Instâncias para o Problema de Steiner em Grafos

Isabel Rosseti
Marcus Poggi de Aragão
Celso C. Ribeiro
Eduardo Uchoa
Renato F. Werneck
{*rosseti, poggi, celso, uchoa, rwerneck*}*@inf.puc-rio.br*

PUC-RioInf.MCC47/00 Dezembro, 2000

Abstract
In this work, 33 Steiner problems in graphs's instances are proposed. They are based on otimization problems as set covering, code covering and Steiner triples systems. These instances are interesting because, in most of them, a good gap between linear relaxation and primal solution was obtained. Moreover, for some of these instances, the optimal solution was easily calculated.

Keywords
Steiner Problems's Instances; Steiner Problems in Graphs; Set Covering Problems; Code Covering Problems; Steiner Triples Systems.

Resumo
Nesse trabalho, são propostas 33 instâncias para o problema de Steiner em grafos. Elas são baseadas em outros problemas de otimização combinatória, a saber: problema de recobrimento de código, problema de recobrimento de conjuntos e sistemas de triplas de Steiner. Essas instâncias são interessantes porque, na grande maioria delas, foi obtido um bom *gap* entre a solução primal e a relaxação linear. Além disso, para algumas delas, foi possível calcular o valor da solução ótima de uma maneira bastante simples.

Palavras-chave
Instâncias para o Problema de Steiner em Grafos; Problema de Steiner em Grafos; Problema de Recobrimento de Conjuntos; Problema de Recobrimento de Código; Sistema de Triplas de Steiner.

1 Introdução

Seja $G = (V_G, E_G)$ um grafo não-direcionado, V_G o conjunto de nós do grafo G, E_G o conjunto de arestas desse grafo, $W : E_G \rightarrow \mathcal{Z}^+$ uma função que atribui um peso a cada aresta do conjunto E_G e $K \subseteq V_G$ um subconjunto de nós, ditos *terminais*. O Problema de Steiner em Grafos (*Steiner Problem in Graphs* - SPG) consiste em encontrar uma sub-árvore $T = (V_T, E_T)$ de G, onde $K \subseteq V_T$, em que a somatória de todos os pesos das arestas contidas no conjunto E_T seja mínima. Esse clássico problema, cuja origem remonta a Fermat no século XVII, foi um dos primeiros a serem provados NP-difíceis [Karp72]. O SPG vem sendo intensamente estudado devido à sua aplicação em uma grande variedade de situações práticas, tais como projeto de redes, desenho de circuitos integrados, biologia computacional, entre outros. Citar todas as centenas de artigos relevantes sobre o SPG está fora do escopo desse artigo, mas bons *surveys* podem ser encontrados em [Macu87, HRW92, Wint87, DSR00].

Artigos que propõem novos algoritmos para o SPG geralmente contêm uma seção de experimentos computacionais, descrevendo os resultados obtidos em algum conjunto de instâncias. Uma exceção são os artigos sobre algoritmos aproximados, nesse caso apenas a prova da aproximação costuma ser fornecida. Até o final dos anos 80, diferentes autores usavam diferentes conjuntos de instâncias para as suas experiências, o que tornava a comparação entre algoritmos bastante difícil. Em 1990, J. Beasley criou a OR-Library [Beas90], um repositório contendo conjuntos de instâncias para vários problemas de otimização clássicos, incluindo o SPG, que podiam ser facilmente acessados através da Internet. Essa iniciativa representou uma pequena revolução na pesquisa sobre o SPG, pois tornou possível acompanhar o progresso dos algoritmos através dos resultados obtidos nessas instâncias. Ao longo dos anos 90, outros conjuntos de instâncias de teste propostos foram usados em artigos de diversos autores e hoje também são reconhecidos (informalmente) como conjuntos de teste padrão. Os mais utilizados são as instâncias de incidência de [Duin93] e as instâncias VLSI de [KM98]. Todos esses conjuntos podem ser buscados eletronicamente na SteinLib [KMV00].

Entretanto, o próprio sucesso da pesquisa sobre o SPG tornou os conjuntos de instâncias de teste atualmente existentes inadequados à tarefa de comparar os novos algoritmos que surgirão ao longo da década de 2000. O ponto é que os algoritmos de estado-da-arte para o SPG já são capazes de resolver à otimalidade quase todas as instâncias da OR-Library em poucos segundos [BL98, KM98, PV00, PUW00]; a única instância que permanece relativamente difícil já pode ser resolvida em cerca de 1 minuto [PV00]. A situação não é muito diferente para os demais conjuntos de instâncias de teste. Todas as instâncias VLSI [PUW00] e quase todas as instâncias de incidência já foram resolvidas [KM98, PUW00], a maioria delas em menos de 1 minuto.

Os melhores algoritmos exatos para o SPG são baseados nas chamadas "formulações fortes". Pelo menos três delas são conhecidas há bastante tempo: a formulação por multifluxo, a formulação por cortes direcionados e a formulação por eliminação de ciclos (ver [GM93] ou [MW95] para uma discussão a esse respeito). Os valores dados pela relaxação linear dessas formulações são sempre iguais, e na grande maioria das atuais instâncias de teste estão bastante próximos, ou até igualam, o valor do ótimo inteiro. Isso faz com que resolver essas instâncias seja praticamente um problema polinomial.

O objetivo desse trabalho é propor novas instâncias de teste para o SPG, relativamente pequenas porém difíceis de serem resolvidas pelas técnicas conhecidas. Procuramos fazer com que existam diferenças significativas entre as relaxações lineares e os ótimos inteiros. É importante salientar que uma diferença de alguns pontos percentuais já é o suficiente para tornar uma instância difícil (não é conhecido nenhum exemplo de instância para o qual essa diferença supere 14%). Espera-se que o desafio de resolver essas instâncias estimule novos desenvolvimentos tanto em algoritmos exatos quanto em heurísticas para o SPG.

As instâncias foram inspiradas em outros problemas de otimização combinatória, tais como Recobrimento de Código [HL74, PS99], Recobrimento de Conjunto [CFT98] (incluindo as conhecidas instâncias baseadas nos Sistemas de Triplas de Steiner [FNT74]).

As instâncias SPG baseadas em recobrimento de código usam um esquema de atribuição de custos as arestas semelhante a de [Duin93] e utilizam um algoritmo heurístico para determinar a disposição dos nós terminais no grafo resultante.

Os problemas SPG criados a partir de problemas de recobrimento de conjuntos podem ser divididos em 2 classes distintas, a saber: instâncias SPG onde não são conhecidos os valores de suas soluções ótimas e instâncias que usam os valores das soluções ótimas dos problemas de recobrimento para saber, previamente, quais os valores das soluções ótimas dos problemas de Steiner. Na primeira classe, o grafo SPG resultante é bipartido e todas as arestas possuem o mesmo custo. Já na segunda classe, os custos das arestas do grafo de Steiner foram baseados nos custos de cobertura das colunas do problema de recobrimento original. Com essa estratégia de atribuição, foi possível usar o valor da solução ótima do problema de recobrimento para calcular o valor da solução ótima para o SPG.

Uma outra classe de instância SPG que foi gerada usa recobrimento de código e sistemas de triplas de Steiner para criar problemas de Steiner via uma transformação intermediária para o problema de recobrimento de conjuntos. Nessa classe, quando a solução ótima do problema de recobrimento de conjuntos gerado é conhecida, também se sabe, previamente, o valor da solução ótima do SPG criado na fase final.

Esse trabalho é organizado como segue. Na seção 2, as instâncias baseadas no problema de recobrimento de código são mostradas. Na seção 3, são apresentadas instâncias a partir do problema de recobrimento de conjuntos. Na seção 4, são discutidas a solução primal e a relaxação linear. Na seção 5, são mostrados os resultados computacionais dessas instâncias. E, finalmente, na seção 6, são apresentadas as considerações finais a respeito dessas novas instâncias.

2 Instâncias Baseadas no Recobrimento de Código

Seja V_q^n o conjunto de todos os vetores n-dimensionais cujos componentes são números inteiros z, tal que $0 \leq z \leq q-1$. A distância de Hamming entre dois vetores x, $y \in V_q^n$, denotada por $d(x,y)$, é o número de componentes na qual x e y são diferentes. Qualquer subconjunto de V_q^n é chamado de código. Um código C é chamado de r-recobrimento de V_q^n se, $\forall y \in V_q^n$, existe um vetor $x \in C$, tal que $d(x,y) \leq r$. O Problema do Recobrimento de Código (*Code Covering Problem* - CCP) consiste em encontrar um recobrimento de código de tamanho mínimo. O CCP é um problema NP-difícil [HL74] e tem sido bastante estudado na literatura.

Uma instância do CCP corresponde a um grafo em que os nós são os vetores de V_q^n e as arestas ligam vetores com distâncias iguais ou menores a r. Esse grafo é *regular*, todos os nós têm o mesmo número de vizinhos N_r. Os arquivos descrevendo esses grafos, em que os nós são numerados de 1 até $|V_q^n|$, têm o seguinte formato: (a) a primeira linha informa $|V_q^n|$ e N_r e (b) as próximas $|V_q^n|$ linhas informam para cada nó qual são os seus vizinhos. A figura 2 mostra o arquivo correspondente ao grafo do CCP $n = 2$, $q = 3$ e $r = 1$.

Geramos instâncias de Steiner sobre grafos de CCP da seguinte maneira: usamos o algoritmo de busca tabu proposto por [PS99] para obter uma solução aproximada para o CCP dado. Os vetores nessa solução são os nós terminais do SPG, os demais vetores de V_q^n são não-terminais. Se a aresta une dois não-terminais, seu custo é 1. Se a aresta é adjacente a exatamente um terminal, seu custo é 2. As arestas entre dois terminais recebem custo 3.

Essa maneira de se atribuir custos as arestas de acordo com os tipos de nós adjacentes é a mesma usada nas instâncias de incidência de [Duin93], mas nesses casos os grafos são gerados aleatoriamente. No caso das instâncias baseadas no CCP, o alto grau de simetria do grafo combinado com uma disposição não tão simétrica dos terminais parece tornar as instâncias bem mais difíceis.

```
E_G ← ∅
V_G ← V_q^n
K ← Busca_Tabu(V_q^n)
Para cada elemento i do conjunto V_q^n faça
    Para j ← 1 até N_r faça
        Leia o elemento k
        Se k é terminal e i é terminal então faça W_{i,k} ← 3
            Se k é não terminal e i é não terminal então faça W_{i,k} ← 1
                Senão faça W_{i,k} ← 2
        E_G ← E_G ∪ {W_{i,k}}
```

Figura 1: Algoritmo de geração das instâncias baseadas no recobrimento de código.

Exemplo: dado o arquivo CCP abaixo:

```
9 4
1 2 3 4 7
2 1 3 5 8
3 1 2 6 9
4 1 5 6 7
5 2 4 6 8
6 3 4 5 9
7 1 4 8 9
8 2 5 7 9
9 3 6 7 8
```

Figura 2: Exemplo de CCP.

Se a busca tabu escolher como solução os vetores 2, 3, e 7, o SPG gerado a partir desse CCP seria:

É importante salientar que não é possível saber *a priori* o valor da solução ótima das instâncias baseadas em CCP.

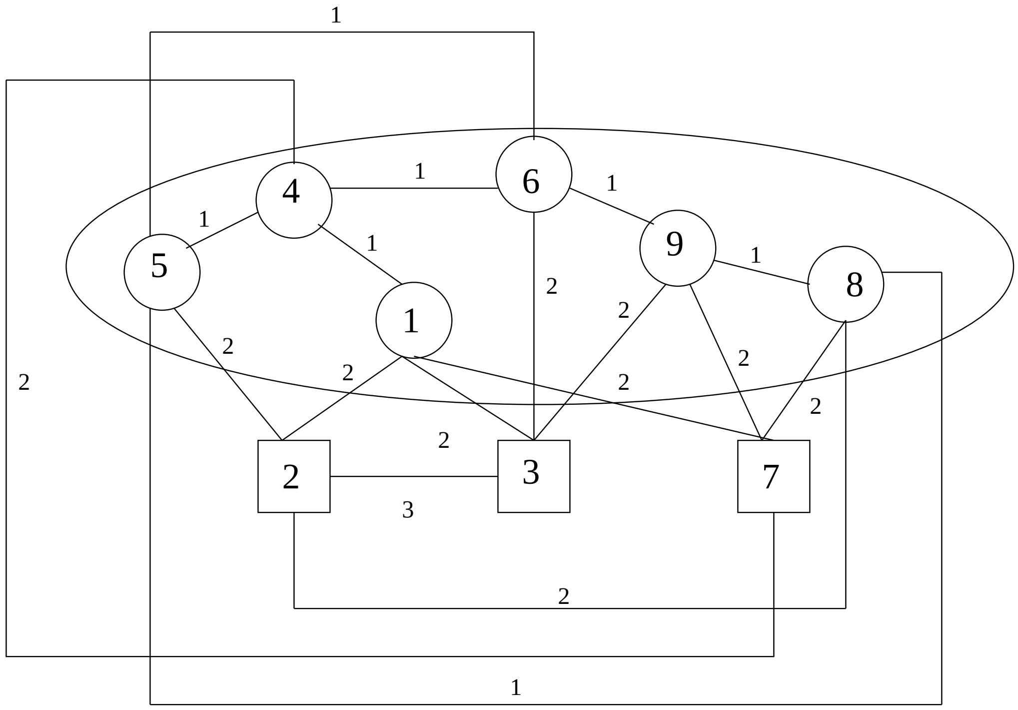

Figura 3: Exemplo de instância SPG gerada a partir de um CCP.

3 Instâncias Baseadas no Recobrimento de Conjuntos

O Problema do Recobrimento de Conjuntos (*Set Covering Problem* - SCP) consiste em minimizar os custos de cobrir, através da escolha de um subconjunto de colunas, todas as linhas de uma matriz 0-1 de m linhas e n colunas (essa matriz é conhecida, para esse problema, como matriz de incidência).

Seja A a matriz de incidência, C o vetor de custos ordenado de maneira crescente e X o vetor solução. Seja $x_j = 1$ se a coluna j pertencer a solução. Caso contrário, $x_j = 0$. O SCP corresponde ao seguinte programa linear inteiro:

$$minimize \sum_{j=1}^{n} c_j x_j \qquad \{1\}$$

sujeito a

$$\sum_{j=1}^{n} a_{ij} x_j \geq 1,\ i = 1, ..., m \qquad \{2\}$$

$$x_j \in \{0, 1\},\ j = 1, ..., n \qquad \{3\}$$

O SCP é amplamente aplicado em recuperação da informação, balanceamento de linhas de montagem, problemas de roteamento, lista de espera de linhas aéreas, entre outros. O SCP é NP-completo, mas diversos algoritmos exatos e heurísticos têm sido aplicados a ele, ver [CFT98] para um *survey* recente.

As instâncias de teste mais usuais para o SCP são as da OR-LIB [Beas90], divididas nas séries 4, 5, 6, A, B, C, D, E, F, G e H. Na maioria dessas instâncias o valor da solução ótima já é conhecido. Essas instâncias são descritas em arquivos com o seguinte formato: (a) o número de linhas, m, (b) o número de colunas, n, (c) o custo de cada coluna C_j, onde $j = 1, 2, ...n$ e, para cada linha i, $i = 1, 2, ..., m$, (d) o número de colunas que recobre a linha i, $ncol_i$, e a lista de colunas que recobre essa linha. Um exemplo desse formato é mostrado na figura 5.

Foram criados dois tipos de instâncias do SPG baseadas em instâncias do SCP: instâncias em que o valor da solução ótima não é conhecido previamente e instâncias que usam o valor da solução ótima do SCP para determinar o valor da sua solução ótima. Nas próximas subseções, cada uma delas é discutida.

3.1 Instâncias sem Conhecimento da Solução Ótima

Essa classe de instâncias é formada por problemas onde não é possível determinar, através da solução ótima do SCP original, os valores das suas soluções ótimas.

O algoritmo que gera esse tipo de instância funciona da seguinte maneira: inicialmente, transformam-se todas as colunas do SCP em nós não terminais e todas as linhas do SCP em nós terminais. Para cada linha do SCP, descobre-se o número de colunas que cobre essa linha. Logo a seguir, criam-se arestas entre cada linha e as colunas que a recobrem. E, finalmente, esse algoritmo faz com que todas as arestas desse SPG passem a ter custo 1.

$V_G \leftarrow \emptyset$
$E_G \leftarrow \emptyset$
$K \leftarrow \emptyset$
Para $i \leftarrow 1$ até n faça
    $V_G \leftarrow V_G \cup \{i\}$
Para $i \leftarrow n+1$ até $m+n$ faça
    $V_G \leftarrow V_G \cup \{i\}$
    $K \leftarrow K \cup \{i\}$
Para $i \leftarrow 1$ até m faça
    $z \leftarrow n+i$
    Para $j \leftarrow 1$ até $ncol_i$ faça
        Leia a coluna k
        $W_{z,k} \leftarrow 1$
        $E_G \leftarrow E_G \cup \{W_{z,k}\}$

Figura 4: Algoritmo de geração das instâncias baseadas no recobrimento de conjuntos sem o conhecimento da solução ótima.

Exemplo: dado o arquivo SCP abaixo:

```
(1) 3 4
(2) 2 3 4 5
(3) 2
(4) 1 3
(5) 2
(6) 1 4
(7) 3
(8) 2 3 4
```

Figura 5: Exemplo de SCP.

A linha (1) equivale ao item (a) e (b) da seção 3. O item (c) da seção 3 está representado na linha (2). Já as linhas (3) a (8) representam o item (d) dessa instância. O SPG gerado a partir desse SCP seria:

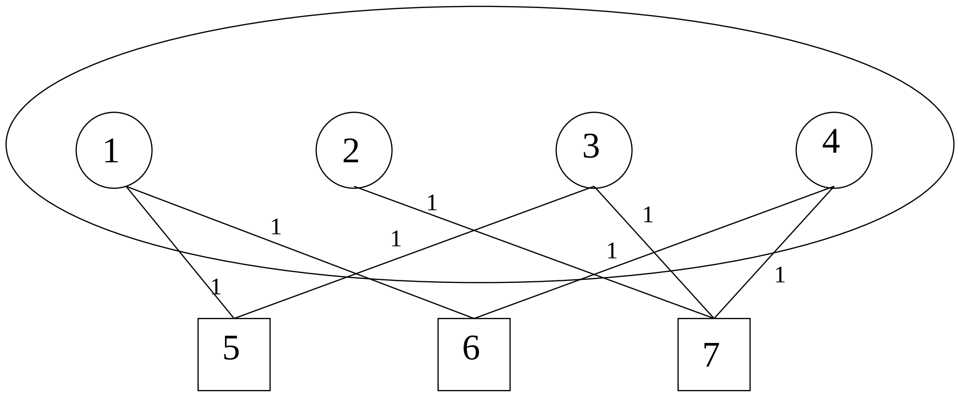

Figura 6: Exemplo de instância SPG gerada a partir de um SCP.

3.2 Instâncias com Conhecimento da Solução Ótima

O algoritmo que gera esse tipo de instância funciona da seguinte maneira: inicialmente, transformam-se todas as colunas do SCP em nós não terminais e todas as linhas do SCP em nós terminais. Cria-se um nó terminal, r, e criam-se arestas desse nó para cada nó não terminal. Essas arestas passam a ter seus custos iguais ao custos do uso das colunas do SCP original. Para cada linha do SCP, descobre-se o número de colunas que cobre essa linha. Logo a seguir, criam-se arestas entre cada linha e as colunas que a recobrem. E, finalmente, esse algoritmo faz com que todas as arestas desse SPG passem a ter custo $maior\_custo\_SCP + 1$.

$V_G \leftarrow \emptyset$
$E_G \leftarrow \emptyset$
$K \leftarrow \emptyset$
Para $i \leftarrow 1$ até n faça
    $V_G \leftarrow V_G \cup \{i\}$
Para $i \leftarrow n+1$ até $m+n+1$ faça
    $V_G \leftarrow V_G \cup \{i\}$
    $K \leftarrow K \cup \{i\}$
Para $i \leftarrow 1$ até n faça
    $W_{(n+1),i} \leftarrow custo\_SCP_i$
    $E_G \leftarrow E_G \cup \{W_{(n+1),i}\}$
Para $i \leftarrow 2$ até $m+1$ faça
    $z \leftarrow n+i$
    Para $j \leftarrow 1$ até $ncol_i$ faça
        Leia a coluna k
        $W_{z,k} \leftarrow (maior\_custo\_SCP + 1)$
        $E_G \leftarrow E_G \cup \{W_{z,k}\}$

Figura 7: Algoritmo de geração das instâncias baseadas no recobrimento de conjuntos sem o conhecimento da solução ótima.

Nessas instâncias é possível calcular o valor ótimo da solução através da fórmula:

$$custo\_SCP + ((total\_terminais - 1) \times (maior\_custo\_SCP + 1)) \qquad \{4\}$$

Demonstração: a solução ótima do SCP da subseção 3.1 é 5. O SPG formado a partir desse exemplo seria:

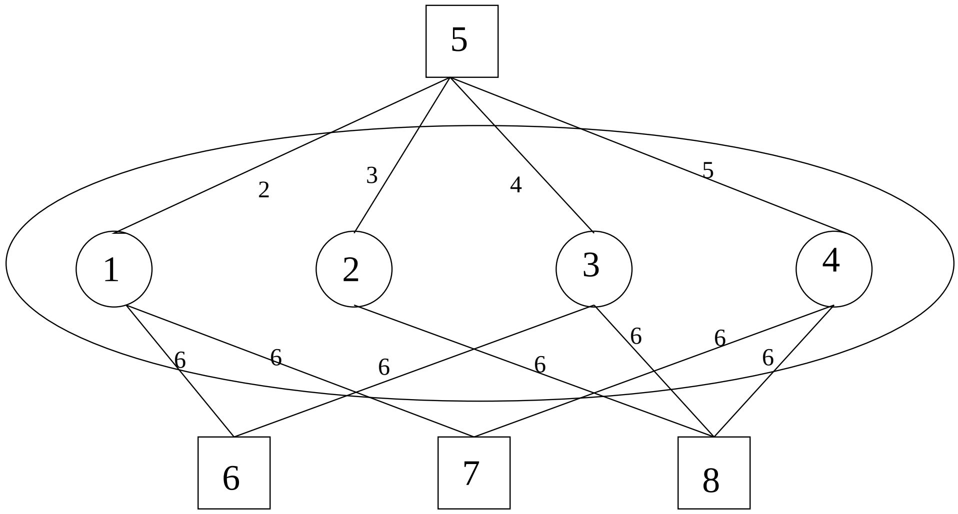

Figura 8: Exemplo de instância SPG gerada a partir de um SCP.

Pela figura 8 pode-se observar que a solução ótima da instância SPG criada é a seguinte:

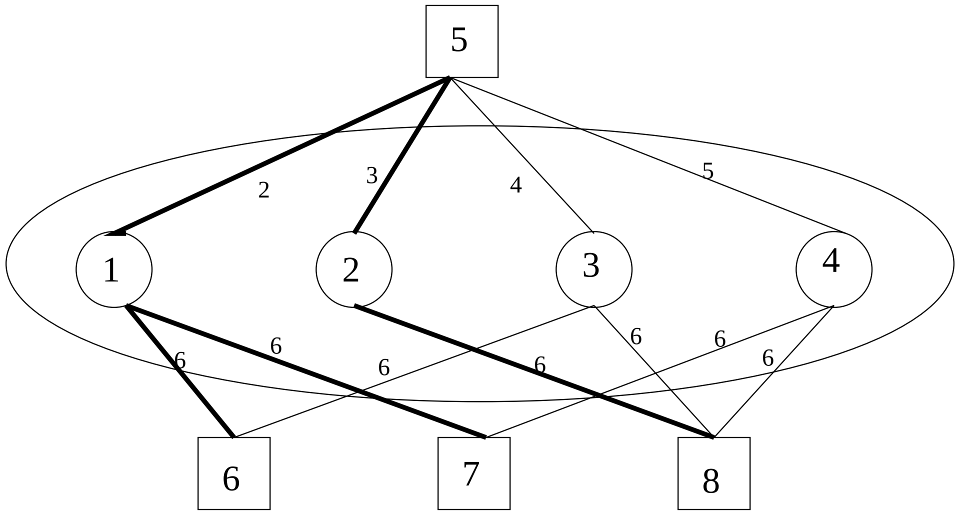

Figura 9: Solução ótima da figura 8.

Observe que, pela figura 9, o valor da solução ótima é 23. Aplicando a fórmula {4}, o mesmo valor é obtido. Portanto, essa fórmula pode ser usada para calcular o valor da solução ótima quando a instância for criada usando as regras descritas nessa subseção.

3.3 Obtendo Instâncias com o Conhecimento da Solução Ótima via SCP

Essa classe de instâncias SPG é formada por instâncias vindas de outros problemas de otimização combinatória, tais como recobrimento de código e sistema de triplas de Steiner, que geram problemas de Steiner via uma transformação prévia para o SCP. Nesses problemas, quando a solução ótima do SCP gerado na fase intermediária é conhecida, também se conhece o valor da solução ótima da instância SPG, porque foi usada a transformação de SCP para SPG discutida na subseção 3.2.

Nas próximas subseções, cada uma dessas instâncias é descrita.

3.3.1 Instâncias Baseadas no Recobrimento de Código via SCP

A transformação de uma instância de recobrimento de código para uma instância SCP foi feita pelo seguinte algoritmo: inicialmente, faça o tamanho do conjunto V_q^n, passar a ser o número de linhas e colunas do SCP. Além disso, assuma que o custo de cada coluna C_j, onde $j = 1, 2, ...|V_q^n|$ passe a ter o valor 1. Logo a seguir, assuma que o número de vetores em que $d(x, y) \leq r$ passe a ser o número de colunas que recobre a linha i, onde $i = 1, 2, ..., |V_q^n|$. E, finalmente, para cada linha i, faça a lista de vetores que distam r de i passar a ser a lista de colunas, L_i, que recobre essa linha.

```
m ← |V_q^n|
n ← |V_q^n|
Para j ← 1 até n faça
    C_j ← 1
Para i ← 1 até m faça
    L_i ← ∅
Para i ← 1 até m faça
    Para j ← 1 até N_r faça
        Leia o vetor k
        L_i ← L_i ∪ {k}
```

Figura 10: Algoritmo de geração das instâncias SCP baseadas em instâncias de recobrimento de conjuntos.

Como foi mencionado na subseção 3.3, aplica-se a transformação de um SCP para um SPG descrita na subseção 3.2.

Exemplo: dado o CCP abaixo:

```
(1) 4 3
(2) 1 2 3
(3) 1 2 4
(4) 1 3 4
(5) 2 3 4
```

Figura 11: Exemplo de um CCP.

o SCP formado seria:

```
(1) 4 4
(2) 1 1 1 1
(3) 3
(4) 1 2 3
(5) 3
(6) 1 2 4
(7) 3
(8) 1 3 4
(9) 3
(10) 2 3 4
```

Figura 12: Exemplo de um SCP a partir do CCP dado.

O SPG formado a partir desse exemplo de SCP, de acordo com a subseção 3.2, seria:

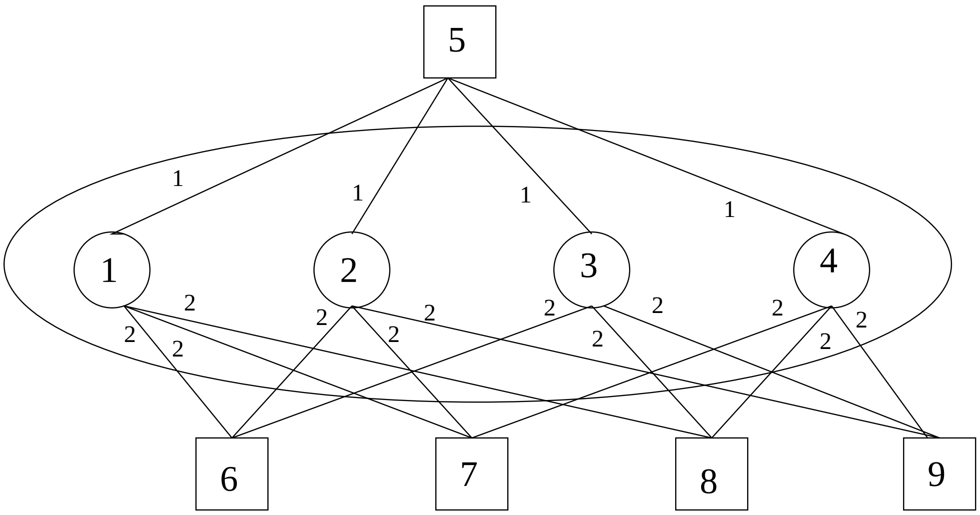

Figura 13: Exemplo de instância SPG gerada a partir de um SCP.

3.3.2 Instâncias Baseadas em Triplas de Steiner via SCP

Um sistema de triplas de Steiner de n elementos é um par (S, F), onde $S = \{1, 2, 3, ..., n\}$ é o conjunto das variáveis de 1 a n e $F = \{S_1, S_2, ..., S_m\}$ é uma família de triplas de S (isto é, subconjuntos de S de cardinalidade 3), que obedece a seguinte condição: $\forall_{x,y}$, $x, y \in S$, x, y estão contidos em uma das triplas do conjunto F [HL74, FNT74, FR89].

Esses sistemas só existem se: (1) $n \geq 3$, (2) o resto da divisão inteira de n por 6 for 1 ou 3 e (3) cada elemento de S aparece em $(n-1)/2$ elementos do conjunto F.

Nesse trabalho foi utilizado o código de [Pogg00] para criar sistemas de triplas de Steiner. Esses sistemas têm o seguinte formato: (a) número de variáveis, ou seja, nv, (b) número de triplas, ou seja, nt e, para cada tripla S_i, (c) a lista de elementos, $x, y, z \in S$, que formam essa tripla.

A transformação de uma instância de sistemas de triplas de Steiner para uma instância SCP foi feita pelo seguinte algoritmo: inicialmente, faça o número de triplas, nt, passar a ser o número de linhas do SCP e assuma que o número de variáveis, nv, passe a ser o número de colunas do SCP. Além disso, assuma que o custo de cada coluna C_j, onde $j = 1, 2, ...nv$ passe a ter o valor 1. Logo a seguir, assuma que o número de colunas que recobre a linha i, $i = 1, 2, ..., nt$, é sempre igual a 3. E, finalmente, para cada linha i, faça a tripla i passar a ser a lista de colunas, L_i, que recobre essa linha.

$m \leftarrow nt$
$n \leftarrow nv$
Para $j \leftarrow 1$ até n faça
    $C_j \leftarrow 1$
Para $i \leftarrow 1$ até m faça
    $L_i \leftarrow \emptyset$
Para $i \leftarrow 1$ até m faça
    Para $j \leftarrow 1$ até 3 faça
        Leia o elemento j da tripla i, k
        $L_i \leftarrow L_i \cup \{k\}$

Figura 14: Algoritmo de geração das instâncias SCP baseadas em instâncias de sistemas de triplas de Steiner.

Como foi mencionado na subseção 3.3, aplica-se a transformação de um SCP para um SPG descrita na subseção 3.2.

Exemplo: dado o arquivo de sistema de triplas de Steiner abaixo:

(1) 3 1
(2) 1 2 3

Figura 15: Exemplo de sistema de triplas de Steiner.

A linha (1) equivale ao item (a) e (b) da seção 3.3.2. Já a linha representa o item (c) desse sistema. O SCP formado a partir desse sistema seria:

(1) 1 3
(2) 1 1 1
(3) 3
(4) 1 2 3

Figura 16: Exemplo de um SCP a partir de um sistema de triplas de Steiner dado.

O SPG formado a partir desse exemplo de SCP, de acordo com a subseção 3.2, seria:

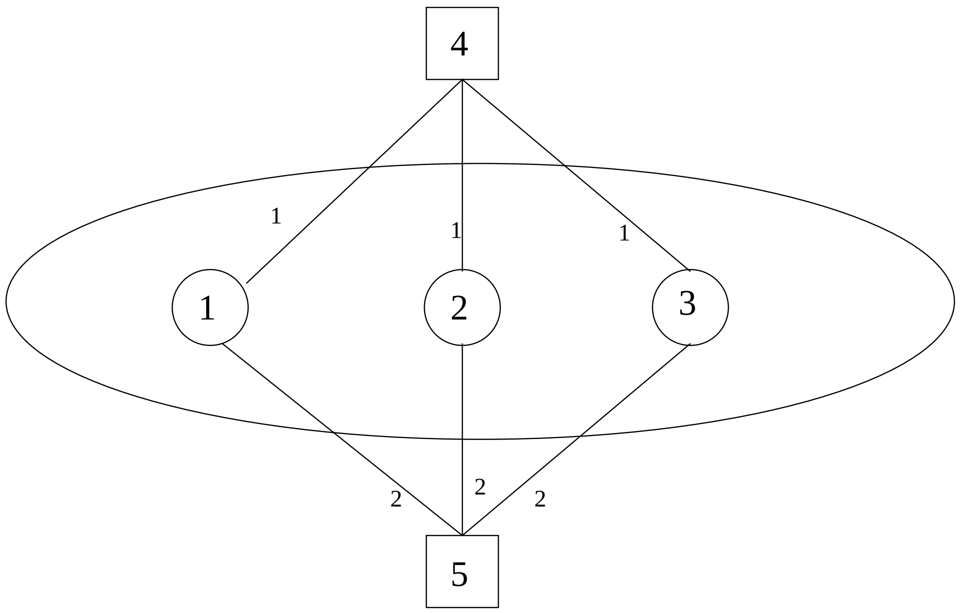

Figura 17: Exemplo de instância SPG gerada a partir de um SCP.

4 Limites Inferiores e Superiores

Nessa seção descreve-se os algoritmos usados para calcular limites inferiores e superiores para as novas instâncias de Steiner propostas.

4.1 Relaxação Linear da Formulação por Cortes Orientados

Essa formulação opera sobre o grafo direcionado $D = (V, A)$ obtido pela substituição de cada aresta $[u, v] \in E$ por 2 arcos (u, v) e (v, u) com mesmo custo da aresta original. Além disso, algum terminal $r \in T$ é escolhido para ser a *raiz*. Uma arborescência de Steiner enraizada em r é um conjunto de arestas $S \subseteq A$, tal que $(V(S), S)$ contém um caminho dirigido de r a t, para todo $t \in T - \{r\}$. Achar uma arborescência de Steiner de custo mínimo em D equivale a achar a árvore de Steiner de custo mínimo do problema original. Defina variáveis binárias y_a, $a \in A$, indicando quais arestas estão contidas na arborescência de Steiner, ou seja, $y_a = 1$ se a aresta a estiver na arborescência. A formulação é a seguinte:

$$minimize \sum_a c_a^{\rightarrow} y_a \qquad \{5\}$$

$$sujeito\ a$$

$$y_a(\delta(W)) \geq 1, \forall\ W \subset V,\ r \in W,\ (V - W) \cap T \neq \emptyset \qquad \{6\}$$

$$0 \leq y_a \leq 1 \qquad \{7\}$$

$$y_a \in \{0, 1\} \qquad \{8\}$$

Onde $\delta(W)$ denota o corte induzido por $W \subseteq V$.

$\delta(W)$ pode ser definido como sendo:

$$\delta(W) = \{(u,v) \in A | u \in X, v \in V - X\} \qquad \{9\}$$

Ou seja, $\delta(W)$ é o conjunto de arestas em que o nó u da aresta pertença ao conjunto X e o nó v pertença ao conjunto $V - X$.

A relaxação linear é obtida retirando-se a restrição $\{8\}$ da úlitma formulação. Além disso, na execução do código de [UW00], não foram inseridos todos os cortes possíveis de cada instância. Ao invés disso, esses cortes são incluídos e/ou retirados da solução e, para cada inserção e/ou retirada de cortes, executa-se algum método de resolução de programação linear, tais como CPLEX ou XPRESS. Essas operações são realizadas até que todos os cortes estejam presentes na solução. Nesse caso, o algoritmo para e o seu resultado indica o valor da relaxação linear.

4.2 Solução Primal

A maneira usada para calcular a solução primal dessas instâncias, descrita em [RUW00], utiliza a estrutura básica e os principais conceitos de um GRASP para desenvolver um algoritmo híbrido com perturbações nos custos das arestas, a fim de encontrar uma solução aproximada para o Problema de Steiner em Grafos. Cada iteração desse algoritmo consiste de uma fase de construção e uma fase de busca local. *Path-relinking* é aplicado como uma estratégia de intensificação de pós-otimização.

Na fase de construção, uma solução viável é criada usando um algoritmo guloso randomizado. Essa fase é implementada usando-se um número pré-definido de heurísticas gulosas combinadas com perturbações nos custos das arestas, randomicamente computados a cada iteração. Essas soluções construídas, em geral, não são ótimos locais. Assim, aplica-se algum algoritmo de busca local para tentar melhorar essas soluções.

Na fase de busca local, 2 definições de vizinhanças são combinadas para oferecer mais eficiência a estratégia de busca: a vizinhança baseada em nós e a a vizinhança baseada em *key-paths* [VSA96]. O algoritmo de busca local inicia com a aplicação de 1 das 2 vizinhanças citadas anteriormente sobre a solução obtida na fase de construção, C, gerando uma solução D. Logo a seguir, usa-se o outro tipo de vizinhança combinado com a busca local sobre a solução D, a fim de se obter uma solução melhorada E.

Três Estratégias de intensificação pós-otimização baseadas em *path-relinking* são usadas para tentar encontrar uma solução melhor que E, a saber: *path-relinking* de movimentos complementares, *path-relinking* de penalização dos custos das arestas e *path-relinking* híbrido adaptativo [RUW00]. Para implementar essas estratégias, mantém-se um conjunto, CE, contendo as melhores soluções, chamadas soluções elite, identificadas ao longo da execução do algoritmo de busca local. Geralmente, as estratégias baseadas em *path-relinking* geram novas soluções pela exploração de caminhos que conectam as soluções pertencentes ao conjunto CE.

5 Resultados Computacionais

Nessa seção são apresentados os resultados computacionais obtidos pelas execuções do código de [UW00] sobre essas instâncias.

Na tabela 1, são mostrados os resultados computacionais das instâncias descritas na seção

2. Essa tabela é formada por 6 colunas, a saber: (1) **n-q-r** que se refere aos números n, q e r usados para gerar o CCP, (2) **‖V‖-‖E‖-‖T‖** que informa o número de vértices, de arestas e de nós terminais da instância, respectivamente, (3) **Ótimo** que mostra qual é o valor da solução ótima para a instância em estudo, (4) **RL** que mostra a relaxação linear encontrada para uma determinada instância, (5) **Primal** que apresenta a melhor solução primal encontrada para uma determinada instância e, finalmente, (6) **Gap** que mostra a diferença entre a solução primal e a relaxação linear. Essa diferença é calculada da seguinte maneira:

$$Gap = ((Primal - \lceil RL \rceil) \times 100)/\lceil RL \rceil \qquad \{10\}$$

Observando a tabela 1, é possível concluir que existe um bom *gap* entre as soluções. Por outro lado, a dificuldade dessa classe de instâncias se concentra na impossibilidade de se determinar, antes da utilização dessa classe, qual o valor da solução ótima de grande parte das instâncias que pertencem a ela.

n-q-r	**‖V‖-‖E‖-‖T‖**	**Ótimo**	**RL**	**Primal**	**Gap(%)**
6-2-1	64-192-12	32	29.8332	32	6.67
3-4-1	64-288-8	23	21	23	9.52
3-5-1	125-750-13	—	$\geq$ 31	36	16.13
5-3-1	243-1215-27	—	$\geq$ 63	72	14.29
9-2-1	512-2304-64	—	$\geq$ 153	169	10.46
6-3-1	729-4368-76	—	$\geq$ 189	200	5.82
10-2-1	1024-5120-135	—	$\geq$ 324	349	7.72
11-2-1	2048-11263-244	—	$\geq$ 576	625	8.51
3-10-1	1000-13500-50	—	$\geq$ 113	127	12.39
7-3-1	2187-15308-222	—	$\geq$ 503	562	11.73
3-11-1	1331-19965-61	—	$\geq$ 137	154	12.41
12-2-1	4096-24574-473	—	$\geq$ 1099	1197	8.92
3-12-1	1728-28512-74	—	$\geq$ 163	186	14.11

Tabela 1: Soluções das instâncias pré-processadas geradas a partir do problema de recobrimento de código em que os custos das arestas são dependentes dos tipos de nós que a formam.

Na tabela 2, são mostrados os resultados computacionais das instâncias descritas na subseção 3.1. O formato dessa tabela é similar ao formato da tabela 1. Entretanto, a diferença que existe entre elas é que a primeira coluna da tabela 1, **n-q-r**, foi substituída pela coluna **nome** que indica o nome da instância SCP usada. Além disso, a coluna **Ótimo** foi eliminada dessa tabela porque, para essa classe de problemas, o valor da solução ótima não pode ser determinado previamente, como foi mencionado na subseção 3.1.

Observando a tabela 2, é possível concluir que existe um bom *gap* entre as soluções. A dificuldade dessa classe de instâncias se concentra no fato de não ser possível determinar a sua solução ótima apenas com o uso da solução ótima do SCP original.

Na tabela 3, são mostrados os resultados das instâncias discutidas na subseção 3.2. O formato dessa tabela é similar ao formato da tabela 1. Entretanto, a diferença que existe entre elas é que a primeira coluna da tabela 1, **n-q-r**, foi substituída pela coluna **nome** que indica o nome da instância SCP usada. Analisando-se essa tabela, pode-se observar que o *gap* entre as soluções é muito pequeno. Isso se deve ao fato de que o custo de inclusão de todos os terminais existentes na instância, isto é, segunda parcela da soma da fórmula {4}, é muito maior que o custo do SCP original (primeira parcela da soma da fórmula {4}).

Porém, a grande facilidade dessa classe de instâncias é poder determinar, através do uso da fórmula {4}, a solução ótima dessas instâncias, com o conhecimento prévio da solução ótima do SCP que originalmente a gerou.

nome	**‖V‖-‖E‖-‖T‖**	**RL**	**Primal**	**Gap(%)**
scp42	990-3610-200	232.055	239	2.58
scpe2	550-5013-50	52.4011	54	1.92
scp52	1819-7326-200	229.17	240	4.35
scp62	1199-10000-200	213.3	224	4.67
scpa2	3140-17795-300	≥ 305	357	17.05

Tabela 2: Soluções das instâncias pré-processadas geradas a partir das instâncias do problema de recobrimento de conjuntos existentes na OR-Library em que os valores das suas soluções ótimas não são conhecidos previamente.

nome	**‖V‖-‖E‖-‖T‖**	**Ótimo**	**RL**	**Primal**	**Gap(%)**
scpe2	551-5513-51	105	103.381	105	0.96
scp52	2110-9816-201	20502	20499.7611	20511	0.05
scp62	1201-11002-201	20346	20340.4564	20357	0.08
scpa2	3259-20990-301	30552	30547.4963	30575	0.09

Tabela 3: Soluções das instâncias pré-processadas geradas a partir das instâncias do problema de recobrimento de conjuntos existentes na OR-Library em que os valores das suas soluções ótimas são conhecidos previamente.

Na tabela 4, são mostrados os resultados das instâncias descritas na subseção 3.3.1. O formato dessa tabela é similar ao formato da tabela 1. Analisando-se essa tabela é possível concluir que, exceto para a instância 3-4-1, ocorre um bom *gap* entre as soluções.

A dificuldade das maiores instâncias dessa classe se resume ao fato de não se poder determinar, de uma maneira simples, quais os valores das soluções ótimas dos SCP's gerados na fase intermediária de criação delas e, portanto, não se poder calcular os valores das suas soluções ótimas.

Contudo, para os menores problemas dessa classe, o cálculo do valor da solução ótima do SCP gerado na fase intermediária é facilmente conhecido com o uso de algum algoritmo de resolução de programação linear, tais como CPLEX ou XPRESS. Assim, a determinação do ótimo desses problemas SPG é trivial.

Na tabela 5, são mostrados os resultados obtidos pelas execuções das instâncias discutidas na subseção 3.3.2. O formato dessa tabela é similar ao formato da tabela 3. Entretanto, a diferença que existe entre elas é que na primeira coluna da tabela 3, a coluna **nome**, que indica o nome da instância SCP usada, é substituída pela coluna **sistema**. Essa coluna se refere ao número de variáveis usado para gerar o sistema de triplas de Steiner.

n-q-r	$\|V\|$-$\|E\|$-$\|T\|$	Ótimo	RL	Primal	Gap(%)
6-2-1	129-512-65	140	137.143	140	1.45
2-7-1	99-686-50	105	101.769	105	2.94
3-4-1	129-704-65	136	134.4	136	0.74
2-10-1	201-2000-101	—	205.263	210	1.94
5-3-1	487-2916-244	—	508.091	519	1.96
9-2-1	1024-5632-513	—	1075.2	1104	2.60

Tabela 4: Soluções das instâncias pré-processadas geradas a partir do problema de recobrimento de código via uma transformação prévia para o SCP.

Observando essa tabela, é possível concluir que existe um bom *gap* entre as soluções. A dificuldade da maior instância dessa classe se resume ao fato de não se poder determinar, de uma maneira simples, qual o valor da solução ótima do SCP gerado na fase intermediária de criação dela e, portanto, não se poder calcular o valor da sua solução ótima.

Contudo, para a maioiria dos problemas dessa classe, o cálculo do valor da solução ótima do SCP gerado na fase intermediária é obtido de maneira simples com a utilização de algum algoritmo de resolução de programação linear, tais como CPLEX ou XPRESS. Assim, a determinação do ótimo desses problemas SPG é simplesmente a aplicação da fórmula $\{4\}$.

sistema	$\|V\|$-$\|E\|$-$\|T\|$	Ótimo	RL	Primal	Gap(%)
31	187-496-156	325	320.333	325	1.25
43	345-946-302	626	616.333	626	1.46
51	477-1326-426	875	867	883	1.85
57	590-1653-533	1093	1083	1103	1.85
79	1107-3160-1028	—	2080.33	2109	1.35

Tabela 5: Soluções das instâncias pré-processadas geradas a partir do problema de triplas de Steiner via uma transformação prévia para o SCP.

6 Considerações Finais

Com um maior entendimento de alguns problemas de otimização combinatória, adquirido no desenvolvimento desse trabalho, foi possível desenvolver instâncias para o problema de Steiner em grafos a partir de outros problemas de otimização, tais como o problema de recobrimento de código, o problema de recobrimento de conjuntos e os sistemas de triplas de Steiner.

Informalmente, nas execuções realizadas, observou-se que, para as instâncias baseadas em recobrimento de código cujos pesos das arestas dependem do tipo de nós que as formam, existe um bom *gap* entre a relaxação linear e a solução primal, para as instâncias onde foi possível calcular essa relaxação. Contudo, a dificuldade dessa classe de instâncias se concentra na impossibilidade de se determinar, antes da utilização dessa classe, qual o valor da solução ótima de grande parte das instâncias que pertencem a ela.

Nas instâncias baseadas em recobrimento de conjuntos em que não se conhecem, previamente, os valores de suas soluções ótimas, foi possível observar que existe um bom *gap* entre as suas soluções. A dificuldade dessa classe de instâncias se concentra no fato de não ser possível determinar a sua solução ótima apenas com o uso da solução ótima do problema de recobrimento original.

Nas instâncias baseadas em recobrimento de conjuntos em que o valor da solução ótima pode ser calculado previamente, foi observado que *gap* entre as soluções é muito pequeno. Porém, a grande facilidade dessa classe de instâncias é poder determinar a solução ótima dessas instâncias, com o conhecimento prévio da solução ótima do problema de recobrimento que originalmente a gerou.

Para as instâncias baseadas em recobrimento de código que sofrem uma transformação intermediária para o problema de recobrimento de conjuntos, observou-se que ocorre um bom *gap* entre as soluções na maioria das instâncias testadas. A dificuldade das maiores instâncias dessa classe se resume ao fato de não se poder determinar, de uma maneira simples, quais os valores das soluções ótimas dos problemas de recobrimento de conjuntos gerados na fase intermediária de criação delas e, portanto, não se poder calcular os valores das suas soluções ótimas. Contudo, para os menores problemas dessa classe, o cálculo do valor da solução ótima do problema de recobrimento de conjunto gerado na fase intermediária é facilmente conhecido e, assim, a determinação do ótimo desses problemas de Steiner é trivial.

Finalmente observou-se que, para as instâncias baseadas em sistemas de triplas de Steiner que sofrem uma transformação prévia para o problema de recobrimento de conjuntos, existe um bom *gap* entre as soluções. A dificuldade da maior instância dessa classe se resume ao fato de não se poder determinar, de uma maneira simples, qual o valor da solução ótima do problema de recobrimento gerado na fase intermediária de criação dela e, portanto, não se poder calcular o valor da sua solução ótima. Contudo, para a maioiria dos problemas dessa classe, o cálculo do valor da solução ótima do problema de recobrimento gerado na fase intermediária é obtido de maneira simples e, portanto, a determinação do ótimo desses problemas de Steiner é trivial.

Bibliografia

[Beas90] J.E. Beasley, "OR-Library: Distributing Test Problems by Electronic Mail", **Journal of the Operational Research Society** 41, 1069-1072, 1990.

[BL98] J.E. Beasley and A. Lucena, "A Branch and Cut Algorithm for the Steiner Problem in Graphs", **Networks**, 31, 39-59, 1998.

[CFT98] A. Caprara, M. Fischetti and P. Toth, "Algorithms for the Set Covering Problems", para ser publicado em **Annals of Operations Research**, 1998.

[DSR00] D. Du, J.M. Smith and J.H. Rubenstein, "Advances in Steiner Trees", D. Du, J.M. Smith and J.H. Rubenstein Eds., Combinatorial Optimization, Volume 6, 2000.

[Duin93] C. Duin, "Steiner's Problem in Graphs", *PhD thesis*, University of Amsterdam, 1993.

[FR89] T.A. Feo and M.G.C. Resende, "A Probalilistic Heuristic for a Computationally Difficult Set Covering Problem", **Operations Research Letters** 8, 67-71, 1989.

[FNT74] D.R. Fulkerson, G.L. Nemhauser and L.E. Trotter, Jr., "Two Computationally Difficult Set Covering Problems that Arise in Computing the 1-Width of Incidence Matrices of Steiner Triple Systems", **Mathematical Programming Study** 2, 72-81, 1974.

[GM93] M.X. Goemans and Y. Myung, "A Catalog of Steiner Tree Formulations", **Networks**, 23, 19-28, 1993.

[HL74] M. Hall and J.H. van Lint, "Combinatorics", **Proceedings of the NATO Advanced Study**, The Netherlands, 1974.

[HRW92] F.K. Hwang, D.S. Richards and P. Winter, "The Steiner Tree Problem", **Annals of Discrete Mathematics**, 53, North-Holland, Amsterdam, 1992.

[Karp72] R.M. Karp, "Reducibility Among Combinatorial Problems", **Complexity of Computer Computations** (R.E. Miller and J.W. Thatcher, Eds.), Plenum Press, New York, 85-103, 1972.

[KM98] T. Koch and A. Martin, "Solving Steiner Tree Problems in Graphs to Optimality", **Networks** 32, 207-232, 1998.

[KMV00] T. Koch, A. Martin and S. Voß, "SteinLib: An Updated Library on Steiner Tree Problems in Graphs", **Konrad-Zuse-Zentrum für Informationstechnik Berlin**, ZIB-Report 00-37, Berlin, 2000.

[Macu87] N. Maculan, "The Steiner Problems in Graphs", **Annals of Discrete Mathematics**, 31, 185-212, 1987.

[MW95] T.L. Magnanti and L.A. Wolsey, "Optimal Trees", **Handbooks in Operations Research and Management Science**, In M.O. Ball et al., editor, volume 7, chapter 9, Elsevier Science, 1995.

[Pogg00] M.V.S. Poggi de Aragão, comunicação pessoal, 2000.

[PS99] M.V.S. Poggi de Aragão and C.C. de Souza, "Upper Bounds for Minimum Covering Codes Via Tabu Search", **3rd Metaheuristics International Conference**, 359-365, Rio de Janeiro, Brazil, 1999.

[PV00] T. Polzin and S. Vahdati, "Improved Algorithms for the Steiner Problem in Networks", para ser publicado em **Discrete Applied Mathematics**, 2000.

[PUW00] M.V.S. Poggi de Aragão, E. Uchoa and R.F. Werneck, "Dual Heuristics on the Exact Solution of Large Steiner Problems", submetido para publicação, 2000.

[VSA96] M.G.A. Verhoeven, M.E.M. Severens and E.H.L. Arts, "Local Search for Steiner Trees in Graphs", In V.J. Rayward-Smith et al., editor, **Modern Heuristics Search Methods**, 117-129, John Wiley and Sons, 1996.

[RUW00] C.C. Ribeiro, E. Uchoa and R.F. Werneck, "A hybrid GRASP with perturbations and adaptive path-relinking for the Steiner problem in graphs", 2000.

[UW00] E. Uchoa and R.F. Werneck, comunicação pessoal, 2000.

[Wint87] P. Winter, "Steiner Problems in Networks: A Survey", **Networks**, 17, 129-167, 1987.