MAC-499: Trabalho de Formatura Supervisionado

Implementação de Algoritmos de Aproximação envolvendo Árvores k-restritas para o Problema de Steiner em Grafos

Michel Vale Ferreira
Orientação: Cristina Gomes Fernandes

Índice

RESUMO: 3

PARTE I: DESCRIÇÃO DO PROJETO 3
- INTRODUÇÃO 3
  - ALGORITMOS IMPLEMENTADOS 3
  - ANÁLISE DOS RESULTADOS 4
- CONCEITOS BÁSICOS 4
  - GRAFO COMPLETO E DESIGUALDADE TRIANGULAR 4
  - RESTRIÇÃO 4
  - CONJUNTO SEPARADOR 4
  - ÁRVORES DE STEINER 4
  - K-ÁRVORES 4
  - ÁRVORES K-RESTRITAS 5
  - CONJUNTO REMOÇÃO E FUNÇÃO ÍNDICE 5
  - FUNÇÃO PROJEÇÃO 5
  - FUNÇÃO GANHO 6
  - FUNÇÃO GANHO RELATIVO 6
- PROBLEMA DE STEINER EM GRAFOS (PSG) 7
  - ALGORITMOS DE APROXIMAÇÃO PARA O PSG 7
- ALGORITMOS 7
  - ALGORITMO DAS ÁRVORES 3-RESTRITAS 7
    - DESCRIÇÃO DO ALGORITMO 7
  - ALGORITMO DAS ÁRVORES 3-RESTRITAS ORIGINAL DE ZELIKOVSKY 8
  - ALGORITMO DO GANHO RELATIVO 9
    - DESCRIÇÃO DO ALGORITMO 9
- IMPLEMENTAÇÕES 10
  - METODOLOGIA E DESENVOLVIMENTO 10
  - EXECUTÁVEIS 11
  - TESTES PRELIMINARES 11
  - OTIMIZAÇÕES 11
  - CONSUMO DE TEMPO ASSINTÓTICO 11
    - ÁRVORES 3-RESTRITAS 12
    - ÁRVORES 3-RESTRITAS ORIGINAL DE ZELIKOVSKY E GANHO RELATIVO (K=3) 12
    - GANHO RELATIVO 12
- TESTES E RESULTADOS 12
  - GRÁFICO COMPARATIVO 13
- CONCLUSÃO 14
  - CUSTO DAS ÁRVORES 14
  - CONSUMO DE TEMPO 14
- BIBLIOGRAFIA 14

PARTE II: O PROJETO E O BCC 15
- DESAFIOS 15
- FRUSTRAÇÕES 15
- DISCIPLINAS MAIS IMPORTANTES 15
- INTERAÇÃO COM A ORIENTADORA 15
- APRIMORAMENTO DOS CONHECIMENTOS 16

CONSIDERAÇÕES FINAIS 16

RESUMO:

Esta monografia é um relato da experiência do Projeto de conclusão de curso, sob a orientação de Cristina Gomes Fernandes, que consiste em implementar a*lgoritmos de aproximação* descritos em "*Árvores k-restritas e aproximações para o Problema de Steiner em Grafos"*, dissertação de mestrado de Eduardo Gondo[1], bem como em analisar os resultados obtidos nos testes. A monografia está dividida em duas partes:

I. Uma descrição do trabalho de implementação dos algoritmos e análise dos resultados.
II. A relação deste trabalho com o BCC.

PARTE I: DESCRIÇÃO DO PROJETO

INTRODUÇÃO

Problemas de *otimização combinatória* têm como objetivo encontrar um ponto ótimo (mínimo ou máximo) de uma função definida sobre um certo domínio finito. Embora os elementos do domínio possam ser facilmente enumerados, testar todos os elementos na busca pelo melhor é inviável do ponto de vista computacional, pois o domínio pode ser muito grande. Vários destes problemas fazem parte do conjunto dos problemas *NP-Difícei*s e o *Problema de Steiner em Grafos* é um deles[1].

Existem duas abordagens para estes problemas[2]:

- Algoritmos que produzam soluções ótimas, mas que, no pior caso, consumam tempo exponencial no tamanho da entrada. (Estes algoritmos são adequados apenas para instâncias pequenas).
- Algoritmos eficientes (isto é, que consomem tempo polinomial no tamanho da entrada) que produzam *soluções aproximadas* com uma garantia de qualidade, ou seja, um limite para o erro entre o custo da solução produzida pelo algoritmo e o custo de uma solução ótima do problema. Estes algoritmos são denominados algoritmos de aproximação e chamamos de razão de aproximação a sua garantia de qualidade.

Segundo Gondo[1], informalmente, o *problema de Steiner em grafos* pode ser definido como: *dado um grafo G com custos não-negativos nas arestas, determinar um subgrafo conexo de G de custo mínimo que contenha um dado conjunto de vértices.* Esse problema aparece no contexto de projetos de circuitos VLSI[3], árvores filogenéticas[4], problemas de conexão[5], etc.

ALGORITMOS IMPLEMENTADOS

Foram implementados, utilizando a base de dados do *Stanford GraphBase* (SGB)[2] e o *GNU project C and C++ Compiler* (GCC), três algoritmos de aproximação projetados por

---

[1] GAREY M. R., GRAHAM R. L. e JOHNSON D. S., *The complexity of computing Steiner minimal trees,* In: GONDO, E. K.: *Árvores k-restritas e aproximações para o problema de Steiner em grafos*. São Paulo, 2002. Dissertação (Mestrado) – Instituto de Matemática e Estatística. Universidade de São Paulo.
[2] CORMEN T. H., RIVEST C. E. e RIVEST R. L., *Introduction to algorithms*. In: Idem.
[3] KAHNG A. B. e ROBINS G., *On optimal interconnections for VLSI.* In: Idem.
[4] HWANG S. e PRÖMEL H. J., *A 1.598 approximation algorithm for the Steiner problem in graphs.* In: Idem.
[5] GRÖTSCHEL M. e MONMA, *Integer polyhedra arising from certain network design problems with connectivity constraints.* In Idem.

Zelikovsky presentes na dissertação de Gondo[1] e um quarto, que é o algoritmo do *ganho relativo* para k=3, uma versão específica e eficiente do algoritmo do *ganho relativo*.

Todos são baseados em árvores *k-restritas*, método comum aos *algoritmos de aproximação* com as melhores razões de aproximação conhecidas para o *PSG*.

1. **Árvores 3-restritas**
2. **Árvores 3-restritas (Original de Zelikovsky)**
3. **Ganho Relativo (k = 3)**
4. **Ganho Relativo**

Análise dos Resultados

Os resultados apresentados pelos algoritmos serão analisados, bem como seus desempenhos como algoritmos de aproximação para o PSG em termos de consumo de tempo e razão de aproximação.

Conceitos Básicos

Aqui seguem conceitos básicos para o entendimento dos algoritmos e suas implementações. Todos esses conceitos foram extraídos de Gondo[1].

Grafo Completo e Desigualdade Triangular

Dizemos que um grafo $G=(V, E)$, é completo se cada vértice de V_G é adjacente a todo outro. Dizemos que G satisfaz a desigualdade triangular se $c(ij) \leq c(ik) + c(kj)$, para quaisquer três vértices distintos i, j, k de V_G

Restrição

Sejam um grafo $G=(V, E)$, um subconjunto W de vértices de G, uma função c de E_G em $Q_{>0}$ e $G[W]$, o grafo induzido por W. A restrição de c a $G[W]$ é a função que coincide com c exceto nas arestas que não estão em $E_{G[W]}$ (isto é, arestas com ambas extremidades em W), onde ela não está definida.

$\dot{c}(e) := c(e)$ para toda aresta e em $E_{G[W]}$

Conjunto Separador

Se $A \subseteq V_G$, denotamos por $\delta(A)$ o conjunto de todas as arestas de G que têm um extremo em A e o outro em $V_G \setminus A$. O conjunto $\delta(A)$ é chamado de *separador* de A em V_G.

Árvores de Steiner

Considere um grafo G, uma função c de E_G em $Q_{>0}$ satisfazendo a desigualdade triangular e um conjunto R não-vazio de vértices terminais. Uma árvore T em G é uma *árvore de Steiner* de (G, R) se $R \subseteq V_T$.

Uma *árvore de Steiner mínima* de (G, c, R) é uma *árvore de Steiner* de (G, R) de custo mínimo. Denotamos o custo mínimo por $smt(G, c, R)$.

Denotamos por $VT(T)$ o conjunto de vértices terminais de T e por $VS(T)$ o conjunto dos demais vértices, chamados de vértices de Steiner de T.

K-Árvores

Uma *k-árvore* de (G, R) é uma árvore em G com k folhas cujos vértices internos são de Steiner e as folhas são terminais.

ÁRVORES K-RESTRITAS

Seja T uma árvore de Steiner de (*G, R*). Se *T'* é uma subárvore maximal de *T* que é uma *i*-árvore de (*G, R*) para algum i, dizemos que *T'* é um *componente cheio de T.* Se para algum *k*, todo componente cheio de *T* é uma *i-árvore* em (*G, R*) com $i \leq k$, dizemos que *T* é uma árvore *k-restrita de* (*G, R*).

Uma *árvore k-restrita mínima* de (*G, c, R*) é uma *árvore k-restrita* de (*G, R*) de custo mínimo. Denotamos o custo mínimo por $smt_k(G, c, R)$.

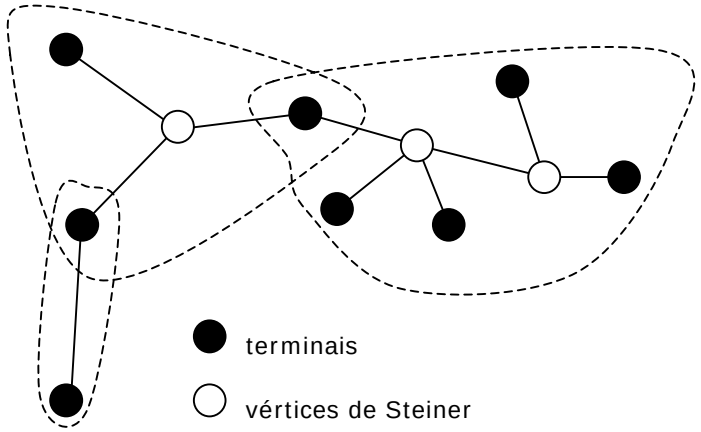


Figura 1: Exemplo de árvore de Steiner. Cada componente cheio está destacado pelas linhas pontilhadas. Note que, neste exemplo, o número máximo de terminais de um componente cheio é 5. Portanto se trata de uma árvore 5-restrita.

CONJUNTO REMOÇÃO E FUNÇÃO ÍNDICE

Sejam *G* um grafo e *c* uma função de E_G em $Q_{>0}$. Considere *M* uma árvore geradora mínima de (*G, c*) e um conjunto τ de vértices de *G*. Um conjunto remoção de τ é um separador *D* de τ em *M* com *c*(*D*) máximo.

O *índice* é o custo de um conjunto remoção. Denotamos por $ind(M, c, \tau) := c(D)$, onde *D* é um conjunto remoção de τ em *M.* Note que $|D| := |\tau| - 1$.

FUNÇÃO PROJEÇÃO

Seja *D* um conjunto remoção de τ em *M.* A função projeção *p* de *D* em τ relativa a *M* é uma função que associa cada aresta de *D* à uma aresta de *G* , definida do seguinte modo:

Para cada aresta *e* de *D*, existem dois componentes de $M - D$, digamos X_e e Y_e, cada um contendo um dos extremos de *e*. Sejam *x* e *y* os vértices de τ em X_e e Y_e respectivamente (cada componente de $M - D$ tem exatamente um vértice de τ , pois *D* é minimal). A função projeção associa a aresta *e* à aresta *xy*.

Obs.: No nosso caso, *G* é um grafo completo, isto é, cada vértice é adjacente a todos os outros.

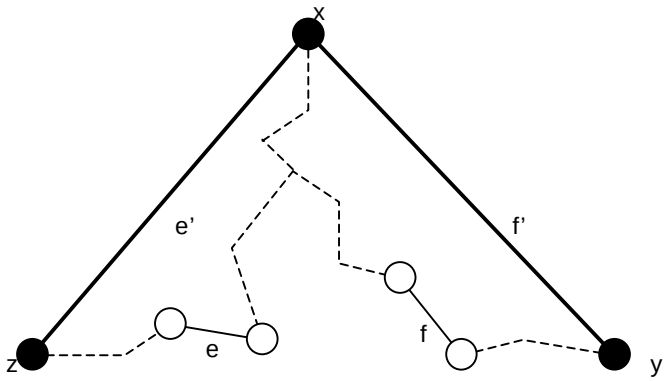


Figura 2: O conjunto remoção $D := \{e, f\}$ e a imagem da função $p(D) := \{e', f'\}$ sobre $\tau := \{x, y, z\}$ em M, onde M são as arestas pontilhadas mais as arestas de D.

FUNÇÃO GANHO

Sejam G um grafo, c uma função de E_G em $Q_{>0}$ satisfazendo a desigualdade triangular e um conjunto R não vazio de vértices terminais. Sejam T_τ uma árvore de Steiner mínima de (G, c, τ), onde τ é um subconjunto de R, $\dot{c}$ a restrição de c a $G[R]$ e M uma árvore geradora mínima de $(G[R], \dot{c})$. O ganho é calculado assim:

$g(M, \tau) := ind(M, \dot{c}, \tau) - smt(G, c, \tau)$

O ganho de τ indica se é ou não vantajoso utilizar T_τ para compor com M uma árvore de Steiner de custo menor. O ganho é a diferença entre o custo do conjunto remoção e as arestas de T_τ.

FUNÇÃO GANHO RELATIVO

Idem à função ganho, exceto que a fórmula do ganho relativo é:

$$g_r(M, \tau) := \frac{smt(G, c, \tau)}{ind(M, \dot{c}, \tau)}$$

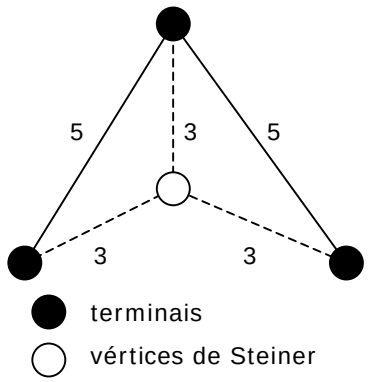


Figura 3: No grafo as linhas contínuas são arestas de uma árvore geradora mínima sobre os terminais e as linhas pontilhadas são as arestas de uma árvore de Steiner mínima. A função *índice* vale 10 e a árvore de Steiner tem custo 9, logo o *ganho* vale 1 e o *ganho relativo* vale 0,9.

PROBLEMA DE STEINER EM GRAFOS (PSG)

Definição: Seja um grafo $G=(V, E)$ conexo, uma função c de E_G em $Q_{>0}$ e um conjunto R não-vazio de vértices de G, os vértices terminais. O objetivo do PSG é encontrar uma árvore de Steiner mínima de (G, c, R).

Note que se $|R| = 2$, o PSG é equivalente a encontrar um caminho de custo mínimo entre dois vértices. Para isto existem algoritmos exatos e eficientes, como o de *Dijkstra*. Se $R = V_G$, então o PSG é equivalente a encontrar uma árvore geradora mínima de (G, c). Para isto também existem algoritmos exatos e eficientes, como o de *Kruskal* (Ambos algoritmos citados estão presentes nesta implementação). Logo, assumimos que $2 < |R| < |V_G|$.

ALGORITMOS DE APROXIMAÇÃO PARA O PSG

Seja (G, c, R) uma instância do PSG. Um algoritmo de aproximação para o PSG que produz uma árvore de Steiner T deve consumir tempo polinomial no tamanho da instância e sua razão de aproximação é um número real $\alpha > 1$, tal que $c(T) \leq \alpha * smt(G, c, R)$.

ALGORITMOS

ALGORITMO DAS ÁRVORES 3-RESTRITAS

O algoritmo das árvores 3-restritas é um algoritmo de Zelikovsky[6] utilizando árvores 3-restritas para inserir vértices de Steiner na árvore geradora mínima M de $(G[R], \dot{c})$. A razão de aproximação deste algoritmo é 1,834.

DESCRIÇÃO DO ALGORITMO

A entrada do algoritmo é um grafo completo G, uma função c de E_G em $Q_{>0}$ e um conjunto R não-vazio de vértices de G, os vértices terminais. Como saída, o algoritmo produz uma árvore de Steiner cujo custo é no máximo 1,834 do custo de uma árvore de Steiner mínima de (G, c, R).

O algoritmo começa com uma árvore geradora mínima M de $(G[R], \dot{c})$ e com um grafo S, que inicialmente é uma cópia de M.

Em cada iteração o algoritmo enumera todas as combinações de triplas τ de R e para cada uma calcula o *índice, o conjunto remoção, a árvore 3-restrita mínima* T_τ de (G, c, τ) e finalmente seu *ganho.* Então guarda a tripla com o maior ganho positivo. Enquanto existir uma tripla τ com ganho positivo, o algoritmo calcula a *função projeção*, substitui as arestas do *conjunto remoção D* em S pela árvore T_τ e atualiza o custo de cada aresta do *conjunto remoção D* em M.

O algoritmo pára quando não existem mais triplas com ganho positivo e devolve S.

[6] ZELIKOVSKY A. Z., An *11/6-approximation algorithm for the network Steiner problem*. In: Idem

Algoritmo das árvores 3-restritas (G, c, R)

Seja $\dot{c}_0$ a restrição de c a $G[R]$;
Seja M_0 a árvore geradora mínima de $(G[R], \dot{c}_0)$;
$S_0 := M_0$
$i := 1$;
Enquanto existir $\tau \subseteq R$, $|\tau| = 3$, com $g(M_{i-1}, \tau) > 0$ **faça**
  Seja $\tau_i \subseteq R$ com $|\tau_i| = 3$ e com $g(M_{i-1}, \tau_i)$ máximo;
  Seja T_i uma árvore 3-restrita mínima de (G, c, τ_i);
  Seja D_i um conjunto remoção de τ_i em M_{i-1};
  Seja p a função projeção de D_i em τ_i, relativa a M_{i-1};
  $A_i := \emptyset$;
  $c_i := c_{i-1}$;
  Para cada e em D_i **faça**
    $\dot{c}_i(p(e)) := \dot{c}_{i-1}(e) - g(M_{i-1}, \tau_i)$;
    $A_i := A_i \cup \{p(e)\}$;
  $M_i := M_{i-1} - D_i + A_i$;
  $S_i := S_{i-1} - D_i + T_i$;
  $i := i + 1$;
$f := i - 1$;
Devolva S_f;

Algoritmo das Árvores 3-restritas Original de Zelikovsky

O algoritmo originalmente proposto por Zelikovsky diferia em vários pontos da versão acima, o principal é que o algoritmo não construía as árvores S_o, S_1, ..., S_f iterativamente, apenas guardava em um conjunto W os vértices de Steiner das árvores T_τ e no fim devolvia uma árvore T geradora mínima de $(G[R \cup W], c)$. Note que $c(T) \leq c(S_f)$.

Algoritmo das árvores 3-restritas original de Zelikovsky (G, c, R)

Seja $\dot{c}_0$ a restrição de c a $G[R]$;
Seja M_0 a árvore geradora mínima de $(G[R], \dot{c}_0)$;
$i := 1$;
$W_0 := \emptyset$;
Enquanto existir $\tau \subseteq R$, $|\tau| = 3$, com $g(M_{i-1}, \tau) > 0$ **faça**
  Seja $\tau_i \subseteq R$ com $|\tau_i| = 3$ e com $g(M_{i-1}, \tau_i)$ máximo;
  Seja T_i uma árvore 3-restrita mínima de (G, c, τ_i);
  $W_i := W_{i-1} \cup VS(T_i)$;
  Seja D_i um conjunto remoção de τ_i em M_{i-1};
  Seja p a função projeção de D_i em τ_i, relativa a M_{i-1};
  $A_i := \emptyset$;
  $c_i := c_{i-1}$;
  Para cada e em D_i **faça**
    $\dot{c}_i(p(e)) := 0$;
    $A_i := A_i \cup \{p(e)\}$;
  $M_i := M_{i-1} - D_i + A_i$;
  $i := i + 1$;
$f := i - 1$;
Seja T uma árvore geradora mínima de $(G[R \cup W_f], c)$;
Devolva T;

ALGORITMO DO GANHO RELATIVO

O algoritmo do ganho relativo também foi proposto por Zelikovsky[7] utilizando árvores k-restritas mínimas para melhorar o custo da solução corrente. A diferença entre este e o anterior está no processo de escolha das árvores k-restritas. A razão de aproximação deste algoritmo é 1,694 para valores grandes de k.

DESCRIÇÃO DO ALGORITMO

A entrada do algoritmo é um grafo completo G, uma função c de E_G em $Q_{>0}$ e um conjunto R não-vazio de vértices de G, os vértices terminais e uma constante $k \geq 3$. Como saída, o algoritmo produz uma árvore de Steiner cujo custo é no máximo 1,694 do custo de uma árvore de Steiner mínima de (G, c, R).

O algoritmo começa com uma árvore geradora mínima M de $(G[R], \dot{c})$ e com um conjunto W de vértices, inicialmente vazio.

Em cada iteração o algoritmo enumera todos os subconjuntos τ de R, com $|\tau| \leq k$, e enumera todos os subconjuntos σ de $V_G \setminus R$, com $1 \leq |\sigma| \leq \tau - 2$ *e* para cada um calcula seu *índice, seu conjunto remoção, a* componente k-restrita T_τ mínima em $(G, c, [\tau \cup \sigma])$ e finalmente seu *ganho relativo.* Então guarda os conjuntos τ e σ com o menor ganho relativo. Enquanto existir conjuntos τ e σ com ganho relativo < 1, o algoritmo calcula a *função projeção*, adiciona os vértices de Steiner de T_τ a W e zera o custo $\dot{c}(e)$ de cada aresta e em $G[R]$ que está na imagem da *função projeção* de D em τ. Além disso, altera M de forma a mantê-la uma árvore geradora mínima de $(G[R], \dot{c})$.

O algoritmo pára quando $c(M)=0$. Neste ponto o algoritmo devolve uma árvore geradora mínima em $(G[R \cup W], \check{c})$, onde $\check{c}$ é a restrição de c a $G[R \cup W]$.

Algoritmo do ganho relativo (G, c, R)
1 Seja $\dot{c}_0$ a restrição de c a $G[R]$;
2 Seja M_0 a árvore geradora mínima de $(G[R], \dot{c}_0)$;
3 $i := 1$;
4 $W_0 := \emptyset$;
5 **Enquanto** $c_{i-1}(M_{i-1}) > 0$ **faça**
6 Seja $\tau_i \subseteq R$ com $2 < |\tau_i| < k$ e com $g_r(M_{i-1}, \tau)$ mínimo;
7 Seja T_i uma árvore k-restrita mínima de (G, c, τ_i);
8 $W_i := W_{i-1} \cup VS(T_i)$;
9 Seja D_i um conjunto remoção de τ_i em M_{i-1};
10 Seja p a função projeção de D_i em τ_i, relativa a M_{i-1};
11 $A_i := \emptyset$;
12 $c_i := c_{i-1}$;
13 **Para** cada e em D_i **faça**
14 $\dot{c}_i(p(e)) := 0$;
15 $A_i := A_i \cup \{p(e)\}$;
16 $M_i := M_{i-1} - D_i + A_i$;
17 $i := i + 1$;
18 $f := i - 1$;
19 Seja T uma árvore geradora mínima de $(G[R \cup W_f], c)$;
20 **Devolva** T;

[7] ZELIKOVSKY A. Z., *Better approximation algorithm for the network and Euclidean Steiner tree problem*. In: Idem

IMPLEMENTAÇÕES

Nesta seção serão comentados o processo de criação dos programas e os detalhes das implementações feitas na disciplina.

METODOLOGIA E DESENVOLVIMENTO

No início, as atividades consistiam no estudo da teoria envolvida no projeto, quase exclusivamente baseada da dissertação de Gondo[1] e alguns detalhes retirados do livro de Hochbaum[3]. Quando as dúvidas se acumulavam, eram agendadas reuniões com a orientadora.

Enquanto os conceitos iam se assentando, as funções mais simples eram implementadas, pois muito código precisaria ser escrito antes que os algoritmos do projeto começassem a rodar. Nesta fase foram implementados algoritmos de enumeração[4], de geração de grafos, de busca (4 diferentes), de Dijkstra, e de Kruskal[3], com cuidado para escrever o código mais eficiente possível. A partir daí foram estudados os manuais do SGB[2] e algumas funções foram substituídas por similares pertencentes ao pacote. Até então a produtividade era baixa.

A partir daí, foi necessário um aprofundamento nos estudos sobre os algoritmos e começou a ser implementado o conjunto principal de funções do algoritmo das *árvores 3-restritas* sem muita preocupação com eficiência do código. O objetivo era fazer o algoritmo funcionar para uma única tripla e posteriormente usar a função de enumeração para fazer o algoritmo rodar para todas as triplas. Esta fase foi a mais longa e foi concluída depois de os conceitos terem sido bem absorvidos. Nesta fase houve muitas mudanças drásticas no código, pois os *utility fields* do grafo não eram suficientes e a produtividade melhorou muito.

Uma vez com o algoritmo rodando, foi necessário criar diversas instâncias de teste e gerar instâncias interessantes para testes foi uma tarefa mais complexa do que se imaginava. Quando os exemplos criados pelo gerador eram usados como teste nos programas, as árvores de Steiner produzidas por eles tinham apenas um vértice de Steiner. Esses grafos tinham que ser pequenos para que fosse possível visualizar se o comportamento do algoritmo era o esperado. O fato de ser um algoritmo de aproximação, que não gera uma solução ótima necessariamente, tornou essa tarefa mais difícil. Nesta fase foram necessárias algumas reuniões com a orientadora.

Com o primeiro algoritmo funcionando, não foi necessário muito trabalho para ter uma versão do algoritmo do *ganho relativo* (k = 3) funcionando, tendo em vista que ambos algoritmos compartilham muito código com poucas diferenças. Curiosamente este último algoritmo é mais parecido com o algoritmo original de Zelikovsky para as *árvores 3-restritas* que o algoritmo das *árvores 3-restritas* descrito na dissertação de Gondo[1].

O próximo passo seria generalizar o *ganho relativo* para *k* arbitrário. Esta tarefa também foi dura, pois para cada subconjunto τ, $|\tau| > 2$, de R enumerado, era necessário enumerar todos subconjuntos $\sigma, 1 \leq |\sigma| \leq |\tau| - 2$ de elementos de $V_G \setminus R$. Trabalhar com duas funções de enumeração concatenadas complica muito a passagem de parâmetros para as funções que devem rodar dentro das enumerações e a obtenção de seus resultados fora delas. Esta parte do trabalho foi feita tendo em mente os resultados de testes obtidos pelo algoritmo, e não a eficiência do código então há muito espaço para otimizações no código deste algoritmo.

Por fim, foi implementado o algoritmo original de Zelikovsky para as *árvores 3-restritas* aproveitando suas semelhanças com o *ganho relativo* (k=3) para comparar os desempenhos de ambos e chegar a conclusões sobre as diferenças práticas entre as funções *ganho* e *ganho relativo*.

Executáveis

Todos os arquivos criados durante a disciplina, incluindo Makefiles podem ser encontrados no diretório "arquivos". Aqui segue uma lista com descrição sucinta

- **steiner**: recebe um arquivo salvo pelo SGB contendo um grafo e os vértices terminais e roda o algoritmo das *árvores 3-restritas*, das árvores 3-restritas original de Zelikovsky, do *ganho relativo* para k=3 e para k=4 e como saída, ele imprime as árvores de steiner obtidas com o algoritmo ou as salva, respectivamente, em arquivos com extensões ".a3r", ".a3rz", ".gr3" e ".grK", onde K define o limite máximo de terminais de um componente cheio. Atualmente K=4, pois o algoritmo já é demasiado lento para instâncias médias. (O valor da constante K pode ser modificado no arquivo "steiner.c")
- **a3rz**: algoritmo das árvores 3-restritas original de Zelikovsky. Funciona como o anterior, mas executa apenas um algoritmo.
- **gr3**: algoritmo do *ganho relativo* (k=3). Funciona como o anterior, mas executa apenas um algoritmo.
- **cria_grafo**: é um gerador de exemplos para teste utilizando os módulos geradores de grafos do SGB e criação de arquivos.
- **converte_stp**: um conversor de arquivos com extensão ".stp", encontrados nas bibliotecas de testes do sítio http://elib.zib.de/steinlib para o formato padrão do SGB.

Obs.: todo arquivo criado por esses programas segue o padrão de arquivos do SGB, portanto podem ser utilizados em qualquer programa que leia esses arquivos.

Obs. 2: cada executável é gerado por apenas um arquivo ".c", pois desde o início dos testes ficou claro que o otimizador de código do GCC mostrava melhores resultados quando lidava com um único arquivo.

Testes Preliminares

Em testes para verificar o consumo de tempo e descobrir um valor razoável para fazer os testes, notou-se que o código precisava ser otimizado, pois para instâncias com $|V_G| = 100$, o algoritmo do *ganho relativo* levava horas e quando $|V_G| = 500$, mesmo os algoritmos mais rápidos já implementado naquele instante: *árvores 3-restritas* e *ganho relativo* (k=3) levavam horas. Por isso foi necessário voltar ao desenvolvimento para otimizar os códigos.

Otimizações

A primeira otimização foi feita em ambos algoritmos do *ganho relativo* (k=3). Ela consistia de eliminar elementos que não apresentam ganho efetivo (isto é, $g_r(M, \tau) < 1$) nas enumerações, diminuindo assim o domínio dos conjuntos para as posteriores.

As últimas otimizações no código foram feitas por sugestão da orientadora. Elas visavam diminuir o consumo de tempo assintótico, melhorando a qualidade do código escrito, diferente da primeira. Foram implementadas apenas para o *ganho relativo* (k=3) e posteriormente para o algoritmo das *árvores 3-restritas original de Zelikovsky*. As instâncias que levavam horas agora levam minutos.

Consumo de Tempo Assintótico

Como algoritmos de aproximação, o consumo de tempo assintótico de cada implementação dos algoritmos é polinomial. As versões atuais têm as seguintes complexidades: (Notação: $n := |V_G|$, $m := |E_G|$)

Algoritmo	**Consumo de Tempo**
Árvores 3-Restritas	$O(n^6)$
Árvores 3-Restritas Original de Zelikovsky	$O(n^5)$
Ganho Relativo (k=3)	$O(n^5)$
Ganho Relativo*	$O(n^{2k+1})$

* Obs.: k define o número máximo de terminais em uma componente cheia antes da chamada a Kruskal.

ÁRVORES 3-RESTRITAS

O consumo de tempo para cada vértice de steiner adicionado à solução é dado pela função de enumeração das triplas τ de R, que consome $O(n^3)$ e pela criação da árvore *3-restrita mínima* T_τ de (G, c, τ), que consome $O(m)$, mas m é no nosso caso $O(n^2)$. Como o número de vértices de Steiner é $O(n)$, o consumo total é $O(n^6)$

A criação da árvore *3-restrita mínima* T_τ de (G, c, τ) pode ser feita em $O(n)$. Esta sugestão foi dada pela orientadora para otimizar o código, porém não foi feita aqui porque este algoritmo produz árvores não melhores que a implementação do algoritmo original.

ÁRVORES 3-RESTRITAS ORIGINAL DE ZELIKOVSKY E GANHO RELATIVO (K=3)

O consumo de tempo para cada vértice de steiner adicionado à solução é dado pela função de enumeração das triplas τ de R, que consome $O(n^3)$ e pela definição do custo da árvore *3-restrita mínima* T_τ de (G, c, τ), que consome $O(n)$. Como o número de vértices de Steiner é $O(n)$, o consumo total é $O(n^5)$. Aqui a sugestão foi implementada.

GANHO RELATIVO

O consumo de tempo para cada conjunto *VS* adicionado à solução é dado pela função de enumeração dos subconjuntos τ de R, $|\tau| = k$, que consome $O(n^k)$, pela enumeração dos subconjuntos σ de $V_G \setminus R$, $1 \leq |\sigma| \leq |\tau| - 2$, que consome $O(n^{k-2})$ e pela indução de $G[\tau \cup \sigma]$, feita por uma função do SGB, que consome $O(n^2)$. Como o número de conjuntos de vértices de Steiner é $O(n)$, o consumo total é $O(n^{2k+1})$.

Podemos trocar a função de indução do SGB por uma específica que faça a indução em $O(n)$, fazendo com que o consumo caia para $O(n^{2k})$. Isto não foi feito por falta de tempo.

Note que, mesmo para $k = 3$, esta implementação consome muito mais tempo que a implementação específica para produzir a mesma árvore de Steiner.

TESTES E RESULTADOS

Foram feitos diversos testes com arquivos retirados da biblioteca de testes no sítio: http://elib.zib.de/steinlib. Estes testes foram escolhidos por terem o valor da árvore de Steiner mínima para cada instância. Foram compiladas tabelas com os resultados destes testes que estão disponíveis junto desta monografia, bem como as instâncias já convertidas para o nosso padrão. Elas podem ser obtidas no diretório “resultados”.

Os testes mostraram fatos curiosos:

- Embora gerando o mesmo conjunto de vértices de Steiner, os algoritmos das *árvores 3-restritas* e a sua versão *original* obtêm resultados muito diferentes. Isso se deve ao fato de que o algoritmo original executa uma chamada de *Kruskal* antes de devolver a árvore de Steiner. Esse passo pode fundir duas ou mais *componentes cheias* com três terminais em uma única *componente cheia* com $k>3$ terminais. Assim, o algoritmo original produz uma árvore de Steiner de custo menor ou igual ao custo de uma árvore gerada pelo algoritmo das *árvores 3-restritas*.

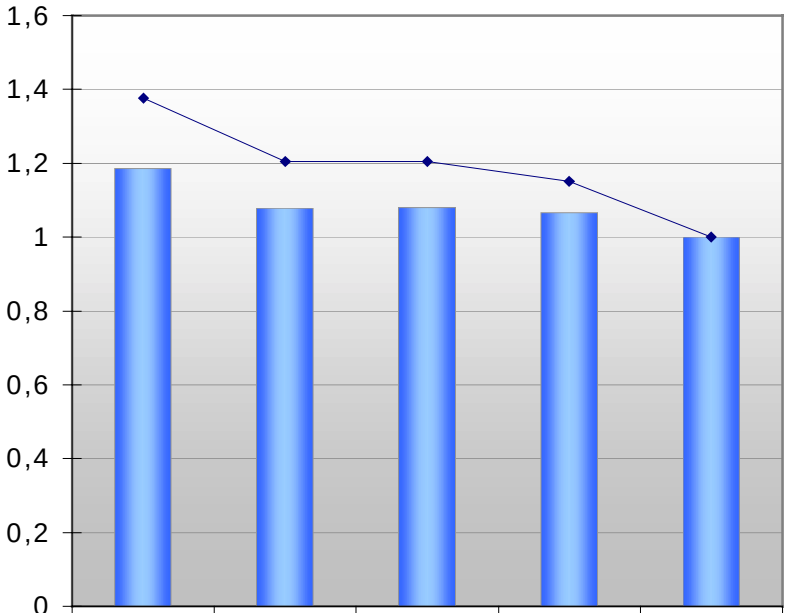
- Seria esperado que a função *ganho* tivesse um desempenho médio melhor que a função *ganho relativo* para $k = 3$, visto que o *ganho relativo* é uma adaptação para comparar árvores k-restritas para diferentes valores de k, porém os desempenhos foram equivalentes.
- Ao contrário do que se imaginava, nem sempre o algoritmo do *ganho relativo* com $k=4$ encontra uma árvore de Steiner de custo menor ou igual a uma árvore gerada pelo algoritmo do *ganho relativo* ($k=3$). Isto também se deve à chamada de Kruskal em ambos algoritmos e a uma imperfeição no conceito da função *ganho relativo*. Esta imperfeição faz com que o *ganho relativo* olhe para apenas uma *componente cheia* com quatro terminais, por exemplo, de *ganho relativo* mínimo e não consiga enxergar que duas outras *componentes cheias* com três terminais cada, ambas com *ganho relativo* maior (menos bom), possam reduzir mais o custo da árvore de Steiner no fim da execução, assim ele escolhe a árvore 4-*restrita* e estraga as duas 3-*restritas* que, produziriam uma árvore de custo inferior. Note que, se $k=6$, essa imperfeição do *ganho relativo* é minimizada.

Gráfico Comparativo

Segue aqui um gráfico do desempenho médio dos algoritmos. Ele foi feito baseado nos resultados das 100 instâncias do conjunto *I080* de testes da biblioteca de testes, que é um conjunto de grafos aleatórios com 80 vértices. Os dados de cada instância podem ser vistos no diretório de testes.

Os algoritmos aqui são tratados por siglas: árvores 3-restritas (**A3R**), árvores 3-restritas original de Zelikovsky (**A3RZ**), ganho relativo (**GRK**) e um algoritmo fictício ótimo (**Opt**), que produz árvores de custo mínimo.

O *custo relativo* é definido como o custo da árvore de Steiner produzida pelo algoritmo dividido pelo custo da árvore de Steiner ótima de cada instância. O *custo relativo médio* é a média aritmética de todas as instâncias e o *custo relativo máximo* é o máximo entre todas as instâncias.



	A3R	A3RZ	GR3	GR4	Opt
Médio	1,186137	1,078709	1,08042	1,065459	1
Máximo	1,376147	1,204867	1,204867	1,151716	1

CONCLUSÃO

CUSTO DAS ÁRVORES

Os custos das árvores produzidas pelos algoritmos estão muito próximos dos custos ótimos, especialmente os algoritmos do ganho relativo. Isso mostra que na prática eles são muito bons e cada um tem uma propriedade interessante:

- Os algoritmos de aproximação baseados em árvores k-restritas realmente produzem, na média, árvores de Steiner, muito próximas da árvore de Steiner de custo mínimo.
- A chamada de Kruskal no fim da execução dos algoritmos melhora muito o custo das árvores de Steiner produzidas por eles, em geral bem próximos do ótimo.
- O algoritmo do *ganho relativo* para *k* arbitrário, no pior caso, não produz uma árvore muito ruim. Isso sugere que à medida que *k* cresce, o algoritmo produz, no pior caso, uma árvore de Steiner mais parecida com uma árvore de Steiner de custo mínimo. Isto é, o algoritmo tem mais recursos para conseguir adicionar um vértice de Steiner.
- Nem sempre o algoritmo do *ganho relativo* para *k* arbitrário, k > 3, produz uma árvore de Steiner de custo menor ou igual ao custo de uma árvore produzida por um algoritmo *ganho relativo* para *k*-1.

CONSUMO DE TEMPO

Os algoritmos das *árvores 3-restritas original de Zelikovsky* e do *ganho relativo* (k=3) têm baixo consumo de tempo e podem ser usados em instâncias grandes, $|V_G| = 500$. Porém o algoritmo do *ganho relativo* tem um consumo de tempo muito alto, ainda que polinomial e acabou por impedir testes em instâncias maiores, com $|V_G| > 100$.

BIBLIOGRAFIA

[1]. GONDO, E. K. *Árvores k-restritas e aproximações para o Problema de Steiner em Grafos*. São Paulo, 2002. Dissertação (Mestrado) – Instituto de Matemática e Estatística. Universidade de São Paulo.

[2]. KNUTH, D. E., *The Stanford GraphBase: a platform for Combinatorial Computing*, ACM Press, 1993.

[3]. HOCHBAUM, D. S., *Approximations Algorithms for NP-Hard Problems,* PWS Publishing (1997).

[4]. FEOFILOFF, P., *Notas de aula de MAC0122* (*Princípios de Desenvolvimento de Algoritmos*). Texto disponível na Internet: http://www.ime.usp.br/~pf/algoritmos/aulas/enum.html

PARTE II: O PROJETO E O BCC

DESAFIOS

Embora já tivesse feito um estágio, desejava me envolver em algum projeto mais complexo, que eu achasse digno de um projeto de conclusão do curso.

O que mais me interessou neste projeto quando a orientadora enviou a proposta para a lista de discussão da disciplina foi a natureza do problema e a área. Eu queria fazer algo na área de grafos, pois esta disciplina foi a que me acordou para o curso. Acredito que o ambiente conturbado daquele semestre contribuiu para isso. Até então eu não havia me interessado tanto por uma matéria.

O projeto apresentou diversos desafios. O principal foi implementa algoritmos de aproximação. Eu gosto de programar apenas depois que todos os passos da solução do problema estão claros para mim e compreender totalmente a solução não foi fácil, pois além de envolver vários grafos com propriedades diferentes simultaneamente, era complicado decidir se a árvore encontrada pelo algoritmo era a árvore esperada quando não era a ótima.

A única experiência prévia com implementação de algoritmos de aproximação foi em Estruturas de Dados, quando implementei um algoritmo de aproximação probabilístico com complexidade de tempo $O(n^2)$ para o problema do Caixeiro Viajante (TSP), pelo qual recebi um prêmio do professor por ter conseguido encontrar o menor caminho entre os alunos da disciplina.

Outro problema foi a falta de prática com programas que rodem por horas a fio. Essa experiência de lidar com problemas grandes não havia sido enfrentada durante todo o curso. Embora conhecesse os conceitos de análise de algoritmos, nunca tinha sentido na pele a diferença prática de reduzir a complexidade de tempo de um programa e sentir a diferença que faz um fator de n.

FRUSTRAÇÕES

Como frustração fica o fato de não ter implementado, da forma mais eficiente, todos os algoritmos da dissertação e de não ter conseguido rodar muitos exemplos de testes para os já feitos (por causa do tempo que levam) para que a comparação pudesse ser feita. Também fica a frustração de não ter passado o código fonte dos programas para CWEB.

DISCIPLINAS MAIS IMPORTANTES

As disciplinas mais importantes para o projeto foram:

- Grafos
- Otimização Combinatória
- Introdução a Autômatos Finitos
- Conceitos Fundamentais de Linguagens

As duas primeiras disciplinas forneceram os conhecimentos básicos sobre grafos, problemas de otimização. Autômatos foi importante por ensinar um método diferente de pensar ao programar em grafos. Ela foi fundamental para a implementação da função de enumeração. Já Conceitos Fundamentais foi importante por causa de um assunto: recursão de cauda, que transforma uma recursão num laço. Também foi utilizada na enumeração.

INTERAÇÃO COM A ORIENTADORA

A interação com a orientadora foi muito boa. Embora não tivessem sido marcadas muitas reuniões, todas foram muito proveitosas. Eu levava uma quantidade razoável de dúvidas,

que eram esclarecidas parcial ou completamente durante as reuniões e as que restavam eram esclarecidas com mais uma lida no material bibliográfico.

APRIMORAMENTO DOS CONHECIMENTOS

Para continuar atuando nesta área, o único passo possível para o aprimoramento dos conhecimentos técnicos e científicos seria um mestrado.

Embora tenha gostado muito do projeto que desenvolvi durante este ano, especialmente nas etapas finais, quando podia ver o resultado de tanto esforço em ação, ainda estou em dúvida se isto é o que quero no momento.

CONSIDERAÇÕES FINAIS

Dentre as implementações que apresentamos, todas tiveram como foco principal a o custo da árvore de Steiner produzido por elas para completar o estudo dos algoritmos iniciado por Gondo[1]. Apenas as versões do algoritmo das *árvores 3-restritas original de Zelikovsky* e do *ganho relativo* (k=3) tiveram como foco, alem deste, a otimização do consumo de tempo.

Desta forma foi possível analisar seus comportamentos e compará-los na prática, como Gondo[1] desejava no final de sua dissertação. O algoritmo que se destacou mais, entre os implementados, em ambos aspectos foi o do *ganho relativo* (k=3).