ← waited_conv - 1;
3 se waited_conv = 0 execute procedimento convergecast_leave(t);
```

(E.6) procedimento convergecast_leave(t):

```
1 se nó é o líder de t
2    envie LEAVE(t, END, sent_reacts(t), rec_reacts) ao raiz;
3    se fragment_state(t) = leaving;
4        fragment_state(t) ← unknown;
5    senão
6        fragment_state(t) ← entering;
7        encaminhe JOIN(t, REQ) ao raiz;
8 senão
9    envie CONV(t, LEAVE, sent_reacts(t)) pelo arco a tal que state(t, a)= to_leader;
```

(E.7) Ao receber REACT(t):

```
1 se nó é líder de t
2    se fragment_state(t) = finished
3        fragment_state(t) ← reactivating;
4        envie LEAVE(t, REQ) ao raiz;
5    senão se fragment_state(t) = leaving ou fragment_state(t) = reactivating
6        rec_reacts ← rec_reacts + 1;
7    senão
8        encaminhe LATE_REACT(t) ao raiz;
9 senão
10     envie REACT(t) pelo arco aa tal que state(t, a)=to_leader;
```

(E.8) Ao receber LEAVE(t, END, s_reacts, r_reacts):

```
1 pending_reacts ← pending_reacts + s_reacts - r_reacts;
2 leaving ← leaving - 1
3 se pending_reacts = 0 e leaving = 0 e joining = 0
4     execute verif_reorg;
```

(E.9) Ao receber LATE_REACT():

```
1 pending_reacts ← pending_reacts - 1
2 se pending_reacts = 0 e leaving = 0 e joining = 0
3     execute verif_reorg;
```

(I.1) Ativação espontânea, executada quando um terminal *t* deseja entrar no conjunto de terminais:

```
1 encaminhe JOIN(t, REQ) ao raiz;
```

(I.2) Ao receber JOIN(t, REQ):

```
1 se reorg ou pending_reacts ≠ 0 ou leaving ≠ 0
2     wait_join ← wait_join ∪ {t}
3 senão
4     encaminhe JOIN(t, OK) a t;
5     joining ← joining + 1;
```

(I.3) Ao receber JOIN(t, OK):

```
1 fragment_state(t) ← joining;
2 execute A.1, iniciando o crescimento de um fragmento
```

(R.1) procedimento verif_reorg:

```
1 se wait_join ≠ ∅
2     encaminhe JOIN(t, OK) para todo t ∈ wait_join;
3     joining ← joining + |wait_join|;
4     wait_join ← ∅;
5 senão
6     gt ← 0;
7     gb ← 0;
8     para cada terminal t :
9         gt ← gt + gt(t);
10        gb ← gb + gb(t);
11    se (gt / gb) > limit
12        reorg ← true;
13        execute reorganização;
14        reorg ← false;
15    se wait_leave ≠ ∅
16        encaminhe LEAVE(t, OK) para todo t ∈ wait_leave;
17        leaving ← leaving + |wait_leave|;
18        wait_leave ← ∅;
19    senão se wait_join ≠ ∅
20        encaminhe JOIN(t, OK) para todo t ∈ wait_join;
21        joining ← joining + |wait_join|;
22        wait_join ← ∅;
```

Para conseguirmos manter o custo da solução atualizada, algumas rotinas do Dual Ascent distribuído devem sofrer alterações. A mensagem *Conv(t, type)* passa a conter o incremento ao custo da solução que vai sendo calculado no último *convergecast*, quando *type=End*. Este valor deve ser armazenado em uma variável($inc\_tree(t)$) local durante o *convergecast*, como indicado:

(A.6) Ao receber CONV(t, type, *value*) pelo arco $a = (v_1, v_2)$:

...

senão se type = END

    end$(t, v_1) \leftarrow$ TRUE;

    inc_tree(t) $\leftarrow$ *value*;

...

Os procedimentos **(A.8) leader-broadcast(t)** e **(A.9) non-leader-convergecast(t)** passam a:

(A.8) procedimento leader-broadcast(t):

se end(t,v) = TRUE para pelo menos um vizinho v;

    envie BROAD(t, END, pb, inc_tree(t)) por cada arco a tal que state(t, a)=to_leaf;

...

(A.9) procedimento non-leader-convergecast(t):

se end(t, v) = TRUE para pelo menos um vizinho v;

    fragment_state(t) $\leftarrow$ finished;

    se não in_tree

        in_tree $\leftarrow$ true;

        inc_tree(t) $\leftarrow$ inc_tree(t) + c_a tal que state(t, a)=to_leader;

    envie CONV(t, END, inc_tree(t)) pelo arco a tal que state(t, a)=to_leader;

...

Na rotina (A.13), onde o nó raiz recebe os dados do terminal conectado, deverá ser processado o acréscimo no custo da solução e não haverá mais terminação. Uma verificação sobre a necessidade de reorganização após cada operação de inclusão deve ser realizada, caso o sistema esteja estável.

(A.13) Ao receber BROAD(t, END, pb, inc_tree) pelo arco $a = (v_1, v_2)$;

...

10 gb(t) ← pb;

11 gt(t) ← inc_tree;

12 joining ← joining - 1;

13 **se** pending_reacts = 0 **e** leaving = 0 **e** joining = 0

14 **execute** verif_reorg;

...

8.4 Exemplos

Na Figura 8.3 exibimos uma possível execução da heurística para o problema *on-line*, fazendo referência à rotina do pseudo código em execução em cada situação. A figura mostra também, quando oportuno, estruturas de controle para o estado do sistema.

Na Figura 8.3 parte "a", um terminal t solicita autorização para inclusão no conjunto de terminais enviando a r uma mensagem *Join(Req)* (rotina I.1). Como o sistema está estável, r autoriza a inclusão respondendo com a mensagem *Join(Ok)* (rotina I.2). O terminal t inicia o crescimento de um fragmento e o processo continua como no DA estático, até que o terminal raiz seja incluído no fragmento. As reticências da figura representam a árvore de saturação que vai sendo criada durante o crescimento do fragmento, cujos nós são omitidos. O terminal t informa então no último *broadcast* o limite inferior parcial (pb) e o custo adicionado à árvore de distribuição (pt) (rotina A.13).

Na parte "b" da Figura 8.3, observamos a exclusão de um terminal. O terminal t pede autorização para exclusão enviando a mensagem *Leave(Req)*(rotina E.1). Admitiremos que nesse ponto o sistema encontre-se em processo de reorganização. Nesse caso r registra que t aguarda por uma autorização para exclusão e envia a mensagem *Leave(Ok)* apenas após o término da reorganização (rotina E.2). Recebendo o *Leave(Ok)*, t informa a seu fragmento a exclusão com um *broadcast* em sua árvore de saturação (rotina E.3). O fragmento informa em *convergecast* que não foram reativados outros terminais (mensagem *Conv(Leave, 0)*). Neste ponto t pode informar a r por meio da mensagem *Leave(End, 0, 0)* que terminou o processo de exclusão (rotina E.6, linha 2). Os dois zeros significam que não foram nem enviadas nem recebidas mensagens *React*.

Mais dois exemplos com situações mais complexas, podem ser observados no apêndice

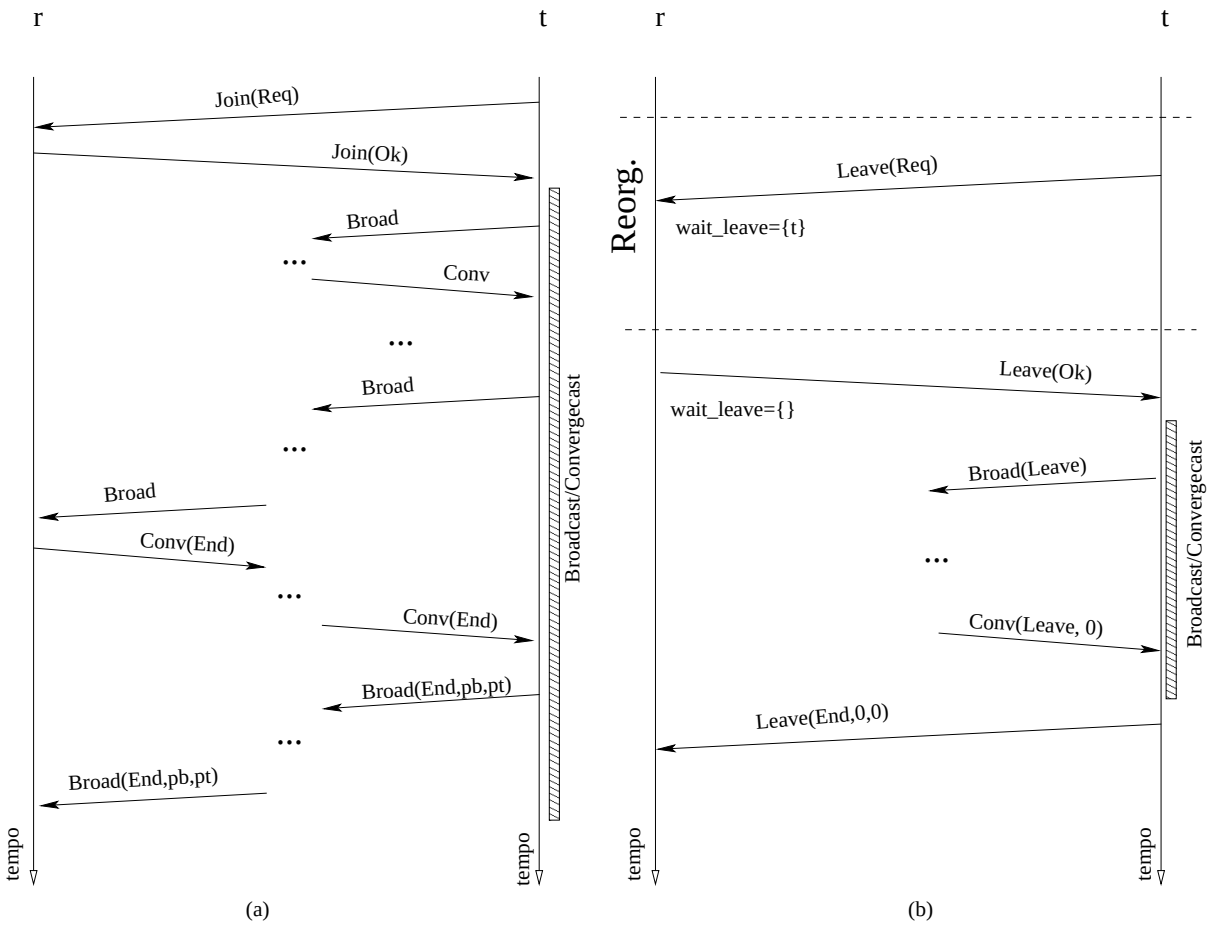

Figura 8.3: Possível execução para o problema *on-line* - inclusão, reorganização e exclusão.

A.

8.5 Análise do Algoritmo

Todo o processamento de inclusão e exclusão de terminais pode ser feito sem qualquer perturbação à solução que esteja sendo utilizada para distribuição de mensagens em *multicast* no momento. Somente a reorganização altera rotas existentes para terminais. Ainda assim, pode-se continuar utilizando a solução antiga enquanto se calcula a nova, resumindo a perturbação ao mínimo, durante a troca de árvores.

8.5.1 ARIES x Dual Ascent *On-Line*

O ARIES parte do pressuposto de que regiões da solução que tenham sofrido muitas alterações serão necessariamente regiões onde a qualidade da solução obtida por heurísticas se afastará do ótimo. Essa presunção é razoável, mas podem existir casos em que

muitas alterações ocorram e as inclusões por caminho mínimo continuem gerando árvores de custo baixo, próximo ao ótimo ou, de forma inversa, uma única alteração gere uma árvore com custo muito superior ao ótimo. Neste trabalho disparamos reorganizações somente quando o custo da solução torna-se muito alto, se comparado ao limite inferior fornecido pelo Dual Ascent.

Outra diferença importante entre o ARIES e nossa abordagem é que o ARIES pressupõe o prévio conhecimento dos caminhos mínimos entre todos os pares de nós, sob a justificativa de que essa informação já consta das tabelas de roteamento nas redes práticas. O Dual Ascent *on-line* calcula os caminhos mínimos durante sua execução, o que permite sua utilização mesmo quando a métrica cuja minimização é desejada seja diferente da utilizada pelos roteadores.

A seguir, analisamos a complexidade de pior caso para as operações de inclusão e exclusão de terminais, quanto ao número de mensagens enviadas e ao tempo, dado pelo tamanho da maior cadeia de mensagens que sustentem relação de causalidade do tipo enviar uma mensagem em decorrência do recebimento de outra.

Como veremos a seguir, reorganizações mais oportunas e a qualidade de solução superior do DA *on-line* cobram seu custo. O DA é mais caro computacionalmente do que os algoritmos mais simples, baseados nos caminhos mínimos como o EBA e o ARIES. Esperamos que, na prática, parte desse custo nas operações de inclusão e exclusão de terminais seja compensada por um número menor de reorganizações desnecessárias e uma reorganização mais rápida por atuar sobre o grafo de saturação.

8.5.2 Complexidades para Inclusão e Exclusão de Terminais

Considerando o procedimento de inclusão de um nó, quanto ao número de mensagens, no pior caso um terminal incluirá todos os nós do grafo que enviará mensagens *Check* para todos os demais. Portanto, a complexidade de mensagens $O(|V|^2)$. Para o tempo, o pior caso ocorrerá quando os nós são incluídos um a um formando um caminho até o nó raiz ser incluído. Nesse caso o número de mensagens da maior cadeia de eventos com relação de causalidade será dado pelo somatório $2 + 4 + 6 + 8 + ... + (V - 2)$, que é também $O(|V|^2)$.

A título de comparação, se considerarmos que os caminhos mínimos já estão calculados, a complexidade de pior caso para uma operação de inclusão no ARIES é $O(|V|)$ para tempo e mensagens. As complexidades para o cálculo dos caminhos mínimos são

$O(|E|.|V|)$ para mensagens e $O(|V|)$ para tempo. No caso *on-line* não é justo que simplesmente acrescentemos este custo à execução do algoritmo, já que ele é cobrado apenas uma vez para todas as operações e, portanto, seu peso é tanto menor quanto maior for o número de operações.

Considerando a exclusão de um terminal, no pior caso, esta ocasionará a reativação de todos os demais terminais, elevando as complexidades de tempo e mensagens aos níveis do DA estático: $O(|T|.|V|^2)$. A complexidade em tempo e mensagem para exclusão de terminais no ARIES é $O(|V|)$.

8.6 Resultados Experimentais

Implementamos o Dual Ascent dinâmico em ANSI C e MPICH2 e executamos em um único computador Pentium Dual Core com 1 GByte de memória e MPICH2 versão 1.0.7 e com as instâncias simuladoras da Internet descritos no Capítulo 6. Para cada grafo, foi gerada uma seqüência de 15 inserções de terminais escolhidos aleatoriamente no conjunto de nós. Os 15 nós escolhidos constituem o conjunto inicial de terminais e foram incluídos todos no instante 0 da simulação, utilizando a operação de inserção dinâmica, descrita anteriormente.

A seguir, foram geradas seqüências de operações de inclusão ou exclusão. Para cada operação, sorteou-se inicialmente se seria uma operação de inclusão ou exclusão. Para o caso de uma operação de exclusão, sorteou-se um terminal e para uma operação de inclusão, um nó não terminal. Foram utilizadas funções de distribuição de probabilidade homogênea em ambos os sorteios.

Para cada grafo, geraram-se duas seqüências de operações. A primeira com 16 operações, separadas por tempos suficientes para que o sistema se estabilizasse e permitisse uma reorganização, caso necessário. Na segunda, foram geradas 60 operações, agrupadas de 6 em 6. As operações de um grupo são disparadas em paralelo. Entre cada grupo de 6 operações, guardou-se um tempo suficiente para a estabilização do sistema.

Foi utilizado o limite de 40% de distância tolerada entre o custo da solução e o limite inferior obtido com o Dual Ascent. Ultrapassado esse limite, uma reorganização seria disparada.

Com a finalidade de comparação, executamos o mesmo algoritmo para inserção e exclusão de nós, porém implementando os contadores do ARIES como critério para disparo

Instância		Número de Reorganizações (%)		Ganho Médio por Reorganização (%)		Distância Média para Ótimo (%)	
		DA	ARIES	DA	ARIES	DA	ARIES
GLP (1 oper. por vez)	1	**0**	8	-	1,4	11,7	**6,1**
	2	**2**	5	**32,7**	12,4	16,2	**15,2**
	3	**0**	6	-	0,3	3,9	**3,4**
	4	**1**	7	**38,0**	7,3	6,3	**5,0**
	5	**4**	6	**24,7**	11,3	**12,0**	21,5
GLP (6 oper. por vez)	1	**0**	8	-	2,4	9,3	**6,7**
	2	**4**	5	**27,2**	20,1	**30,4**	46,3
	3	**0**	6	-	0,3	0,7	**0,6**
	4	**2**	6	**25,2**	14,0	34,6	**21,8**
	5	**4**	7	**28,8**	8,6	**11,7**	22,8
SW (1 oper. por vez)	1	**6**	9	**32,1**	26,6	**30,8**	41,1
	2	**4**	5	**27,5**	23,6	**15,7**	18,0
	3	**3**	6	**41,6**	20,7	**13,5**	27,8
	4	**1**	5	**20,6**	2,3	**20,0**	24,3
	5	**0**	6	-	2,7	7,4	**3,6**
SW (6 oper. por vez)	1	**7**	**7**	**37,5**	24,9	**10,1**	21,9
	2	**2**	7	**24,7**	6,4	24,7	**10,7**
	3	**5**	8	**16,6**	12,5	24,5	**19,6**
	4	**1**	8	**19,9**	6,0	15,5	**7,9**
	5	**0**	8	-	1,7	4,5	**3,1**
MÉDIA		2,3	6,7	28,4	10,3	15,2	16,4

Tabela 8.1: Comparação entre o critério para disparo de reorganizações baseado em contadores do ARIES e o critério baseado no limite inferior oferecido pelo Dual Ascent (DA).

de reorganizações. Neste caso, desprezamos o limite inferior e executamos reorganizações quando um contador ultrapassava o limite de 6, valor indicado em [Bauer e Varma 1997] como ideal.

Utilizamos a mesma rotina para reorganização em ambos os casos: o Prim-SPH descrito em [Novak et al. 2001a], principalmente por este algoritmo ser adaptável para o problema direcionado e apresentar excelentes resultados práticos.

Como podemos observar pela Tabela 8.1, utilizando o limite inferior do Dual Ascent como critério para disparo de reorganizações, reduziu-se o número de disparos a praticamente um terço, mantendo-se a distância média entre a solução heurística e ótima após cada reorganização em um patamar ligeiramente inferior. Isso foi possível porque realizando as reorganizações no momento em que realmente são necessárias, conseguindo um ganho médio por reorganização muito superior: 28,4% utilizando o Dual Ascent contra 10,3% utilizando o método do ARIES.

Nas figuras 8.4 e 8.5 podemos observar curvas com o ótimo (calculado com o algoritmo *branch-and-bound* descrito em [Poggi de Aragão et al. 2001]), limite inferior e soluções heurísticas durante um período de tempo em que o conjunto de terminais sofre inclusões ou exclusões. Reorganizações são disparadas com base no limite inferior ou nos contadores do ARIES. Círculos próximos ao eixo das ordenadas indicam reorganizações disparadas pelo método do ARIES e quadrados indicam reorganizações disparadas pelo método do limite inferior (DA). Em ambos os casos, as reorganizações foram feitas utilizando o algoritmo Prim-SPH [Novak et al. 2001a].

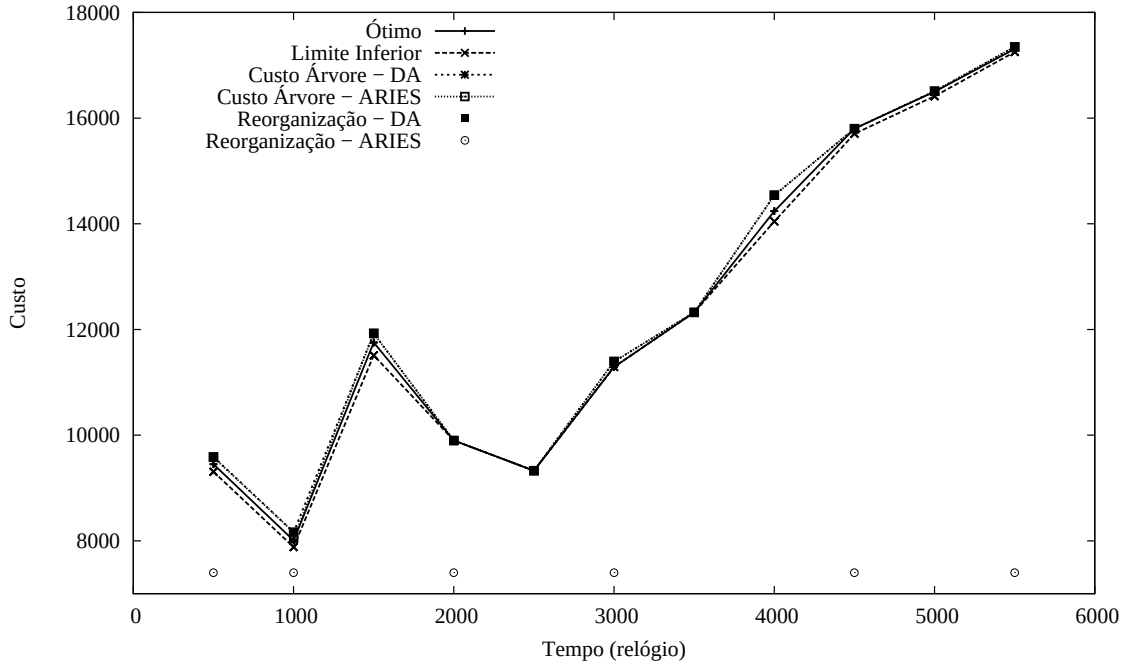

Figura 8.4: Instância GLP-3 com operações em grupos de seis.

Para algumas instâncias, o método guloso produz resultados muito bons, como na instância GLP-3, com operações realizadas em grupos de seis. Podemos observar no gráfico da Figura 8.4, que solução ótima, limite inferior e soluções heurísticas apresentam diferenças muito pequenas, sendo as reorganizações perfeitamente dispensáveis.

Para outras instâncias, no entanto, as soluções obtidas pelo método guloso não são tão boas. Este é o caso da instância SW-1 com operações feitas em grupos de seis. Como podemos observar pela Figura 8.5, as operações degradam muito rapidamente a qualidade da solução do problema. No início da simulação, ambas as estratégias disparam reorganizações que aproximam a qualidade da árvore de distribuição do ótimo. No entanto, a partir do instante 1.744, o método dos contadores não dispara reorganizações necessárias e o sistema fica com uma solução pobre por um grande período de tempo.

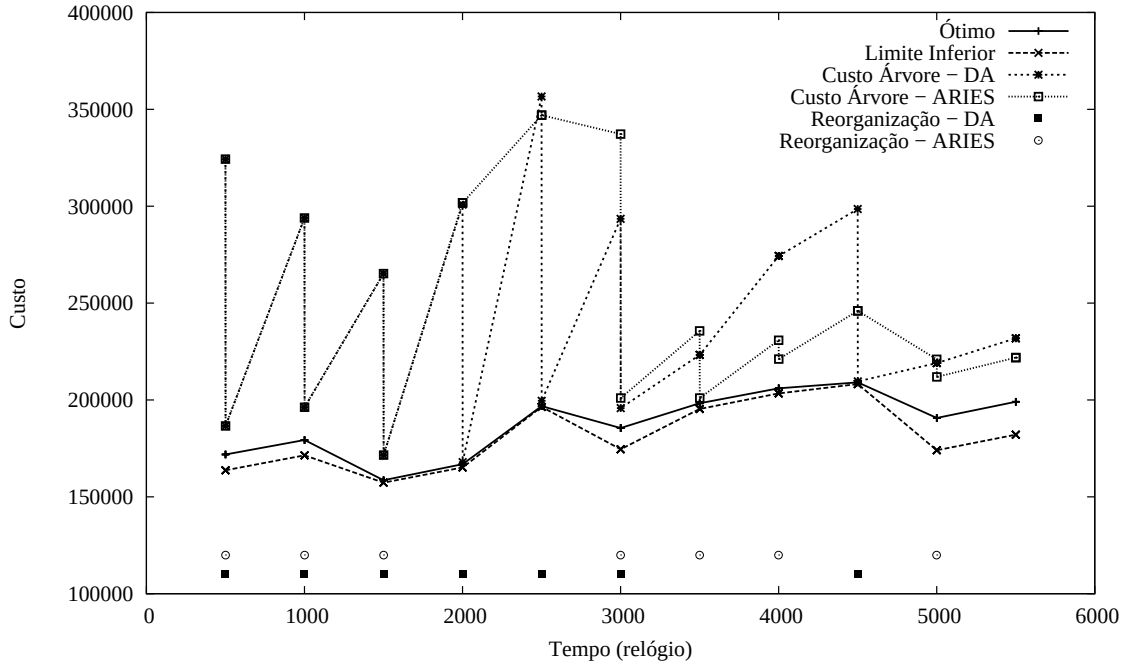

Figura 8.5: Instância SW-1 com operações em grupos de seis.

As duas instâncias escolhidas para a construção dos gráficos das figuras 8.4 e 8.5 são casos extremos. As demais instâncias apresentaram resultados intermediários nos testes práticos. Os gráficos de todas as instâncias podem ser observados no Apêndice B.

8.7 Conclusão

Foi apresentada uma possível adaptação do Dual Ascent para o problema *on-line*, onde nós podem ser incluídos e excluídos do conjunto de terminais com o passar do tempo, variação de grande interesse prático do Problema de Steiner em Grafos.

Exploramos a utilização do limite inferior fornecido pelo algoritmo como critério para realização de reconstruções parciais ou totais da solução quando o conjunto de terminais é alterado. Mostramos que este critério é mais eficiente do que os baseados em número de alterações sofridas em determinadas regiões do grafo, pois nem sempre o simples fato de haver ocorrido muitas alterações significa que a qualidade da solução tenha diminuído, justificando o custo da reconstrução e, muitas vezes, uma simples alteração pode degradar bastante a qualidade da solução, justificando a reconstrução. Como o limite inferior oferecido pelo Dual Ascent apresenta qualidade muito boa, apresentando valores muito próximos ao ótimo, sua utilização leva a reconstruções em situações muito mais oportunas do que o critério baseado em número de alterações.

Capítulo 9

Conclusão

Durante este trabalho desenvolvemos uma versão distribuída para o algoritmo Dual Ascent, cuja versão seqüencial, proposta por [Wong 1984], tem se mostrado uma eficiente heurística para solução do Problema de Steiner em Grafos. O interesse no desenvolvimento de boas heurísticas para este problema reside no fato do roteamento *multicast* poder ser modelado como um problema dessa classe. Várias aplicações como jogos *on-line*, bate-papo em tempo real, vídeo conferência e banco de dados distribuído, que dependem de roteamento *multicast* eficiente estão sendo mais utilizados com a expansão da internet. O algoritmo apresentado provê meios para obtenção de uma boa solução para o problema com grafos orientados e não orientados, além de fornecer um limite inferior de excelente qualidade, tipicamente 3% abaixo do ótimo, que pode ser utilizado para avaliação de soluções obtidas com o Dual Ascent ou quaisquer outras heurísticas.

Foram apresentadas análises teóricas e experimentais sobre um número razoavelmente grande e variado de instâncias. Obtivemos melhores soluções quando comparamos com as obtidas pelas principais heurísticas para o problema e verificamos que os custos de tempo e mensagens apresentados em situações práticas de execução do algoritmo são mais baixos que os limites teóricos calculados.

Foi mostrado como a versão distribuída do Dual Ascent pode ser utilizada para o problema de Steiner com limitação de saltos. Essa variação do problema é de grande interesse prático, pois modela muito bem a transmissão de multimídia em *multicast* pela Internet, quando as aplicações não toleram uma latência acima de determinados limites. A melhor qualidade da solução obtida com a utilização do novo algoritmo implica em aumento de custos, principalmente quando o limite de saltos tolerado for alto, mas não detectamos diferenças significativas no ritmo de crescimento desses custos. O algoritmo foi testado em instâncias que mimetizam a Internet (BRITE-GLP, SW) com resultados

muito semelhantes aos obtidos em instâncias geradas para avaliação do problema de um ponto de vista mais teórico (*SteinLib*), o que reforça nossa crença de que o Dual Ascent é uma alternativa a ser considerada para a distribuição de dados em *multicast* pela Internet.

Finalmente foi apresentada uma possível adaptação do Dual Ascent para o problema *on-line*, onde nós podem ser incluídos e excluídos do conjunto de terminais com o passar do tempo. Exploramos a utilização do limite inferior fornecido pelo algoritmo como critério para realização de reconstruções parciais ou totais da solução quando o conjunto de terminais é alterado. Mostramos que este critério é mais eficiente do que os baseados em número de alterações sofridas em determinadas regiões do grafo, pois nem sempre o simples fato de haver ocorrido muitas alterações significa que a qualidade da solução tenha diminuído, justificando o custo da reconstrução e, muitas vezes, uma simples alteração pode degradar bastante a qualidade da solução, justificando a reconstrução. Como o limite inferior oferecido pelo Dual Ascent apresenta qualidade muito boa, apresentando valores muito próximos ao ótimo, sua utilização leva a reconstruções em situações muito mais oportunas do que o critério baseado em número de alterações.

Os principais algoritmos encontrados na literatura partem de pressuposto de que as distâncias mínimas entre um nó e todos os demais do grafo estão disponíveis sem custos adicionais para o a resolução do problema de Steiner. Esta presunção baseia-se na situação prática onde algoritmos de roteamento *unicast* como o RIP e o OSPF já constroem tabelas de roteamento que possuem estas informações. Se desconsiderarmos o custo do cálculo da distância mínima nos algoritmos utilizados para comparação com o DA, conseguiremos uma diminuição substancial no tempo global e no número de mensagens enviadas por esses algoritmos. Admitimos inclusive que o aumento relativo nos custos do Dual Ascent possam não ser compensados totalmente pelo ganho na qualidade árvore de distribuição *multicast*, o que pode levar muitos projetistas de sistemas a optar por algoritmos mais simples. No entanto, a interdependência entre algoritmos para *unicast* e *multicast* não é desejada. Alterações nas estruturas do sistema de comunicação *unicast* podem inviabilizar a utilização dos algoritmos *multicast*. Nesta situação acreditamos que o Dual Ascent apresenta-se como uma alternativa valiosa na criação de protocolos *multicast* para utilização em redes com o porte da internet.

APÊNDICE A – Exemplos Adicionais de Possíveis Execuções do algoritmo On Line

Neste apêndice incluímos dois exemplos adicionais aos apresentados no Capítulo 8 do texto principal, ilustrando situações um pouco mais complexas do que a apresentada no texto principal.

Na Figura A.1 terminal t_1 solicita e recebe autorização para exclusão, o nó raiz registra a existência de um terminal saindo, somando 1 na variável *leaving* (rotinas E.1 e E.2). Os nós v_1 e v_2 fazem parte da árvore de saturação de t_1 e, ao receberem a comunicação da exclusão de t_1, detectam que o terminal t_2 utiliza reduções em custos de arcos feitas por t_1, portanto, enviam mensagens *React* ao terminal t_2 (rotina E.4). O terminal t_2 acumula o total de mensagens *React* recebidas em uma variável *rec_reacts*, pois terá de informar posteriormente ao terminal raiz esse valor (rotina E.5).

O terminal t_2 pede então autorização a r para sair do sistema (mensagem *Leave(Req)*) (rotina E.7). O nó r concede a autorização (*mensagem Leave(OK)*) e registra mais um terminal em processo de exclusão (*leaving*=2) (rotina E.2). O terminal t_2 executa o procedimento de exclusão sem reativar outros terminais, portanto, a mensagem *Leave(End, 0, 2)* é remetida a r, informando que não foram enviadas mensagens *React* e foram recebidas duas mensagens desse tipo. O terminal raiz subtrai 2 de *pending_reacts* e 1 de *leaving* em decorrência do término do processo de exclusão deste terminal. A variável *pending_reacts* passa a armazenar o -2 e *leaving* passa a conter 1 (rotina E.8).

Como observamos, *pending_reacts* pode conter valores negativos, portanto, uma outra exclusão que enviasse exatamente duas reativações e estivesse sendo processada em paralelo poderia causar que a variável contivesse zero em uma situação em que o sistema não estivesse estável. Observe, no entanto, que essa situação só pode ocorrer quando um terminal que tenha sido excluído do conjunto de terminais ainda não tenha informado na mensagem *Leave(End)* o total de reativações realizadas. Nesse caso, *leaving* será diferente de zero, prevenindo reorganizações com o sistema em estado transiente.

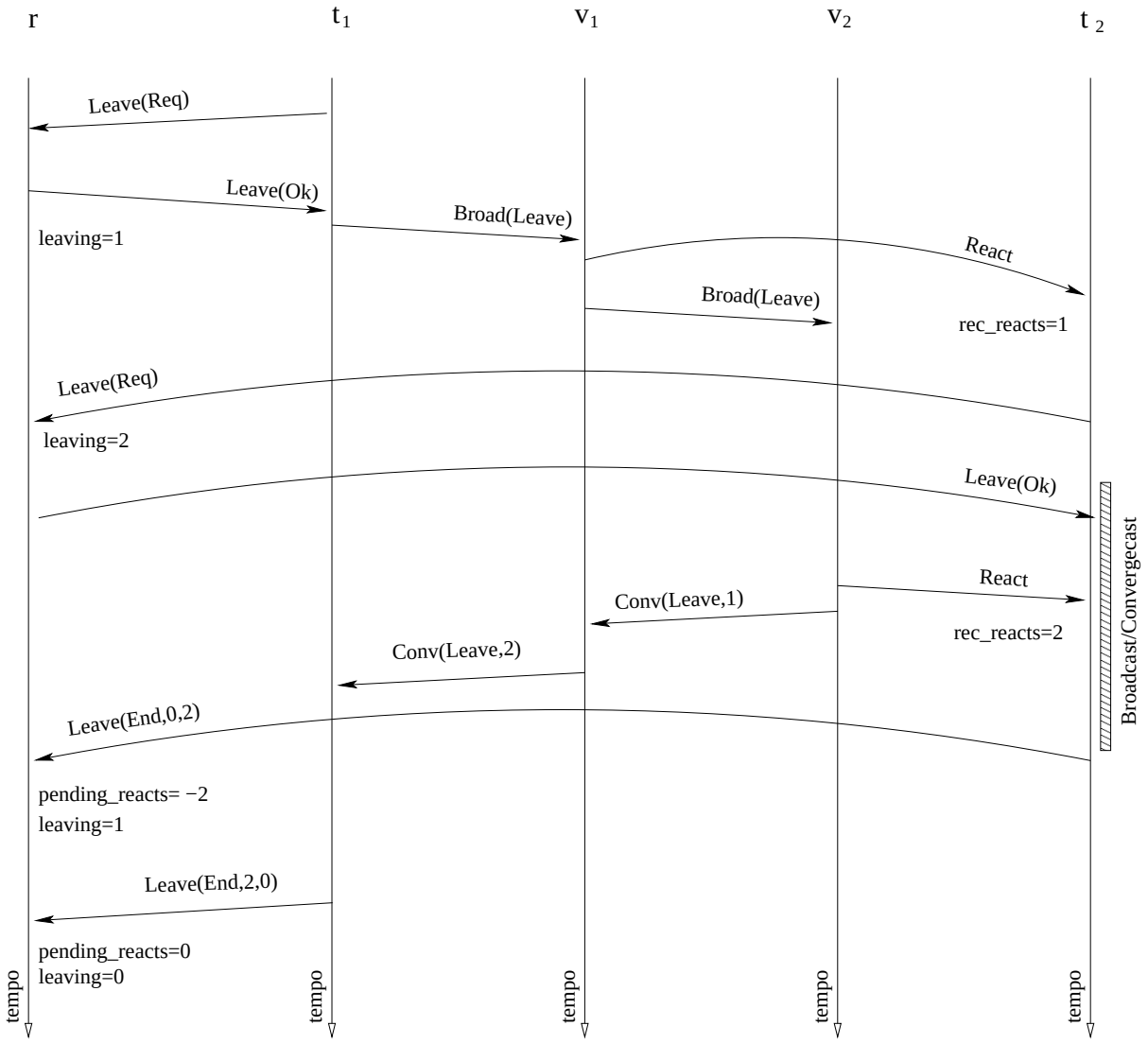

Figura A.1: Possível execução para o problema *on-line* - exclusão com uma reativação.

O *convergecast* de t_1 termina e ele sabe agora que foram enviadas duas mensagens *React* em decorrência de sua exclusão. Esse fato é informado então a r com uma mensagem *Leave(End, 2, 0)* (rotina E.6))que causa a soma de 2 a $pending\_reacts$ e a subtração de 1 de *leaving* (rotina E.8). As variáveis $pending\_reacts$ e *leaving* têm seus valores ajustados para 0, indicação segura de que não existem exclusões em processamento.

A Figura A.2 ilustra uma situação onde dois terminais são reativados, e um deles deve esperar que não haja mais exclusões em processamento para poder iniciar o processo de reinclusão. O terminal t_1 recebe autorização para exclusão e envia uma mensagem *Broad(Leave)* para um membro v_1 de sua árvore de saturação que está sendo desfeita (rotinas E.1, E.2 e E.3). O nó v_1 detecta que dois terminais, t_2 e t_3, faziam uso das saturações realizadas por t_1 e envia mensagens *React* para ambos (rotina E.4).

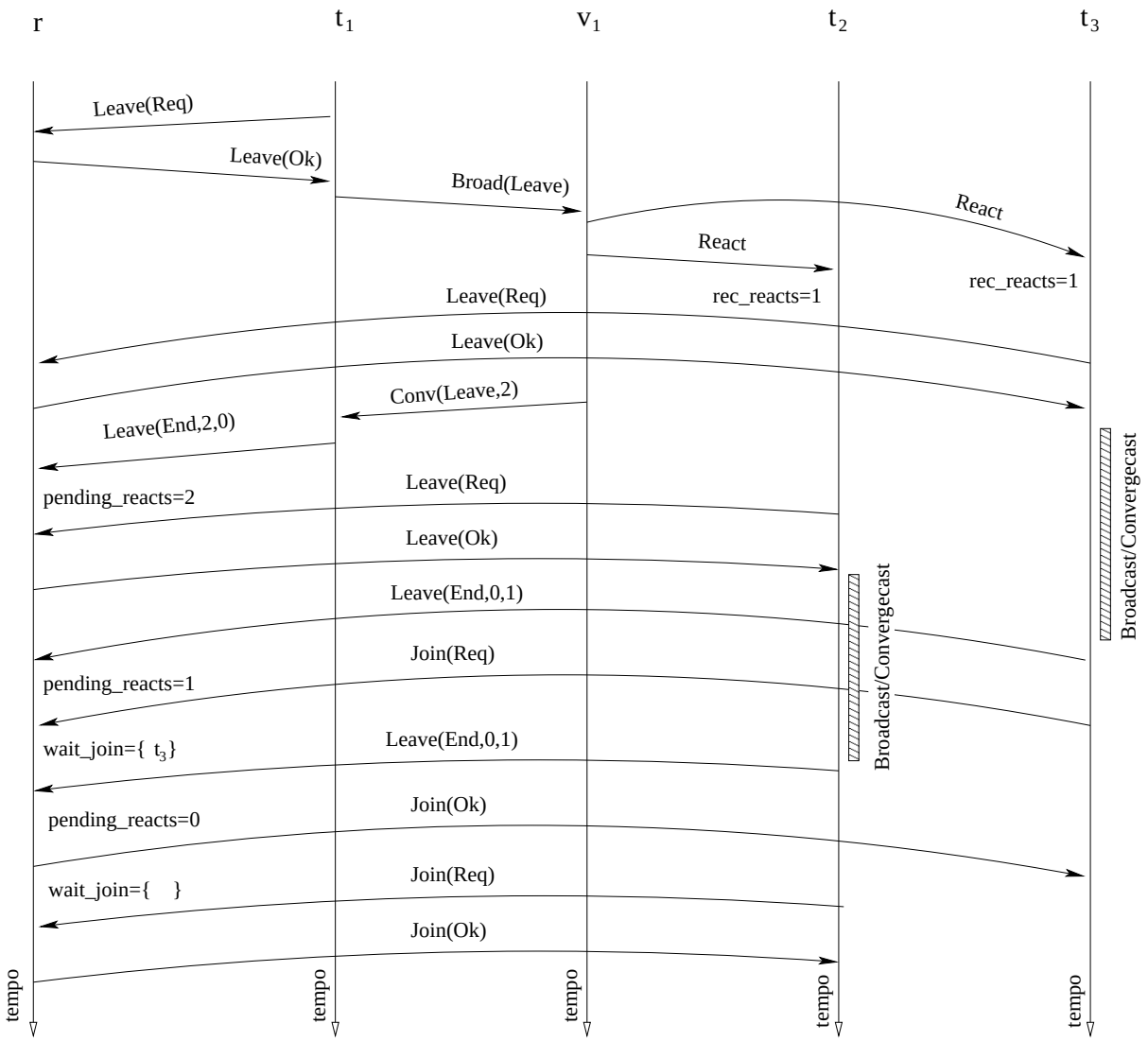

Figura A.2: Possível execução para o problema *on-line* - exclusão com duas reativações.

O terminal t_3 reage primeiro, enviando uma requisição para exclusão a r que autoriza imediatamente (rotinas E.1 e E.2). O terminal t_1 finaliza o *convergecast* de exclusão e informa na mensagem *Leave(End, 2, 0)* que foram enviadas duas mensagens *React* sem o recebimento de nenhuma (rotina E.6). O nó r ajusta o valor de *pending_reacts* para 2 (rotina E.8).

O terminal t_2 inicia então o processamento da mensagem *React* a ele direcionada por v_1 (rotina E.7), solicita e consegue de r autorização para exclusão (rotinas E.1 e E.2). Agora, t_3, que já terminou o processo de exclusão, informa com a mensagem *Leave(End, 0, 1)* que não enviou novas mensagens *React* e recebeu apenas uma (rotina E.6). O terminal raiz subtrai 1 de *pending_reacts* (rotina E.8). Nesse ponto *pending_reacts* contém 1. O terminal t_3 está pronto para retornar ao sistema, portanto, envia a mensagem *Join(Req)* a r (rotina E.6), que consulta a variável *pending_reacts* e verifica que ainda existem

exclusões em processamento, logo, ao invés de autorizar a reinclusão de t_3, inclui esse terminal no conjunto de terminais que aguardam autorização para inclusão (conjunto $wait\_join$) (rotina I.2).

Neste ponto o terminal t_2 encerra o *broadcast/convergecast* de exclusão e informa ter recebido apenas uma mensagem *React* por meio do envio da mensagem *Leave(End, 0, 1)* a r (rotina E.6). O terminal raiz subtrai 1 de $pending\_reacts$ que passa a conter 0, indicando que não existe nenhum terminal em processo de exclusão e inclusões pendentes podem ser processadas, portanto, r envia a mensagem *Join(Ok)* aguardada por t_3 (rotinas E.8 e R.1). Em seguida t_2 solicita e recebe autorização para reinclusão (rotinas I.1 e I.2).

Agora, t_2 e t_3 iniciarão em paralelo a recriação de seus grafos/árvores de saturação (rotina I.3), portanto, a exclusão de t_1 causa uma reorganização dessas estruturas em uma parte específica do grafo. Essa reorganização minimiza o efeito de inclusões feitas a partir de grafos de saturação grandes, que levam a resultados de pior qualidade segundo [Werneck 2001].

APÊNDICE B – Reorganização Baseada no Limite Inferior - Resultados Práticos

Nas figuras B.1 a B.20 podemos observar curvas com o ótimo (calculado com o algoritmo *branch-and-bound* descrito em [Poggi de Aragão et al. 2001]), limite inferior e soluções heurísticas com reorganizações baseadas no limite inferior e nos contadores do ARIES durante variações no conjunto de terminais. Círculos próximos ao eixo das ordenadas indicam reorganizações disparadas pelo método dos contadores (ARIES) e quadrados indicam reorganizações disparadas pelo método do limite inferior (DA). As reorganizações foram feitas em todos os casos utilizando o Prim-SPH [Novak et al. 2001a].

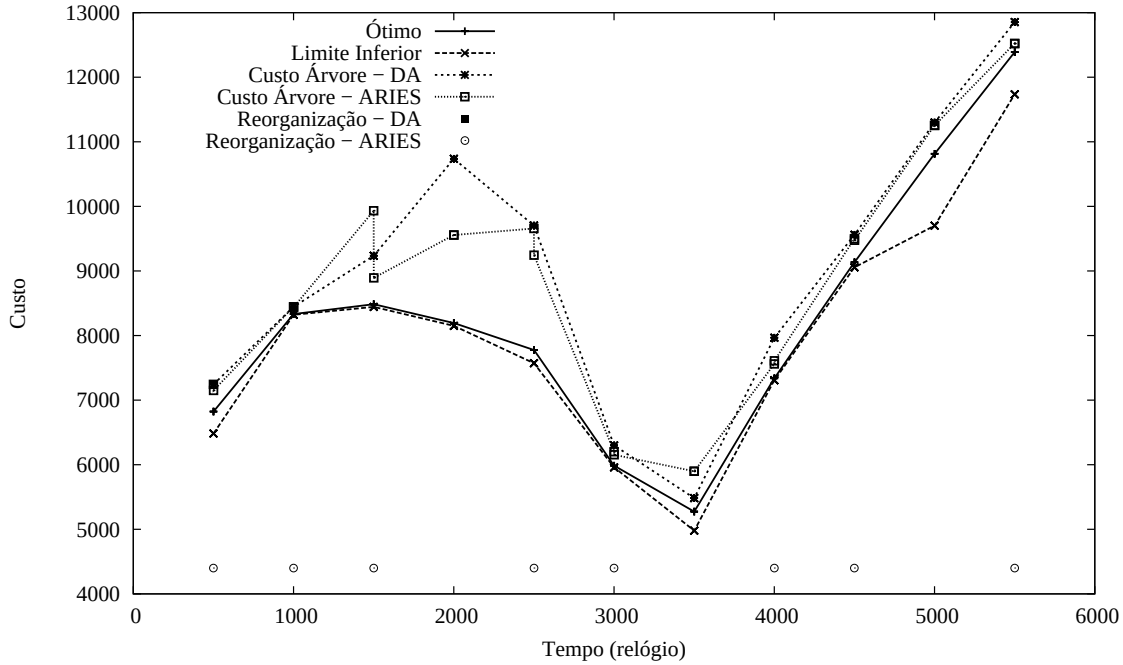

Figura B.1: Instância GLP-1 com operações em grupos de seis.

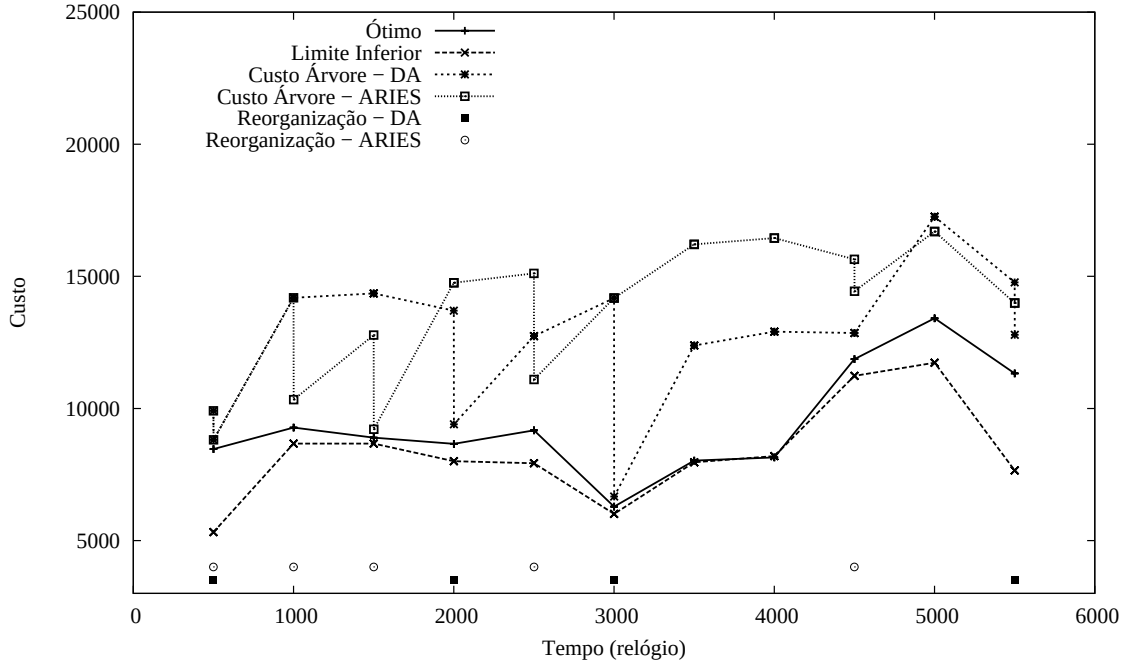

Figura B.2: Instância GLP-2 com operações em grupos de seis.

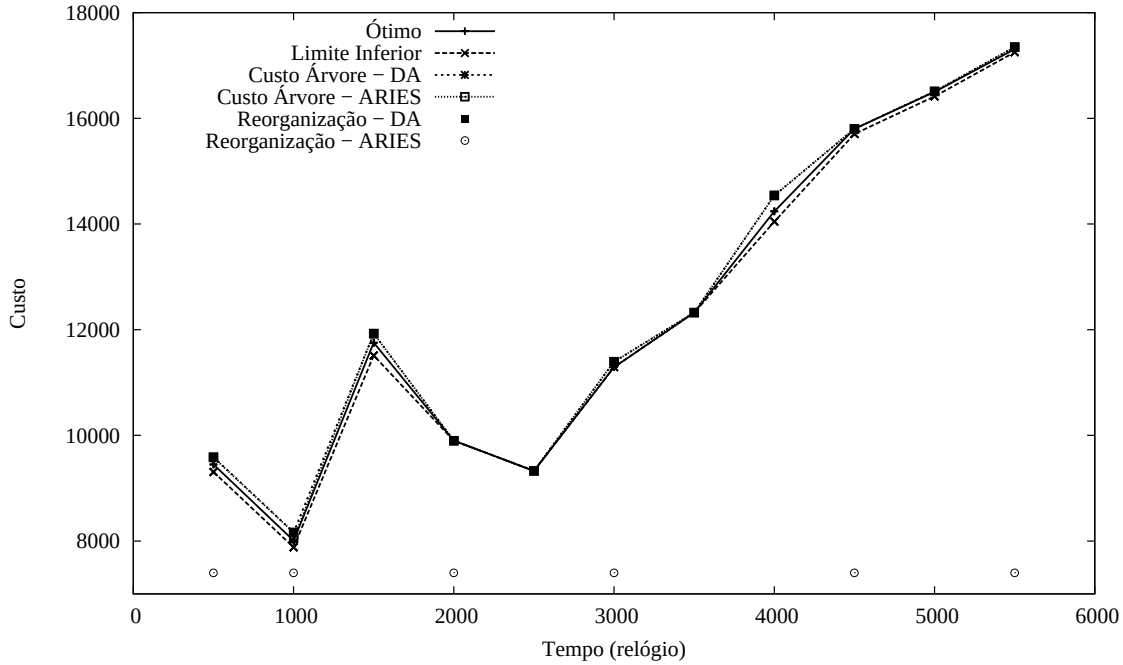

Figura B.3: Instância GLP-3 com operações em grupos de seis.

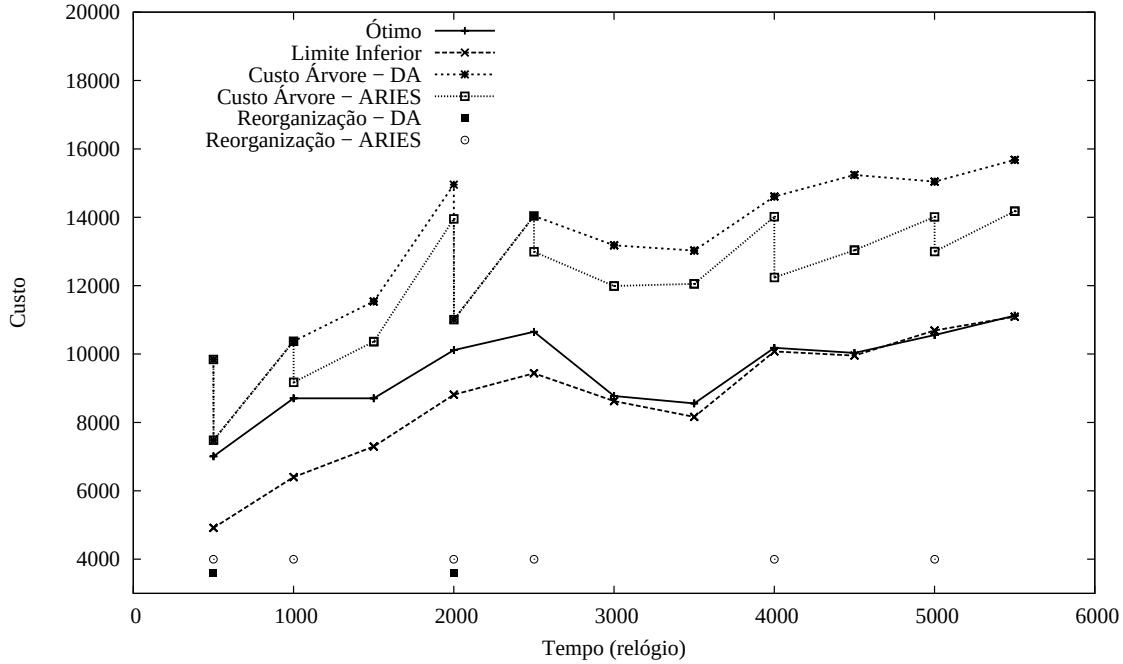

Figura B.4: Instância GLP-4 com operações em grupos de seis.

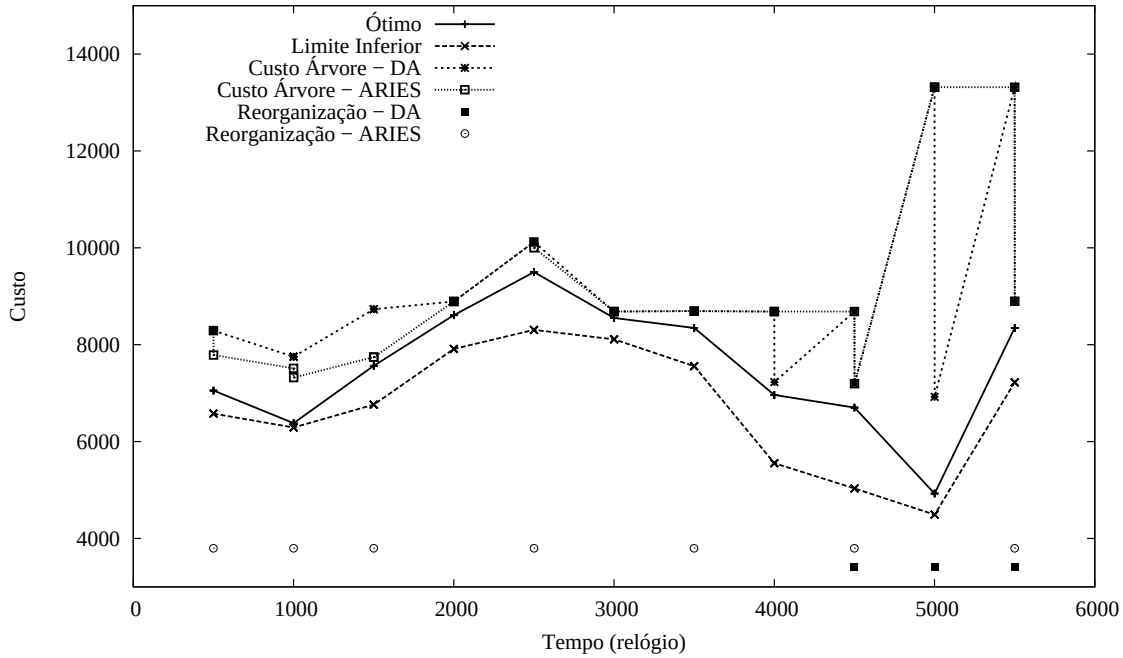

Figura B.5: Instância GLP-5 com operações em grupos de seis.

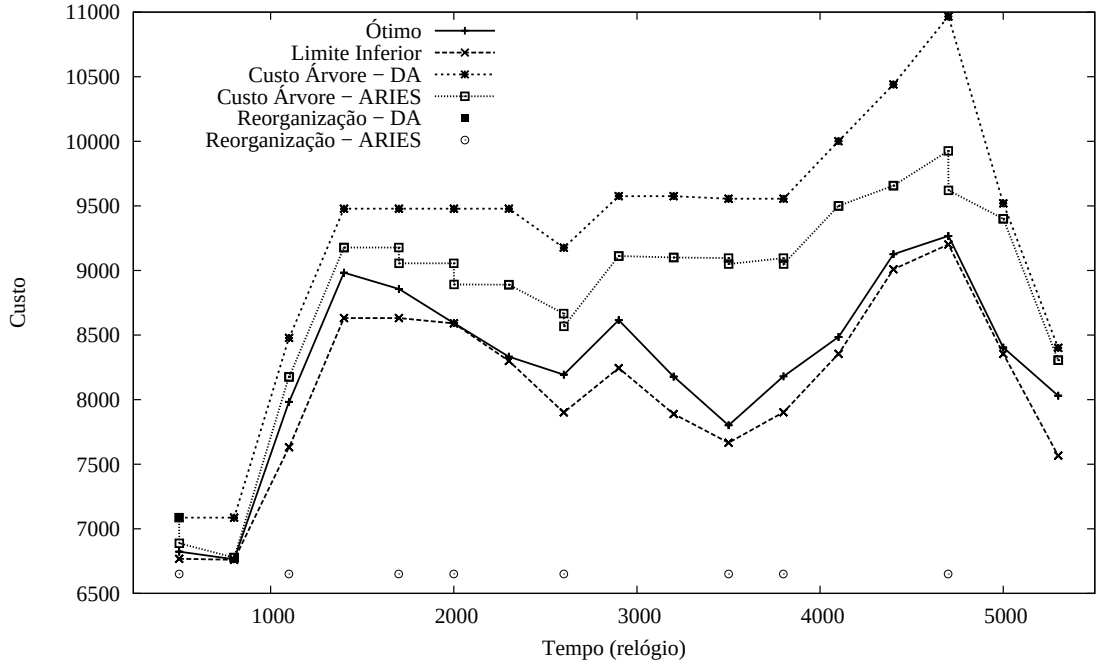

Figura B.6: Instância GLP-1 com operações processadas uma a uma.

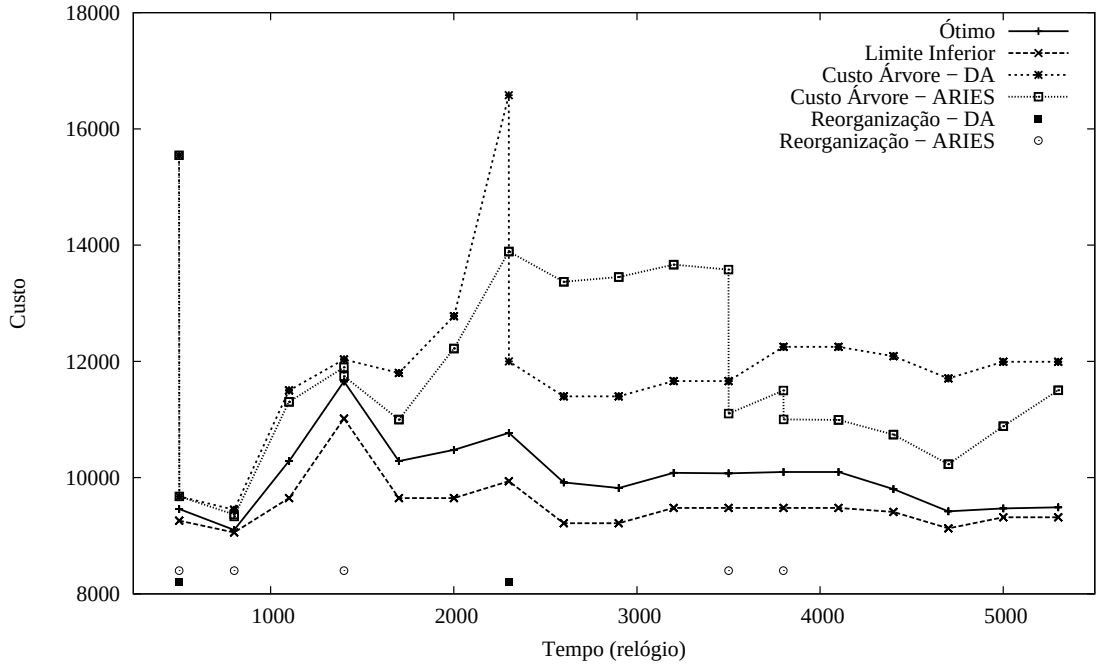

Figura B.7: Instância GLP-2 com operações processadas uma a uma.

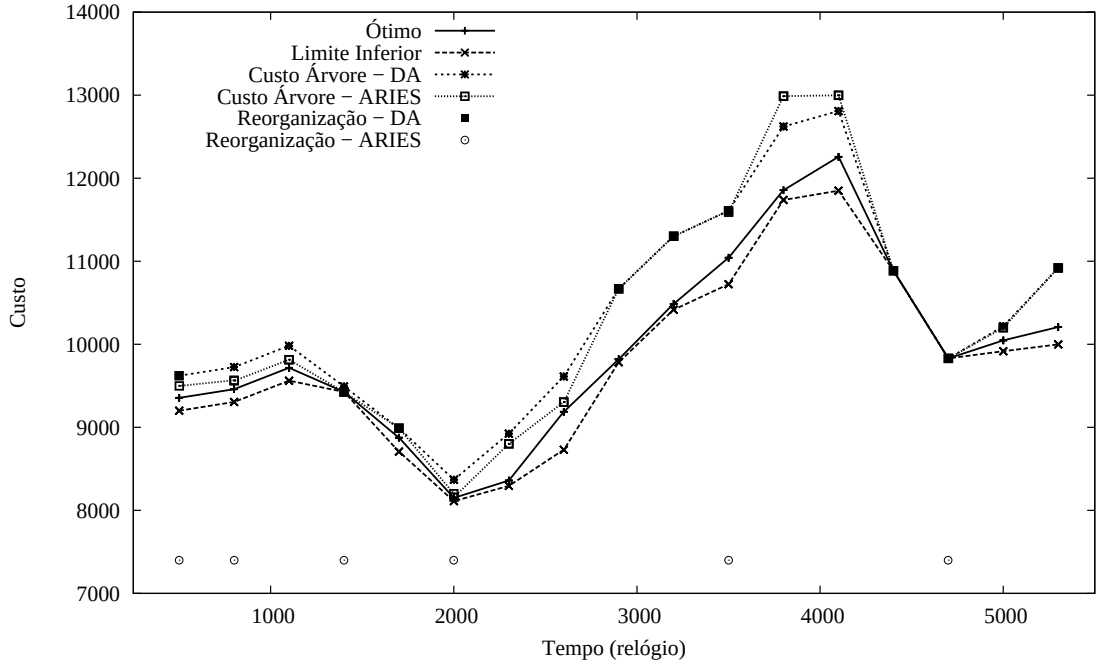

Figura B.8: Instância GLP-3 com operações processadas uma a uma.

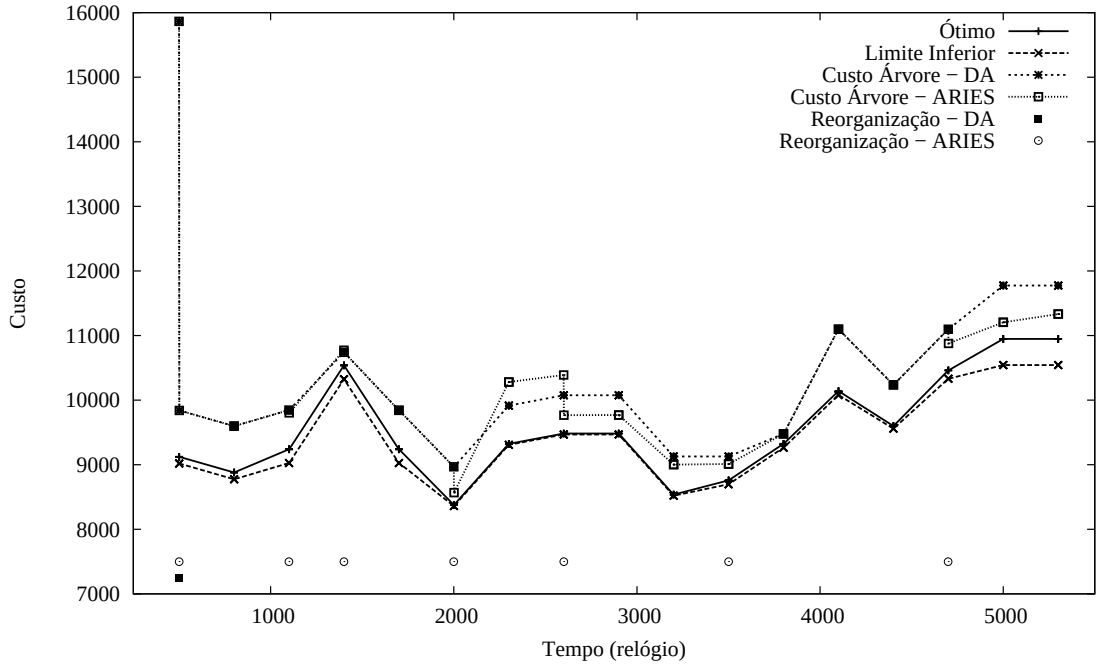

Figura B.9: Instância GLP-4 com operações processadas uma a uma.

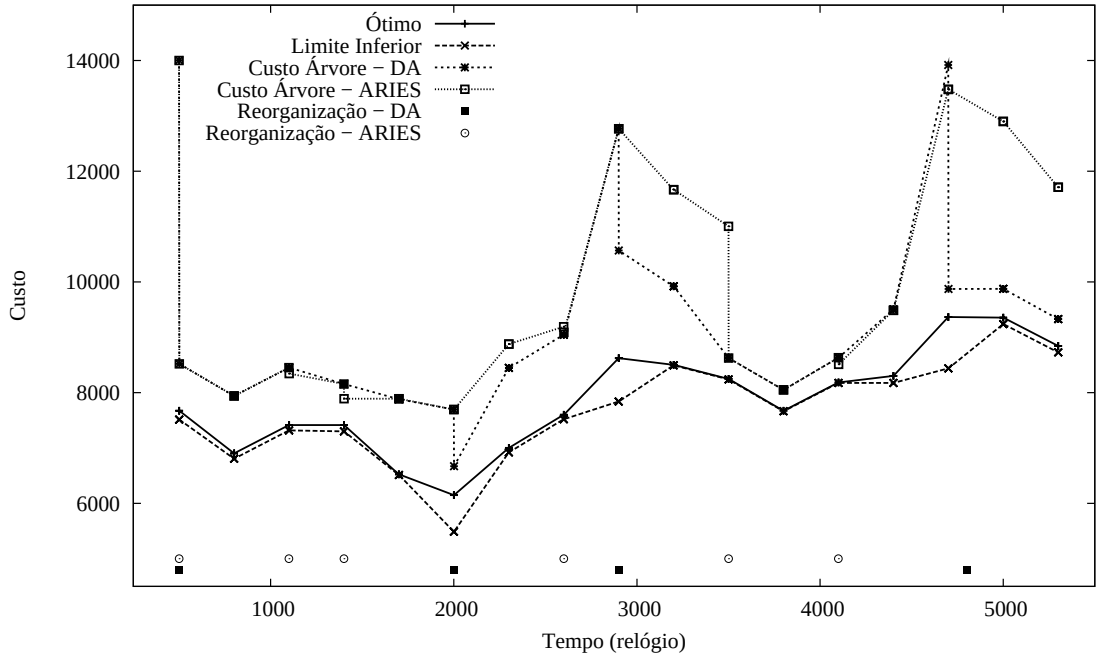

Figura B.10: Instância GLP-5 com operações processadas uma a uma.

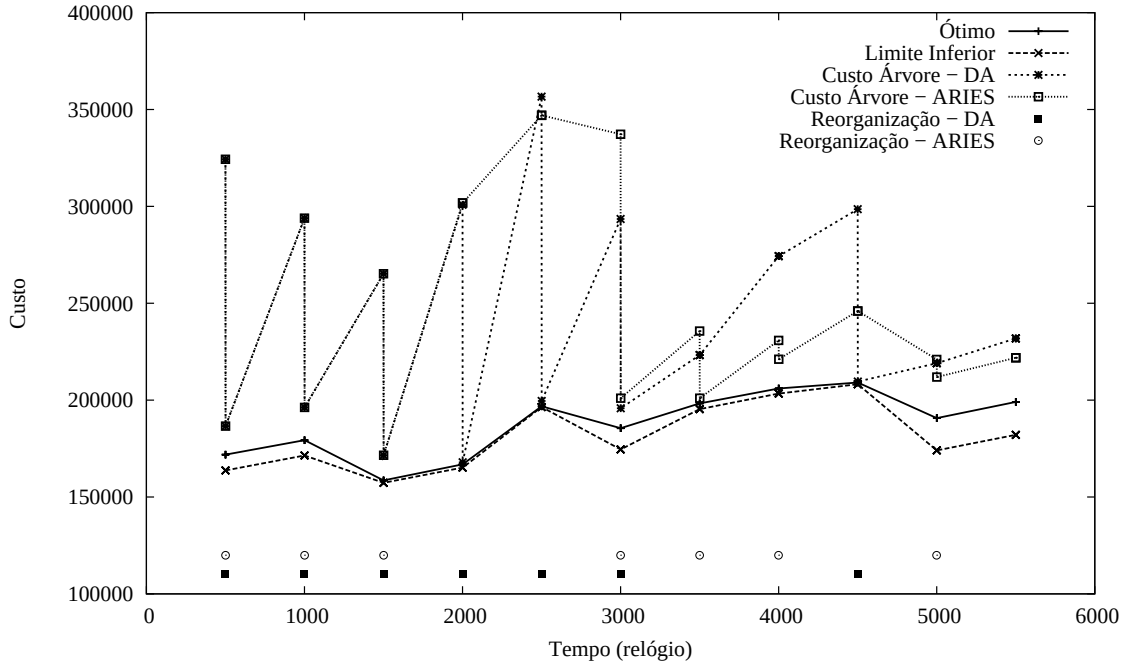

Figura B.11: Instância SW-1 com operações em grupos de seis.

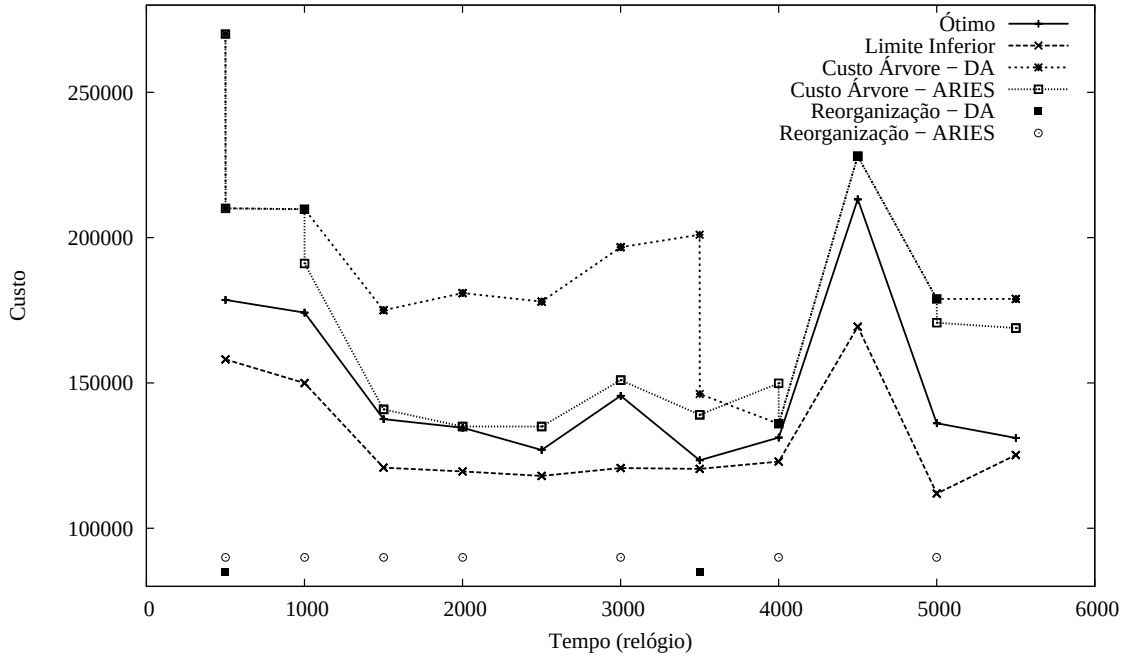

Figura B.12: Instância SW-2 com operações em grupos de seis.

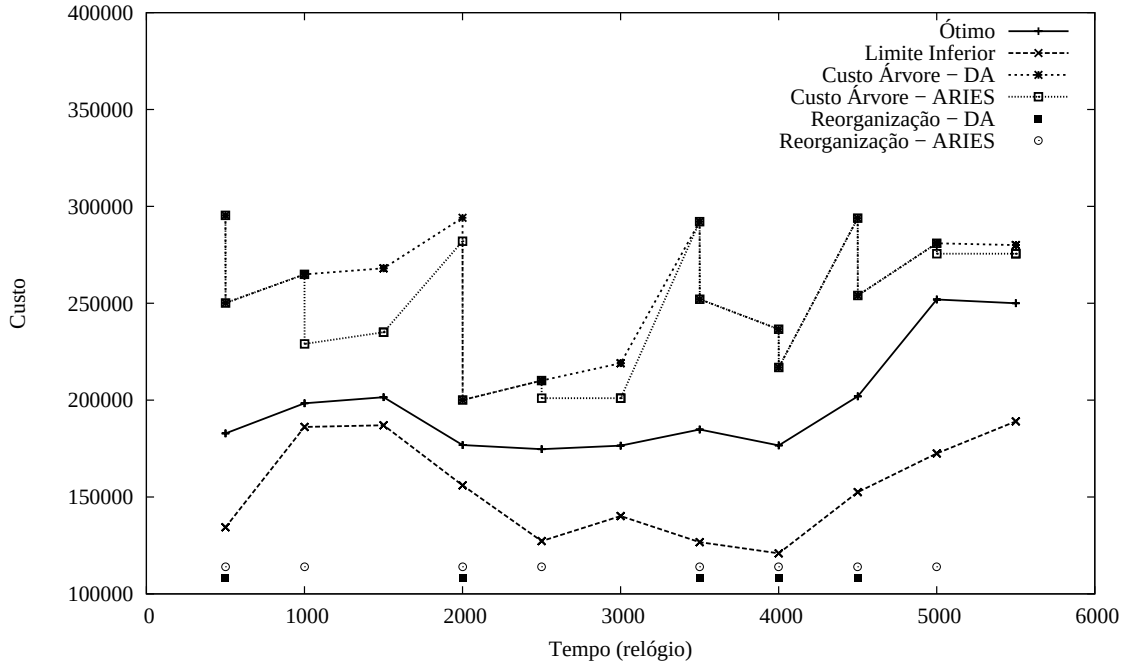

Figura B.13: Instância SW-3 com operações em grupos de seis.

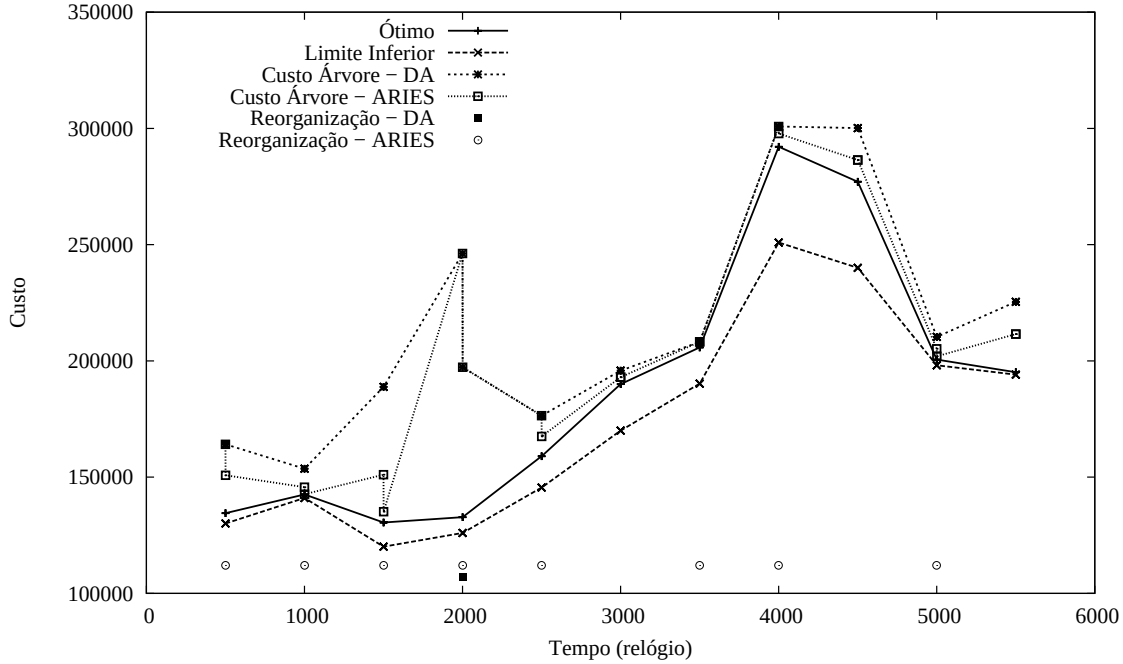

Figura B.14: Instância SW-4 com operações em grupos de seis.

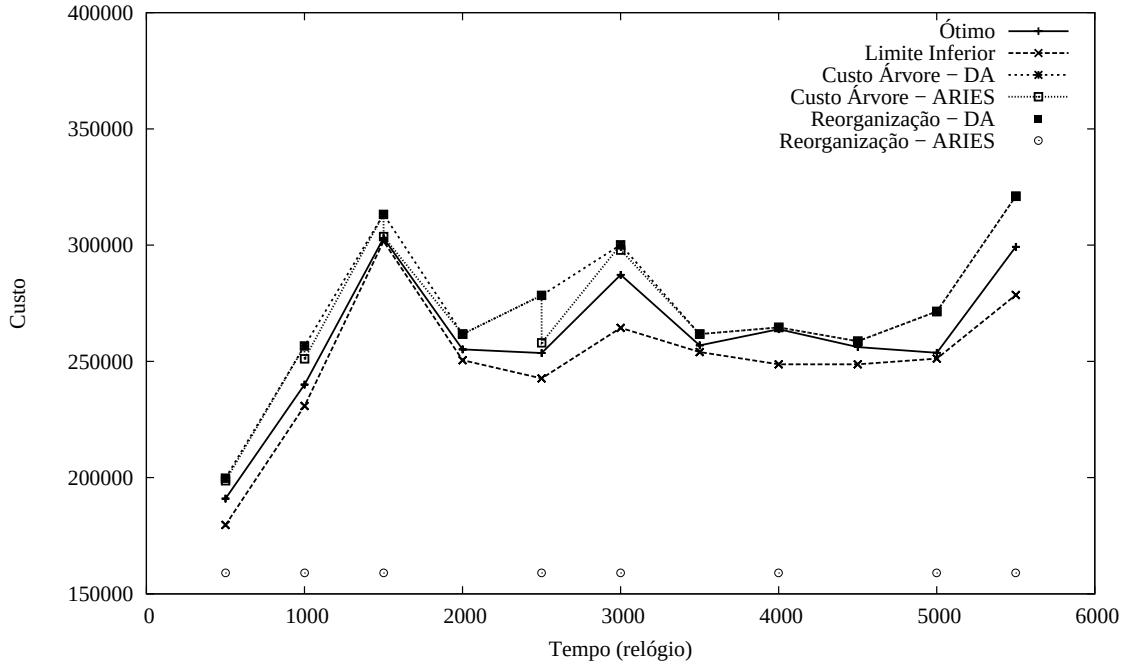

Figura B.15: Instância SW-5 com operações em grupos de seis.

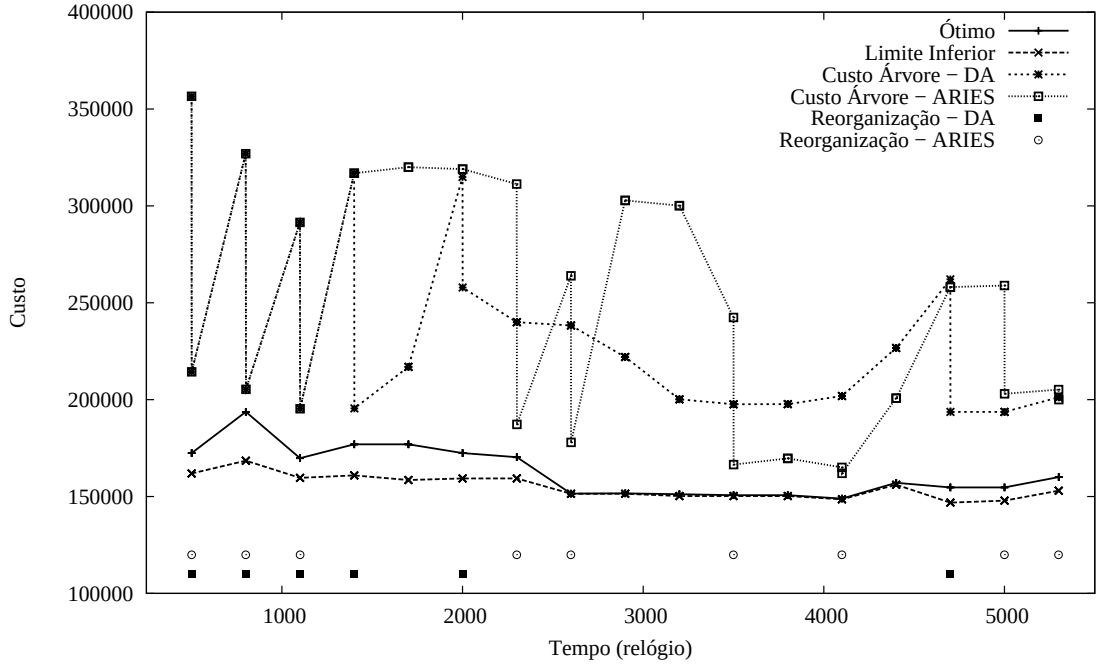

Figura B.16: Instância SW-1 com operações processadas uma a uma.

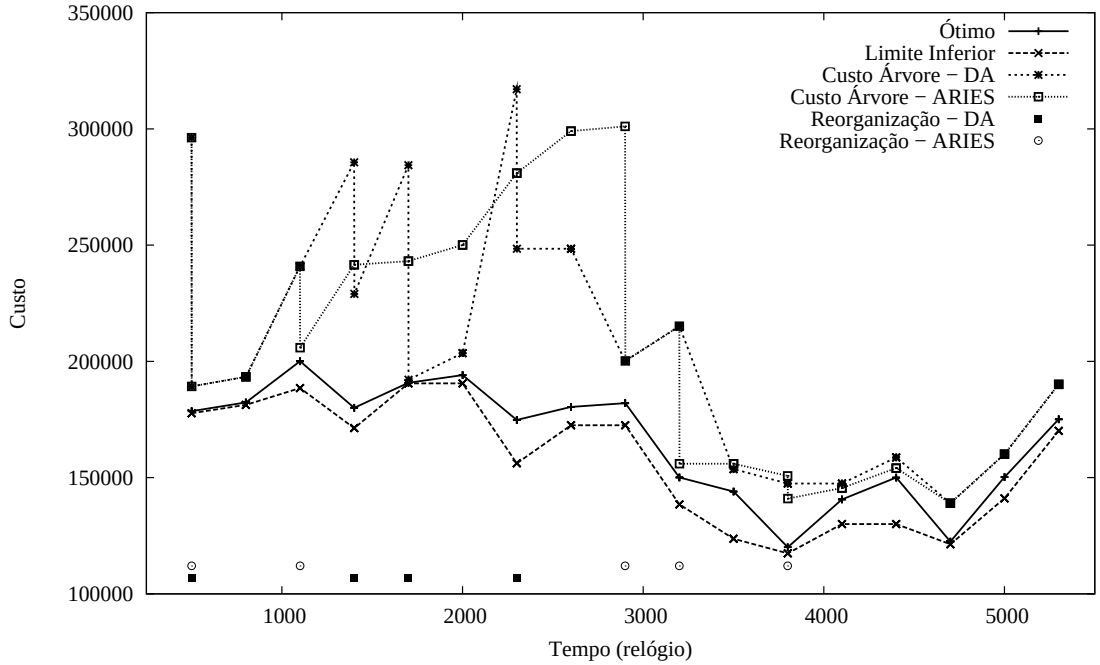

Figura B.17: Instância SW-2 com operações processadas uma a uma.

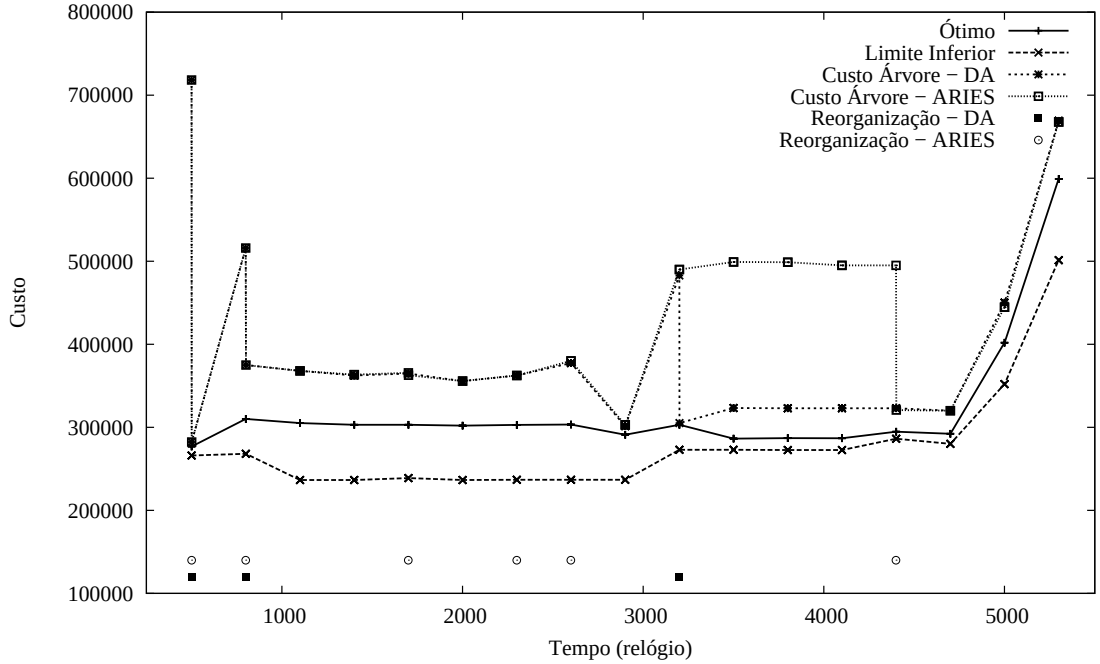

Figura B.18: Instância SW-3 com operações processadas uma a uma.

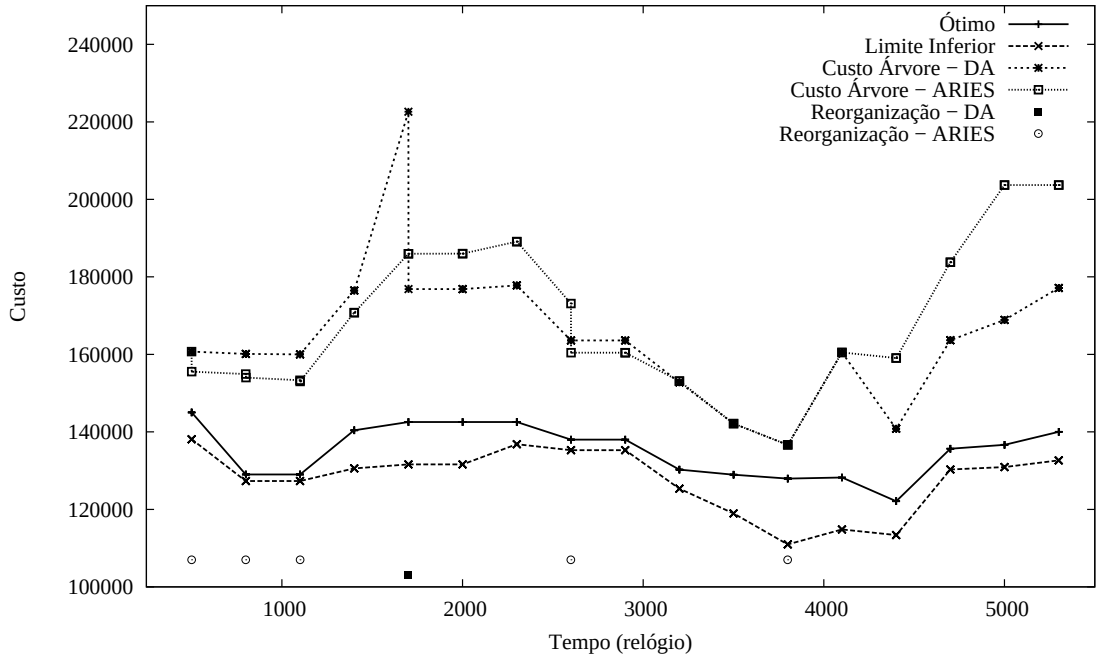

Figura B.19: Instância SW-4 com operações processadas uma a uma.

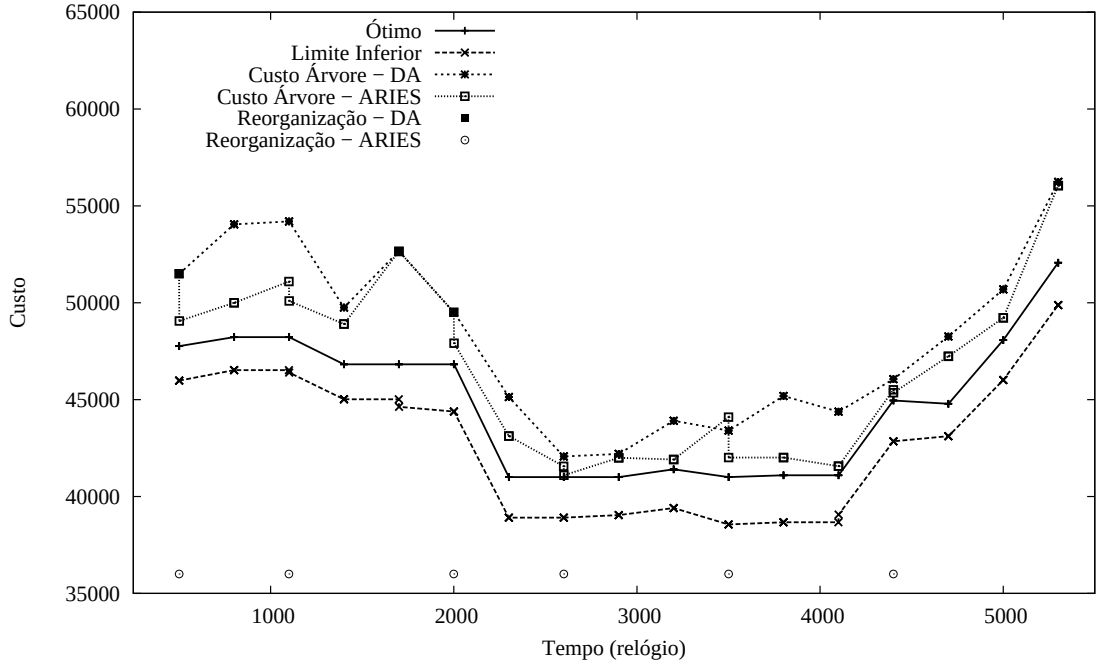

Figura B.20: Instância SW-5 com operações processadas uma a uma.

Referências

[Adelstein et al. 2003] Adelstein, F.; Richard III, G. G. e Schwiebert, L. **Distributed multicast tree generation with dynamic group membership**. *Computer communications*, v. 26, n. 10, p. 1105–1128, 2003.

[Alrabiah 1999] Alrabiah, T. **Multicast routing for real-time communication in wide-area high speed networks**. Dissertação (Mestrado) — University of Pittsburgh, 1999.

[Aneja 1980] Aneja, Y. **An integer linear programming approach to the Steiner problem in graphs**. *Networks*, v. 10, n. 2, p. 167–178, 1980.

[Barbosa 1996] Barbosa, V. C. **An introduction to distributed algorithms**. Cambridge,The MIT Press, 1996. ISBN 0262024128.

[Bauer e Varma 1996] Bauer, F. e Varma, A. **Distributed algorithms for multicast path setup in data networks**. *IEEE/ACM Transactions on networking*, v. 4, n. 2, p. 181–191, 1996.

[Bauer e Varma 1997] Bauer, F. e Varma, A. **ARIES: a rearrangeable inexpensive edge-based on-line Steiner algorithm**. *IEEE Journal on selected areas in communications*, v. 15, n. 3, p. 382–397, 1997.

[Beasley 1984] Beasley, J. **An algorithm for the Steiner problem in graphs**. *Networks*, v. 14, n. 1, p. 147–159, 1984.

[Bu e Towsley 2002] Bu, T. e Towsley, D. F. **On distinguishing between internet power law topology generators**. In: *Proceedings of IEEE annual joint conference of the IEEE computer and communications societies INFOCOM'02*, v. 2, p. 638–647, 2002.

[Chen 1983] Chen, N. P. **New algorithms for Steiner tree on graphs**. In: *Proceedings of the IEEE international symposium on circuits and systems ISCAS'83*, p. 1217–1219, 1983.

[Chopra et al. 1992] Chopra, S.; Gorres, E. R. e Rao, M. R. **Solving the Steiner tree problem on a graph using branch and cut**. *ORSA Journal on computing*, v. 4, n. 3, p. 320–335, 1992.

[Choukhmane 1978] Choukhmane, E. A. **Une heuristique pour le probleme de l´arbre de Steiner**. *RAIRO Recherche operationnelle*, v. 12, p. 207–212, 1978.

[Dijkstra 1959] Dijkstra, E. W. **A note on two problems in connexion with graphs**. *Numerische mathematik*, v. 1, p. 269–271, 1959.

[Doar e Leslie 1993] Doar, M. e Leslie, I. **How bad is naive multicast routing?** In: *Proceedings of IEEE conference on computer communications INFOCOM'93*, v. 1, p. 82–89, 1993.

[Drummond et al. 2008] Drummond, L.; Santos, M. e Uchoa, E. **A distributed dual ascent algorithm for Steiner problems in multicast routing**. *Networks*, 2008. A ser publicado, disponível em http://www.ic.uff.br/∼lucia/dual-ascent.pdf.

[Duin 1993] Duin, C. **Steiner problems in graphs**. Tese (Doutorado) — University of Amsterdam, 1993.

[M. Faloutsos 1999] M. Faloutsos, P. Faloutsos, C. F. **On power-Law relationships of the internet topology**. In: *Proceedings of the conference on applications, technologies, architectures, and protocols for computer communication SIGCOMM'99*, p. 251–262, 1999.

[di Fatta e Re 1998]di Fatta, G. e Re, G. L. **Efficient tree construction for the multicast problem**. In: *Proceedings of the international telecommunications symposium ITS'98*, v. 2, p. 632–637, 1998.

[Feng e Yum 1999] Feng, G. e Yum, T. P. **Efficient multicast routing with delay constraints**. *International journal of communications systems*, v. 12, n. 3, p. 181–195, 1999.

[Gallager et al. 1983] Gallager, R. G.; Humblet, P. A. e Spira, P. M. **A distributed algorithm for minimum-weight spanning trees**. *ACM Transactions on programming languages and systems*, v. 5, n. 1, p. 66–77, 1983.

[Garey e Johnson 1979] Garey, M. R. e Johnson, D. S. **Computers and intractability: a guide to the theory of NP-completeness**. New York,W. H. Freeman & Company, 1979. ISBN 0716710455.

[Gatani et al. 2005] Gatani, L.; Re, G. L. e Gaglio, S. **An efficient distributed algorithm for generating multicast distribution trees**. In: *Proceedings of the international conference on parallel processing workshops ICPPW'05*, p. 477–484, 2005.

[Gatani et al. 2006] Gatani, L.; Re, G. L. e Gaglio, S. **An efficient distributed algorithm for generating and updating multicast trees**. *Parallel computing*, v. 32, n. 11–12, p. 777–793, 2006.

[Goemans e Williamson 1996] Goemans, M. X. e Williamson, D. P. **The primal-dual method for approximation algorithms and its application to network design problems**. In: *Approximation algorithms for NP-hard problems*. Boston,PWS Publishing Co., 1996. p. 144–191.

[Gouveia et al. 2007] Gouveia, L.; Simonetti, L. e Uchoa, E. **Modelling the hop-constrained minimum spanning tree problem over layered graph**. In: *Proceedings of the international network optimization conference INOC'07*, p. 1–6, 2007.

[Hakimi 1971] Hakimi, S. L. **Steiner's problem in graphs and its implications**. *Networks*, v. 1, n. 2, p. 113–133, 1971.

[Huang e Lee 2005] Huang, T.-L. e Lee, D. T. **Comments and an improvement on "A distributed algorithm of delay-bounded multicast routing for multimedia applications in wide area networks"**. *IEEE/ACM Transactions on networking*, v. 13, n. 6, p. 1410–1411, 2005.

[Hwang et al. 1992] Hwang, F.; Richards, D. e Winter, P. **The steiner tree problem (Annals of discrete mathematics)**. Amsterdam,North-Holland Publishing Company, 1992. ISBN 044489098X.

[Imase e Waxman 1991] Imase, M. e Waxman, B. **Dynamic Steiner tree problems**. *SIAM Journal of discrete mathematics*, v. 4, n. 3, p. 369–384, 1991.

[Jia 1998] Jia, X. **A distributed algorithm of delay-bounded multicast routing for multimedia applications in wide area networks**. *IEEE/ACM Transactions on networking*, v. 6, n. 6, p. 828–837, 1998.

[Jin e Bestavros 2006] Jin, S. e Bestavros, A. **Small-world characteristics of internet topologies and implications on multicast scaling**. *Computer networks: the international journal of computer and telecommunications networking*, v. 50, n. 5, p. 648–666, 2006.

[Karp 1972] Karp, R. M. **Reducibility among combinatorial problems**. In: *Complexity of computer computations*. New York,Plenum Press, 1972. p. 85–103.

[Koch et al. 2000] Koch, T.; Martin, A. e Voss, S. **SteinLib: an updated library on Steiner problems in graphs**. Konrad-zuse-zentrum für informationstechnik Berlin ZIB-report, 2000. 00-37 p. Disponível em http://elib.zib.de/steinlib.

[Kompella et al. 1993a] Kompella, V. P.; Pasquale, J. C. e Polyzos, G. C. **Multicast routing for multimedia communication**. *IEEE/ACM Transactions on networking*, v. 1, n. 3, p. 286–292, 1993.

[Kompella et al. 1993b] Kompella, V. P.; Pasquale, J. C. e Polyzos, G. C. **Two distributed algorithms for multicasting multimedia information**. In: *Proceedings of the international conference on computer communications and networking ICCCN'93*, p. 343–349, 1993.

[Kou et al. 1981] Kou, L.; Markowsky, G. e Berman, L. **A fast algorithm for Steiner trees**. *ACTA Informatica*, v. 15, n. 2, p. 141–145, 1981.

[Kruskal 1956] Kruskal, J. B. **On the shortest spanning subtree of a graph and the traveling salesman problem**. In: *Proceedings of the american mathematical society*, v. 7, n. 1, p. 48–50, 1956.

[Lin e Lai 1998] Lin, H. e Lai, S. **VTDM-A dynamic multicast routing algorithm**. In: *Proceedings of annual joint conference of the IEEE computer and communications societies INFOCOM'98*, v. 3, p. 1426–1432, 1998.

[Medina et al. 2001] Medina, A.; Lakhina, A.; Matta, I. e Byers, J. **BRITE: An approach to universal topology generation**. In: *Proceedings of the international symposium on modeling, analysis and simulation of computer and telecommunications systems MASCOTS'01*, p. 346–353, 2001.

[Melzak 1961] Melzak, Z. A. **On the problem of Steiner**. *Canadian mathematical bulletin*, v. 4, p. 143–148, 1961.

[Mokbel et al. 1999] Mokbel, M. F.; Elhaweet, W. A. e Elderini, M. N. **A delay-constrained shortest path algorithm for multicast routing in multimedia applications**. In: *Proceedings of IEEE middle east workshop on networking*, 1999.

[Narang et al. 1999] Narang, N.; Kumar, G. e Ravikumar, C. P. **Efficient algorithms for delay bounded multicast tree generation for multimedia applications**. In: *Proceedings of the international conference on high performance computing HiPC'99*, p. 169–173, 1999.

[Novak et al. 2001a] Novak, R.; Rugelj, J. e Kundus, G. **A note on distributed multicast routing in point-to-point networks**. *Computers and operations research*, v. 28, n. 12, p. 1149–1164, 2001.

[Novak et al. 2001b] Novak, R.; Rugelj, J. e Kundus, G. **Steiner tree based distributed multicast routing in networks**. In: *Steiner trees in industries, Combinatorial optimization*. Dordrecht,Kluwer Academic Publishers, 2001. v. 11, p. 327–351.

[Oliveira e Pardalos 2005] Oliveira, C. A. S. e Pardalos, P. M. **A survey of combinatorial optimization problems in multicast routing**. *Computers and operations research*, v. 32, n. 8, p. 1953–1981, 2005.

[Oliveira et al. 2005] Oliveira, C. A. S.; Pardalos, P. M. e Resende, M. G. **Optimization problems in multicast tree construction**. In: *Handbook of optimization in telecommunications*. Dordrecht,Kluwer Academic Publishers, 2005. p. 701–733.

[Plesnik 1981] Plesnik, J. **A bound for the Steiner tree problem in graphs**. *Mathematica slovaca*, n. 31, p. 155–163, 1981.

[Poggi de Aragão et al. 2001] Poggi de Aragão, M.; Uchoa, E. e Werneck, R. **Dual heuristics on the exact solution of large Steiner problems**. In: *Eletronic notes in discrete mathematics GRACO'01*, v. 7, p. 150–153, 2001.

[Poggi de Aragão e Werneck 2002] Poggi de Aragão, M. e Werneck, R. **On the implementation of MST-based heuristics for the Steiner problem in graphs**. In: *Revised papers from the international workshop on algorithm engineering and experiments ALENEX'02*, p. 1–15, 2002.

[Polzin e Vahdati 2001] Polzin, T. e Vahdati, S. **Improved algorithms for the Steiner problem in networks**. *Discrete applied mathematics*, v. 112, n. 1–3, p. 263–300, 2001.

[Prim 1957] Prim, R. C. **Shortest connection networks and some generalizations**. *Bell system technical journal*, v. 36, n. 6, p. 1389–1401, 1957.

[Raghavan et al. 1999] Raghavan, S.; Manimaran, G. e Murthy, C. S. R. **A rearrangeable algorithm for the construction delay-constrained dynamic multicast trees**. *IEEE/ACM Transactions on networking*, v. 7, n. 4, p. 514–529, 1999.

[Rayward-Smith 1983] Rayward-Smith, V. **The computation of nearly minimal steiner trees in graphs**. *International journal of mathematical education in science and technology*, v. 14, p. 15–23, 1983.

[Rugelj e Klavzar 1997] Rugelj, J. e Klavzar, S. **Distributed multicast routing in point-to-point networks**. *Computers and operations research*, v. 24, n. 6, p. 521–527, 1997.

[Santos et al.] Santos, M.; Drummond, L. M. A. e Uchoa, E. **Distributed dual ascent algorithm for steiner problems in networks**. In: *Anais do simpósio brasileiro de redes de computadores e sistemas distribuídos SBRC'07*.

[Santos et al. 2007] Santos, M.; Drummond, L. M. A. e Uchoa, E. **A distributed primal-dual heuristic for Steiner problems in networks**. In: *Proceedings of the 6th international workshop on experimental algorithms WEA'07, Lecture notes in computer science*, v. 4525, p. 175–188, 2007.

[Santos et al. 2008] Santos, M.; Drummond, L. M. A.; Uchoa, E. e Simonetti, L. **Dual-ascent distribuído para multicast com restrição de saltos**. In: *Anais do simpósio brasileiro de redes de computadores e sistemas distribuídos SBRC'08*, p. 231–244, 2008.

[Segall 1983] Segall, A. **Distributed network protocols**. *IEEE Transactions on information theory*, v. 29, n. 1, p. 23–35, 1983.

[Shaikh e Shin 1997] Shaikh, A. e Shin, K. G. **Destination-driven routing for low-cost multicast**. *IEEE Journal on selected areas in communications*, v. 15, n. 3, p. 373–381, 1997.

[Singh e Vellanki 1998] Singh, G. e Vellanki, K. **A distributed protocol for constructing multicast trees**. In: *Proceedings of the international conference on principles of distributed systems OPODIS'98*, p. 61–76, 1998.

[Takahashi e Matsuyama 1980] Takahashi, H. e Matsuyama, A. **An approximate solution for the Steiner problem in graphs**. *Mathematica japonica*, v. 24, p. 573–577, 1980.

[Voss 1992] Voss, S. **Steiner's problem in graphs: heuristic methods**. *Discrete applied mathematics*, v. 40, n. 1, p. 45–72, 1992.

[Watts e Strogatz 1998] Watts, D. e Strogatz, S. **Collective dynamics of small-world networks**. *Nature*, v. 363, p. 202–204, 1998.

[Waxman 1988] Waxman, B. **Routing of multipoint connections**. *IEEE Journal on selected areas in communications*, v. 6, n. 9, p. 1617–1622, 1988.

[Werneck 2001] Werneck, R. F. F. **Problema de steiner em grafos: algoritmos primais, duais e exatos**. Dissertação (Mestrado) — Pontifícia Universidade Católica do Rio de Janeiro, 2001.

[Williamson 2002] Williamson, D. **The primal-dual method for approximation algorithms**. *Mathematical programming*, v. 91, n. 3, p. 447–478, 2002.

[Winter 1987] Winter, P. **Steiner problem in networks: a survey**. *Networks*, v. 17, n. 2, p. 129–167, 1987.

[Wong 1984] Wong, R. **A Dual ascent approach for Steiner tree problems on a directed graph**. *Mathematical programming*, v. 28, n. 3, p. 271–287, 1984.