Um algoritmo randomizado para o problema de determinação do corte $s-t$ mínimo em grafos direcionados

Anselmo A. Montenegro
Celso da Cruz Carneiro Ribeiro
Marcus Vinicius S. Poggi de Aragão

Pontifícia Universidade Católica do Rio de Janeiro - PUC-Rio
e-mail: {anselmo,celso,poggi}@inf.puc-rio.br

PUC-RioInf.MCC36/01 Julho, 2001

Abstract: The minimum cut problem in a direct graph consists basically in finding a partitioning of the set of vertices of the graph in two subsets, in such a way that the number (or the sum of the weights) of the edges that connect the partitions is minimized. A important variation of this problem is the s-t mimimum cut problem, in which we impose the restriction that each candidate cut separate two special vertices s and t of the input set. As far as we are know, no eficient and simple randomized algorithm has been proposed to solve this problem. In this work we investigate the aplicability of randomization techniques to the $s-t$ minimum, proposing a algorithm and analising its complexity.

Keywords: graph cuts, $s-t$ minimum cut, randomized algorithms, probabilistic algorithms, maximum flow problem.

Resumo: O problema de determinação do corte de capacidade mínima em grafos não direcionados consiste, basicamente, na determinação de uma partição dos vértices do grafo em dois subconjuntos, de forma que o número(ou a soma dos pesos) de(as) arestas que conectam as duas partições seja minimizado(a). Uma variação importante desse problema é o chamado problema de determinação do *corte $s-t$ mínimo*, no qual impomos a restrição de que os cortes candidatos separem dois vértices especiais s e t, pertencentes ao conjunto de vértices do grafo de entrada, em duas partições distintas. Diferentemente do problema anterior, não foi verificado até o presente momento a existência de algoritmos radomizados simples e eficientes para a sua solução. Neste trabalho investigamos a aplicabilidade das técnicas de randomização para o problema de corte $s-t$ mínimo em grafos direcionados com pesos, propondo um algoritmo e avaliando sua complexidade.

Palavras-chave: cortes em grafos, corte $s-t$ mínimo, algoritmos randomizados, algoritmos probabilísticos, fluxo máximo.

Conteúdo

1 Introdução **4**

2 O problema de corte mínimo em um grafo **6**
2.1 Definição do problema 6
2.2 Algoritmos existentes 7

3 Algoritmo randomizado por contrações **9**
3.1 O algoritmo 9
3.2 Análise probabilística 12
3.3 Grafos com arestas com pesos 13
3.3.1 Seleção dos vértices 14
3.3.2 A operação de contração 17
3.3.3 Complexidade 19

4 Acelerando o algoritmo randomizado **21**
4.0.4 O Algoritmo de contração recursivo 21
4.0.5 Complexidade 23
4.0.6 Análise probabilística 24

5 Resolvendo o problema de corte s-t mínimo em grafos direcionados **26**
5.1 Características do problema 26
5.2 A estrutura de dados 27
5.3 A operação de seleção 30
5.4 A operação de validação de uma seleção 30
5.5 A operação de contração 31
5.6 Algoritmo 33
5.7 Análise de complexidade 34
5.8 Análise probabilística 36
5.9 Testes 37
5.10 Observações 39

Lista de Figuras

3.1 Sequência de contrações 11
3.2 Grafo 15

4.1 Árvore de recursão 23

5.1 Teste 1 37
5.2 Teste 2 38

Lista de Tabelas

3.1 W(u,v) 15
3.2 WS(u,v) 15
3.3 D(u) e DS(u) 16

5.1 Tempo de processamento 37
5.2 Custo 38

Capítulo 1

Introdução

A necessidade de se resolver problemas complexos em intervalos de tempo bastante reduzidos pode tornar inviável a utilização de métodos exatos, mesmo quando estes possuem complexidade polinomial. A utilização de *Algoritmos Aproximativos*, que procuram determinar soluções de boa qualidade de forma bastante rápida, tem se apresentado como uma alternativa promissora nesses casos críticos. Dentro dessa classe de algoritmos, podemos destacar os *Algoritmos Randomizados ou Probabilísticos* que, baseados no conceito estatístico de amostragem aleatória, procuram realizar uma investigação em uma subpopulação representativa do espaço de soluções de um problema, de forma a determinar dentre estas, aquela que possui o menor custo segundo uma determinada função objetivo.

Vários problemas têm sido tratados de forma bastante satisfatória através da utilização de algoritmos randomizados, como por exemplo, o problema de determinação de *árvores geradores de custo mínimo* e o problema de determinação do *corte de capacidade mínima em grafos não direcionados*, os quais, por sua vez, encerram inúmeras aplicações à problemas reais. Um estudo bastante completo sobre a aplicação de algoritmos randomizados para problemas de otmização em grafos pode ser encontrado em [1].

O problema de determinação do corte de capacidade mínima em grafos não direcionados consiste, basicamente, na determinação de uma partição dos vértices do grafo em dois subconjuntos, de forma que o número(ou a soma dos pesos) de(as) arestas que conectam as duas partições seja minimizado(a). Uma variação importante desse problema é o chamado problema de determinação do *corte* $s-t$ *mínimo*, no qual impomos a restrição de que os cortes candidatos separem dois vértices especiais s e t, pertencentes ao conjunto de vértices do grafo de entrada, em duas partições distintas. Diferentemente do problema anterior, não foi verificado até o presente momento a existência de algoritmos radomizados simples e eficientes para a sua solução.

Neste trabalho investigaremos a aplicabilidade das técnicas de randomização para o problema de corte $s - t$ mínimo em grafos direcionados com pesos, propondo um algoritmo e avaliando sua complexidade.

Capítulo 2

O problema de corte mínimo em um grafo

Os problemas de determinação de cortes de capacidade mínima em grafos são problemas clássicos em otimização combinatória, tendo aplicações em diversas áreas, como por exemplo, na construção de compiladores para linguagens paralelas, no projeto de redes confiáveis e em visão computacional [15, 16, 17, 18, 19]. Neste capítulo definiremos o problema de determinação do corte de custo mínimo e descreveremos, de forma resumida, as principais técnicas utilizadas para resolvê-lo e como estas evoluiram ao longo do tempo.

2.1 Definição do problema

Primeiramente, para que possamos definir o problema, iremos enunciar alguns conceitos básicos.

Definições:

- Um corte $(C, \bar{C})$ em um grafo $G(V, A)$ conexo é uma partição do conjunto de vértices V em dois subconjuntos não vazios C e $\bar{C}$.
- Uma aresta $(u, v) \in A$ cruza um corte $(C, \bar{C})$ de G somente se $u \in C$ e $v \in \bar{C}$.
- O *valor* de um corte $(C, \bar{C})$ é dado pelo número(soma dos pesos) de(as) arestas que o cruzam.

Com base nesses conceitos iremos agora definir o problema.

Definição: Seja $G(V, A)$ um grafo conexo onde V é seu conjunto de vértices e A seu conjunto de arestas. Então, *o problema de determinação*

do corte mínimo em $G(V, A)$, consiste em determinar dentre todos os cortes $(C, \bar{C})$ possíveis, aquele cujo valor é minimo.

Uma variação bastante importante desse problema é o problema de determinação do corte $s-t$ mínimo. Seja $G(V, A)$ um grafo direcionado conexo, no qual distingüimos dois vértices especiais s e t pertencentes a V. Então, um *corte* $s-t$ $(C, \bar{C})$ é determinado por uma partição de V em dois subcojuntos não vazios C e $\bar{C}$ tal que $s \in C$ e $t \in \bar{C}$.

Por motivos de economia e clareza, utilizaremos a notação PCM para denominar o problema de determinação do corte de capacidade mínima e a notação $PCMs$-t para o problema de determinação do corte de capacidade mínima $s-t$.

2.2 Algoritmos existentes

Existem diversas abordagens para solucionar o problema de corte mínimo em um grafo, sendo a mais antiga baseada na solução de problemas de *fluxo máximo*. A relação entre esses dois problemas é estabelecida através do teorema *corte mínimo - fluxo máximo*, o qual garante que o fluxo máximo entre dois vértices s e t de um grafo é determinado pelo menor corte $s-t$ existente no mesmo.

Os primeiros algoritmos propostos para solucionar o problema de fluxo máximo em um grafo foram baseados na idéia de *caminhos aumentantes* proposta por Ford e Fulkerson[12]. A complexidade dos algoritmos que solucionam esse problema vem sendo reduzida gradativamente à medida que novas pesquisas são realizadas.

A técnica que tem demonstrado melhores resultados é a chamada *push-relabel* de Goldberg e Tarjan [7]. Sua implementação mais antiga possui complexidade de $O(mn \log \frac{n^2}{m})$, sendo que posteriormente, King et al. desenvolveram uma versão que alcança a complexidade $O(nm(\log \frac{m}{n \log n} n))$ [9]. Recentemente, através do estudo das funções de distâncias utilizadas no processo de rotulação do algoritmo push-relabel, Goldberg e Rao desenvolveram um algoritmo de complexidade $O(min(n^{2/3}, m^{1/2}) m \log(\frac{n^2}{m}) logU)$, onde n é o número de vértices do grafo, m o número de arcos e U o maior valor de capacidade definida sobre os arcos [8]. Um introdução ao problema de fluxo máximo pode ser encontrada em [10, 11, 12, 13, 14]; já o leitor interessado em detalhes mais avançados pode obtê-los nos trabalhos [6, 8].

Devemos lembrar que no caso do problema de determinação do corte mínimo, no qual não há a restrição de que as partições contenham vertices específicos, é necessário a resolução de $C_{n,2}$ problemas de fluxo máximo, o que aumenta consideravelmente a complexidade do algoritmo final. Na verdade,

Gomory e Hu verificaram que esse esforço computacional pode ser reduzido ao demonstrar, através do conceito de *árvore de fluxo equivalente*(*flow equivalent tree*) [2], que o corte mínimo pode ser encontrado resolvendo-se apenas $n-1$ problemas de fluxo máximo .

Devido à essa dificuldade, foram propostos alguns algoritmos que procuram resolver diretamente o problema de corte mínimo, sem recorrer à solução de problemas de fluxo máximo. Podemos citar como exemplos o *Round-Robin Algorithm* de Gabow [3] e o algoritmo de Nagamochi e Ibaraki [4, 5].

O *Round-Robin Algorithm* é baseado em uma caracterização do problema de corte mínimo como um matróide e é análogo ao algoritmo por caminhos aumentantes para o problema de fluxo máximo. No problema de fluxo máximo, o valor do corte mínimo $s-t$ é igual ao número máximo de caminhos direcionados $s-t$ disjuntos que podem ser empacotados no grafo. De maneira análoga, o corte mínimo corresponde ao maior empacotamento possível de árvores direcionadas disjuntas.

O algoritmo de Gabow é voltado para grafos direcionados, mas pode ser utilizado para grafos não-direcionados. Para isto, basta, acrescentar as arestas inexistentes, no sentido contrário ao das arestas que estão presentes. Sua complexidade é $O(cm \log \frac{n^2}{m})$, onde c é o valor do corte mínimo, o que o faz o algoritmo mais rápido para grafos com cortes pequenos.

O algoritmo de Nagamochi e Ibaraki é um algoritmo baseado em contrações que se utiliza do conceito de *certificados esparsos de conectividade*. Dada uma propriedade monotônicamente crescente P de um grafo G, dizemos que um certificado esparso é um subgrafo G' de G para o qual uma propriedade P é satisfeita para G' e também para G. A vantagem do certificado esparso é a de que nele, uma propriedade pode ser muito mais facilmente verificada do que no grafo completo. Através de um algoritmo capaz de encontrar certificados de conectividade esparsos é possível determinar as arestas que não fazem parte do corte desejado, acelerando significativamente os algoritmos que requerem uma varredura de todas as arestas do grafo a cada iteração, como é o caso por exemplo, dos algoritmos baseados em caminhos aumentantes.

Nagamochi e Ibaraki desenvolveram um algoritmo para determinação de certificados de conectividade em tempo linear, através do qual, foi possível construir um algoritmo para determinação do corte mínimo através de uma sequência de operações de contração. Nesse algoritmo, uma aresta que não faz parte do corte mínimo é removida a cada passo, repetindo-se o processo até que restem somente as arestas que façam parte do corte desejado.

Iremos, no próximo capítulo, enfocar a construção de algoritmos randomizados que se baseiam na idéia de contrações de Nagamochi e Ibaraki.

Capítulo 3

Algoritmo randomizado por contrações

Neste capítulo descreveremos uma solução para o *PCM* através de um algoritmo randomizado, analisando a probabilidade como o qual este é capaz de retornar a solução ótima. Em seguida, descreveremos como o algoritmo pode ser utilizado para solucionar o *PCM* sobre grafos cujas arestas possuem pesos, detalhando cada uma de suas operações e avaliando sua complexidade.

3.1 O algoritmo

O algoritmo probabilístico que iremos descrever foi proposto por David Karger [1] e é baseado nas idéias de contração propostas por Nagamochi e Ibaraki. Sua principal diferença em relação a este último consiste no fato de que ao invés de utilizar um procedimento de identificação de uma aresta que não faz parte no corte mínimo do grafo, simplesmente seleciona uma aresta aleatoriamente e a considera como não pertencente ao corte mínimo.

A razão pela qual é possível utilizar um procedimento de seleção aleatória para remoção de uma aresta do grafo é que podemos assumir que uma aresta "típica"não faz parte do corte mínimo, já que normalmente, o número de arestas de um corte é pequeno em relação à cardinalidade do conjunto de arestas de um grafo.

A contração de uma aresta (u, v) pode ser efetuada através do seguinte procedimento:

Procedimento Contrai-Aresta(G,u,v)

Faça $G \leftarrow G \setminus (u, v)$

Substitua os vértices u e v por um novo vértice z.

Para cada vértice w, $w \notin \{u, v\}$, substitua as arestas (u, w) e (v, w) pela aresta (z, w).

Fim Procedimento

Denominaremos $G(V, A) \setminus F$, o novo grafo obtido através da aplicação de um sequência de operações de contração de um subconjunto de arestas $F \in A$. Chamaremos de $V \setminus F$ o conjunto de vértices e de $A \setminus F$, o conjunto de arestas do grafo resultante $G(V, A) \setminus F$. O conjunto $V \setminus F$ é formado pelos meta-vértices gerados pela contração de vértices de G, enquanto que o conjunto de arestas $A \setminus F$ é formado pelas arestas que sobreviveram ao processo de contração das arestas F de $G(V, A)$.

O processo de contração deve ser repetido até que restem somente dois vértices. Desta forma, supondo que removemos apenas arestas que não fazem parte do corte mínimo, então, o valor do corte mínimo do grafo $G'(V, A)$ obtido será exatamente igual ao valor do corte mínimo do grafo original $G(V, A)$.

Suponha que o valor do corte mínimo em G' é maior que k. Então, de alguma forma, o número de arestas conectando os vértices u' (resultantes da contração de vértices $u \in C$ de $G(V, A)$) a vértices v'(resultantes da contração de vértices $v \in \bar{C}$) aumentou. Porém, como a operação de contração não cria novas arestas, temos, por contradição que o corte mínimo não pode ter aumentado.

Suponha agora que o valor do corte mínimo em G' diminuiu. Então, foi criado, a partir das operações de contração em $G(V, A)$, um subgrafo em $G'(V, A)$ cujo grau de conectividade é menor que k. Como todos os subgrafos em $G(V, A)$ tem conectividade maior que k e como as operações de contração geram vértices com grau maior ou igual aos graus dos vértices em $G(V, A)$, então não é possível a geração de subgrafos deste tipo, logo temos que o corte não diminui.

Consequentemente, temos que o valor do corte mínimo do novo grafo obtido é igual ao valor do corte mínimo do grafo original.

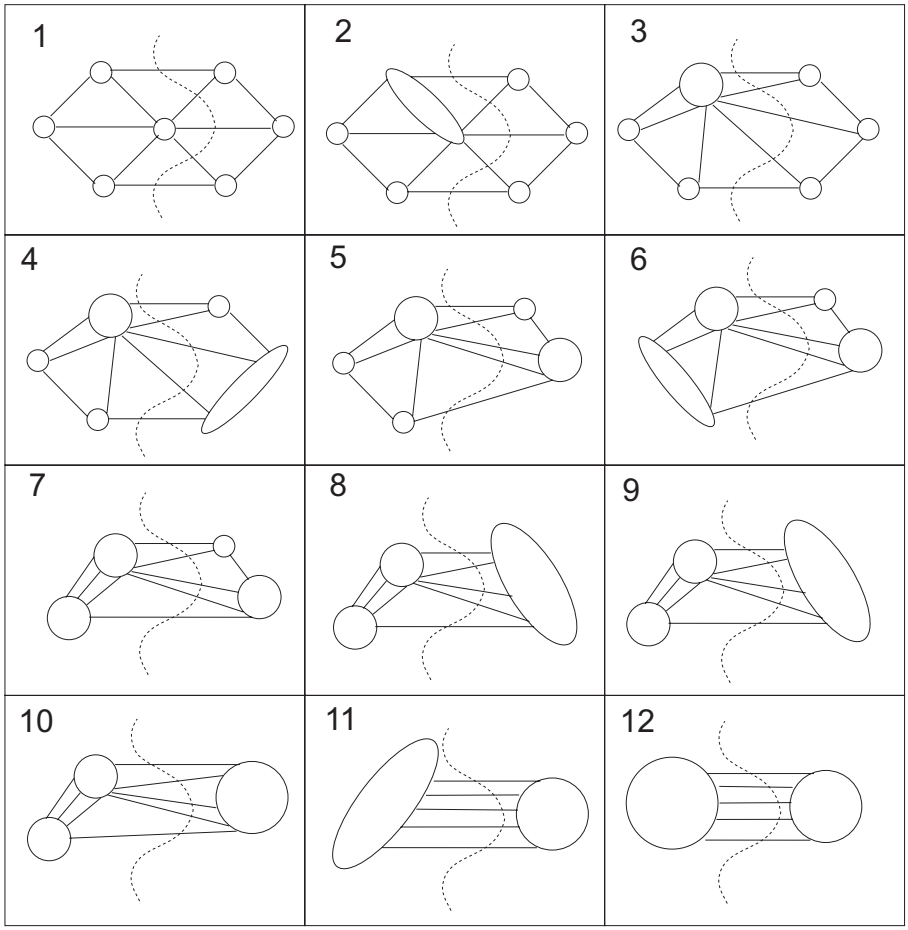

Figura 3.1: Sequência de contrações

O algortimo que apresentamos abaixo se aplica a multigrafos, isto é, a grafos que podem conter mais de uma aresta conectando dois vértices.

Procedimento Contração-Randomizada(G,n)

repita n vezes

Escolha uma aresta (u, v) de G aleatoriamente
Contrai-Aresta(G,u,v)

fim repita

retorne G

Fim Procedimento

Algoritmo Corte-Mínimo-Randomizado(G)

$G' \leftarrow$ Contração-Randomizada(G,$|V| - 2$)

$valor \leftarrow$ ObtemValor(G')

Fim Algoritmo

3.2 Análise probabilística

Para verificarmos a qualidade da solução gerada pelo algoritmo *Corte-Mínimo-Randomizado* é necessário calcular a probabilidade de selecionarmos um conjunto de arestas que não faz parte do corte mínimo do grafo de entrada.

Teorema 1 *A probabilidade com a qual o algoritmo de contração determina o corte mínimo é maior ou igual a* $\Omega(n^{-2})$.

Prova: Considere um multigrafo $G(V, A)$ que possui um corte mínimo K, de tamanho k. A cada iteração o número de vértices de $G(V, A)$ diminui de uma unidade, sendo, na i-ésima iteração, o número de vértices igual a $n_i = n - i + 1$. O número de arestas $|A|$ em $G(V, A)$ na i-ésima iteração deve ser no mínimo $n_i \frac{k}{2}$, já que o menor grau de cada vértice deve ser maior ou igual a k. Considerando esse fato, temos que a probabilidade de escolhermos uma aresta pertencente ao corte mínimo K é igual à

$$\frac{k}{n_i \frac{k}{2}} = \frac{2}{n_i} \tag{3.1}$$

Logo, a probabilidade de escolhermos uma aresta que não faz parte do corte mínimo K, na i-ésima iteração, é maior ou igual a $\left(1 - \frac{2}{n_i}\right)$. Consequentemente, a probabilidade de escolhermos todas as arestas que não fazem parte do corte mínimo de $G(V, A)$ é maior ou igual a:

$$\begin{aligned}
\left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right)\cdots\left(1 - \frac{2}{3}\right) &= \left(\frac{n-2}{n}\right)\left(\frac{n-3}{n-1}\right)\cdots\left(\frac{2}{4}\right)\left(\frac{1}{3}\right) \\
&= \frac{1}{\binom{n}{2}} \\
&= \Omega(n^{-2})
\end{aligned}$$

Como todo grafo possui pelo menos um corte mínimo, temos que a probabilidade com a qual o algoritmo de contração determina esse corte é maior ou igual a $\Omega(n^{-2})$.

Como conseqüência do teorema acima, temos o seguinte resultado:

Corolário 1 *Um corte mínimo $(C, \bar{C})$ sobrevive a uma contração de k vértices com probabilidade maior ou igual a $C_{k,2}/C_{n,2} = \Omega((k/n)^2)$.*

Prova: Seguindo as idéias da prova anterior, um corte $(C, \bar{C})$ sobrevive a x contrações com probabilidade

$$Pr = \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right)\ldots\left(1 - \frac{2}{n-x+1}\right) =$$

$$\frac{(n-2)(n-3)\ldots(n-x-1)}{(n)(n-1)\ldots(n-x+1)} =$$

Multiplicando o numerador e o denominador da fração pelo fator $(n - x - 2)!(n - x)!.2$ temos

$$\frac{(n-2)(n-3)\ldots(n-x-1)(n-x-2)!(n-x)!.2!}{(n)(n-1)\ldots(n-x+1)(n-x-2)!(n-x)!.2!} =$$

$$\frac{2!(n-2)!(n-x)!}{2!n!(n-x-2)!}$$

$$\frac{\left(\begin{array}{c} n-x \\ 2 \end{array}\right)}{\left(\begin{array}{c} n \\ 2 \end{array}\right)}$$

Fazendo $x = n - k$, de forma que restem k vértices após as contrações, temos a seguinte expressão para a probabilidade

$$\frac{\left(\begin{array}{c} n-(n-k) \\ 2 \end{array}\right)}{\left(\begin{array}{c} n \\ 2 \end{array}\right)} = \frac{\left(\begin{array}{c} k \\ 2 \end{array}\right)}{\left(\begin{array}{c} n \\ 2 \end{array}\right)} \tag{3.7}$$

3.3 Grafos com arestas com pesos

Podemos adaptar facilmente o algoritmo descrito anteriormente para que o mesmo possa ser aplicado a grafos cujas arestas possuem pesos. Para isto, basta considerarmos que uma determinada aresta com peso p corresponde a um conjunto de p arestas paralelas em um multigrafo correspondente.

O algoritmo original proposto por Karger representa o grafo de entrada através de uma matriz de adjacências W, onde cada um de seus elementos $W(u,v)$ representa o peso de uma aresta (u,v). A inexistência de uma aresta (u,v) é representada na matriz através da atribuição do valor zero ao elemento $W(u,v)(W(u,v)=0)$. Além disso, para cada vértice, é definido um peso $D(u)$ que é igual a soma dos pesos de todas arestas a ele incidentes.

Para realizarmos a escolha aleatória de uma aresta podemos proceder da seguinte forma: primeiramente escolhemos um vértice u com probabilidade $Pr[\text{escolher } u]$ proporcional a $D(u)$. Em seguida, escolhemos dentre os vértices sucessores de u, um vértice v com probabilidade $Pr[\text{escolher } v]$ proporcional a $W(u,v)$. Podemos verificar, a partir do seguinte teorema, que a probabilidade de escolha de uma aresta através desse procedimento é proporcional ao seu peso.

Teorema 2 *Se uma aresta é escolhida através do procedimento descrito acima, então , a probabilidade* $Pr[escolher(u,v)\ ]$ *é proporcional a* $W(u,v)$.

Prova: seja $\delta = \sum_u D(u)$. Então

$$\begin{aligned} Pr[\text{escolher } (u,v)] &= Pr[\text{escolher } u].Pr[\text{escolher } (u,v)|\text{escolhido } u] \\ &\quad + Pr[\text{escolher } v].Pr[\text{escolher } (u,v)|\text{escolhido } v] \\ &= \frac{D(u)}{\sigma}.\frac{W(u,v)}{D(u)} + \frac{D(v)}{\sigma}.\frac{W(u,v)}{D(v)} \\ &= \frac{2W(u,v)}{\sigma} \qquad (3.8) \\ &\propto W(u,v) \qquad (3.9) \end{aligned}$$

3.3.1 Seleção dos vértices

A seleção de um vértice com probabilidade proporcional ao seu peso pode ser feita através da utilização de uma lista $DS(u)$ de somas acumuladas dos pesos de cada um dos vértices do grafo. Através desta lista, associamos intervalos a cada um dos vértices, cujos tamanhos são proporcionais aos seus pesos. Em seguida, através de um gerador de números aleátorios, geramos um número n entre 0 e $DS(u) = \sum_{u \in V} D(u)$.

Através de um procedimento de busca, por exemplo, uma busca binária, podemos determinar o intervalo $[DS(i), DS(i+1)]$, no qual o valor sorteado n está contido e por conseguinte, o vértice i escolhido. Através da utilização de uma lista de somas acumaladas $WS(u,v)$, podemos proceder da mesma forma para sortear um vértice v sucessor do vértice u escolhido.

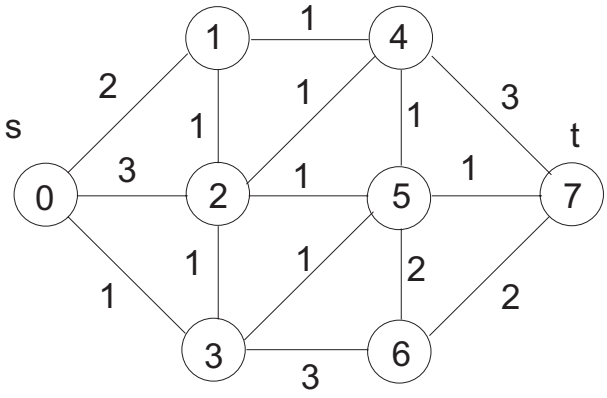

Figura 3.2: Grafo

	0	1	2	3	4	5	6	7
0	0	2	3	1	0	0	0	0
1	2	0	1	0	1	0	0	0
2	3	1	0	1	1	1	0	0
3	1	0	1	0	0	1	3	0
4	0	1	1	0	0	1	0	3
5	0	0	1	1	1	0	2	1
6	0	0	0	3	0	2	0	2
7	0	0	0	0	3	1	2	0

Tabela 3.1: W(u,v)

	0	1	2	3	4	5	6	7
0	0	2	5	6	6	6	6	6
1	2	2	3	3	4	4	4	4
2	3	4	4	5	6	7	7	7
3	1	1	2	2	2	3	6	6
4	0	1	2	2	2	3	3	6
5	0	0	1	2	3	3	5	6
6	0	0	0	3	3	5	5	7
7	0	0	0	0	3	4	6	6

Tabela 3.2: WS(u,v)

u	D(u)	DS(u)
0	6	6
1	4	10
2	7	17
3	6	23
4	6	29
5	6	35
6	7	42
7	6	48

Tabela 3.3: D(u) e DS(u)

Procedimento Seleciona-Vertice(DS)

$min \leftarrow 0$

$max \leftarrow DS(u)$

$n \leftarrow$ Gera-Número-Aleatório(min,max)

Se $n < DS(0)$

então retorne 0

senão

Para $i = 0$ até $|V| - 2$ faça

Se $DS(i) \leq i < DS(i+1)$ **então retorne** $i + 1$

fim para

Fim Procedimento

Procedimento Seleciona-Vertice-Adjacente(u,D,WS)

$min \leftarrow 0$

$max \leftarrow WS(u, |V| - 1)$

$n \leftarrow$ Gera-Numero-Aleatório(min,max)

Se $n < WS(u, 0)$

então retorne 0

senão

Para $i = 0$ até $|V| - 2$ faça

Se $WS(u, i) \leq i < WS(u, i+1)$ **então retorne** $i + 1$

fim para

Fim Procedimento

Procedimento Seleciona-Aresta(D,DS,WS)

$u \leftarrow$ Seleciona-Vértice(DS)

$v \leftarrow$ Seleciona-Vértice-Adjacente(u,D,WS)

retorne u,v

Fim Procedimento

3.3.2 A operação de contração

Uma vez tendo sido determinada a aresta que deve ser colapsada na iteração corrente, devemos atualizar o grafo de tal forma que o mesmo reflita o resultado da operação de contração. Essa operação pode ser realizada facilmente através de operações aritméticas envolvendo os elementos da matriz W e do vetor de pesos D.

Procedimento Contrai-Aresta-com-Pesos(u,v)

$D(u) \leftarrow D(u) + D(v) - 2W(u,v)$

$D(v) \leftarrow 0$

$W(u,v) \leftarrow W(v,u) \leftarrow 0$

Para cada vértice w, excetuando-se u e v

$W(u,w) \leftarrow W(u,w) + W(v,w)$

$W(w,u) \leftarrow W(w,u) + W(w,v)$

$W(v,w) \leftarrow W(w,v) \leftarrow 0$

Fim Para

Fim Procedimento

Após a operação de contração, as listas de somas acumuladas se tornarão inconsistentes, sendo necessário atualizá-las para que possam refletir a configuração do novo grafo. Suponha, por exemplo, que uma aresta u, v foi contraída, então devemos efetuar as seguintes operações para atualizar as listas de somas parciais:

Procedimento Atualiza-Somas-Parciais(WS,DS,u,v)

Recalcular as somas parciais WS(u,i) dos vértices adjacentes a u

Recalcular as somas parciais DS(u) a partir do vértice u.

Fim Procedimento

Finalmente chegamos à especificação do algoritmo para o tratamento de grafos com pesos associados às arestas:

Procedimento Contração-Randomizada-com-Pesos(G,n)

repita n vezes

$u, v \leftarrow$ Seleciona-Aresta(D,DS,WS)
Contrai-Aresta-com-Pesos(u,v)
Atualiza-Somas-Acumuladas(WS,DS,u)

fim repita

retorne G

Fim Procedimento

Algoritmo Corte-Mínimo-Randomizado-com-Pesos(G)

$G' \leftarrow$ Contração-Randomizada-com-Pesos(G,$|V| - 2$)
$valor \leftarrow$ ObtemValor(G')

Fim Algoritmo

3.3.3 Complexidade

A complexidade do algoritmo *Corte-Mínimo-Randomizado-com-Pesos* é determinada pelo procedimento de *Contração-Randomizada-com-Pesos*. Esse procedimento envolve $n-2$ repetições, onde n é o número de vértice do grafo, dos procedimentos *Seleciona-Aresta*, *Contrai-Aresta-com-Pesos* e *Atualiza-Somas-Acumuladas*.

Primeiramente avaliaremos o total de operações executadas em cada um dos procedimentos em uma determinada iteração i do algoritmo. Para uma determinada iteração i, utilizaremos a notação n_i para número de vértices do grafo.

O procedimento *Seleciona-Aresta* envolve a escolha de um vértice u através do procedimento *Seleciona-Vértice* e a escolha de um vértice v, adjacente a u, através do procedimento *Seleciona-Vértice-Adjacente*. Ambos os procedimentos requerem a geração de um número aleatório contido em um intervalo definido pelas somas acumuladas, o qual assumiremos aqui que é realizada por uma rotina do tipo caixa-preta em O(1). Além disso, cada um dos procedimentos de seleção requer a realização de uma busca para determinação do vértice escolhido. Essa tarefa pode ser implementada eficientemente através

de uma pesquisa binária a qual requer $\log n_i$ operações tanto para a determinação do vértice u escolhido quanto para a a seleção do vértice v adjacente a u. Por sua vez, os procedimentos *Contrai-Aresta-com-Pesos* e *Atualização-de-Somas-Parciais* requerem um total de operações da ordem de n_i.

Como o número total de operações dos três procedimentos é da ordem do número de vértices do grafo na iteração corrente e como a cada repetição o número de vértices é reduzido em uma unidade, chegamos à complexidade $O(n^2)$ através de uma análise de complexidade amortizada.

Pelo teorema 1, temos que a probabilidade de sucesso do algoritmo de contração é $\Omega(n^{-2})$, o que nos obriga a executar o algoritmo $n^2 \log n$ vezes de forma que possamos sobrepassar a probabilidade de sucesso por um fator de $\log n$. Consequentemente, chegamos a um valor de complexidade final igual a $O(n^4 \log n)$.

A complexidade de um algoritmo para o *PCM* baseado em algoritmos para o problema de fluxo máximo $s-t$ está em torno de n^4; $O(n^3)$ para solucionar um problema de fluxo máximo $s-t$ e $O(n)$ necessário para resolver cada problema de corte mínimo $s-t$. Logo, a complexidade do algoritmo de contração randomica é ainda bastante alta se levarmos em consideração que este não é um algoritmo exato.

Capítulo 4

Acelerando o algoritmo randomizado

No capítulo anterior verificamos que a complexidade do algoritmo proposto anteriormente é ainda bastante elevada para o objetivo a que nos propomos. Iremos propor uma modificação de forma que seja possível aumentar a probabilidade de sucesso de uma execução do algoritmo, o que consequentemente, nos permitirá reduzir significativamente a complexidade total do algoritmo. Em seguida, descreveremos sua complexidade, assim como a probabilidade com a qual o mesmo é capaz de retornar a solução exata.

4.0.4 O Algoritmo de contração recursivo

Grande parte da complexidade apresentada pelo algoritmo anterior é resultante da baixa probabilidade de sucesso de uma execução do algoritmo, o que nos obriga a executá-lo um número de vezes da ordem de $n^2 \log n$. Logo, se de alguma forma conseguirmos aumentar a probabilidade de sucesso de uma execução, então poderemos reduzir consideravelmente a complexidade total do algoritmo.

Podemos observar que as contrações iniciais possuem grande probabilidade de não selecionar uma aresta pertencente ao corte mínimo do grafo, contudo, à medida que este se torna cada vez mais compacto, essa probabilidade se torna cada vez menor, chegando ao extremo de apenas 1/3 de sucesso.

Uma possível estratégia para amenizar esse problema consiste em reduzir o grafo somente até o ponto em que a probabilidade de escolhas bem sucedidas ainda seja elevada. A partir desse ponto devemos utilizar um algoritmo exato que retorne o corte mínimo para o grafo corrente. Por outro lado, sabemos que os melhores algoritmos exatos possuem complexidade em torno

de $O(n^3)$, o que torna esta idéia pouco eficaz na prática.

Uma alternativa mais promissora consiste em substituir a execução de um algoritmo exato por duas chamadas recursivas do algoritmo randomizado para o grafo corrente. Dessa forma, iremos executar um número de vezes relativamente grande para os grafos muito colapsados - para os quais, verificamos que a probabilidade de sucesso é baixa - enquanto executaremos um número de vezes reduzido para os grafos cujos níveis de contração possuem grande probabilidade de não conterem as arestas que fazem parte do corte mínimo.

As idéia do algoritmo pode ser resumida da seguinte forma: invocamos a rotina de contração para que esta contraia um número t de arestas do grafo original, por exemplo $n/\sqrt{2}$. Então, para o grafo obtido G', executamos duas vezes, de forma recursiva, a rotina que determina o corte mínimo , e escolhemos dentre os dois cortes obtidos, aquele que forneceu o melhor resultado. Abaixo, descrevemos o algoritmo em pseudo-código.

Procedimento Contração-Recursiva(G)

Se $|V| = 2$

então

$valor \leftarrow ObtemValor(G)$

retorne $valor$

senão

$G' \leftarrow$ Contração-Randomizada-com-Pesos(G , $n/\sqrt{2}$)

$valorG' \leftarrow$ Contração-Recursiva (G', $n/\sqrt{2}$)

$G'' \leftarrow$ Contração-Randomizada-com-Pesos(G , $n/\sqrt{2}$)

$valorG'' \leftarrow$ Contração-Recursiva (G'', $n/\sqrt{2}$)

Se $valorG' < valorG''$ então retorne $valorG'$

senão retorne $valorG''$

Fim Procedimento

Através de uma árvore de computação binária podemos chegar à uma compreensão bastante intuitiva do processo. Nessa árvore, o nó raiz corresponde ao grafo de entrada, enquanto que os nós intermediários correspondem aos diferentes estágios de contração. Já as folhas da árvore correspondem aos grafos colapsados contendo apenas dois vértices, os quais definem a base da recursão. Para um determinado nó G_j^i em um nível de recursão i, associamos dois filhos, a saber, G_j^{i+1} e G_{j+1}^{i+1}. Considerando que G_j^i contém t vértices,

então cada um de seus filhos é obtido através da contração de $t/\sqrt{2}$ vértices, o que faz com que a árvore possua profundidade $2 \log n$ e número de folhas igual à $O(n^2)$.

Podemos considerar o caso em que executamos, n^2 vezes, a versão não recursiva do algoritmo , como sendo equivalente a execução do algortimo recursivoa sobre n^2 árvores de altura unitária, no qual a contração é completa($|V|-2$ vértices são contraídos). Logo, o processo recursivo não leva à geração de um número menor de cortes candidatos, e sim, a um compartilhamento do esforço computacional.

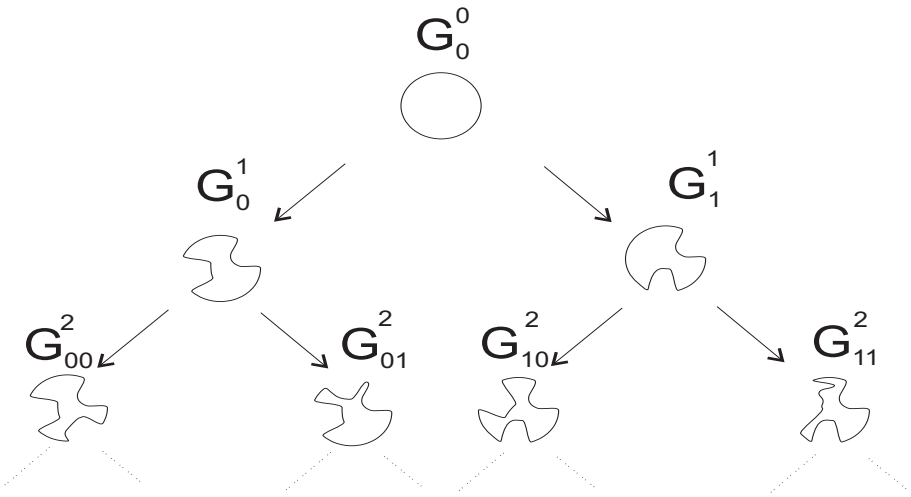

Figura 4.1: Árvore de recursão

4.0.5 Complexidade

A complexidade do algoritmo recursivo é expressa através da seguinte fórmula de recorrência:

$$T(n) = 2(n^2 + T(\frac{n}{\sqrt{2}})) \tag{4.1}$$

Desenvolvendo essa fórmula, aplicando, por exemplo, o método de substituições, chegamos ao seguinte valor de complexidade

$$T(n) = O(n^2 \log n) \tag{4.2}$$

Uma questão importante diz respeito ao espaço de armazenamento necessário para o armazenamento do grafo nos diversos níveis de contração. Podemos observar que para cada nível de recursão precisamos armazenar somente

um grafo, por exemplo, a sequência $(G_0^0, G_1^0, G_2^0, ...)$ na figura 4.1. Como os grafos no nível de recursão i possuem $O(n/2^{\frac{i}{2}})$ temos que o espaço total de armazenamento é

$$O(\sum_{i=0}^{h} \frac{n^2}{2^i}) = O(n^2) \tag{4.3}$$

Além disso, precisamos, em cada nível de recursão, registrar o melhor corte, porém isto pode ser feito gastando um espaço da ordem de $O(n^2)$.

4.0.6 Análise probabilística

Para calcularmos a complexidade total do algoritmo, precisamos determinar o número de execuções necessárias para se obter uma solução com grau de probabilidade de sucesso suficiente. Iremos mostrar, que o algoritmo proposto acima encontra o corte mínimo com probabilidade $\Omega(1/\log n)$.

Prova: Suponha que um determinado corte $(C, \bar{C})$ tenha sobrevivido até um determinado nó G_i^j na árvore de recursão. Então, esse corte sobreviverá até alguma folha da árvore se os seguintes critérios forem satisfeitos simultâneamente:

- O corte $(C, \bar{C})$ sobrevive à sequência de contrações em G_j^i e/ou G_j^{i+1}, ambos filhos de G_j^i.
- O corte $(C, \bar{C})$ é retornado pela chamada recursiva que é executada após as sequências de contrações em G_j^i e/ou G_j^{i+1}.

Sabemos, pelo corolário 1, que a probabilidade de que um corte sobreviva a uma sequência de k contrações é igual a

$$\frac{C_{k,2}}{C_{n,2}} \tag{4.4}$$

Fazendo $k = n/\sqrt{2}$, e simplificando as frações, temos

$$\frac{(n/\sqrt{2})(n/\sqrt{2}-1)}{n(n-1)} = \frac{1}{2} - O(\frac{1}{n}) \tag{4.5}$$

Logo, a probabilidade de que $(C, \bar{C})$ seja retornado pelo algoritmo é

$$P(n) = 1 - (1 - \frac{1}{2}P_{filho\,esquerdo}(\frac{n}{\sqrt{2}}))(1 - \frac{1}{2}P_{filho\,direito}(\frac{n}{\sqrt{2}})) - O(\frac{1}{n}) \tag{4.6}$$

$$P(n) = 1 - (1 - \frac{1}{2}P(\frac{n}{\sqrt{2}}))^2 - O(\frac{1}{n}) \tag{4.7}$$

Assumindo que o termo $O(\frac{1}{n})$ é desprezível, podemos resolver a recorrência através de uma troca de variáveis. Fazendo $p_k = P(\sqrt{2^k})$ podemos reescrever a fórmula como:

$$p_{k+1} = p_k - \frac{1}{4}p_k^2 \tag{4.8}$$

Fazendo $z_k = 4/p_k - 1$, temos que $p_k = 4/(z_k + 1)$. Então, substituíndo na formula de recorrência temos:

$$z_{k+1} = z_k + 1 + \frac{1}{z_k} \tag{4.9}$$

Através de indução chegamos a expressão

$$k < z_k < k + H_{k-1} + 3 \tag{4.10}$$

onde H_k é o k_{th} número harmônico. Logo $z_k = k + O(\log k)$ e $p_k = 4/(z_k + 1) = 4/(k + O(\log k) + 1) = \Theta(1/k)$. Consequentemente,

$$P(n) = p_{2\log_2 n} = \Theta(\frac{1}{\log n}) \tag{4.11}$$

Finalmente, chegamos a conclusão de que o algoritmo recursivo retorna o corte de custo mínimo com probabilidade igual a $\Omega(1/\log n)$. Consequentemente, basta executarmos um número de vezes da ordem de $\log^2 n$ para alcançarmos a mesma probabilidade de sucesso do algoritmo anterior. Como o algoritmo recursivo possui complexidade $O(n^2)$, chegamos finalmente a complexidade total de $O(n^2 \log^2 n)$.

Capítulo 5

Resolvendo o problema de corte s-t mínimo em grafos direcionados

Nos capítulos anteriores descrevemos a utilização de técnicas de randomização na solução do problema de determinação do corte mínimo em grafos não-direcionados. Neste capítulo, veremos como aplicar as mesmas idéias no desenvolvimento de um algoritmo para o versão $s-t$ do problema de corte mínimo em um grafo direcionado. Descreveremos as estruturas de dados e as funções utilizadas pelo algoritmo, explicitando as principais diferenças entre o algoritmo para PCM e a nova versão proposta. Finalmente, descrevemos o algoritmo em pseudo-código e avaliamos sua complexidade teórica.

5.1 Características do problema

Como vimos anteriormente, a principal diferença entre o PCM e o $PCMs$-t está na introdução, nesse último, de uma restrição sobre as partições geradas pelos cortes candidatos. No $PCMs$-t especificamos, em um grafo de entrada $G(V, A)$, dois vértices especiais denominados $s-t$, e impomos a restrição de que todo corte que compõe uma solução viável deve separar s e t em duas partições distintas.

Essa restrição nos leva a tomar certos cuidados adicionais na utilização das idéias de contração probabilística para o $PCMs{-}t$. Por exemplo, durante a contração de uma aresta (u, v), se exatamente um dos vértices for igual a s ou t, devemos torná-lo representante do conjunto formado pela contração dos dois vértices, de forma que possamos identificar quando um conjunto de vértices contém s ou t. Já no algortimo para o PCM, qualquer vértice pode

se tornar o representante de um conjunto formado pela contração de vértices, já que não fazemos nenhuma distinção especial sobre os vértices do grafo.

Um outro problema se deve ao fato de que a seleção de uma aresta u, v conectando um conjunto de vértices representado por s a um conjunto de vértices representado por t, deve ser invalidada já que esta causaria a contração dos vértices colapsados em s e t em um único conjunto de vértices impossibilitando assim obtenção de uma solução viável.

Antes de prosseguirmos com a descrição da estrutura de dados, definiremos os conceitos de *metavértice* e *vértice representante*:

Definição: Um *metavértice* é um conjunto de vértices gerado pela contração de vértices e/ou metavértices.

Definição: Um *vértice representante* ou *vértice pai* de um métavértice $\bar{u}$ contendo um conjunto de vértices U é:

- O vértice s, caso $s \in U$.
- O vértice t, caso $t \in U$.
- Qualquer vértice $u \in U$ caso $s \notin U$ e $t \neq U$.

Ao longo do texto, algumas vezes denominaremos um *metavértice* por simplesmente um vértice, já que ele pode ser visto dessa forma se considerarmos um grafo isoladamente ao longo do processo de contração.

Iremos agora descrever a estrutura de dados utilizada pelo algoritmo de forma a possibilitar o tratamento adequado das restrições e caracteríticas do $PCMs - t$.

5.2 A estrutura de dados

A estrutura de dados utilizada para representar um grafo de entrada $G(V, A)$ possui os seguintes campos:

G.n - número de vértices.

G.na - número de vértices ativos.

G.nad - número de vértices admissíveis.

G.V - vetor de vértices.

Em nossa representação classificamos os vértices em três grupos: *vértices ativos*, *vértices admissíveis* e *vértices inativos*.

Um *vértice ativo* é um vértice(metavértice) que possui pelo menos um sucessor. Esses vértices são aqueles que podem ser selecionados como o vértice u na seleção de uma aresta (u, v). No início do processo, todos os vértices que possuem algum sucessor são considerados ativos, já que cada um deles é representante de pelo menos um conjunto unitário que é determinado por eles mesmos.

Um *vértice admissível* é um vértice que pode ser escolhido como um sucessor v de um vértice ativo u na seleção de uma aresta (u, v). Um vértice ativo, após uma operação de contração, se tornará um vértice admissível se não possuir mais nenhum sucessor. Em outras palavras, isto significa que ele poderá ser escolhido como o vértice v mas não mais como o vértice u de uma aresta selecionada. Podemos verificar que todo vértice ativo é também um vértice admissível.

Os *vértices inativos* são os vértices que passaram a fazer parte de um metavértice e não mais fazem parte do grafo corrente como um objeto individual.

O conjunto de vértices do grafo é armazenado em um vetor V no qual cada elemento possui as seguintes informações:

V[u].r - rótulo do vértice.

V[u].ns - número de sucessores imediatos.

V[u].nsa - número de sucessores ativos.

V[u].Su - vetor de sucessores do tamanho do grau de saída do vértice u.

V[u].peso - peso do vértice u.

V[u].sp - soma parcial dos pesos até o vértice u.

V[u].suAnt - ponteiro para a lista de sucessores anterior.

V[u].suPos - ponteiro para a lista sucessores posterior.

V[u].rep - representante do conjunto do qual o vértice u faz parte.

V[u].ind - índice que indica a posição do vértice u.

V[u].indInv - índice invertido, isto é, dada uma posição i, retorna o rótulo do vértice em i.

Alguns destes campos, como o rótulo, o peso e a soma parcial, são óbvios ou então são análogos aos que já existiam na estrutura utilizada no algoritmo anterior. Contudo, outros requerem uma explicação mais detalhada. Iniciaremos com o vetor de sucessores por ser o campo mais complexo, o qual irá introduzir uma série de modificações no algoritmo original. Os demais campos serão descritos à medida que as operações que as utilizam vão sendo abordadas.

O vetor de sucessores $V[u].su$ de um vértice u possui os seguintes campos:

V[u].Su[i].v - rótulo do vértice sucessor v na posição i.

V[u].Su[i].peso - peso da aresta (u, v).

V[u].Su[i].sp - soma parcial dos pesos dos sucessores de u até o sucessor na posição i.

A primeira vista, parece que nada de novo foi introduzido, porém devemos considerar o seguinte fato: o vetor tem tamanho fixo e não é capaz de representar todas os possíveis sucessores que um vértice pode vir a ter após uma operação de contração de um par de vértices, já que este contém somente entradas para sucessores definidos segundo a topologia do grafo original(não contraído). Suponha que dois vértices u e v de uma aresta (u, v) são colapsados e que u passe a ser o rótulo do conjunto formado pela união do dois vértices. Então, o metavértice u terá como sucessores a união dos sucessores do antigo vértice u e do vértice v. Assim, o vetor de sucessores de um vértice u, em geral não poderá representar os seus sucessores na sequência de grafos que serão gerados durante o processo.

Não entraremos neste momento nos detalhes que envolvem a operação de contração pois estes serão abordados nas seções que se seguem. O que queremos esclarecer neste instante, é que é necessário utilizar um esquema para que seja possível percorrer todos os sucessores de um vértice, imediatos ou então provenientes do conjunto de vértices o qual representa.

Resolvemos esse problema através da adição, para cada vértice u, de dois campos, a saber, $V[u].suAnt$ e $V[u].suPos$. O campo $V[u].suPos$ contém o rótulo de um vértice inativo no qual um subconjunto de seus sucessores foi herdado por u após uma operação de contração. O campo $V[u].suAnt$, funciona de forma inversa, isto é, ele contém o rótulo de um vértice que herdou um subconjunto dos sucessores de um vértive u inativo. Naturalmente, os vértices poderão ter ou não vetores de sucessores anteriores e/ou posteriores. No caso em que um vértice não possui algum destes vetores, iremos atribuir o valor -1 ao campo associado ao vetor que não está definido.

Iremos de agora em diante abordar as diversas operações necessárias para a descrição do algoritmo de contração randomizada para o $PCMs - t$.

5.3 A operação de seleção

A seleção de uma aresta pode ser feita de forma análoga à realizada pelo procedimento de seleção do algoritmo anterior. Devemos apenas lembrar que quando contraímos uma aresta (u, v) então u passa a representar o conjunto determinado pela contração de u e v. Se o vértice v for s ou t, então devemos trocar u por v de forma que os vértices s e t possam sempre representar o resultado da contração. Desta forma obedecemos a definição de *representante* conforme vimos anteriormente.

Procedimento Seleciona-Aresta-ST(G)

$u \leftarrow$ Seleciona-Vértice-ST(G)

$v \leftarrow$ Seleciona-Vértice-Adjacente-ST(G)

Se $(v = s) \vee (v = t)$ **então** Troque (u, v)

retorne u,v

Fim Procedimento

Não consideramos aqui o caso em que u também for igual a s ou t, já que se essas condições forem verdadeiras deveremos invalidar a seleção. Veremos em seguida como tratar essa situação.

5.4 A operação de validação de uma seleção

Uma questão crucial é a de como evitar que as operações de contração acabem por colocar os vértices s e t em uma mesma partição, gerando desta forma uma solução inviável. Ao selecionarmos um arco (u, v) para contração devemos verificar se a condição $(u = s \wedge v = t) \vee (u = t \wedge v = s)$ é satisfeita.

Algoritmo Validação(u,v)

Se $(u = s \wedge u = t) \vee (u = t \wedge u = s)$ **então retorne** VERDADEIRO

Senão retorne FALSO

Fim Algoritmo

Caso a condição seja satisfeita, devemos evitar a contração de (u, v) e remover v da lista de sucessores de u, de forma que (u, v) não mais possa ser escolhida em uma futura seleção. A remoção de um vértice v do vetor de sucessores é feita de forma bastante simples, bastando para isso fazer uma troca entre as posições de v e do último sucessor ativo do vetor de sucessores de u, e em seguida reduzir o número de sucessores ativos $V[u].nsa$ de u.

Algoritmo Remove-Sucessor(u,v)

Troque as posições de v e do último sucessor ativo de u

$V[u].nsa \leftarrow V[u].nsa - 1$

Fim Algoritmo

5.5 A operação de contração

A operação de contração de uma aresta é o núcleo do algoritmo. Basicamente esta operação consiste em substituir os vértices u e v por um novo vértice z, o qual deve receber a união dos sucessores e antecessores de u e v. Aqui começam a surgir alguns problemas devido a estrutura que utilizamos e ao fato de trabalharmos com um grafo direcionado.

Como nossa estrutura é basicamente estática, procuraremos modificar o grafo original o mínimo possível. Primeiramente, como já mencionamos anteriormente de forma indireta, não criaremos um novo vértice z, e sim, utilizaremos o próprio vértice u como o métavértice gerado a partir da contração de u e v. Em segundo lugar, a operação de união não terá efeito direto sobre os antecessores de v que não sejam o próprio vértice u. Desta forma, quando um vértice v se torna inativo, após a contração com um vértice u, outros vértices ativos w poderão ainda tê-lo como sucessor. Na verdade, isto significa que os vértices $w \neq u$ apontam indiretamente para u através de v. Logo, no processo de seleção de um vértice v como sucessor de um vértice u, devemos buscar o representante do conjunto do qual v faz parte de forma a determinar o verdadeiro sucessor de v, o que pode ser feito através do campo $V[v].rep$.

Descrevemos abaixo, em pseudo-código a rotina de união entre os sucessores de dois vértices u e v.

Procedimento União(u,v)

Remova o arco (u, v)

Para todos os sucessores si de v **faça**

Caso si seja

· u ou filho de u:
Remova o arco (v, si)

· sucessor de u:
Some as capacidades dos arco (u, si) e (v, si) e remova o arco (v, si)

Fim Caso

Fim Para

Para todos os filhos fv de v **faça**

Caso fv seja

· sucessor de u:
Remova o arco (u, fv)

Fim Para

Fim Procedimento

Para efetuar a operação de união utilizaremos um vetor auxiliar $VetUni$ cujo tamanho é igual ao número de vértices do grafo original. Sua função é ser capaz de armazenar, de forma ordenada, todos os elementos relacionados ao vértice u de uma aresta (u, v) escolhida, permitindo assim uma fácil comparação com os elementos associados a vértice v, e consequentemente, a identificação das ações necessárias para efetuar a união. Cada elemento do vetor possui os seguintes campos:

VetUni[v].u - rótulo do vértice que contém o sucessor.

VetUni[v].indiceV - posição do vértice v na lista de sucessores de u.

VetUni[v].tipo - tipo do elemento em VetUni. Os tipos podem ser ARCO, REPRESENTANTE , FILHO OU NENHUM.

Os campos $VetUni[u].u$ e $VetUni[u].indiceV$ são utilizados para guardar um vértice sucessor. Em nossa estrutura um vértice sucessor é determinado por um par $(u, vIndex)$ onde u é um vértice que aponta para v e $vIndex$ a posição de v no vetor de sucessores de u. O metavértices e os vértices filhos são representados diretamente pelo índice do elemento no vetor sendo que nesse casos atribuímos um valor não utilizado, por exemplo -1, aos campos $VetUni[u].u$ e $VetUni[u].v$.

Após a contração dos vértices u, v devemos remover v da lista de vetores ativos. Isto é feito na nossa estrutura através da troca das posições de v com o último vértice ativo, de forma similar a que é feita pelo procedimento *Remove-Sucessor*. Esta troca de posições não necessariamente deve ser feita de forma explicita sobre o vetor de vértices e sim através de uma manipulação de índices. Este é o motivo pelo qual definimos os campos $V[u].ind$ e $V[u].invInd$.

Além disso, após a operação de união, devemos atualizar os representantes do conjunto de vértices representado por v. Infelizmente, esta rotina deve ser feita sobre todos os vértices do grafo original, o que irá comprometer a complexidade do algoritmo aqui proposto.

Uma vez especificadas todas as operações básicas podemos descrever finalmente o algoritmo de contração para o problema *PCMs-t*.

5.6 Algoritmo

Procedimento Contração-Randomizada-com-pesos-ST(G,s,t,n)

Repita até que $G.nad == n$

$u, v \leftarrow$ Seleciona-Aresta-ST()

Se Validação(u,v)= FALSO então

Remove-Sucessor(u,v)

Atualize as somas parciais dos sucessores de u

senão

União(u,v)

Atualize o vetor de somas parciais

Atualize os representantes de cada vértice

Fim repita

retorne G

Fim Procedimento

O algoritmo recursivo pode ser obtido através da substituição do procedimento *Contração-Randomizada-com-Pesos* pelo procedimento *PCMs-t Contração-Randomizada-com-Pesos-ST*.

Procedimento Contração-Recursiva-ST(G)

Se $|V| = 2$

então

$valor \leftarrow ObtemValor(G)$

retorne $valor$

senão

$G' \leftarrow$ Contração-Randomizada-com-Pesos-ST(G , $n/\sqrt{2}$)

$valorG' \leftarrow$ Contração-Recursiva-ST (G', $n/\sqrt{2}$)

$G'' \leftarrow$ Contração-Randomizada-com-Pesos-ST(G , $n/\sqrt{2}$)

$valorG'' \leftarrow$ Contração-Recursiva-ST (G'', $n/\sqrt{2}$)

Se $valorG' < valorG''$ então retorne $valorG'$

senão retorne $valorG''$

Fim Procedimento

Obviamente, durante a implementação, devemos fazer copias dos grafos na passagem de um nível de recursão para o outro. Um forma simples de realizar este processo é alocar todo espaço de memória suficiente para armazenar os grafos em cada nível de recursão que será gerado. Como sabemos a priori que o nível de recursão máximo é igual a profundidade da árvore, podemos alocar um espaço de tamanho *Altura(árvore)*Tamanho(Grafo)*.

As cópias podem ser feitas desprezando-se os vértices sucessores inativos, já que estes não mais serão acessados. Já os vértices inativos devem ser copiados, pois com vimos anteriormente, alguns vértices ativos podem possuir vértices inativos como sucessores.

5.7 Análise de complexidade

A análise de complexidade do algorimto *Contração-Randomizada-com-pesos-ST* é bastante similar a análise de complexidade do algoritmo *Contração-Randomizada-com-pesos*. A principal diferença surge devido a introdução da operação de atualização dos representantes dos vértices, a qual requer um número de operações da ordem do número de vértices do grafo original.

Desta forma, a complexidade do algoritmo recursivo fica determinada pela expressão

$$T(n) = 2(N^2 + T(n)), \tag{5.1}$$

onde N é o número de vértices do grafo original e não varia com o nível de recursão.

Desenvolvendo a formula de recorrência acima temos:

$$\begin{aligned} T(n) &= 2(N^2 + T(\frac{n}{\sqrt{2}})) \\ &= 2(N^2 + 2(N^2 + T(\frac{n}{\sqrt{2}^2}))) \\ &= 2(N^2 + 2(N^2 + 2(N^2 + T(\frac{n}{\sqrt{2}^3})))) \\ &= \sum_{i=0}^{i=k} 2^k N^2 \end{aligned}$$

onde k é tal que $\frac{n}{\sqrt{2}^k} = 2$.

Logo, temos que

$$\begin{aligned} \log n &= \log 2^{\frac{k+2}{2}} \\ k &= 2\log n - 2 \end{aligned}$$

Voltando a fórmula de recorrência temos:

$$
\begin{aligned}
T(n) &= \sum_{i=0}^{2\log n-2} 2^k N^2 \\
&= 2N^2(\log n - 1) \sum_{i=0}^{2\log n-2} 2^k \\
&= (2N^2(\log n - 1))(\frac{2^{2\log n-2}-1}{2}) \\
&= N^2(\log n - 1)(\frac{2^{\log n^2}}{2} - 1) \\
&= N^2(\log n - 1)(\frac{n^2}{2} - 1) \\
&= (N^2 \log n - N^2)(\frac{n^2}{2} - 1) \\
&= \frac{1}{2}N^2n^2 \log n - \frac{1}{2}N^2n^2 - N^2 \log n + N^2 \\
&= O(n^4 \log n)
\end{aligned}
$$

5.8 Análise probabilística

Apesar de conseguirmos, a grosso modo, adaptar o algoritmo de contração randomizado para o problema *PCMs-t* de forma relativamente simples, o mesmo não pode ser dito a repeito de sua análise de probabilidade. A análise da probabilidade de sucesso do algoritmo de contração randomizado para o PCM é bastante simples, bastando para isso estimarmos a razão entre o número de arestas no corte mínimo e o número de arestas presentes no grafo a cada iteração. A estimativa do número de arestas no PCM era feita com base no número de vértices, o qual é bem conhecido a cada iteração, e no valor mínimo de seus graus, uma propriedade que é invariante na sequência de grafos obtidos.

No caso do problema *PCMs-t*, esta propriedade não é mais válida pois nada podemos afirmar sobre os graus dos vértices que não fazem parte da fronteira entre as duas partições. A questão sobre a análise probabilística para algoritmos randomizados para o *PCMs-t* parece estar em aberto e não será abordada neste trabalho.

5.9 Testes

Os testes foram efetuados sobre dois grafos em forma de grade tridimensional com dimensões 8x8x8 e 16x16x16 em uma máquina PC Pentium III 600 Mhz. Os resultados foram comparados com uma implementação do algoritmo push-relabel de Andrew Goldberg. Infelizmente os resultados obtidos são bastante inferiores aos resultados fornecidos pelo algoritmo push-relabel, tanto no que diz respeito ao tempo de processamento quanto à qualidade do custo obtido.

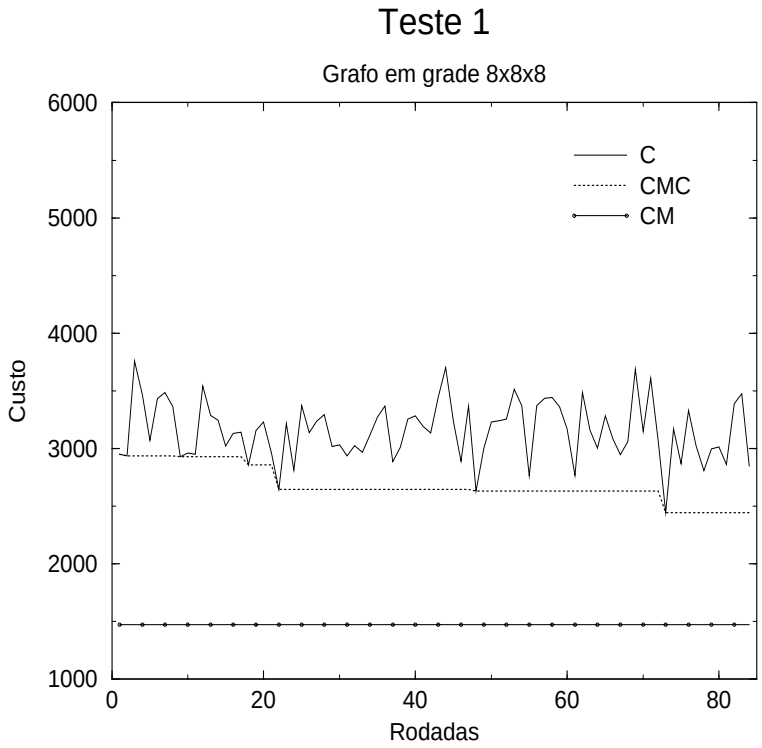

Figura 5.1: Teste 1

Grafo	Random MinCut	Push-Relable
8x8x8	39s	0.1 s
16x16x16	7854s	2 s

Tabela 5.1: Tempo de processamento

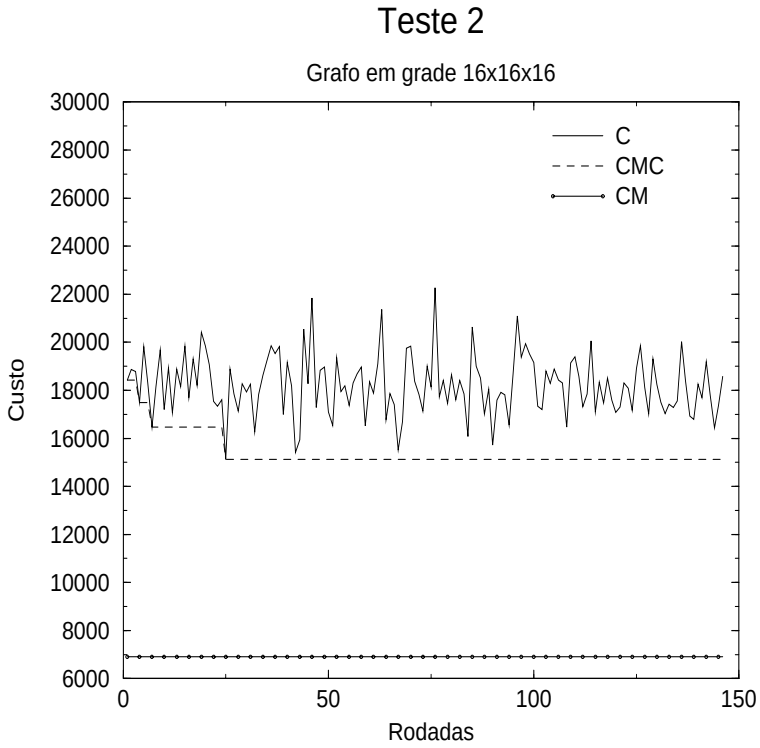

Figura 5.2: Teste 2

Grafo	Random MinCut	Push-Relable
8x8x8	2443	1474
16x16x16	15127	6414

Tabela 5.2: Custo

Nos gráficos acima temos que C representa a curva de custos obtidos em cada uma das execuções do algoritmo randomizado, CMC representa o menor custo obtido até a execução corrente e CM o custo mínimo obtido através de um algoritmo push-relabel exato.

Grande parte do elevado tempo de processamento requerido pelo algoritmo randomizado se deve à detalhes de implementação que poderiam ser melhorados(na seção final deste trabalho descrevemos algumas modificações que podem ser realizadas para que o algoritmo se torne mais eficiente). Entretanto, podemos verificar que o algoritmo push-relabel parece possuir um comportamento bastante eficiente quando aplicado a grafos em grade. Se considerarmos que esses algoritmos apresentam complexidade de pior caso em torno de n^3, podemos notar que o tempo de processamento gasto por esse algoritmo sobre grafos em forma de grade é bem menor que o esperado. Essa observação já foi feita no trabalho de Cox e Roy [17], os quais aplicaram o algoritmo push-relabel sobre grafos em grade com o intuito de resolver o problema de determinação de mapas de profundidade a partir de visão estéreo. Desta forma, aparentemente, os algoritmos do tipo "push-relabel"apresentam um comportamento bastante eficiente quando aplicados a grafos em grade, enquanto que os algoritmos randomizados tratam esta espécie de grafo da mesma forma que um grafo sem estrutura especial.

5.10 Observações

Neste trabalho abordamos a utilização de técnicas de randomização para o problema de determinação do corte de capacidade mínima em um grafo direcionado e em seguida procuramos extender essas técnicas para o problema de determinação do corte minimo s-t.

Apesar de ser possível adaptar o algoritmo original para o problema s-t de forma aparentemente simples, verificamos que a análise da probabilidade com o qual o algoritmo retorna a solução exata é bastante complexa, sendo ainda um problema em aberto. Infelizmente, sem conhecimento da probabilidade de sucesso, temos apenas um algoritmo heurístico, já que não conhecemos o número de vezes necessário para obtenção da solução exata com alto grau de probabilidade. Uma outra dificuldade surgiu devido a necessidade de aplicarmos o algoritmo à grafos esparsos com número de vértices muito grande, o que nos obrigou a modificar a estrutura de dados original.

No estágio atual, a limitação de espaço de armazenamento nos levou a aumentar a complexidade do algoritmo. Porém, sugerimos uma modificação que pode nos permitir reduzir esta complexidade de forma siginificativa. Propomos que a operação de união atue sobre os antecessores dos vértices, impedin-

do desta forma que vértices ativos apontem indiretamente para outros vértice ativos através de um vértice inativo. Naturalmente, essa modificação requer o armazenamento dos antecessores de um vértice na estrutura de dados de forma que a operação de união possa ser realizada eficientemente.

Um efeito colateral dessa modificação é que deveremos renomear os vértices sucessores, o que causará uma maior modificação no grafo. Por outro lado, através do tratamento dos antecessores, será também possível descartar completamente os vértices inativos de um nível para o outro, tornando assim a execução mais ágil. Um outro problema deverá ser abordado com cuidado, diz respeito a forma como o corte será extraído a partir do grafo colapsado uma vez que modificarmos a operação de união.

Propomos como trabalhos futuros, a determinação de uma cota para a probabilidade de sucesso do algoritmo, permitindo desta forma que tenhamos um algoritmo aproximativo. Além disso propomos a implementação das modificação sugeridas em uma tentativa de reduzir a complexidade do algoritmo de forma que esta se apresente compatível com a complexidade apresentada pelo algoritmo proposto por Karger para o *PCM*.

Bibliografia

[1] Karger, D. Random Sampling in Graph Optimization Problems. Dissertation submited to the Department of Computer Science and the Commitee on Graduate Studies of Standford University. January 1995.

[2] Gomory, R. E. and Hu, T. C. Multi-terminal network flows. Journal of the Society of Industrial and Applied Mathematics 9,4(Dec. 1961), 551-570.

[3] Gabow, H. N. A matroid approach to finding edge connectivity and packing arborescences. In Proceedings of the $23rd$ ACM Symposium on the Foundations of Computer Science (Oct. 1991), IEEE Computer Society Press, pp. 812-821.

[4] Nagamochi, H. and Ibaraki, T. Linear time algorithms for finding k-edge connected and k-node connected spanning subgraphs. Algorithmica 7 (1992), 583-596.

[5] Nagamochi, H. and Ibaraki, T. Computing edge connectivity in multigraphs and capacitated graphs. SIAM Journal of Discrete Mathmatics 5,1 (Feb. 1992), 54-66.

[6] Cherkassky, B. V. and Goldberg, A. V. On Implementing Push-Relabel Method for the Maximum Flow Problem. In *Proc. 4th Int. Programming and Combinatorial Optimization Conf.*, pages 157-171, 1995.

[7] Goldberg, A. V. and Tarjan, R. E. A new approach to the maximum flow problem. Journal of the ACM 35 (1988), 921-940.

[8] Goldberg, A.V. and Rao, S. Length Functions for Flow Computations. Tehcnical report # 97-055. NEC Research Institute, Inc. March 1997.

[9] King, V., Rao, S., and Tarjan, R. E. A faster deterministic maximum flow algorithm. In Proceedings of the 3^{rd} Annual ACM-SIAM Symposium on Discrete Algorithms (jan. 1992), ACM-SIAM, pp. 157-164.

[10] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. McGraw-Hill, New York, 1990.

[11] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[12] L. ford and D. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[13] W.J. Cook, W. H. Cunningham, W. R. Pulleyblank, A. Schrijver *Combinatorial Optimization*. Wiley-Interscience. 1998

[14] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice Hall, New Jersey, 1982.

[15] Y. Boykov, O. Veksler and R. Zabih. Energy Minimization with Discontinuities.

[16] Y. Boykov, O. Veksler and R. Zabih. A New Algorithm for Energy Minimization with Discontinuities.

[17] S. Roy and I. Cox. A Maximum-Flow Formulation of the N-camera Stereo Correspondence Problem. In *Proc. Int. Conf. on Computer Vision, ICCV'98*, Bombay, India, 1998.

[18] H. Ishikawa and D. Geiger. Oclusions, Discontinuities, and Epipolar Lines in Stereo. In *Proc. Fifth European Conference on Computer Vision, (ECCV'98)*, Freiburg, Germany, 2-6 June 1998.

[19] H. Ishikawa and D. Geiger. Segmentation by Grouping Junctions. In *IEEE Conference on computer Vision and Patter Recognition*, 1998.