EDUARDO UCHOA BARBOZA

ALGORITMOS PARA PROBLEMAS DE STEINER COM APLICAÇÕES EM PROJETO DE CIRCUITOS VLSI

TESE DE DOUTORADO

DEPARTAMENTO DE INFORMÁTICA

Rio de Janeiro, Maio de 2001

EDUARDO UCHOA BARBOZA

ALGORITMOS PARA PROBLEMAS DE STEINER COM APLICAÇÕES EM PROJETO DE CIRCUITOS VLSI

Tese apresentada ao Departamento de Informática da PUC-Rio como parte dos requisitos para a obtenção do título de Doutor em Ciências em Informática.

Orientador: Prof. Marcus Poggi de Aragão

Departamento de Informática

Pontifícia Universidade Católica do Rio de Janeiro

Rio de Janeiro, Maio de 2001

Esta tese é dedicada a minha esposa India e aos meus pais, irmãos e tios, que tanto me apoiaram e incentivaram.

Eduardo Uchoa Barboza

Agradecimentos

Ao meu orientador Marcus Poggi de Aragão que esteve presente em todos os momentos importantes deste trabalho.

Aos brasileiros que pagam impostos, por me sustentarem durante os últimos quatro anos através do CNPq.

Ao professor Sérgio Colcher pelo estilo Latex usado nesta tese.

Aos meus colegas da PUC, em particular Adriana Alvim, Christina Flach, Maira Medina, Maurício Souza, Sérgio Colcher e Simone Martins, pela sua amizade dentro e fora do ambiente de trabalho.

Ao meu colega Renato Werneck pela amizade e pela importante colaboração na pesquisa realizada no último ano do meu doutorado.

Aos professores Abílio Lucena, Nelson Maculan, Oscar Porto, Celso Ribeiro e Cid de Souza pelos ensinamentos, incentivos e valiosas discussões durante o doutorado.

Ao Lacerda e aos meus incontáveis companheiros das quartas-feiras no Hipódromo, sem os quais esta tese teria sido feita do mesmo jeito, mas eu teria bebido muito menos.

RESUMO

Esta tese está dividida em duas partes. A primeira é um estudo do empacotamento de duas árvores de Steiner, um problema NP-difícil com aplicações em projeto de circuitos VLSI. É proposta uma nova formulação e a partir de um estudo teórico dos poliedros associados, novas classes de desigualdades definindo facetas são encontradas. Isso resulta em um algoritmo de *branch-and-cut* para o problema. É feito um detalhado estudo de técnicas práticas para acelerar esse algoritmo. O código resultante consegue um desempenho muito bom, resolvendo instâncias com 10.000 vértices em alguns minutos. A segunda parte da tese trata do problema de Steiner em grafos, também NP-difícil. A fase de pré-processamento é uma das técnicas decisivas na resolução de grandes instâncias desse problema. Entretanto, nas instâncias originadas de projeto de circuitos VLSI, uma das mais importantes aplicações do problema de Steiner, as técnicas de pré-processamento tradicionais não funcionam bem. Propomos novos testes de redução particularmente adequados a esses casos. Testando o procedimento resultante em instâncias reais disponíveis na internet, obtivemos significativas reduções. Finalmente, estudamos a resolução exata do problema de Steiner em grafos. Analisamos o comportamento de uma conhecida heurística dual para esse problema e propomos três novas heurísticas duais para melhorar o seu desempenho. Essas heurísticas levam a dois algoritmos exatos, um *branch-and-bound* que não usa programação linear e um *branch-and-cut*. Esses algoritmos foram testados em um variado conjunto de instâncias da literatura e obtiveram bons resultados, resolvendo várias dessas instâncias pela primeira vez.

ABSTRACT

The first part of this thesis addresses the Packing of Two Steiner Trees Problem, which is NP-hard and has applications on VLSI layout. We present a new formulation and a theoretical investigation on the corresponding polyhedra. New families of valid inequalities, including proven facets, are identified. We also discuss branch-and-cut implementation issues. The resulting code has a good practical performance and can solve some instances with up to 10,000 vertices in minutes. The second part of the thesis addresses the Steiner Problem in Graphs, which is also NP-hard. VLSI layout applications yield Steiner instances considered hard to be solved by current methods. In particular, preprocessing techniques developed for Steiner problems over general graphs are not likely to reduce significantly such instances. We propose a new preprocessing procedure effective for VLSI problems. Testing this procedure on real-world instances available on the web, significant reductions were obtained within reasonable computational times. Finally, we study the exact solution of the Steiner Problem in Graphs. We analyze a well-known dual heuristic for this problem and propose three new dual heuristics to improve its performance. These heuristics lead to two exact algorithms, a branch-and-bound without linear programming and a branch-and-cut. The algorithms were tested over instances from the literature and obtained good results, solving several instances for the first time.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A otimização combinatória é o ramo da matemática aplicada e da ciência da computação que trata de problemas em que se buscam os mínimos (ou máximos) de funções definidas sobre um domínio discreto. Alguns desses problemas têm recebido uma atenção especial dos pesquisadores devido a sua capacidade de modelar um grande número de aplicações práticas, seja na indústria, no governo, na tecnologia ou em qualquer área que necessite de planejamento no uso de recursos escassos. Esta tese apresenta trabalhos sobre dois importantes problemas de otimização combinatória: o problema de empacotamento de árvores de Steiner e o problema de Steiner em grafos.

1.1 O Problema de Empacotamento de Árvores de Steiner

O Problema de Empacotamento de Árvores de Steiner (PEAS) é definido como: dados um grafo G e K conjuntos disjuntos de vértices, conhecidos como conjuntos de terminais, encontrar K sub-árvores de G disjuntas, de forma que cada uma delas conecte um conjunto de terminais. Determinar a existência de solução para uma instância do PEAS é um problema NP-completo, mesmo no caso em que $K = 2$ (Korte et al. [41]). Existe também uma segunda versão do PEAS, também NP-completa, em que só se exige que as arestas de cada árvore sejam disjuntas, os vértices podem ser compartilhados. A referência mais abrangente sobre o PEAS ainda é o livro editado em 1989 por Korte, Lovász, Prömel e Schrijver [40], em particular os artigos [23, 37, 41, 56, 57].

A aplicação do PEAS com maior destaque é o projeto de circuitos VLSI, pois o problema de roteamento das ligações elétricas dentro de um chip pode ser modelado de forma bastante realista como um PEAS (ver por exemplo, Lengauer [42] ou Sherwani[58]). A seguir fazemos uma descrição mais detalhada dessa aplicação:

APLICAÇÃO 1.1. *No roteamento de ligações elétricas em projeto de circuitos VLSI, o grafo G tem uma correspondência física direta com o substrato do chip: as arestas representam trilhas pelas quais pode passar uma ligação elétrica e os vértices são os pontos de intersecção entre as trilhas. Alguns desses vértices definem os K conjuntos de terminais, e estão associados aos componentes do circuito que devem ser conectados entre si para que o chip desempenhe a sua função. O substrato do chip é dividido em camadas (tipicamente de 2 a 8) e dentro de cada camada as trilhas só podem correr em duas direções ortogonais. Há também trilhas verticais ligando vértices em diferentes camadas. Dessa forma, o grafo G corresponde a alguns grafos de grade "empilhados". Esses grafos de grade quase sempre têm "buracos", que correspondem a regiões do substrato em que não podem haver trilhas.*

O tamanho das instâncias vindas diretamente dessa aplicação pode ser muito grande: nos mais modernos microprocessadores a dimensão de G chega a $40.000 \times 40.000 \times 8$ e K chega a 10^6 [73]. Deve-se notar que a grande maioria desses 10^6 conjuntos de terminais é formada por 2 ou 3 vértices muito próximos entre si, relativamente poucos conjuntos de terminais estão distribuídos pelo chip. Assim, o PEAS muitas vezes é aplicado no chamado *roteamento global*, que consiste em se rotear esses conjuntos globais em um grafo simplificado, onde cada vértice corresponde a uma sub-área inteira do grafo original. Posteriormente, no chamado *roteamento detalhado*, aplica-se o PEAS novamente em cada sub-área, para transformar o roteamento global em um roteamento válido para o problema original.

Mas tambem há outras aplicações do PEAS em contextos fora do projeto de circuitos. Por exemplo:

APLICAÇÃO 1.2. *Uma rede já existente de distribuição de algum produto (como gás, energia elétrica ou ligações telefônicas) deve ser dividida entre K empresas (digamos porque o serviço foi privatizado), mas de modo que cada empresa consiga conectar certas localidades. Aqui G representa a rede e os conjuntos de terminais correspondem às localidades que cada empresa deve conectar.*

Todos os problemas NP-completos são equivalentes do ponto de vista da solubilidade em tempo polinomial no pior caso: um hipotético algoritmo polinomial para um desses problemas seria um algoritmo polinomial para todos os demais. Mas do ponto de vista prático a questão é mais complexa. Existem alguns problemas NP-completos importantes para os quais já existem algoritmos que, apesar de exponenciais, são capazes de resolver de forma exata muitas instâncias grandes em tempo razoável. O exemplo mais célebre dessa situação é o problema do caixeiro-viajante, para o qual a abordagem da combinatória poliedral (que leva aos chamados algoritmos de *branch-and-cut*) tem sido muito bem sucedida. Por outro lado, existem vários problemas NP-completos que continuam sendo realmente intratáveis na prática.

O PEAS parece se encontrar nessa segunda categoria e, de fato, todos os algoritmos para esse problema usados na prática são heurísticas. A convicção de que é difícil resolver instâncias do PEAS de forma exata foi reforçada com os recentes trabalhos de Grötschel et al. [28, 27], que se utilizam das técnicas de combinatória poliedral. O algoritmo de *branch-and-cut* proposto teve dificuldades para resolver 7 instâncias de tipo *switchbox* (G é um grafo de grade e todos os terminais estão junto a borda), a maior delas tinha 345 vértices, 24 conjuntos de terminais e um total de 68 terminais. É interessante ler o que os próprios autores escreveram na conclusão desse artigo:

"We have encountered the Steiner tree packing problem in practice and considered it worthwhile to engage in a study of Steiner tree packing polytopes (...) As can be seen from the results of this paper, all expected difficulties turned up. (...) Finding rich classes of joint inequalities and describing them in an understandable manner is a considerable challenge; characterizing the facet-defining inequalities in these classes leads to tremendous technical difficulties. (...) The most important objective of our research project, however, was to see whether the machinery of polyhedral combinatorics can help solving practical instances of Steiner tree packing problems. Indeed, it can — to some extent."

O segundo capítulo desta tese consiste em um estudo do PEAS restrito ao caso $K = 2$, que chamaremos de 2PEAS. Alguns pontos que justificam o estudo desse subproblema:

- Em casos como o descrito na Aplicação 1.2, o valor de K costuma ser pequeno. Quando $K = 2$, o problema é exatamente um 2PEAS.
- Nas aplicações em projeto de circuitos VLSI (Aplicação 1.1), o valor de K costuma ser grande. Mas existem subproblemas a serem resolvidos em que $K = 2$. Por exemplo, no chamado *roteamento de terra e* V_{cc} (ver Sherwani [58], páginas 440–444, ou Syed e El Gamal [61]) há um grande número de terminais que devem ser conectados ao potencial terra e outro grande número de terminais que devem ser conectados ao V_{cc}, essas ligações são feitas em uma ou duas camadas especiais. Esse problema é um 2PEAS.
- Talvez alguns dos resultados obtidos ao se estudar o 2PEAS possam ser generalizados para o PEAS.

De qualquer forma, *a principal contribuição alcançada por esse estudo foi mostrar que o 2PEAS, apesar de NP-difícil, é um problema que pode ser atacado de forma razoável, tanto do ponto de vista teórico (através da descrição de importantes famílias de facetas dos seus poliedros), quanto do ponto de vista prático (pela resolução exata de instâncias grandes).* As contribuições específicas são resumidas nos seguintes tópicos:

- Foi proposta uma nova formulação para o 2PEAS que se revelou bastante conveniente para o estudo poliedral e para o algoritmo de *branch-and-cut* subsequentes.
- Foi realizada uma investigação teórica dos poliedros do 2PEAS correspondentes à nova formulação. As facetas do 2PEAS foram conceitualmente divididas em duas classes:

aquelas herdadas do poliedro de Steiner em grafos usual e as novas facetas que surgem a partir da junção entre dois poliedros de Steiner. Foram encontradas condições para que facetas do Steiner usual também sejam facetas do 2PEAS. Quando isso não acontece, há espaço para melhorar essas desigualdades. Novas classes de facetas, como as p-cruzes, surgem através desse raciocínio.

- A formulação e as novas desigualdades descobertas foram empregadas na construção de um algoritmo de *branch-and-cut* para o 2PEAS. É sabido que implementar um algoritmo desse tipo, de forma que ele seja eficiente na prática, é uma tarefa nada trivial e que pode envolver pequenos "truques computacionais". No caso do 2PEAS esse efeito foi dramático. Instâncias que inicialmente demoravam horas, passaram a ser resolvidas em segundos. É possível que alguns dos novos truques utilizados também funcionem em algoritmos de *branch-and-cut* para outros problemas.

As seguintes publicações foram geradas a partir do trabalho feito nessa parte da tese:

- "Branch-and-cut for vertex-disjoint packing of two Steiner trees", anais do IX CLAIO, Buenos Aires, 1998 [63].
- "Vertex-disjoint packing of two Steiner trees: polyhedra and branch-and-cut", anais do VII IPCO, Graz, 1999 [64].
- "Vertex-disjoint packing of two Steiner trees: polyhedra and branch-and-cut", *Mathematical Programming*, 2001 [65].

É interessante notar que alguns resultados sobre o PEAS já aparecem em forma embrionária em um artigo escrito em 1997:

- "The γ-connected assignment problem", *European Jounal of Operational Research*, 1999 [49].

Todos esses artigos foram escritos em co-autoria com o orientador desta tese.

1.2 Problema de Steiner em Grafos

O Problema de Steiner em grafos (PSG) é definido como: dado um grafo G, um conjunto de vértices terminais e uma função atribuindo custos positivos às arestas, encontrar uma subárvore de G conectando todos os terminais a custo mínimo. O PSG consta da primeira lista de problemas a serem provados NP-difíceis (Karp [36]). Bons *surveys* sobre o PSG podem ser encontrados em Maculan [46], Winter [70], Hwang et al. [32] e, mais recentemente, no livro editado por Du et al. [15]. Esse problema continua sendo intensamente estudado na atualidade: pelo menos 25 artigos diretamente relacionados ao PSG têm sido publicados por ano em revistas ou congressos conceituados.[1] Como resultado do conhecimento acumulado em todas essas pesquisas, o PSG já é um problema razoavelmente bem tratado na prática. Existem vários algoritmos exatos capazes de resolver muitas instâncias da ordem de alguns milhares de vértices e algumas dezenas de milhares de arestas.

O trabalho realizado nessa parte da tese visou aprimorar o estado-da-arte dos algoritmos para o PSG, permitindo que instâncias maiores e mais difíceis pudessem ser resolvidas.

1.2.1 Pré-processamento de instâncias VLSI

Uma das mais importantes aplicações práticas do PSG ocorre em roteamento de ligações elétricas em projeto de circuitos VLSI. Como vimos na Aplicação 1.1, na verdade esse problema é um PEAS sobre um grafo de grade com algumas camadas. Mas como não é possível resolver esse PEAS, usa-se frequentemente a seguinte heurística: (i) para cada conjunto de terminais, resolve-se um PSG num grafo de grade de uma única camada; (ii) distribuem-se as árvores obtidas nas várias camadas; (iii) sempre que houver cruzamentos entre árvores na mesma camada, tenta-se usar partes não usadas das camadas adjacentes para fazer "pequenos desvios". Normalmente várias tentativas são feitas até que se chegue a uma solução factível. Mas de qualquer maneira, essa heurística exige a solução de instâncias do PSG em que G é um grafo de grade com buracos. Essas instâncias são conhecidas como instâncias VLSI.

Um dos pontos chave para o sucesso dos modernos algoritmos exatos para o PSG é a fase de pré-processamento, que consiste em aplicar algoritmos polinomiais na tentativa de transformar a instância original em uma instância reduzida com tamanho significativamente menor.

O modo usual de executar esse pré-processamento é através de testes "condição-ação". Um exemplo de teste muito simples: se v é um vértice não-terminal de grau 2 (condição), então v e as duas arestas adjacentes a v podem ser substituídas por uma única aresta com o custo dessas duas arestas (ação). Existem muitos outros tipos de testes com condições e ações bem mais complexas. Cada um desses testes pode ser visto como um teorema sobre o PSG: se *condição* é verdadeira, então uma solução ótima da instância reduzida (obtida após se aplicar *ação* na instância original) corresponde a uma solução ótima da instância original. O trabalho mais completo nessa linha foi realizado por Duin e Volgenant [17, 16], esses autores generalizaram e sistematizaram uma série de testes já existentes na literatura, e assim criaram um algoritmo de pré-processamento muito eficaz para vários tipos de instâncias. Por exemplo,

[1] Esse número foi estimado pelo autor através de pesquisa em sítios gratuitos da Internet que fornecem referências bibiliográficas. É possível que existam artigos não listados nesses sítios.

nas instâncias da OR-Lib [8], a taxa de redução típica está entre 70% e 90%. Entretanto, esses testes praticamente não funcionam para as instâncias VLSI.

A contribuição descrita no terceiro capítulo desta tese foi desenvolver novos testes de redução particularmente efetivos para as instâncias VLSI. O algoritmo de pré-processamento resultante foi testado nas instâncias VLSI reais disponíveis na SteinLib [39], conseguindo taxas de redução muito boas. Essas reduções foram fundamentais para que todas essas instâncias pudessem ser resolvidas de forma ótima.

Uma segunda contribuição mais abstrata vem do fato que os novos testes adotam uma abordagem distinta dos testes tradicionais e que pode ser comparada a uma "prova automática de teorema por contradição". Por exemplo, um teste pode assumir como hipótese inicial "a aresta e é usada em todas as soluções ótimas". A partir daí, tenta-se usar condições (semelhantes àquelas usadas nos testes tradicionais) para derivar implicações lógicas, e assim, demonstrar que outras arestas pertencem ou não a uma solução ótima. Os fatos já demonstrados anteriormente servem de apoio para novas demonstrações, até que uma das seguintes situações ocorra: (a) Não é possível concluir mais nada. Nesse caso diz-se que o teste "falhou" e a instância permanece inalterada. (b) Chega-se a uma contradição. Nesse caso fica provado que a hipótese inicial era falsa e pode-se adotar a ação correpondente. No caso do exemplo, remover a aresta e.

O seguinte artigo foi produzido a partir desse trabalho:

- E. Uchoa, M. Poggi de Aragão e C.C. Ribeiro, "Preprocessing Steiner problems from VLSI layout", submetido para publicação, 1999 [66].

1.2.2 Heurísticas duais em algoritmos exatos para o PSG

Os melhores algoritmos exatos para o PSG são baseados nas chamadas "formulações fortes" para o problema. Pelo menos três delas são conhecidas há bastante tempo: a formulação por multifluxo (Claus e Maculan [13] e Wong [72]), a formulação por cortes direcionados (Wong [72], Aneja [2] usa cortes não-direcionados) e a formulação por eliminação de ciclos generalizada (Lucena [44], Goemans [25] e Margot at al. [47]). Essas formulações são equivalentes, no sentido em que o limite inferior dado por suas relaxações lineares é sempre igual. Na grande maioria das instâncias citadas na literatura, esses limites estão bastante próximos, ou até igualam, o valor do ótimo inteiro, de forma que resolver uma dessas relaxações lineares implica em estar próximo da resolução do PSG correspondente. Entretanto essa resolução, que é um problema polinomial, pode ser uma tarefa computacionalmente muito cara para instâncias grandes ou degeneradas.

Em 1984, Wong [72] propôs uma heurística de *dual ascent* para se obter rapidamente uma solução aproximada do dual da formulação de multifluxo (a mesma heurística pode ser adaptada para as demais formulações equivalentes). Desde então, outros autores, como Voss [67], Duin [16] e Polzin e Vahdati [53], confirmaram a qualidade dos limites inferiores dados por essa heurística. Por exemplo, ela já é suficiente para resolver de forma ótima cerca de metade das instâncias da OR-Lib. Entretanto, em várias outras instâncias a qualidade dos limites não é boa o suficiente para ajudar na resolução ótima.

A contribuição descrita no quarto capítulo dessa tese foi criar novas heurísticas duais que aperfeiçoam a heurística de Wong e integrá-las em algoritmos exatos para o PSG, de forma a obter significativas melhorias nos tempos de resolução. Dessa forma foi possível resolver um grande número de instâncias difíceis que não puderam ser resolvidas por outros métodos.

A seguinte publicação foi gerada a partir desse trabalho:

- M. Poggi de Aragão, E. Uchoa e R. F. Werneck, "Dual Heuristics on the Exact Solution of Large Steiner Problems", Anais do GRACO, Fortaleza, 2001 [50].

Deve-se destacar que a pesquisa com as heurísticas duais foi feita em estreita colaboração com o aluno de mestrado Renato F. Werneck, que inclusive escreveu grande parte do código usado nos experimentos. Esse código está integrado a um complexo pacote computacional, chamado de Bossa, que implementa uma série de algoritmos para o PSG, incluindo testes de redução, sofisticadas heurísticas primais, e também algoritmos exatos. Planeja-se que em breve o Bossa esteja disponível na internet para uso de outros pequisadores.

1.3 Notas

Nesta tese assumimos que o leitor tenha familiaridade com os conceitos básicos relacionados à otimização combinatória. A menos que exista alguma indicação em contrário, seguimos as definições dadas nas seguintes referências clássicas:

- Complexidade computacional - Garey e Johnson [24].
- Teoria de grafos - Bondy e Murty [11].
- Teoria poliedral e programação inteira - Nemhauser e Wolsey [48].

Uma referência em português sobre esse último tópico, muito consultada pelo autor, é o livro de Ferreira e Wakabayashi [21].

Muitos termos técnicos desta tese aparecem em português. Isso pode causar algum estranhamento inicial, por exemplo ao se ler *empacotamento* no lugar de *packing* ou *levantamento* no lugar de *lifting*. Mas não vemos razão para deixar de traduzir termos com clara equivalência em português. Outros termos, que já são de uso consagrado e não tem essa clara equivalência, aparecem na forma original. Por exemplo, *branch-and-cut* ou *dual ascent*.

Capítulo 2

O Problema do Empacotamento de Duas Árvores de Steiner

2.1 Introdução

2.1.1 Definição do problema

Sejam $G = (V, E)$ um grafo conexo e N_1 e N_2 dois conjuntos de vértices disjuntos, conhecidos como *conjuntos de terminais*. O problema de decisão associado ao empacotamento de duas árvores de Steiner (2PEAS) é definido como: *existem dois conjuntos de vértices disjuntos $T_1 \supseteq N_1$ e $T_2 \supseteq N_2$ tais que $G[T_1]$ e $G[T_2]$ (os subgrafos induzidos por T_1 e T_2) sejam conexos ?* Esse problema foi provado ser NP-completo por Korte et al.[41].

Nesta tese é abordada uma versão de otimização do 2PEAS que generaliza o problema acima. Suponha que $G = (V, E)$ é um grafo conexo com pesos inteiros w associados aos vértices em V. Defina N_1 como o conjunto de vértices com peso negativo e N_2 como os vértices de peso positivo. O problema é *encontrar dois conjuntos de vértices disjuntos T_1 e T_2 tais que $G[T_1]$ e $G[T_2]$ sejam conexos e que minimizem a quantidade $z = \sum_{i \in (N_2 \setminus T_2)} w_i - \sum_{i \in (N_1 \setminus T_1)} w_i$*. Dada uma solução (T_1, T_2) desse problema, dizemos que terminais de N_1 em T_1 e terminais de N_2 em T_2 estão *conectados*. Ou seja, o problema é achar uma solução que deixe de conectar o menor número possível de terminais, de acordo com as prioridades dadas pelos pesos. Caso todos os pesos sejam -1, 0 ou 1, a instância é dita *não-ponderada* e a função objetivo se reduz a minimizar $z = |N_1 \setminus T_1| + |N_2 \setminus T_2|$.

Algoritmos para essa versão de otimização podem ser muito úteis para as aplicações práticas do 2PEAS citadas no Capítulo 1. Por exemplo, o processo de projeto de circuitos integrados costuma ser feito por tentativas: posicionam-se os terminais, se não for possível conectá-los muda-se ligeiramente o posicionamento e assim por diante. Nesse caso, o valor ótimo z^*

não apenas é um bom indicador de quão próximo se está da factibilidade, mas a própria solução (T_1, T_2) já sugere quais terminais devem ser movidos. A idéia é que se usem pesos menores (em valor absoluto) para os terminais que dispõem de mais alternativas de posicionamento.

Um exemplo de uma instância do 2PEAS é mostrado na Figura 2.1. O conjunto N_1 é formado pelos vértices $\{c, d, f, g, k\}$ e N_2 por $\{a, e, i, j\}$. Uma solução ótima da instância, com $z = 3$, é dada por $T_1 = \{d, f, g, h, j, k\}$ e $T_2 = \{a, b, c, e, i\}$. Nessa solução, os terminais c e j não estão conectados.

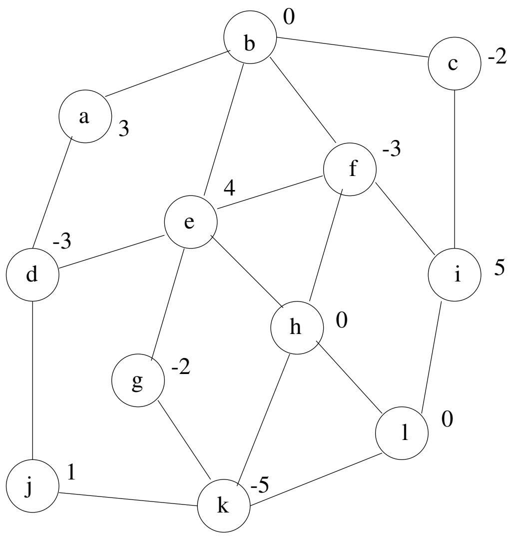

Figura 2.1: Exemplo de uma instância do 2PEAS.

Um exemplo de uma instância não ponderada do 2PEAS é mostrado na Figura 2.2. Nesse caso, G é um grafo de grade 40×40. Os quadradinhos pretos representam terminais em N_1 e os vazados, os terminais em N_2. As arestas de G, que seguem as coordenadas da grade, e os vértices não-terminais, que estão nas intersecções de coordenadas em que não há quadradinhos, não estão desenhados.

A Figura 2.3 mostra uma solução ótima dessa mesma instância. As linhas cheias indicam as arestas em uma árvore geradora de $G[T_1]$, enquanto as linhas pontilhadas indicam uma árvore geradora para $G[T_2]$ (dessa forma fica claro quem são os vértices em T_1 e T_2). Os terminais nas coordenadas (4,33), (8,21), (33,36) e (34,15) não puderam ser conectados, esses terminais estão indicados na figura com um círculo. Como a instância é não ponderada, $z^* = 4$.

2.1.2 Complexidade

Como o problema de decisão associado ao 2PEAS é NP-completo, o problema de otimização tratado nesta tese é claramente NP-difícil. Mas como nas aplicações do PEAS em desenho de circuitos, o grafo G frequentemente é uma grade retangular, achamos relevante incluir uma

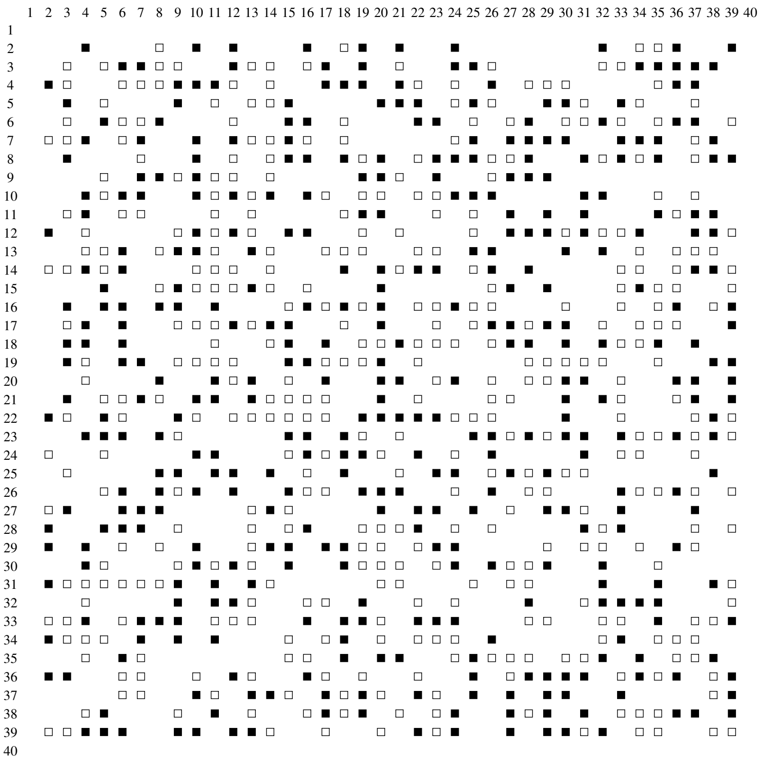

Figura 2.2: Exemplo de uma instância não-ponderada do 2PEAS.

prova de que o problema permanece NP-difícil quando restrito a esse tipo de grafo. Como os grafos de grade são planares e bipartidos, fica também provado que o 2PEAS permanece NP-difícil nesses casos.

2PEAS em grafos de grade
Instância: Um grafo de grade $G = (V, E)$ de dimensão $m \times n$ (os vértices são identificados como $v(i, j)$, $1 \leq i \leq m$, $1 \leq j \leq n$) e pesos inteiros w associados aos vértices (N_1 é definido como o conjunto de vértices com peso negativo e N_2 como os vértices de peso positivo).
Solução: Dois conjuntos de vértices disjuntos T_1 e T_2 tais que $G[T_1]$ e $G[T_2]$ sejam conexos.
Medida: Minimizar $\sum_{v \in (N_2 \setminus T_2)} w_v - \sum_{v \in (N_1 \setminus T_1)} w_v$.

TEOREMA 2.1. *O 2PEAS em grafos de grade é fortemente NP-difícil.*

Prova. Sabe-se que o problema abaixo é NP-difícil (ver problema ND13 em [24]).

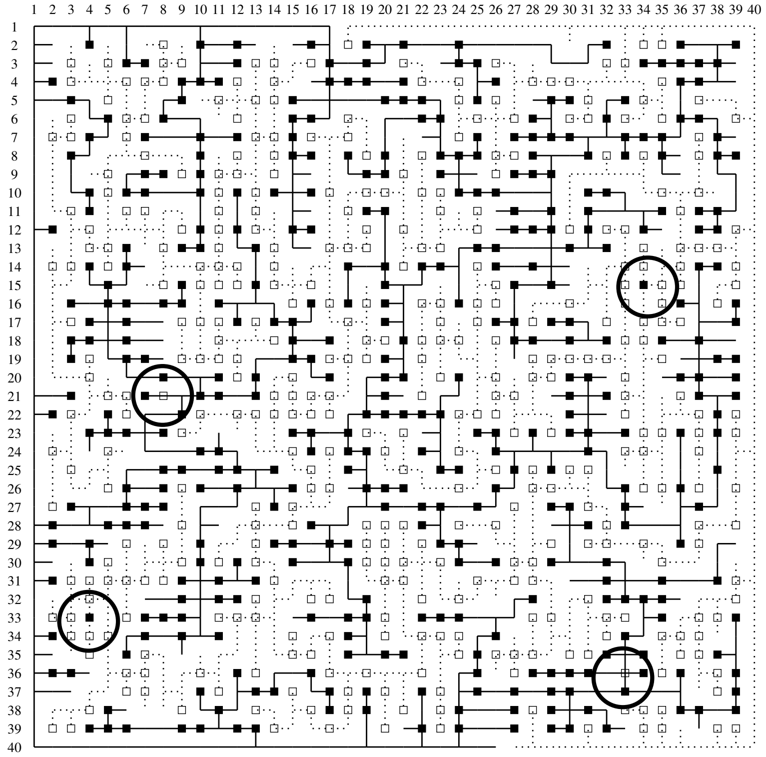

Figura 2.3: Uma solução ótima da instância mostrada na Figura 2.2.

Problema de Steiner em grafos de grade com custos unitários
Instância: Um grafo de grade $G' = (V', E')$ de dimensão $m' \times n'$ (os vértices são identificados como $v'(i, j)$, $1 \leq i \leq m'$, $1 \leq j \leq n'$) e um conjunto de vértices terminais S'.
Solução: Uma sub-árvore $T' = (U', F')$ de G' contendo todos os vértices terminais S'.
Medida: Minimizar $|F'|$.

Dada uma instância $I' = \langle G', S' \rangle$ do problema de Steiner em grafos de grade com custos unitários, pode-se produzir uma instância $I = \langle G, w \rangle$ do 2PEAS em grafos de grade da seguinte maneira. O grafo G é um grafo de grade $m \times n$, onde $m = 2m' + 1$ e $n = 2n' + 1$. Defina C como $|E'|$. Vértices $v(i, j)$ em que i e j são ambos pares, recebem peso $-C$ caso $v'(i/2, j/2) \in S'$ e peso 0 caso contrário. Vértices $v(i, j)$ em que i é par, j é ímpar e $2 < j < n - 1$, recebem peso 1. Vértices $v(i, j)$ em que j é par, i é ímpar e $2 < i < m - 1$, também recebem peso 1. Os vértices restantes recebem peso C. As soluções ótimas de I' e I têm o mesmo custo. Como o valor de C é limitado por um polinômio no tamanho de I', fica provado o teorema. $\square$

A Figura 2.4 ilustra a redução descrita acima. A parte esquerda da figura mostra uma instância de um problema de Steiner, os vértices em cinza são os terminais. A parte direita da figura mostra a instância do 2PEAS gerada.

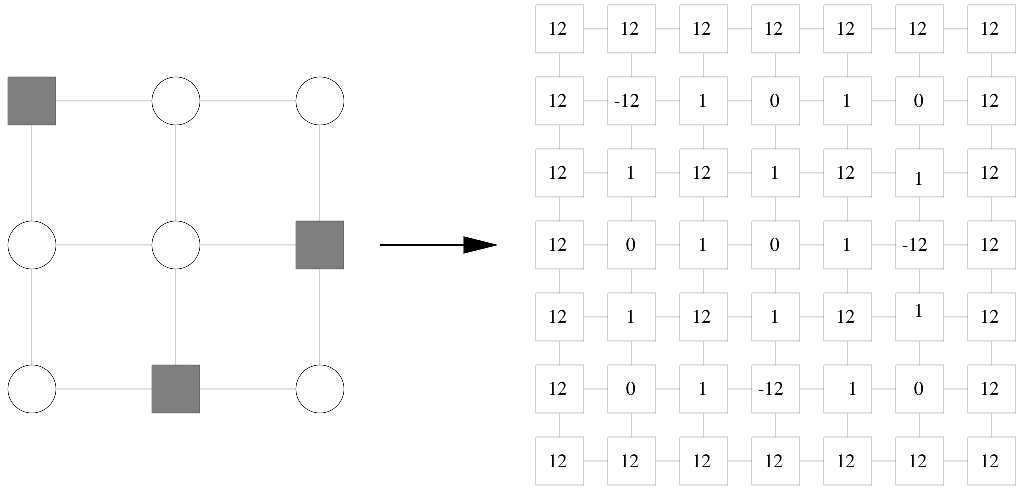

Figura 2.4: Redução de um problema de Steiner a um 2PEAS.

Essa prova pode ser estendida para mostrar que o 2PEAS em grafos de grade permanece NP-difícil quando restrito a instâncias não-ponderadas. A idéia é transformar o grafo de grade G obtido na redução acima em um grafo de grade $(2C + 1)m \times (2C + 1)n$, trocando cada vértice de G por um "bloco" (uma subgrade) de dimensão $(2C + 1) \times (2C + 1)$. Os vértices com peso $-C$ são substituídos por um bloco preenchido com vértices de peso -1, os de peso C por blocos com vértices de peso 1, os de peso 0 por blocos com vértices de peso 0. Já os blocos correspondentes aos vértices de peso 1 têm uma estrutura mais complexa, mostrada na Figura 2.5. A solução do 2PEAS não-ponderado resultante tem o mesmo custo da solução do problema de Steiner em grafos de grade com custos unitários original. Como $C = |E'|$, o tamanho da instância reduzida é limitado por um polinômio no tamanho da instância original.

2.1.3 Formulação

Propomos a seguinte formulação do 2PEAS como um problema de programação linear inteira. Para cada vértice $i \in V$, defina uma variável binária x_i, em que $x_i = 1$ se i pertence ao conjunto T_1 e $x_i = 0$ se i pertencer a T_2. O conjunto de terminais N_1 corresponde aos vértices com peso w_i negativo e N_2 aos vértices com peso positivo. Um conjunto de vértices R é um *separador* do conjunto de vértices S se qualquer caminho unindo dois vértices S passa por pelo menos um vértice de R. O símbolo $\Delta(S)$ representa o conjunto dos separadores minimais

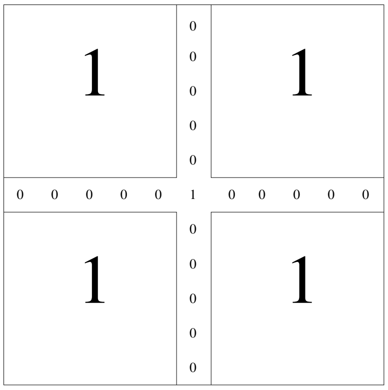

Figura 2.5: Redução a um 2PEAS não-ponderado: esquema do bloco correspondente a um vértice de peso 1.

de S.

$$
(\mathrm{F})\left\{\begin{array}{lllr}
\text{Min } z = & -\sum\limits_{i\in N_1} w_i + \sum\limits_{i\in V} w_i x_i & & (1)\\
\text{s.t.} & \sum\limits_{i\in S} x_i - \sum\limits_{i\in R} x_i \leq 1 & \forall S\subseteq V,\ |S|=2;\ \forall R\in\Delta(S) & (2)\\
 & -\sum\limits_{i\in S} x_i + \sum\limits_{i\in R} x_i \leq |R|-1 & \forall S\subseteq V,\ |S|=2;\ \forall R\in\Delta(S) & (3)\\
 & x_i \in \{0,1\} & \forall i\in V & (4)
\end{array}\right.
$$

As restrições (F.2) dizem que se ambos os vértices de S estão em T_1, então pelo menos um vértice em cada separador de S deve estar também em T_1. As restrições (F.3) são complementares a (F.2): se ambos os vértices de S estão em T_2, um vértice em cada separador de S deve estar em T_2.

A definição das variáveis em (F) leva a soluções em que (T_1, T_2) obrigatoriamente particionam V. Isso não muda o problema, pois para qualquer solução ótima do 2PEAS de "tipo empacotamento" existe também uma solução de "tipo partição" com o mesmo custo. Basta adicionar os vértices de $V - T_1 - T_2$ arbitrariamente a T_1 ou a T_2. Por exemplo, na instância mostrada na Figura 2.1, a solução $T_1 = \{d, f, g, h, j, k\}$ e $T_2 = \{a, b, c, e, i\}$ pode ser convertida numa solução de tipo partição de mesmo custo incluindo-se o vértice l em T_1 ou em T_2.

2.2 Estudo Poliedral

Estudamos o poliedro associado à formulação acima do 2PEAS, com objetivo de identificar novas desigualdades válidas para o problema e assim reforçar essa formulação. É particularmente interessante identificar classes de desigualdades que definem facetas, pois tais desigualdades são as melhores possíveis do ponto de vista teórico. Iniciamos esse estudo com uma pequena digressão conceitual.

DEFINIÇÃO 2.1. *Sejam P_1 e P_2 dois poliedros em $\mathbb{R}^n$. Definimos a* junção *de P_1 e P_2 como o poliedro dado pela envoltória convexa dos pontos inteiros que pertencem tanto a P_1 quanto a P_2. Se P é a junção de P_1 e P_2 (ou seja, $P = Conv(P_1 \cap P_2 \cap Z^n)$, podemos dizer que P_1 e P_2 são superpoliedros (ou relaxações) de P. Nesse contexto, as facetas de P são classificadas em:*

1. Facetas de superpoliedro*, que são as facetas definidas por desigualdades que também definem facetas de P_1 ou de P_2.*
2. Facetas de junção*, que são as facetas definidas por desigualdades que não são válidas nem para P_1 nem para P_2.*

As desigualdades apenas válidas para P, mas que não são válidas nem para P_1 nem para P_2, são chamadas de desigualdades de junção*.*

A Figura 2.6 contém uma ilustração desse conceito em $\mathbb{R}^2$. Se o poliedro P_1 ocupa as áreas marcadas com 1, 2 e 3 e P_2 as áreas 1, 2 e 4, a junção de P_1 e P_2 é o poliedro P que ocupa apenas a área 1. As facetas desse poliedro são descritas pelas desigualdades $x_1 \geq 0$, $x_1 \leq 1$, $x_2 \geq 0$ e $x_2 \leq 1$. As três primeiras desigualdades definem facetas de superpoliedro, enquanto a última define uma faceta de junção. Devido a existência dessa faceta de junção, mesmo uma descrição completa de P_1 e de P_2 não é suficiente para descrever P. Um exemplo de desigualdade de junção que não chega a definir uma faceta de P seria $-x_1 + 2x_2 \leq 2$.

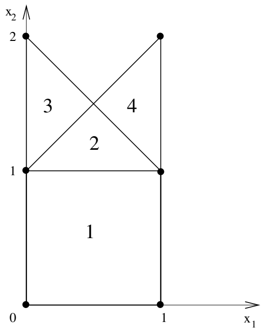

Figura 2.6: Exemplo de poliedro de junção.

Muitos poliedros associados a problemas de otimização combinatória NP-difíceis podem ser vistos como a junção de superpoliedros mais simples. Por exemplo, o poliedro do caixeiro-viajante num grafo $G = (V, E)$ é a junção de um poliedro do problema da *1-árvore* (encontrar um subconjunto $E' \subseteq E$ de custo mínimo em que $|E'| = |V|$ e $G' = (V, E')$ seja conexo) com um poliedro do problema do *2-emparelhamento* (encontrar um subconjunto $E' \subseteq E$

de custo mínimo em que cada vértice de $G' = (V, E')$ tenha grau 2). Nesse caso, ambos os superpoliedros estão associados a problemas polinomiais e são completamente conhecidos (ver [30] por exemplo).

Vamos agora definir o poliedro do 2PEAS como uma junção de poliedros.

DEFINIÇÃO 2.2. *Dado um grafo $G = (V, E)$, uma* v-árvore de *G é um conjunto $T \subseteq V$ tal que $G[T]$ é conexo. O* complemento de uma v-árvore *de G é um conjunto $T \subseteq V$ tal que $G[V \setminus T]$ é conexo. Uma* dv-árvore de *G é um conjunto T que é tanto uma v-árvore quanto o complemento de uma v-árvore.*

A expressão *v-árvore* é uma abreviação para *árvore de vértices*, que pela definição acima é um conjunto de vértices que induz um subgrafo de G contendo uma árvore. Já *dv-árvore* abrevia *árvore dupla de vértices*.

DEFINIÇÃO 2.3. *Seja χ^T o vetor de incidência de um conjunto de vértices T. Defina $P_1(G)$ como a envoltória convexa dos vetores χ^T em que T é uma v-árvore de G e $P_2(G)$ como a envoltória dos χ^T em que T é o complemento de uma v-árvore de G. Defina $P(G)$ como a junção de P_1 e P_2 (ou seja, a envoltória convexa dos vetores χ^T em que T é uma dv-árvore de G).*

Algumas observações:

- Os vetores que definem os poliedros P correspondem às soluções da formulação (F), assim P está associado ao 2PEAS.

- Os superpoliedros P_1 e P_2 estão associados a um tipo de problema de Steiner em grafos com pesos nos vértices: dado um grafo com pesos positivos ou negativos nos vértices, encontrar uma v-árvore com o mínimo (ou máximo) peso total. Esse problema é NP-completo e, portanto, não esperamos encontrar uma descrição completa desses poliedros.

- Os superpoliedros P_1 e P_2 são essencialmente idênticos, sua única diferença é estarem complementados: um vetor x, $0 \leq x \leq 1$, pertence a P_1 se e somente se $1 - x$ pertence a P_2. Isso implica que qualquer desigualdade válida para um deles pode ser convertida em uma desigualdade válida para o outro por complementação de variáveis, ou seja, a substituição de x por $1 - x$. Por exemplo, as desigualdades (F.2) são válidas para P_1 e e as desigualdades complementares (F.3) são válidas para P_2. Dessa forma, sempre que falarmos sobre o poliedro P_1, fica implícito que resultados similares também valem para P_2.

Há casos, como o problema do caixeiro-viajante, em que uma descrição completa dos superpoliedros já é suficiente para se ter uma descrição bastante razoável do poliedro de junção.

No caso do 2PEAS isso não acontece, como ficou mostrado em experiências com geração de colunas [49] (através dessa técnica é possível trabalhar com uma descrição completa implícita de superpoliedros associados a problemas NP-difíceis, pelo menos para instâncias pequenas). O uso de desigualdades de junção é absolutamente essencial para o bom funcionamento de um algoritmo de *branch-and-cut* para o 2PEAS.

Usamos os superpoliedros P_1 e P_2 como ponto de partida conceitual para descobrir algumas dessas desigualdades de junção. A idéia foi tentar descobrir em que condições algumas facetas importantes de P_1 ou P_2 também definem facetas de superpoliedro de P. Quando essas condições não são satisfeitas, há espaço para fazer um levantamento[1], gerando novas desigualdades de junção. Através desse tipo de raciocínio foi possível encontrar importantes famílias de facetas de junção do 2PEAS.

TEOREMA 2.2. *Os poliedros $P(G)$, $P_1(G)$ e $P_2(G)$ têm dimensão plena.*

Prova. Seja T uma árvore geradora de G, enraizada num vértice arbitrário. Para cada $i \in V$, o conjunto de vértices U_i contendo i e seus descendentes em T é uma dv-árvore, assim $\chi^{U_i} \in P(G)$. Como $\chi^{\emptyset} \in P(G)$, existem $|V| + 1$ vetores afim independentes em $P(G)$. Já que $P(G) \subseteq P_1(G) \cap P_2(G)$, fica provado o teorema. □

2.2.1 As facetas de p-separadores de P_1

DEFINIÇÃO 2.4. *Seja S um conjunto de vértices tal que $|S| = p \geq 2$. Um conjunto de vértices R é um* separador de *S se qualquer caminho unindo um par de vértices em S tem pelo menos um vértice em R. Um separador R de S é* próprio *se $S \cap R = \emptyset$ e* minimal *se nenhum outro separador de S está contido em R. O símbolo $\Delta(S)$ indica o conjunto de todos os separadores próprios e minimais de S. Dado um separador R de S e um vértice $i \in V$, defina S_i^R como o maior subconjunto de vértices em S que podem pertencer a uma v-árvore T_i contendo i e tal que $T_i \cap R \subseteq \{i\}$. Nota-se que um separador R de S é minimal se e somente se $|S_i^R| \geq 2$ para cada $i \in R$.*

Por exemplo, no grafo $G1$ (Figura 2.7) tomando $S = \{a, c, h\}$, o conjunto $R = \{b, e, k, i\}$ é um separador em $\Delta(S)$, $S_b^R = \{a, c, h\}$, $S_e^R = \{a, h\}$, $S_j^R = \{a\}$ e $S_g^R = \emptyset$.

DEFINIÇÃO 2.5. *Um separador R de $S = \{s_1, \ldots, s_p\}$ induz a seguinte estrutura em G. O grafo $G[V \setminus R]$ é separado em um certo número de componentes conexos: aqueles contendo exatamente um vértice em S são chamados de* A-componentes *e aqueles sem vértices em S de* B-componentes. *Defina A_i como o conjunto de vértices do A-componente contendo s_i (quando R não é próprio alguns A-componentes não existem). Os conjuntos de vértices que correspondem aos B-componentes são numerados arbitrariamente como $B_1, \ldots, B_q$. Notar que $|S_i^R| = 0$ se i pertence a um B-componente*

[1]Do inglês *lifting*.

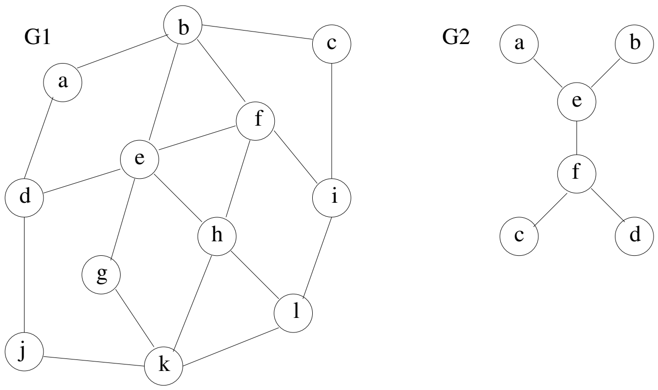

Figura 2.7: Grafos para exemplos.

e $|S_i^R| = 1$ *se i pertence a um A-componente.*

Por exemplo, no grafo $G1$, tomando $s_1 = a$, $s_2 = c$, $s_3 = h$ e $R = \{b, e, k, i\}$, $A_1 = \{a, d, j\}$, $A_2 = \{c\}$, $A_3 = \{f, h, l\}$ e $B_1 = \{g\}$.

O seguinte teorema sugere que o conceito de separador é fundamental para se entender a estrutura facial de P_1.

TEOREMA 2.3. *Todas as facetas de $P_1(G)$ podem ser definidas por uma desigualdade trivial ($-x_i \leq 0$ ou $x_i \leq 1$) ou por uma desigualdade no formato $\sum_{i \in S} \beta_i x_i - \sum_{i \in R} \alpha_i x_i \leq 1$, onde β e α são vetores de números positivos e R é um separador próprio de S.*

Prova. Suponha que uma faceta F de $P_1(G)$ é definida por $a^t x \leq a_0$. O lado direito a_0 é não-negativo, senão essa desigualdade não seria válida para a v-árvore vazia. Sem perda de generalidade podemos supor que $a_0 = 0$ ou $a_0 = 1$. Seja S o conjunto dos vértices que correspondem a coeficientes positivos em a e R os vértices correspondendo a coeficientes negativos.

Caso (a): $a_0 = 0$. O conjunto S deve ser vazio, de outra forma $a^t x \leq 0$ não seria válida para a v-árvore formada por um único vértice em S. O conjunto R deve conter um único vértice u, pois se existir algum outro vértice em R, $a^t x \leq 0$ seria dominada por $-x_u \leq 0$. Assim F é definida por essa desigualdade trivial.

Caso (b): $a_0 = 1$. S não pode ser vazio. Se S é o conjunto unitário $\{u\}$, F pode ser definida pela desigualdade trivial $x_u \leq 1$. Suponha que $|S| \geq 2$. O grafo $G[V \setminus R]$ é dividido em um certo número de componentes conexos. Nenhum par de vértices u e v em S pode pertencer ao mesmo componente, senão $a^t x \leq 1$ seria uma combinação linear das desigualdades válidas $\sum_{i \in S \setminus \{u,w\}} a_i x_i + (a_u + a_w) x_u + \sum_{i \in R} a_i x_i \leq 1$ e $\sum_{i \in S \setminus \{u,w\}} a_i x_i + (a_u + a_w) x_w + \sum_{i \in R} a_i x_i \leq 1$. Logo R é um separador próprio de S. $\square$

Definimos agora uma classe de desigualdades com o potencial de definir facetas de P_1:

DEFINIÇÃO 2.6. *Seja S um conjunto de vértices, onde $|S| = p \geq 2$, e $R \in \Delta(S)$. As desigualdades no formato abaixo são chamadas de p-*separadores*, onde α_i é um inteiro maior ou igual a $|S_i^R| - 1$.*

$$\sum_{i \in S} x_i - \sum_{i \in R} \alpha_i x_i \leq 1$$

As desigualdades em (F.2) são 2-separadores. No grafo $G1$, $x_a + x_f - x_b - x_e - x_j \leq 1$ é um exemplo dessa desigualdade. No mesmo grafo, $x_a + x_f + x_g - x_b - 2x_e - 2x_k \leq 1$ é um 3-separador válido para $P_1(G1)$. Tomar $\alpha_i = |S_i^R| - 1$, para todo $i \in R$, nem sempre leva a uma desigualdade válida para P_1. Por exemplo, no grafo $G2$ (Figura 2.7), se $S = \{a, b, c, d\}$ e $R = \{e, f\}$, o 4-separador $\sum_{i \in S} x_i - x_e - x_f \leq 1$ não é válido para $P_1(G2)$. Mas os 4-separadores $\sum_{i \in S} x_i - 2x_e - x_f \leq 1$ e $\sum_{i \in S} -x_e - 2x_f \leq 1$ são válidos e definem facetas.

TEOREMA 2.4. *Para cada S e cada $R \in \Delta(S)$, existe pelo menos uma desigualdade de p-separador que é válida e define uma faceta de $P_1(G)$.*

A prova desse teorema se utiliza de um resultado e de uma definição adicionais.

LEMA 2.1. *(Lema do Levantamento Máximo Sequencial, páginas 261–263 de [48]) Seja P um poliedro em $\mathbb{R}^n$. Seja N o conjunto dos índices $\{1, \ldots, n\}$ e Z_0 um subconjunto de N. Defina P^{Z_0} como o poliedro $P \cap \{x \in \mathbb{R}^n | x_i = 0, i \in Z_0\}$. Suponha que $\sum_{i \in N \setminus Z_0} a_i x_i \leq a_0$ é uma desigualdade válida para P^{Z_0} definindo uma face de dimensão d. Seja $\sum_{i \in N \setminus Z_0} a_i x_i + \sum_{i \in Z_0} m_i x_i \leq a_0$ qualquer desigualdade obtida por um levantamento sequencial máximo de um coeficiente em Z_0 por vez (em qualquer ordem). Essa nova desigualdade define uma face de P de dimensão $d + |Z_0|$, e se a desigualdade original definia uma faceta de P^{Z_0}, a nova desigualdade define uma faceta de P.*

DEFINIÇÃO 2.7. *Seja T uma v-árvore de G e t um vértice em T. Uma* sequência de v-árvores aninhadas de t até T *é qualquer sequência de v-árvores $T_1, \ldots, T_{|T|}$ tal que $\{t\} = T_1 \subset \ldots \subset T_{|T|} = T$. (ou seja, para cada i, $1 < i \leq |T|$, T_{i-1} é obtido de T_i pela remoção de um vértice distinto de t.) Notar que dados uma v-árvore T e o vértice t, é sempre possível construir tal sequência.*

Prova. (Teorema 2.4) Considere a estrutura induzida por R (Definição 2.5). A desigualdade $\sum_{i \in S} x_i \leq 1$ é válida para $P_1(G)^{Z_0}$, $Z_0 = R \cup (\cup_{j=1}^{q} B_j)$. Defina $F = \{x \in P_1(G)^{Z_0} | \sum_{i \in S} x_i = 1\}$. Para cada $j \in \{1, \ldots, p\}$, A_j é uma v-árvore. Construa uma sequência de v-árvores aninhadas de s_j até A_j. Para cada T nessas sequências, $\chi^T \in F$. Assim existem $\sum_{j=1}^{p} |A_j|$ vetores afim independentes em F e $\sum_{i \in S} x_i \leq 1$ define uma faceta de $P_1(G)^{Z_0}$.

Numere os vértices de R de r_1 até $r_{|R|}$. Defina $R_0 = \emptyset$ e R_j como o conjunto $\{r_1, \ldots, r_j\}$. Para cada $k \in \{1, \ldots, q\}$, defina $f(k)$ como o menor número de um vértice em R adjacente a um B-componente B_k. Defina $W_j = R_j \cup \{B_k | f(k) \leq j\}$ e $Z_j = Z_0 \setminus W_j$. Mostraremos que para cada $j \in \{0, \ldots, |R|\}$, existe uma desigualdade $\sum_{i \in S} x_i - \sum_{i \in R_j} \alpha_i x_i \leq 1$, onde cada α_i é inteiro e satisfaz $\alpha_i \geq |S_i^R| - 1$, que define uma faceta de $P_1(G)^{Z_j}$.

Isso é verdadeiro quando $j = 0$. Suponha que $1 \leq j \leq |R|$. Considere a desigualdade correspondente a $j - 1$. Primeiro faça um levantamento máximo do coeficiente de r_j. Como existe uma v-árvore T_j tal que $T_j \cap R = \{r_j\}$, $T_j \cap S = S_j^R$ e $T_j \cap (\cup_{j=1}^{q} B_j) = \emptyset$, $\alpha_j \geq |S_j^R| - 1$. Como o levantamento é máximo, α_j é inteiro e existe uma v-árvore $U \subseteq (\cup_{j=1}^{p} A_j \cup W_{j-1} \cup \{r_j\})$ tal que $r_j \in U$ e $|S \cap U| - \sum_{i \in R_j \cap U} \alpha_i = 1$. Para cada B-componente B_k tal que $f(k) = j$, escolha um vértice $t \in B_k$ adjacente a r_j. Construa uma sequência de v-árvores aninhadas de t até B_k. Faça um levantamento máximo dos coeficientes dos vértices em B_k na mesma ordem em que esses vértices aparecem na sequência. Como $U \cup T$ é uma v-árvore, para cada T na sequência, todos esses coeficientes devem ser zero. Repetindo esse procedimento até que $j = |R|$, um p-separador definindo uma faceta de $P_1(G)$ é obtido. □

Uma consequência imediata desse resultado é a seguinte:

COROLÁRIO 2.1. *Se um p-separador $\sum_{i \in S} x_i - \sum_{i \in R} (|S_i^R| - 1) x_i \leq 1$ é válido para $P_1(G)$, então também define uma faceta de $P_1(G)$.*

A prova do Teorema 2.4, devido ao uso do Lema 2.1, não fornece um algoritmo polinomial para calcular os coeficientes α para que um p-separador defina uma faceta. Mas dados os conjuntos S e $R \in \Delta(S)$, os coeficientes de um p-separador válido para P_1 podem ser computados pelo seguinte procedimento.

```
Procedimento Calcula α
   M ⟵ ∅;
   Para cada i ∈ R {
         Se (existe j ∈ M tal que S_i^R ∩ S_j^R = ∅)
               Então { α_i ⟵ |S_i^R|; }
               Senão { α_i ⟵ |S_i^R| − 1; M ⟵ M ∪ {i}; }
   }
```

Por exemplo, considere o grafo G2 com $S = \{a, b, c, d\}$ e $R = \{e, f\}$. Tomando os vértices na ordem (e, f), o procedimento retorna $\alpha_e = 1$ e $\alpha_f = 2$. Se a ordem é invertida, o procedimento retorna $\alpha_f = 1$ e $\alpha_e = 2$.

TEOREMA 2.5. *O procedimento* Calcula α *está correto.*

Prova. Seja U uma v-árvore qualquer e seja $Q = R \cap U$. Vamos mostrar que $|S \cap U| - \sum_{i \in Q} \alpha_i \leq 1$. Assuma que Q é não-vazio, caso contrário essa desigualdade é claramente verdadeira. Ordene os vértices em Q como $q_1, \ldots, q_{|Q|}$, usando a mesma ordem em que esses vértices foram considerados no procedimento. Defina Q_i como o conjunto $\{q_1, \ldots, q_i\}$ e $S^R_{Q_i}$ como $\cup_{j \in Q_i} S^R_j$.

Afirmamos que para cada $i \in \{1, \ldots, |Q|\}$, $\gamma_i = |S^R_{Q_i} \cap U| - \sum_{i \in Q_i} \alpha_i \leq 1$. Isso é verdadeiro quando $i = 1$. Suponha que $1 < i \leq |Q|$. Se $\alpha_i = |S^R_i|$, então $\gamma_i \leq \gamma_{i-1} \leq 1$. Suponha que $\alpha_i = |S^R_i| - 1$. O procedimento garante que se existe um vértice $j \in Q_{i-1}$ com $\alpha_j = |S^R_j| - 1$, então $S^R_i \cap S^R_j \neq \emptyset$, e $\gamma_i \leq \gamma_{i-1} \leq 1$. Se não há tal vértice, $\gamma_{i-1} \leq 0$, logo $\gamma_i \leq 1$. Como $|S \cap U| - \sum_{i \in Q} \alpha_i \leq \gamma_{|Q|} \leq 1$, o resultado fica provado. □

Pode-se notar que se $p \leq 3$, o procedimento sempre retorna $\alpha_i = |S^R_i| - 1$ para todo $i \in R$. Assim, pelo Corolário 2.1, todos esses 2-separadores e 3-separadores definem facetas de $P_1(G)$. O procedimento *Calcula* α não é ótimo, se $p \geq 4$ nem sempre se consegue chegar a um p-separador que defina faceta.

Encerramos aqui o estudo dos poliedros P_1. Os resultados obtidos já tornam aceitável a afirmação que os p-separadores são as "principais" facetas de P_1, e que portanto, essas desigualdades podem fornecer um bom ponto de partida para a investigação do poliedro do 2PEAS.

2.2.2 De P_1 a P: p-separadores, p-separadores levantados e p-cruzes

Um p-separador que define uma faceta de $P_1(G)$ pode definir uma faceta de superpoliedro de $P(G)$, assim é interessante descobrir em que condições isso acontece. Ainda mais interessante é descobrir condições em que isso não acontece: nesse caso há espaço para levantar o p-separador e obter novas desigualdades de junção.

DEFINIÇÃO 2.8. *As desigualdades válidas para $P(G)$ que podem ser obtidas através do levantamento de coeficientes α de um p-separador que defina uma faceta de $P_1(G)$ ou de $P_2(G)$, são chamadas de* p-separadores levantados.

Uma instância do PEAS sobre um grafo G que tenha um vértice separador pode ser dividida em duas ou mais instâncias menores. Dessa forma, os teoremas dessa seção podem supor que G é biconexo sem perda de generalidade do ponto de vista algorítimico. A definição e o lema seguintes são usados nas provas desses resultados.

DEFINIÇÃO 2.9. *Seja T uma dv-árvore de G e t um vértice em T. Uma* sequência de dv-árvores aninhadas de t até T *é qualquer sequência de dv-árvores $T_1, \ldots, T_{|T|}$ tal que $\{t\} = T_1 \subset \ldots \subset T_{|T|} = T$.*

O procedimento de construção de sequências de dv-árvores aninhadas não parece ser trivial.

LEMA 2.2. *Suponha que G é biconexo. Dada uma dv-árvore T de G e $t \in T$, é sempre possível construir uma sequência de dv-árvores aninhadas de t até T.*

Prova. Se $T = \{t\}$, o resultado é imediato. Senão, suponha que uma sequência parcial $T = T_{|T|} \supset \ldots \supset T_i$, $|T| \geq i > 1$, já está formada. Vértices na dv-árvore T_i, distintos de t, mas adjacentes a algum vértice em $V \setminus T_i$ são chamados de *vértices candidatos* (se $T = V$, todos os vértices em $V - t$ são candidatos). Como G é biconexo, existe pelo menos um vértice candidato. Mostraremos que é sempre possível escolher um candidato para ser removido de T_i para formar a dv-árvore T_{i-1}.

Seja v_0 um candidato. Se $T_i - v_0$ é uma v-árvore, v_0 é escolhido. Caso contrário $G[T_i - v_0]$ está separado em $j \geq 2$ componentes conexos, cujos conjuntos de vértices são $C_1^1, \ldots, C_j^1$. Como T_i é uma v-árvore, pelo menos um vértice em C_j^1 é adjacente a v_0. Como G é biconexo, pelo menos um vértice em cada C_j^1 é adjacente a $V \setminus T_i$. Ajuste a notação para que C_1^1 seja um conjunto que não contenha t. Seja v_1 um vértice candidato em C_1^1. O subgrafo $G[C_1^1 - v_1]$ está separado em $j \geq 0$ componentes conexos, cujos conjuntos de vértices são $C_1^2, \ldots, C_j^2$. Como C_1^1 é uma v-árvore, pelo menos um vértice em cada C_j^2 é adjacente a v_1. Como G é biconexo, pelo menos um vértice em cada C_j^2 é adjacente a v_0 ou a $V \setminus T_i$. Se pelo menos um vértice em cada C_j^2 é adjacente a v_0 (ou $j = 0$), v_1 é escolhido. Caso contrário, ajuste a notação para que C_1^2 seja um conjunto contendo um vértice candidato.

Repetindo esse procedimento, obtemos uma sequência $v_1, \ldots, v_k$ de vértices candidatos contidos em conjuntos C_1^k de tamanho decrescente. Se cada componente de $G[C_1^k - v_k]$ é adjacente a $T_i \setminus C_1^k$, v_k é escolhido e $T_i - v_k$ é uma v-árvore. No pior caso, o procedimento é repetido até que $C_1^k = \{v_k\}$. □

TEOREMA 2.6. *Suponha que G é biconexo. Seja $\sum_{i \in S} x_i - \sum_{i \in R}(|S_i^R| - 1)x_i \leq 1$ um p-separador válido para $P(G)$, tal que R induza uma estrutura sem B-componentes e que cada A-componente seja uma dv-árvore. Se para cada $i \in R$, existir uma dv-árvore U_i contendo $\{i\} \cup S_i^R$ tal que $U_i \cap R = \{i\}$, essa desigualdade define uma faceta de $P(G)$.*

Prova. Seja $a^t x \leq a_0$ uma desigualdade válida para $P(G)$ tal que

$$F = \{x \in P(G) \mid \sum_{i \in S} x_i - \sum_{i \in R}(|S_i^R| - 1)x_i = 1\} \subseteq F_a = \{x \in P(G) \mid a^t x = a_0\} \ .$$

Mostraremos que $a^t x \leq a_0$ é um múltiplo escalar desse p-separador. Considere a estrutura induzida por R. Para cada A-componente A_j, construa uma sequência de dv-árvores aninhadas de s_j até A_j. Para cada dv-árvore T nessa sequência, $\chi^T \in F$. Logo, $a_{s_j} = a_0$ e para cada $i \neq s_j \in A_j$, $a_i = 0$. Para cada vértice $i \in R$, $\chi^{U_i} \in F$, onde U_i é a dv-árvore dada pela hipótese. Assim $a_i = -(|S_i^R| - 1)a_0$. □

Por exemplo, no grafo G1, o 2-separador $x_a + x_f - x_b - x_e - x_j \leq 1$, o 3-separador $x_a + x_f + x_g - x_b - 2x_e - 2x_k \leq 1$ e o 4-separador $x_a + x_c + x_g + x_h - 2x_b - 2x_e - x_i - 2x_k \leq 1$ definem facetas de $P(G1)$. O reverso do Teorema 2.6 não vale mesmo quando $p = 3$. É necessária uma condição adicional para garantir que se pode fazer o levantamento de um coeficiente α.

TEOREMA 2.7. *Suponha que G é biconexo. Seja $\sum_{i\in S} x_i - \sum_{i\in R}(|S_i^R| - 1)x_i \leq 1$ um p-separador válido para $P(G)$, tal que R induza uma estrutura sem B-componentes e que cada A-componente seja uma dv-árvore. Suponha que exista um vértice $i \in R$ que não satisfaz as condições do Teorema 2.6. Se $|S_i^R| = p$, o coeficiente α_i pode ser levantado em pelo menos uma unidade.*

Prova. Mostraremos que para cada dv-árvore T contendo i, $\gamma_T = |S \cap T| - \sum_{i\in R\cap T} \alpha_i \leq 0$. Se T contém S_i^R, $|S \cap T| = |S_i^R|$. Como T deve conter outro vértice j de R, $\gamma_T \leq |S_i^R| - (|S_i^R| - 1) - (|S_j^R| - 1) = 2 - |S_j^R| \leq 0$. Se T não contém S_i^R, $\gamma_T \leq (|S_i^R| - 1) - (|S_i^R| - 1) \leq 0$. □

Se $p = 2$, os dois teoremas anteriores podem ser resumidos da seguinte forma:

COROLÁRIO 2.2. *Suponha que G é biconexo. Seja $\sum_{i\in S} x_i - \sum_{i\in R} x_i \leq 1$ um 2-separador, onde R induz uma estrutura sem B-componentes. Essa desigualdade define uma faceta de $P(G)$ se e somente se para cada $i \in R$, existir uma dv-árvore U_i contendo $\{i, s_1, s_2\}$ tal que $U_i \cap R = \{i\}$.*

Por exemplo, no grafo G1, o 2-separador $x_a + x_c - x_b - x_e - x_h - x_l \leq 1$ não define uma faceta de $P(G1)$, pois pelo Corolário 2.2, α_e ou α_h podem ser levantados a 0. O 3-separador $x_a + x_c + x_g - x_b - 2x_e - 2x_k \leq 1$ também não define uma faceta de $P(G1)$, pois pelo Teorema 2.7, α_e pode ser levantado a 1.

As restrições à estrutura induzida por R no enunciado nos Teoremas 2.6 e 2.7 têm a sua razão de ser. Por exemplo, no grafo G3 (Figura 2.8), o 2-separador $x_a + x_b - x_d - x_e - x_f \leq 1$ não define faceta, apesar de satisfazer as condições do Corolário 2.2, exceto que R induz uma estrutura com B-componentes. Esse 2-separador é dominado pelo 3-separador levantado $x_a + x_b + x_c - x_d - x_e - x_f \leq 1$ (do 3-separador $x_a + x_b + x_c - 2x_d - 2x_e - 2x_f \leq 1$). No grafo G4 (Figura 2.8), o 3-separador $x_a + x_c + x_e - x_b - x_d - x_f - x_h \leq 1$ não define faceta, apesar de satisfazer as condições do Teorema 2.6, exceto que o A-componente formado por $\{c, g\}$ não é uma dv-árvore. Esse 3-separador é dominado por $x_a + x_c + x_g + x_e - x_b - x_d - x_f - x_h \leq 1$.

Certamente seria desejável melhorar os resultados obtidos até aqui, para se chegar a uma caracterização completa de quando um p-separador define faceta de P. Mais importante ainda, seria determinar quais os máximos levantamentos que podem ser feitos a um p-separador, de modo a se obter facetas de junção de P. Fizemos uma tentativa nesse sentido, mas muitas dificuldades técnicas apareceram. Os teoremas resultantes eram pouco gerais e de demonstração complexa. Mas havia uma "lógica" que parecia estar subjacente aos casos em que um

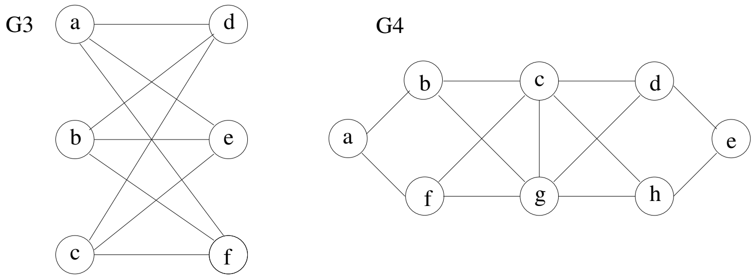

Figura 2.8: Grafos para exemplos.

p-separador podia ser levantado: *os possíveis caminhos entre pares de vértices em R funcionavam como parte de um "separador implícito" entre os vértices de S.* A partir dessa lógica foi possível chegar a uma definição direta e elegante de uma importante subclasse de p-separadores levantados:

DEFINIÇÃO 2.10. *Sejam $S = \{s_1, \dots, s_p\}$ e $R = \{r_1, \dots, r_p\}$ dois conjuntos disjuntos de vértices, tais que $p \geq 2$. Suponha que para quaisquer inteiros i_1, i_2, i_3, i_4, dentro da faixa $\{1, \dots, p\}$ e satisfazendo $i_1 \leq i_2 < i_3 \leq i_4$, não existam dois caminhos disjuntos nos vértices unindo s_{i_1} a s_{i_3} e r_{i_2} a r_{i_4} (ou seja, os vértices em cada caminho de r_{i_2} a r_{i_4} definem um separador entre $\{s_{i_1}, s_{i_3}\}$ e vice-versa). Então a seguinte desigualdade é chamada de p-*cruz

$$\sum_{i \in S} x_i - \sum_{i \in R} x_i \leq 1$$

No grafo G1, $x_e + x_j - x_d - x_g \leq 1$ é uma 2-cruz e $x_a + x_c + x_l + x_j - x_b - x_i - x_k - x_d \leq 1$ é 4-cruz. As p-cruzes são basicamente p-separadores levantados, basta "completar" R (ou S) até que um separador de S (ou R) apareça. Por exemplo, a 3-cruz $x_a + x_c + x_k - x_b - x_l - x_d \leq 1$ é um levantamento do 3-separador $x_a + x_c + x_k - 2x_b - x_f - x_l - x_d \leq 1$ (faceta de $P_1(G1)$) ou do 3-separador $-x_b - x_l - x_d + x_a + x_c + 2x_e + x_f + x_k \leq 4$ (faceta de $P_2(G1)$).

É importante saber que a propriedade dos grafos usada na Definição 2.10 está bem caracterizada:

TEOREMA 2.8 (THOMASSEN, SEYMOUR). *(Ver página 65 de [40]) Seja G um grafo tal que nenhum vértice i separa ao mesmo tempo s_{i_1} de s_{i_3} e s_{i_2} de s_{i_4}. Não existem caminhos disjuntos nos vértices unindo s_{i_1} a s_{i_3} e s_{i_2} a s_{i_4} se e somente se G é obtido de um grafo planar G' onde esses quatro vértices estão na face externa na ordem $s_{i_1}, s_{i_2}, s_{i_3}, s_{i_4}$, colocando-se um grafo arbitrário em algumas faces de G' limitadas por 3 arestas.*

Por exemplo, o grafo obtido colocando-se um K_5 dentro da face de $G1$ limitada por (e, f), (f, h) e (h, e) (ou seja, adicionando-se algumas arestas ligando o K_5 a e, f ou h) é não-planar. Entretanto, continuam não existindo caminhos disjuntos nos vértices entre b e k e entre i e d

nesse novo grafo.

TEOREMA 2.9. *As p-cruzes são válidas para P.*

Prova. Por indução em p. As 2-cruzes são claramente válidas. Suponha que $\sum_{i=1}^{p} x_{s_i} - \sum_{i=1}^{p} x_{r_i} \leq 1$ é uma p-cruz com $p \geq 3$. Pela hipótese de indução, as $(p-1)$-cruzes $\sum_{i=1}^{p-1} x_{s_i} - \sum_{i=1}^{p-1} x_{r_i} \leq 1$ e $\sum_{i=2}^{p} x_{s_i} - \sum_{i=2}^{p} x_{r_i} \leq 1$ são válidas. Somando essas desigualdades com a 2-cruz $x_{s_1} + x_{s_p} - x_{r_1} - x_{r_p} \leq 1$, dividindo o resultado por 2 e arredondando para baixo o lado direito, a p-cruz desejada é obtida. □

Por exemplo, no grafo G1, a 3-cruz $x_a + x_c + x_k - x_b - x_l - x_d \leq 1$ pode ser obtida pela soma e arredondamento das 2-cruzes $x_a + x_c - x_b - x_l \leq 1$, $x_c + x_k - x_b - x_l \leq 1$ e $x_a + x_k - x_b - x_d \leq 1$.

TEOREMA 2.10. *Se G é biconexo, todas as 2-cruzes definem facetas de $P(G)$.*

Prova. Seja $x_{s_1} + x_{s_2} - x_{r_1} - x_{r_2} \leq 1$ uma 2-cruz e seja $a^t x \leq a_0$ uma desigualdade válida para $P(G)$ tal que

$$F = \{x \in P(G) \,|\, x_{s_1} + x_{s_2} - x_{r_1} - x_{r_2} = 1\} \subseteq F_a = \{x \in P(G) \,|\, a^t x = a_0\} \ .$$

Mostraremos que $a^t x \leq a_0$ é um múltiplo escalar dessa 2-cruz.

Como G é biconexo, o Teorema 2.8 implica na existência de um ciclo C contendo s_1, r_1, s_2, r_2 nessa ordem. Seja I_1 o caminho sobre C de r_1 até r_2 passando por s_1 e seja I_2 o caminho sobre C de r_1 até r_2 por s_2. Seja A_1 o conjunto dos vértices que podem ser alcançados a partir de s_1 sem passar por I_2 e A_2 é o conjunto dos vértices que podem ser alcançados a partir de s_2 sem passar por I_1. A_1 e A_2 são dv-árvores (que podem não ser disjuntas). Construa sequências de dv-árvores aninhadas de s_1 até A_1 e de s_2 até A_2. Para cada T nessas sequências, $\chi^T \in F$. Logo, $a_{s_1} = a_{s_2} = a_0$ e $a_i = 0$ se $i \in (A_1 \cup A_2) \setminus S$. Como $\chi^{A_1 \cup A_2 \cup \{r_1\}}$ e $\chi^{A_1 \cup A_2 \cup \{r_2\}} \in F$, $a_{r_1} = a_{r_2} = -a_0$. Os únicos vértices ainda não cobertos estão agrupados em componentes conexos, numerados de B_1 até B_q. Cada um desses componentes deve ser adjacente tanto a r_1 quanto a r_2. Seja j um vértice no componente B_k adjacente a r_1. Construa uma sequência de dv-árvores aninhadas de j até B_k. Para cada T nessa sequência $U_T = A_1 \cup A_2 \cup \{r_1\} \cup T$ também é uma dv-árvore e $\chi^{U_T} \in F$. Assim, $a_i = 0$, para todo $i \in B_k$. □

Uma prova muito semelhante mostra que as p-cruzes com $p \geq 3$ também definem facetas de $P(G)$ quando G é biconexo e existe um ciclo cobrindo os conjuntos S e R. Pode-se dizer que praticamente todas as p-cruzes definem facetas.

Terminamos essa subseção com uma ilustração esquemática do processo de levantamento de p-separadores no caso planar, que fornece intuição para o processo de levantamento de

p-separadores geral. Na Figura 2.9, o círculo representa a face externa de um grafo planar, s_1 está nessa face, mas s_2 e s_3 não. Os segmentos de reta representam um separador R de S. Nesse caso, o 3-separador correspondente define uma faceta (supondo que algumas "condições técnicas" descritas no enunciado do Teorema 2.6 estejam satisfeitas, o objetivo desse parágrafo é justamente ignorar esses detalhes). Já na esquerda da Figura 2.10, s_1 e s_2 estão na mesma face. Isso faz com que o 3-separador possa ser levantado para a desigualdade esquematizada na direita da mesma figura, o vértice r_1 é a "ponta" do separador na parte da face entre s_1 e s_2. Finalmente na esquerda da Figura 2.11, os três vértices S estão na mesma face, e o 3-separador pode ser levantado até uma 3-cruz.

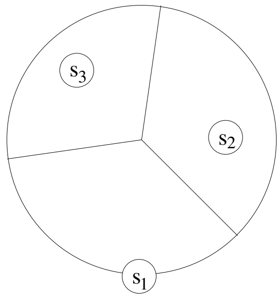

Figura 2.9: Esquema de um 3-separador que define faceta em um grafo planar.

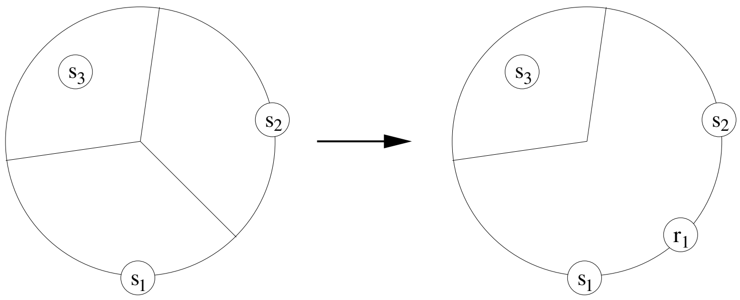

Figura 2.10: Esquema do levantamento de um 3-separador em um grafo planar.

2.3 Algoritmos de Separação

As classes de desigualdades encontradas na seção anterior só podem ter uso prático em um algoritmo de *branch-and-cut* caso se disponha de correspondentes algoritmos de separação. Ou seja, algoritmos capazes de identificar desigualdades pertecentes a uma certa classe que estejam sendo violadas por $\bar{x}$, a solução do programa linear corrente. Nessa seção mostramos algoritmos de separação exatos para 2-separadores e p-cruzes e um algoritmo de separação heurístico para 3-separadores. Como os nossos experimentos computacionais foram realizados

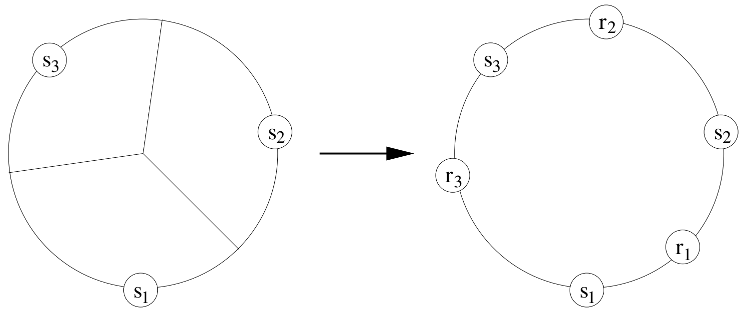

Figura 2.11: Esquema do levantamento de um 3-separador até uma 3-cruz.

em instâncias onde G é um grafo de grade, também desenvolvemos algoritmos de separação mais eficientes para o caso planar.

2.3.1 p-separadores

Separar os 2-separadores sobre um dado conjunto $S = \{s_1, s_2\}$, equivale a encontrar um separador R de S de custo mínimo. Isso pode ser feito por qualquer dos algoritmos para o problema do corte mínimo, aplicado numa rede convenientemente construída de forma a converter custos de vértices em arestas.

Mas quando G é planar, existe uma maneira mais eficiente de encontrar esses separadores, usando-se como subrotina um algoritmo de caminho mínimo. Esse método é uma adaptação do algoritmo descrito por Reif [54]:

Caso 1: *Os vértices s_1 e s_2 estão na mesma face f de G.* Crie um novo grafo G', modificando G com as seguintes operações: (i) Se $g \neq f$ é uma face de G delimitada por vértices no conjunto $V(g)$, crie novas arestas de modo que $G'[V(g)]$ seja completo; (ii) crie dois novos vértices v_1 e v_2, onde v_1 é adjacente a todos os vértices de um dos caminhos sobre f ligando s_1 até s_2 e v_2 é adjacente a todos os vértices do outro caminho sobre f; e (iii) retire os vértices s_1 e s_2. O 2-separador mais violado sobre S corresponde a um caminho de custo mínimo nos vértices entre v_1 e v_2 em G', onde os vértices originais de G tem custo $\bar{x}$ e os novos vértices tem custo 0. Por exemplo, no grafo $G1$ com $S = \{a, l\}$, o grafo $G1'$ resultante é mostrado na Figura 2.12. Se uma face g tem $|V(g)| \geq 5$, pode ser vantajoso mudar o passo (i), ao invés de completar $G'[V(g)]$, adicionar um novo vértice adjacente a todos os vértices em $V(g)$.

Caso 2: *Os vértices s_1 e s_2 não estão na mesma face de G.* Suponha, sem perda de generalidade, que s_1 está na face externa f de G. A idéia é fazer uma *secção* em G, dividindo os vértices que fazem parte de um caminho de custo mínimo nos vértices de s_2 até a face externa. Por exemplo, se $G5$ é o grafo de grade 5×5 mostrado na Figura 2.13, $s_1 = u$ e $s_2 = m$, uma possível secção pode dividir os vértices n e o, como indicado pela linha pontilhada. Daí se aplica uma transformação semelhante a usada no caso 1. O grafo resultante $G5'$ é também

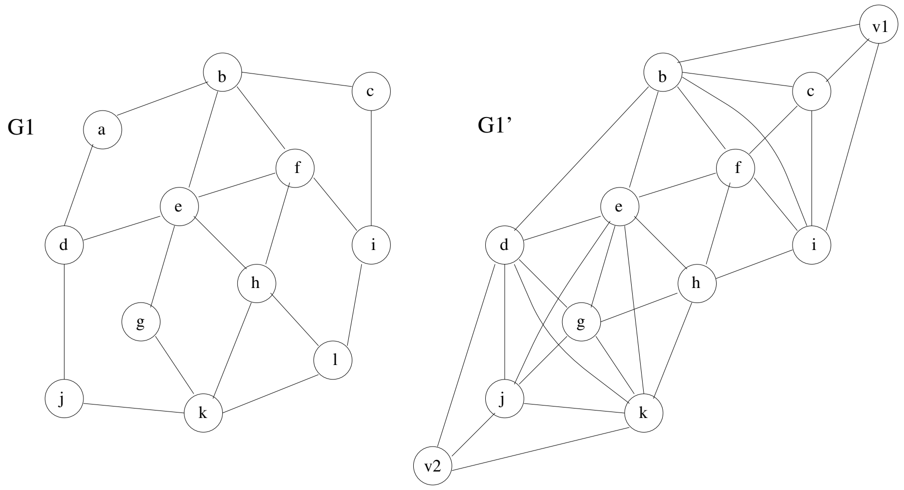

Figura 2.12: Encontrando um separador de 2 vértices na mesma face. $S = \{a, l\}$.

mostrado na Figura 2.13. Nesse caso, o separador de S de mínimo custo pode ser encontrado aplicando-se um algoritmo de caminho mínimo por 3 vezes. Através de um caminho mínimo entre v_1 e v_2, pode-se encontrar um separador como $\{k, q, w\}$. Através do caminho entre n e n', separadores como $\{n, h, l, r\}$, $\{n, i, d, p, q, r\}$ ou $\{n, h, l, q, v, t\}$ podem ser encontrados. Finalmente, acha-se o caminho mínimo entre o e o'.

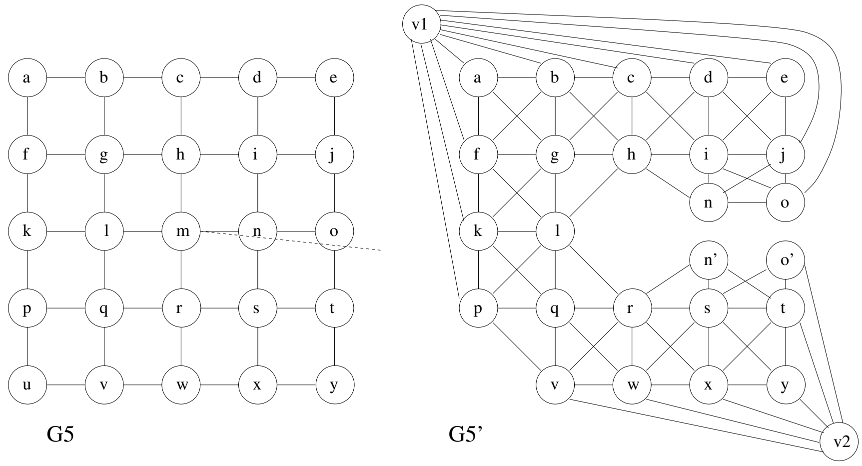

Figura 2.13: Encontrando um separador de 2 vértices em faces diferentes. $S = \{u, m\}$.

Parece ser bem mais difícil fazer procedimentos para encontrar p-separadores violados, quando $p \geq 3$. Propomos uma heurística simples para encontrar 3-separadores sobre um conjunto $S = \{s_1, s_2, s_3\}$ dado: (i) encontre um separador R_1 de $\{s_1, s_2\}$ com o mínimo custo $\bar{x}(R_1)$; (ii) se existir um componente de $G[V \setminus R_1]$ formado pelos vértices A contendo dois vértices de S, digamos s_2 e s_3, encontre em $G[A \cup R_1]$ um separador R_2 de $\{s_2, s_3\}$ com custo

$\bar{x}(R_2)$. Assim, $R = R_1 \cup R_2$ é um separador de S e se $\bar{x}_{s_1} + \bar{x}_{s_2} + \bar{x}_{s_3} - \bar{x}(R_1) - \bar{x}(R_2) > 1$, a desigualdade do 3-separador correspondente está violada. Notar que se um vértice $i \in R$ tem coeficiente $\alpha_i = 2$ no 3-separador, seu custo é corretamente contado duas vezes. Por exemplo, no grafo $G5$ com $S = \{a, i, q\}$, R_1 pode ser $\{d, h, l, p\}$ e R_2 pode ser {l,r,x}.

2.3.2 p-cruzes

As p-cruzes podem ser separadas em tempo polinomial em qualquer grafo G, para qualquer valor *fixo* de p. Essa afirmação segue diretamente da Definição 2.10, pois existe um algoritmo com complexidade $O(|V| \cdot |E|)$ para determinar se é possível ou não conectar dois pares de vértices por caminhos disjuntos (Shiloach [59]). Entretanto, caso se considere a estrutura particular de G pode-se construir algoritmos de separação muito mais eficientes.

No caso em que G é planar, podemos procurar apenas por p-cruzes violadas onde a sequência $s_1, r_1, \ldots, s_p, r_p$ aparece nessa ordem cíclica em alguma face f de G. Se G é também 3-conexo, essas são as únicas p-cruzes. Mesmo que G não seja 3-conexo, as poucas p-cruzes com vértices em faces distintas parecem ser equivalentes ou dominadas por p-separadores. Por exemplo, no grafo $G5$, a 2-cruz $x_u + x_m - x_p - x_v \leq 1$ é também um 2-separador. As p-cruzes sobre a mesma face de G podem ser separadas em tempo $O(|V|)$ por programação dinâmica:

```
Procedimento de separação de p-cruzes em grafos planares
  Para cada face f de G {
      A[0] ⟵ −∞; B[0] ⟵ 0;
      C[0] ⟵ 0; D[0] ⟵ −∞;
      Para cada vértice i na ordem cíclica de f (numerados de 1 até |V(f)|) {
          A[i] ⟵ MAX(A[i − 1], B[i − 1] + x̄_i − 1);
          B[i] ⟵ MAX(B[i − 1], A[i − 1] − x̄_i + 1);
          C[i] ⟵ MAX(C[i − 1], D[i − 1] + x̄_i − 1);
          D[i] ⟵ MAX(D[i − 1], C[i − 1] − x̄_i + 1);
      }
      Se (MAX(B[|V(f)|], C[|V(f)|]) > 1), existe uma p-cruz violada em f;
  }
```

Há duas recursões independentes. A primeira é sobre os vetores A e B. $A[i]$ dá o maior valor (menos 1) do lado esquerdo de uma p-cruz parcial até o vértice i *desbalanceada para* S, ou seja, uma p-cruz parcial com formato $s_1, r_1, \ldots, s_j$. Já $B[i]$ é o maior valor de uma p-cruz parcial até o vértice i *balanceada*, ou seja, $s_1, r_1, \ldots, s_j, r_j$. Dessa forma, a violação é indicada por $B[|V(f)|] > 1$. A outra recursão é similar. $D[i]$ é o maior valor (mais 1) de uma p-cruz parcial até o vértice i desbalanceada para R e $C[i]$ é o valor de uma p-cruz $r_1, s_1, \ldots, r_j, s_j$. A violação acontece quando $C[|V(f)|] > 1$.

2.4 Implementação do Algoritmo de Branch-and-Cut

A formulação para o problema, as demais classes de desigualdades encontradas e os algoritmos de separação correspondentes são a base para a construção de um algoritmo de *branch-and-cut* para o 2PEAS. Mas para que esse algoritmo tivesse um bom desempenho prático e fosse capaz de resolver instâncias grandes, também foi necessário um cuidado especial com detalhes de implementação e até o uso de pequenos "truques" computacionais.

2.4.1 Usando informação primal para levantar desigualdades

Se uma solução com custo z_{inc} já é conhecida, existe uma maneira efetiva de levantar p-separadores, mesmo quando eles definem facetas.

TEOREMA 2.11. *Seja* $\sum_{i\in S} x_i - \sum_{i\in R} \alpha_i x_i \leq 1$ *um* p*-separador válido. Defina* S^1 *como* $\{s_j \in S \mid -(\sum_{i\in A_j\cap N_1} w_i) \geq z_{inc}$ *ou* A_j *tem uma variável fixada a* $1\}$. *O* p*-separador pode ser levantado para* $\sum_{i\in S\setminus S^1} x_i - \sum_{i\in R} \alpha_i x_i \leq 1 - |S^1|$.

Prova. Qualquer dv-árvore T correspondente a uma solução com custo melhor do que z_{inc}, contém pelo menos um vértice ${s_j}^T$ em cada A-componente A_j com vértices em S^1. Seja S^T o conjunto desses vértices ${s_j}^T$. Como o p-separador $\sum_{i\in S\setminus S^1} x_i + \sum_{i\in S^T} x_i - \sum_{i\in R} \alpha_i x_i \leq 1$ é válido para T, então a nova desigualdade levantada também é válida para T. □

Por exemplo, na Figura 2.1 (página 9) $x_a + x_h - x_b - x_e - x_j \leq 1$ é uma 2-cruz que define uma faceta. Mas se uma solução com custo $z_{inc} = 3$ já é conhecida, a desigualdade levantada $x_b + x_e + x_j \geq 1$ poderia ser utilizada no algoritmo de *branch-and-cut*, pois o teorema acima garante que ela não corta nenhuma solução com custo menor do que 3.

2.4.2 Heurística primal

Antes do início do algoritmo de *branch-and-cut*, constrói-se uma solução gulosa do problema e aplica-se $2|V|$ iterações de uma busca tabu para melhorá-la. Cada iteração consiste em uma pesquisa na vizinhança definida por todas as trocas de um único vértice de T_1 para T_2 ou vice-versa, mas sem desconectar $G[T_1]$ ou $G[T_2]$.

Essa heurística não funcionou bem, normalmente as soluções obtidas estão bastante longe do ótimo. Mesmo assim essas soluções são suficientes para obter um valor inicial de z_{inc} que já permite o levantamento de muitos p-separadores, como descrito na Subseção 2.4.1.

2.4.3 Estratégia de separação

A estratégia de separação é a decisão de quantas e quais desigualdades violadas se deve tentar separar em cada rodada de separação. O objetivo é resolver cada nó da árvore de *branch-and-cut* no menor tempo possível. A escolha de tal estratégia pode ser uma questão complexa. Mais desigualdades por rodada de separação reduzem o número de programas lineares (PLs) a serem resolvidos em cada nó. Por outro lado, fazem com que cada PL demore mais, além do que, a própria separação adicional consome um tempo significativo. Ainda mais importante do que a questão da quantidade, é a questão da qualidade do conjunto das desigualdades separadas. Uma boa estratégia de separação deve fazer com que os valores dos sucessivos PLs resolvidos aumentem significativamente, mesmo que cada rodada só adicione um conjunto relativamente pequeno de desigualdades violadas.

Adotamos a seguinte estratégia geral:

```
Estratégia Geral de Separação
 Para cada "família" de desigualdades {
        desmarque todas as variáveis;
        Repita
               ache (i) a desigualdade mais violada sobre as variáveis não-marcadas,
                     (ii) usando o mínimo número de variáveis,
                     (iii) que force mais a função objetivo;
               marque as variáveis nessa desigualdade;
        Até que nenhuma desigualdade violada seja encontrada, ou que
          algum limite para essa família seja atingido;
 }
```

Além de procurar desigualdades mais violadas, há um critério de desempate que busca desigualdades com o conjunto suporte (o conjunto dos coeficientes não-nulos) de menor cardinalidade. Essa idéia foi usada com sucesso, por exemplo, no algoritmo que Koch e Martin [38] propuseram para o problema de Steiner em grafos. Já o terceiro critério, "forçar a função objetivo", significa que buscamos desigualdades $a^T x \leq 1$ que minimizem $a^T w$. Mesmo como o critério de menor prioridade, essa idéia se mostrou muito efetiva. Pelo menos no caso do 2PEAS, é importante não se esquecer dos pesos originais ao se fazer a separação. Desigualdades satisfazendo esses critérios de desempate adicionais podem ser encontradas pelos procedimentos de separação normais, após a introdução de perturbações adequadas no vetor $\bar{x}$. É recomendada a utilização de algoritmos de separação fortemente polinomiais, pois tais perturbações aumentam bastante o tamanho dos números a serem tratados. É o caso dos algoritmos mostrados na seção anterior.

A idéia de separar vários cortes violados por família, desde que seus conjuntos suporte tenham pouca ou nenhuma sobreposição, é usada em algumas implementações recentes de algoritmos de *branch-and-cut*, incluindo [38].

2.4.3.1 p-cruzes

Definimos as p-cruzes sobre cada face f de G como uma família de cortes e seguimos a estratégia geral de separação sem nenhum limite. Em outras palavras, em cada rodada de separação, adicionamos tantas p-cruzes violadas sem sobreposição na mesma face quantas forem possíveis encontrar. Essa estratégia é eficiente porque as p-cruzes são: (i) muito baratas de separar em grafos planares; (ii) muito esparsas, assim elas não sobrecarregam os PLs; e (iii) desigualdades muito fortes na prática.

2.4.3.2 2-separadores

Usamos o seguinte procedimento para escolher um pequeno conjunto de vértices M para formar os conjuntos $S = \{s1, s2\}$. Seja $\bar{D}(i)$ o conjunto de vértices que pode ser alcançado de i por um "caminho decrescente", isto é, um caminho orientado de P a partir i, tal que para cada aresta $(a, b) \in P$, $\bar{x}_a \geq \bar{x}_b$.

```
Procedimento Escolhe M
  Ordene os vértices por:
    (i) vértices fixados a 1 primeiro,
    (ii) maiores valores de x̄_i em seguida,
    (iii) Uma combinação linear de −w_i e da distância de i para a face externa;
  M ⟵ ∅;
  Repita
      i ⟵ o primeiro vértice não-marcado;
      M ⟵ M ∪ {i};
      Marque os vértices em D̄(i);
  Até que todos os vértices estejam marcados;
```

A separação ocorre sobre as $|M| - 1$ famílias de 2-separadores sobre $S = \{s1, s2\}$, onde s_1 é sempre o primeiro vértice incluído em M e s_2 é um dos demais vértices de M. A estratégia geral de separação é aplicada sobre cada uma dessas famílias, com uma pequena ressalva: vários 2-separadores sobre o mesmo S podem ser separados, desde que os correspondentes conjuntos R não tenham sobreposição. Ou seja, os vértices de S não são marcados no procedimento da estratégia geral. Como o algoritmo de separação de 2-separadores não é tão barato, e os 2-separadores podem ser densos o suficiente para carregar os PLs, adotamos um limite: continuamos separando sobre uma família S até que um corte violado por menos que $1/2$ unidade seja encontrado.

Em quase todas as rodadas de separação, s_1 é um vértice na face externa. O motivo pelo qual preferimos que os outros vértices em M estejam próximos a face externa, é que isso melhora o desempenho do algoritmo de separação para grafos planares descrito na Seção 2.3.1.

2.4.3.3 3-separadores

A heurística de separação de 3-separadores é um pouco cara e tais cortes parecem ter uma importância prática limitada. Assim, a separação dessas desigualdades é feita apenas uma vez por nó, na primeira vez em que nenhuma outra desigualdade violada é encontrada.

A separação de 3-separadores tenta aproveitar as informações já obtidas ao se tentar separar 2-separadores. Se o separador de custo mínimo do conjunto $S = \{s1, s2\}$ é R_1 com $\bar{x}_{s_1} = \bar{x}_{s_2} = \bar{x}(R_1) = 1$ e R_1 tem vértices com $\bar{x}$ fracionário, selecionamos um terceiro vértice $s_3 \in M$ com $\bar{x}_{s_3} = 1$. Aplicando-se a heurística descrita na Seção 2.3.1, se $\bar{x}(R_2) < 1$ então um 3-separador violado é encontrado.

2.4.4 Estratégia de enumeração

A estratégia de enumeração indica a ordem em que os nós da árvore de *branch-and-cut* são explorados. Como no nosso caso não dispomos de boas heurísticas primais, a solução ótima do problema é sempre encontrada pelo próprio *branch-and-cut*, como solução inteira de algum nó. Assim, a estratégia de enumeração passa a ser muito importante para o desempenho do algoritmo: quanto antes se chegar a um nó com solução inteira ótima, melhor.

Nos testes realizados com o nosso algoritmo, o valor da relaxação linear z_r obtido no nó raiz, normalmente estava muito próximo ou até igualava z^*. Mas mesmo nesse último caso, a solução do nó raiz costuma ser muito fracionária. De fato, logo ficou claro que as soluções inteiras só apareciam em nós a partir de uma certa profundidade na árvore de enumeração. Por outro lado, foi constatado que existem tantas soluções ótimas alternativas, que diversos ramos da árvore contêm nós com soluções inteiras ótimas.

A estratégia de enumeração *melhor-primeiro* consiste em se escolher o nó ativo com o menor limite inferior (na verdade, o limite do seu nó pai, o limite inferior dado pelo próprio nó só vai ser conhecido depois) para ser explorado primeiro. A estratégia melhor-primeiro é provadamente ótima, no sentido em que nenhuma outra estratégia garante explorar menos nós no pior caso. Ela também costuma funcionar bem na prática e é adotada como *default* em vários códigos comerciais para programação inteira (como o CPLEX [34] ou XPRESS [14]). Mas ela não se mostrou adequada ao nosso algoritmo. Como a estratégia melhor-primeiro tende a explorar a árvore em largura, ela demora a chegar na profundidade em que existem soluções inteiras ótimas.

A estratégia de enumeração *mais-profundo-primeiro* consiste em se escolher o nó ativo com maior profundidade na árvore para ser explorado. No nosso algoritmo, essa estratégia funciona bem na maioria dos casos, pois ela pode fazer com que se chegue rapidamente a um nó com solução inteira ótima. Entretanto, em outros casos, essa estratégia pode ser muito ruim, desperdiçando bastante tempo na exploração de ramos que não contêm soluções ótimas.

Optamos por usar uma estratégia mista: explorar primeiro o nó mais profundo dentre aqueles cujo limite inferior *arredondado para cima* seja mínimo. Por exemplo, um nó na profundidade 4 com limite de 60 é explorado antes do que um nó na profundidade 2 com limite de 59,1. Já um nó na profundidade 5 com limite 60,1 é explorado depois desses dois nós. A esperança é que se encontre uma solução inteira de custo 60 e esse último nó nunca precise ser explorado.

Essa estratégia mista funcionou bem, pois ela permite que se desça rapidamente até as profundidades em que existem soluções inteiras, sem o risco de gastar um tempo excessivo em ramos ruins. Essa estratégia tem a mesma garantia teórica da estratégia melhor-primeiro.

2.4.5 Perturbação dinâmica dos pesos

As instâncias do 2PEAS, principalmente as não-ponderadas, podem ter um número enorme de soluções alternativas com o mesmo custo. Esse tipo de simetria pode prejudicar bastante o desempenho de um algoritmo de *branch-and-cut* ingênuo: os PLs são muito degenerados e podem ser necessárias muitas rodadas de separação para mexer significativamente nos limites inferiores, principalmente quando se está próximo de resolver um nó (esse último efeito é conhecido na literatura como *tailing-off*). Nesses casos, é usual adicionar pequenas perturbações aleatórias aos custos das variáveis para reduzir a degenerescência (por exemplo, [27] e [38]).

A introdução de perturbações aleatórias já melhora razoavelmente o desempenho do algoritmo de *branch-and-cut* para o 2PEAS, de fato, sem esse recurso o nosso código simplesmente não consegue resolver instâncias grandes. Mas verificamos experimentalmente que era melhor que as perturbações não fossem completamente aleatórias, mas que vértices em regiões do grafo com predominância de vértices positivos deveriam receber perturbações positivas e vice-versa. A maneira escolhida para fazer isso foi a seguinte. Para cada vértice i ainda não perturbado, calcula-se $n_i = \sum_{j \in V(i)} w_j$, onde $V(i)$ é o conjunto dos vértices adjacentes a i. Se n_i é positivo, adiciona-se uma pequena perturbação aleatória positiva a w_i; se n_i é negativo, uma perturbação aleatória negativa. O procedimento é repetido até que todos os vértices estejam perturbados.

Mas uma surpreendente melhoria foi obtida com a idéia de perturbação dinâmica dos pesos: quando o algoritmo faz uma ramificação, inverte-se, em ambos os nós filhos, o sinal da perturbação de todos os vértices não-terminais i em que $\bar{x}_i$ é fracionário. Essa idéia mostrou-se bastante eficiente no sentido de transformar uma solução com um grande número de variáveis fracionárias, numa solução quase inteira de mesmo custo. Isso reduziu muito as profundidades máximas atingidas na árvore de enumeração.

Arriscamos uma explicação ao porquê da perturbação dinâmica funcionar. Atribuir uma perturbação negativa a um certo vértice não-terminal i, equivale a colocá-lo no conjunto N_1, uma perturbação positiva o coloca em N_2. Isso faz com que essa decisão arbitrária tenha um impacto sério na "integralidade da instância" (ao contrário, decidir se um terminal vai ter um peso ligeiramente menor ou maior não muda os conjuntos de terminais). O fato de um vértice não-terminal i estar com $\bar{x}_i$ fracionário talvez seja uma indicação de que i foi colocado no conjunto de terminais "errado". Assim, a inversão das perturbações tende a corrigir isso e gerar instâncias mais inteiras nos filhos.

2.4.6 Miscelânea

Iniciamos o nó raiz com um PL vazio, sem nenhuma restrição além dos limites inferiores e superiores das variáveis. A fim de evitar o crescimento excessivo dos PLs, depois de cada resolução, removemos todas as restrições cujas variáveis de folga são básicas. As restrições removidas são descartadas, não há armazenamento de restrições fora dos PLs. Adotamos o tradicional esquema de ramificação sobre variáveis, dando preferência a variáveis i com $\bar{x}_i$ próximo a $1/2$, com valor absoluto de custo grande e na face externa do grafo planar.

2.5 Experimentos Computacionais

O código do algoritmo de *branch-and-cut* foi escrito em C++, usando também funções prontas oferecidas pelo pacote ABACUS [62]. O resolvedor de PLs utilizado foi o CPLEX 3.0 [33] e os tempos foram obtidos numa estação de trabalho Sun ULTRA 167 MHz.

Realizamos os experimentos computacionais em instâncias em que G é um grafo de grade quadrado, motivados pelas aplicações VLSI do 2PEAS sobre esse tipo de grafo [61]. Os experimentos de Grötschel et al. [27] também se limitam a grafos de grade. Geramos dois tipos de instâncias:

Instâncias não-ponderadas. Escolhemos aleatoriamente $|V|/4$ vértices em G para formar o conjunto de terminais N_1 e outros $|V|/4$ vértices para ser N_2.Vértices em N_1 tem pesos -1 e os vértices em N_2 tem peso 1. Nos referimos a essas instâncias como u_n_$Inst$, onde $n = |V|$ e $Inst$ é o número da instância. O valor z^* indica o número de terminais não-conectados em uma solução ótima.

Instâncias ponderadas. Para cada instância não-ponderada, geramos uma instância ponderada correspondente, escolhendo pesos inteiros w_i de uma distribuição uniforme na faixa $[-10, -1]$ se $i \in N_1$ e na faixa $[1, 10]$ se $i \in N_2$. Essas instâncias são referenciadas como w_n_$Inst$. O valor z^* indica a soma dos pesos absolutos dos terminais não-conectados numa solução ótima.

Os resultados obtidos pelo algoritmo de *branch-and-cut* sobre essas instâncias são mostrados na Tabela 2.1. A coluna $\mathbf{z}_r$ mostra o valor do PL final do nó raiz, esse valor não é apenas a relaxação linear da formulação (F), mas inclui as p-cruzes e alguns 3-separadores obtidos heuristicamente. A coluna **Nd** é o número de nós explorados na árvore de *branch-and-cut*. A coluna **nPLs** é o número de PLs resolvidos (e também o número de rodadas de separação), enquanto **nCortes** é o número total de cortes adicionados. A coluna **Tempo** indica o tempo total de cpu em segundos e **%PL** é o percentual desse tempo usado na resolução dos PLs. Praticamente todo o tempo restante foi gasto em procedimentos de separação.

Alguns comentários sobre esses resultados:

- Os valores z_r estão bastante próximos do valor do ótimo inteiro z^*. Isso só acontece devido ao uso das p-cruzes. Por exemplo, na instância u1600_1, o valor da relaxação linear da formulação (F) (ou seja, apenas os 2-separadores) é de apenas 8,7. O efeito da separação heurística de alguns 3-separadores é marginal.

- Nenhuma instância foi resolvida no nó raiz, embora 9 das 28 instâncias tenham sido resolvidas no segundo nó. Esse comportamento pouco usual se deve à perturbação dinâmica dos custos: mesmo quando $z_r = z^*$, o nó raiz dificilmente produz soluções inteiras, mas elas podem ser encontradas logo no primeiro nó filho, após as mudanças nas perturbações.

- Os detalhes de implementação descritos na seção anterior são fundamentais para o bom desempenho do algoritmo de *branch-and-cut*. Por exemplo, a primeira implementação ingênua desse algoritmo não conseguiu resolver a instância u1600_1 em 6 horas de cpu, mesmo já usando os algoritmos de separação descritos na Seção 2.3.

A figura 2.14 mostra a solução ótima da instância u1600_1, as linhas cheias indicam as arestas de uma árvore geradora de $G[T_1]$ e as linhas pontilhadas uma árvore geradora de $G[T_2]$.

Uma experiência adicional interessante é gerar um posicionamento aleatório de terminais evitando as duas causas mais óbvias de infactibilidade, terminais na face externa ou dois pares de terminais opostos cruzados no mesmo quadrado da grade. A instância mostrada na Figura 2.2 (página 10) foi gerada dessa maneira, ela contém 400 vértices em cada conjunto de terminais. A solução ótima mostrada na Figura 2.3 foi obtida em 21 segundos. Mas o que surpreende é que z^* é apenas 4, o posicionamento aleatório de 800 terminais em um grafo de 1600 vértices gerou uma instância quase factível. Isso tem implicações para as aplicações do

Instância	z_r	z^*	Nd	nPLs	nCortes	Tempo	%PL
u1600_1	45.7	46	4	96	2881	20	43
u1600_2	40.0	40	2	40	529	8	24
u2500_1	46.0	46	2	56	1118	33	20
u2500_2	55.0	55	3	49	2049	32	31
u3600_1	76.5	77	2	45	2084	41	37
u3600_2	74.0	74	4	57	1432	50	33
u4900_1	99.1	100	4	116	4380	170	32
u4900_2	92.0	92	3	66	2809	108	32
u6400_1	104.0	104	3	126	5808	443	30
u6400_2	127.0	128	4	123	6384	322	36
u8100_1	131.5	132	4	217	10271	1207	25
u8100_2	115.0	115	6	125	6734	713	22
u10000_1	159.8	161	8	120	6440	676	34
u10000_2	151.5	152	4	157	9870	991	27
w1600_1	222.0	222	2	35	1174	9	40
w1600_2	218.0	218	2	25	602	5	34
w2500_1	233.0	233	2	37	1113	14	42
w2500_2	227.5	228	3	39	810	17	32
w3600_1	364.0	364	2	48	1598	33	40
w3600_2	311.0	311	2	74	1715	54	29
w4900_1	420.0	420	2	77	2847	130	29
w4900_2	400.0	400	3	53	2305	90	34
w6400_1	436.0	436	3	106	3509	295	28
w6400_2	534.0	534	3	128	4915	367	28
w8100_1	549.5	550	4	113	4020	563	18
w8100_2	467.0	467	3	82	3175	264	28
w10000_1	685.0	685	3	60	3544	345	27
w10000_2	632.5	634	7	116	4907	621	38

Tabela 2.1: Resultados do algoritmo de *branch-and-cut*.

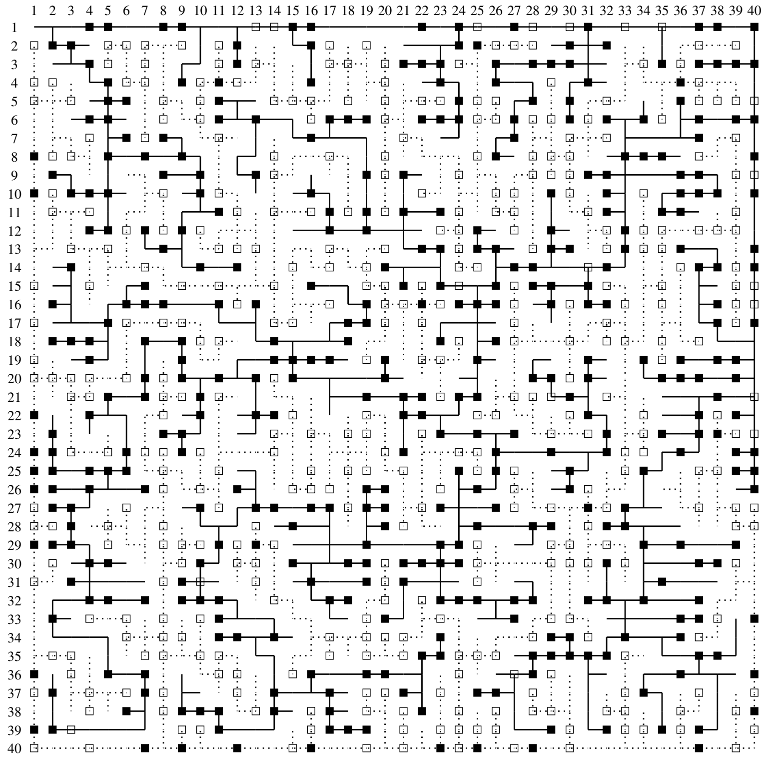

Figura 2.14: Uma solução ótima da instância u1600_1, $z^* = 46$.

2PEAS. Por exemplo, não deve ser difícil achar um posicionamento dos terminais terra e V_{cc} em um circuito que permita fazer todas essas conexões em uma única camada.

2.6 Comentários Finais

- Os experimentos computacionais foram realizados em grafos de grade, mas os mesmos algoritmos poderiam ser aplicados a grafos planares quaisquer. Já um teste do *branch-and-cut* para o 2PEAS em grafos não-planares dependeria de um procedimento geral de separação de p-cruzes, possivelmente baseado no algoritmo de Shiloach [59]. Não conhecemos nenhuma implementação desse algoritmo, que é bastante complexo.

- Uma outra formulação do 2PEAS, similar a usada em [28], usa variáveis binárias x_i^1 e x_i^2, onde $x_i^1 = 1$ se o vértice i pertence a T_1 e $x_i^2 = 1$ se i pertence a T_2; e restrições do tipo $x_i^1 + x_i^2 \leq 1$. A formulação (F) seria equivalente a uma formulação nessas variáveis em que as restrições fossem $x_i^1 + x_i^2 = 1$.

  Acreditamos que a decisão de usar uma formulação que só permite soluções de tipo partição tenha consequências mais profundas do que uma mera economia de variáveis: ela leva a poliedros mais compactos. Isso significa que as desigualdades que definem facetas desses poliedros têm maior chance de serem efetivas em um algoritmo de *branch-and-cut* do que as desigualdades que definem facetas dos poliedros mais relaxados, que também incluem soluções do tipo empacotamento.

Esse princípio parece ser válido para qualquer problema de otimização combinatória, o ideal seria estudar os poliedros definidos pela envoltória convexa de um conjunto "mínimo" de soluções, evitando-se incluir as soluções claramente redundantes. Só que esses poliedros mais compactos costumam ser difíceis de serem analisados. Assim, muitos autores chegam a fazer exatamente o contrário: definir os poliedros de forma a também incluir até soluções infactíveis para o problema.[2] Se pode ser bem mais fácil provar que certas desigualdades definem uma faceta desses poliedros relaxados, é mais duvidoso que essas desigualdades funcionem bem nos algoritmos de *branch-and-cut*.

- Existem formulações estendidas que fornecem uma descrição mais precisa dos super-poliedros P_1 e P_2. Tais formulações podem ser adaptadas para o 2PEAS. Um exemplo desse tipo de formulação é apresentado a seguir. Para cada aresta $e \in E$, defina variáveis adicionais z_e^1 e z_e^2, onde $z_e^1 = 1$ caso e pertença a uma árvore geradora de T_1 e $z_e^2 = 1$ caso e pertença a uma árvore geradora de T_2. Dado um conjunto de vértices S, definimos $E(S)$ como o conjunto das arestas com ambos os extremos em vértices de S.

$$
\text{(EF)} \left\{
\begin{array}{llll}
\text{Min } z = & -\sum_{i \in N_1} w_i + \sum_{i \in V} w_i x_i & & (1) \\
\text{s.t.} & \sum_{e \in E} z_e^1 \geq \sum_{i \in V} x_i - 1 & & (2) \\
& \sum_{e \in E(S)} z_e^1 \leq \sum_{i \in S \setminus \{d\}} x_i & \forall S \subseteq V,\ \forall d \in S & (3) \\
& \sum_{e \in E} z_e^2 \geq -\sum_{i \in V} x_i + |V| - 1 & & (4) \\
& \sum_{e \in E(S)} z_e^2 \leq -\sum_{i \in S \setminus \{d\}} x_i + |S| - 1 & \forall S \subseteq V,\ \forall d \in S & (5) \\
& z_e^1, z_e^2 \geq 0;\ x_i \in \{0, 1\} & \forall e \in E;\ \forall i \in V & (6)
\end{array}
\right.
$$

As restrições (EF.3) e (EF.5) são conhecidas como *restrições de eliminação de ciclos generalizadas* e foram propostas de modo independente por Lucena [44], Goemans [25] e Margot at al. [47] para o problema de Steiner em grafos. Elas podem ser separadas em tempo polinomial, usando-se até $O(|V|)$ chamadas a um algoritmo de corte mínimo.

[2] Isso costuma ser feito quando o poliedro original não tem dimensão cheia.

As restrições (EF.2) e (EF.3) fornecem uma descrição de P_1 mais precisa do que as restrições (F.2). Da mesma forma, (EF.4) e (EF.5) são melhores do que (F.3). Entretanto, nos testes realizados, isso não levou a uma melhoria significativa na descrição de P. Ambas as formulações são realmente reforçadas pela adição de desigualdades de junção, como as p-cruzes. Dessa forma, preferimos trabalhar com (F), pois ela leva a PLs com bem menos variáveis e restrições. É possível que em outros tipos de instâncias a situação seja outra, e passe a valer a pena resolver PLs mais pesados em troca de limites inferiores mais fortes.

- A contribuição global dada por esse capítulo da tese foi mostrar que o PEAS restrito ao caso $K = 2$ é um problema que pode ser atacado de forma razoável, tanto do ponto de vista teórico (através da descrição de facetas importantes), quanto prático (pela construção de um algoritmo de *branch-and-cut* efetivo). Mas será que os resultados obtidos no 2PEAS podem ser estendidos ao PEAS geral ? Encerramos o capítulo com especulações sobre essa questão.

  O principal ponto é que muitas desigualdades de junção do 2PEAS, como as p-cruzes, estão relacionadas à existência ou não de caminhos disjuntos entre 2 pares de vértices, um problema polinomial. Já as generalizações diretas dessas desigualdades para o PEAS com K conjuntos de terminais, parecem envolver o Problema dos K Caminhos Disjuntos (KPCD), que é NP-completo (problema ND40 em [24]).[3] Por outro lado, principalmente quando K é pequeno e G é planar, são conhecidas muitas condições simples para garantir a não existência de tais caminhos (ver [23], por exemplo). É possível que um algoritmo de *branch-and-cut* possa usar desigualdades de junção obtidas a partir dessas condições. Para compensar a maior dificuldade em se obter desigualdades de junção, é possível que valha a pena usar formulações estendidas que forneçam uma descrição melhor dos superpoliedros.

  Em resumo, o autor acredita que os resultados obtidos no 2PEAS podem ser generalizados, com alguma dificuldade, para o PEAS com poucos conjuntos de terminais, digamos $K \leq 5$, principalmente se as instâncias forem planares. Já para atacar o PEAS com muitos conjuntos de terminais e grafos não-planares, seria necessária uma abordagem essencialmente diferente.

[3] Seymour e Robertson provaram que o KPCD é polinomial para qualquer K fixo [56], mas essa prova leva a algoritmos envolvendo constantes astronomicamente grandes e é impraticável mesmo para $K = 3$.

Capítulo 3

Pré-processamento de Instâncias VLSI

3.1 Introdução

O Problema de Steiner em Grafos (PSG) é definido da seguinte maneira. Seja $G = (V, E)$ um grafo conexo com custos inteiros positivos $c(e)$ nas arestas (também denotados como $c(u, v)$ quando $e = (u, v)$) e $T \subseteq V$ um conjunto de vértices, conhecidos como *terminais*. Uma *árvore de Steiner* é uma sub-árvore (V', E') de G tal que $T \subseteq V'$. O PSG é o problema de encontrar uma *Árvore de Steiner Mínima* (ASM), isto é, uma árvore de Steiner que minimize $\sum_{e \in E'} c(e)$.

Um *procedimento de pré-processamento* é um algoritmo polinomial que tenta transformar a instância original em uma instância equivalente reduzida, com tamanho siginificativamente menor. Naturalmente deve haver um mecanismo "de volta", para converter soluções ótimas obtidas na instância reduzida em soluções ótimas da instância original. Os procedimentos de pré-processamento são usualmente construídos a partir dos chamados *testes de redução*, teoremas que garantem que se uma certa *condição* for satisfeita, então é possível realizar uma certa *ação* para reduzir a instância. Os testes vão sendo aplicados sucessivamente nas instâncias já reduzidas e o procedimento continua até que nenhum teste tenha as suas condições de aplicação satisfeitas. Em alguns casos é até possível que a instância seja totalmente resolvida apenas pelo pré-processamento.

O trabalho mais completo em pré-processamento para o SPG foi realizado por Duin e Volgenant [17, 16], que generalizaram e sistematizaram vários testes apresentados anteriormente na literatura (por exemplo, Beasley [6, 7] e Balakrishnan e Patel [5]). A partir desses testes é possível construir procedimentos de pré-processamento bastante efetivos para várias classes de instâncias. Por exemplo, nas instâncias de teste presentes na OR-Lib [8] a taxa de redução típica é de 70–90%.

Conforme detalhado na Seção 1.2.1, uma das mais importantes aplicações práticas do PSG

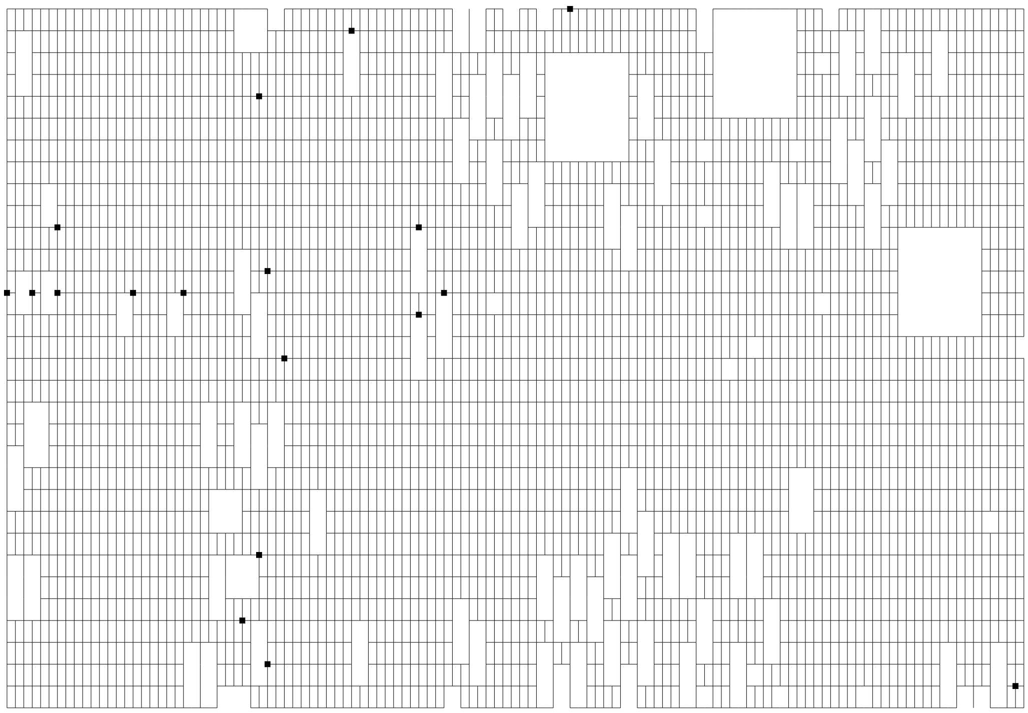

Figura 3.1: diw0559 original, $|V| = 3738$, $|E| = 7013$, $|T| = 18$.

ocorre no roteamento de ligações elétricas em projeto de circuitos VLSI e leva a instâncias em que G é um grafo de grade com buracos. Em 1996, Koch e Martin disponibilizaram na internet um conjunto de 116 instâncias VLSI reais, divididas em 7 séries: alue, alut, diw, dmxa, gap, msm e taq. Essas instâncias são descritas em [35, 38] e podem ser encontradas na SteinLib [39]. A Figura 3.1 mostra a instância diw0559, as arestas são os segmentos de reta, os vértices são os cruzamentos entre segmentos e os terminais são indicados por quadradinhos preenchidos. Os custos das arestas são proporcionais ao comprimento dos segmentos correspondentes. Outras instâncias VLSI maiores e mais complexas são mostradas nas figuras 3.2 e 3.3.

As instâncias VLSI são consideradas de difícil resolução pelos algoritmos existentes.

- Em parte isso ocorre porque a estrutura dessas instâncias faz com que existam muitas soluções distintas com o mesmo custo. Esse tipo de simetria compromete o desempenho dos algoritmos baseados em programação linear, pois surgem problemas ligados à degenerescência.

- Essa estrutura também faz com que os testes de redução de Duin e Volgenant sejam praticamente inúteis. Isso causa sérias dificuldades, pois o tamanho dessas instâncias pode ser muito grande. (Nas séries alue e alut já há instâncias com várias dezenas de milhares de vértices e arestas. É provável que os circuitos mais modernos levem a instâncias ainda maiores.)

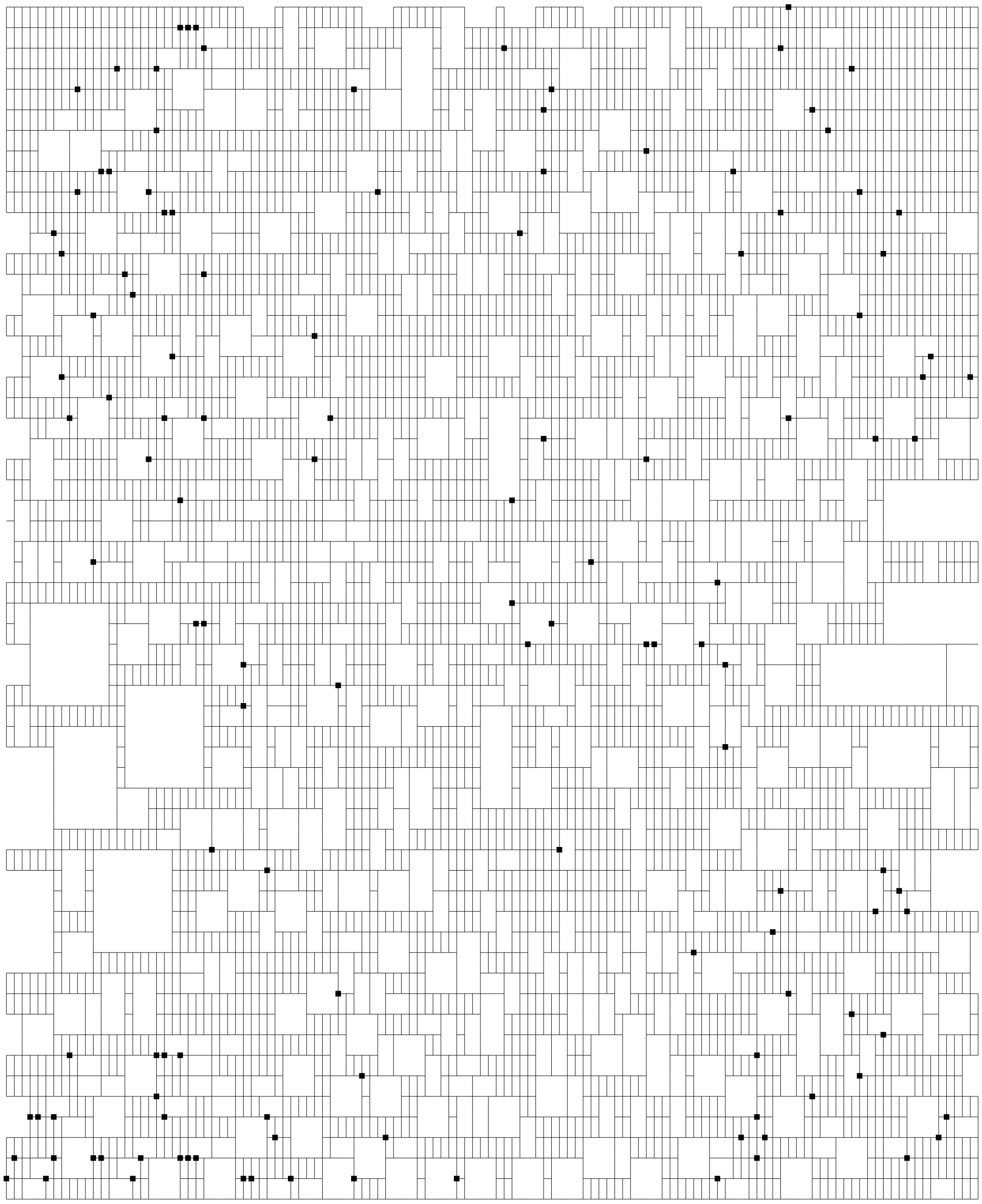

Figura 3.2: taq0903 original, $|V| = 6163$, $|E| = 10.490$, $|T| = 130$.

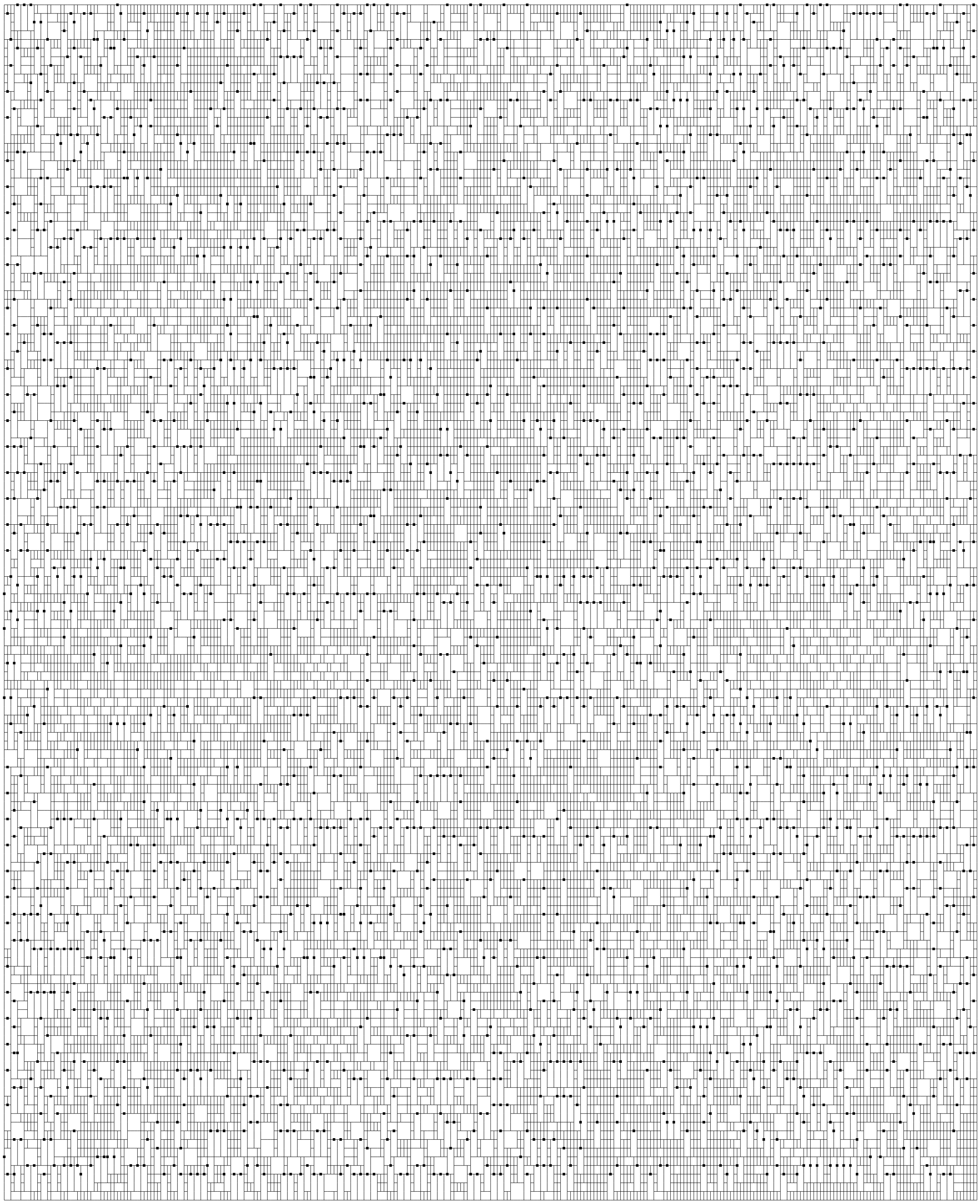

Figura 3.3: alue7080 original, $|V| = 34.479$, $|E| = 55.494$, $|T| = 2344$.

Este capítulo da tese tenta atacar essa última dificuldade, através da construção de um procedimento de pré-processamento que se utiliza de novos testes de redução particularmente efetivos para instâncias VLSI.

3.2 Testes de Redução Tradicionais

3.2.1 Testes simples

Começamos apresentando três testes muito simples:

TESTE 3.1. Non-Terminal Degree 1 (NTD1) - *Um vértice não-terminal u de grau 1 e sua aresta adjacente (u, v) podem ser removidos.*

TESTE 3.2. Non-Terminal Degree 2 (NTD2) - *Um vértice não-terminal u de grau 2 e suas arestas adjacentes (u, v) e (u, w) podem ser subtituídos por uma única aresta (v, w) com custo $c(u, v) + c(u, w)$.*

Uma aresta $e \in E$ é dita *escolhível* se existir pelo menos uma ASM contendo e. Uma vez que uma aresta escolhível $e = (u, v)$ é identificada, ela já pode ser colocada na solução e os vértices u e v contraídos a um único vértice terminal. Uma aresta e é dita *redundante* se existir pelo menos uma ASM que não a contenha. Arestas redundantes são simplesmente removidas.

TESTE 3.3. Terminal Degree 1 (TD1) - *Se $|T| \geq 2$, a aresta adjacente a um terminal de grau 1 é escolhível.*

3.2.2 Testes baseados em alternativas

Os testes de redução baseados em alternativas são aqueles que exploram a existência de soluções alternativas para o problema. Por exemplo, "se *condição*, então a aresta e é redundante" é um teste por alternativa caso exista o teorema "se *condição*, para qualquer árvore de Steiner R que use e, existe outra árvore de Steiner R' que não usa e e tal que $c(R') \leq c(R)$".

Antes de se apresentar os principais testes baseados em alternativas criados por Duin e Volgenant, algumas definições são necessárias. Seja $\mathcal{P}(u, v)$ o conjunto dos caminhos entre dois vértices u e v. A *distância* usual entre u e v pode ser definida como

$$d(u, v) = \min\{c(P) \mid P \in \mathcal{P}(u, v)\},$$

onde $c(P)$ é a soma dos custos das arestas em P. Se $P \in \mathcal{P}(u, v)$, defina $T(P)$ como $\{u, v\} \cup (T \cap P)$, isto é, o conjunto de vértices formado por u, v e pelos terminais em P. A *distância de Steiner* $SD(P)$ ao longo de P é o custo do maior subcaminho em P ligando dois vértices em $T(P)$, consecutivos em P. A *distância de Steiner gargalo* entre u e v é definida como

$$B(u,v) = \min\{SD(P) \mid P \in \mathcal{P}(u,v)\}.$$

Uma boa ilustração do conceito de distâncias gargalo é considerar $\langle G, c\rangle$ como um sistema rodoviário, T como postos de gasolina e um motorista que sai do ponto u com o tanque cheio e deseja chegar em v. Nesse caso, $B(u,v)$ é a menor autonomia (distância que se consegue rodar sem reabastecer) que seu carro precisa ter para que isso seja possível.

A distância de Steiner gargalo sem passar por uma dada aresta e é definida como

$$B(u,v)^{-e} = \min\{SD(P) \mid P \in \mathcal{P}(u,v);\ e \notin P\}.$$

Se e desconecta u e v, $B(u,v)^{-e} = \infty$.

TESTE 3.4. Special Distance (SD) - *Seja (u,v) uma aresta em E. Se $B(u,v) < c(u,v)$, então (u,v) é redundante.*

O teste SD também aparece na literatura com o nome de *Paths with m Terminals (PTm)*, onde m é o número de terminais no caminho usado para calcular a distância de Steiner gargalo. Uma pequena generalização do teste SD considera o caso da igualdade.

TESTE 3.5. Special Distance with Equality (SDE) - *Seja (u,v) uma aresta em E. Se $B(u,v)^{-(u,v)} \leq c(u,v)$, então (u,v) é redundante.*

TEOREMA 3.1. *Os testes SD e SDE são válidos.*

Prova. Suponha que $B(u,v)^{-(u,v)} \leq c(u,v)$ (se $B(u,v) < c(u,v)$, $B(u,v)^{-(u,v)} = B(u,v)$). Seja $P \in \mathcal{P}(u,v)$, $(u,v) \notin P$, um caminho tal que $SD(P) = B(u,v)^{-(u,v)}$ e seja $T(P) = \{u,v\} \cup (T \cap P)$. Suponha que uma árvore de Steiner R use a aresta (u,v). Removendo (u,v) de R, obtemos duas sub-árvores, a que contém o vértice u será chamada de R_u e a que contém v de R_v. Escolha dois vértices x e y de $T(P)$, consecutivos em P, tal que $x \in R_u$ e $y \in R_v$. Seja $P(x,y)$ o subcaminho em P de x até y. Como $c(P(x,y)) \leq B(u,v)^{-(u,v)} \leq c(u,v)$, $R' = R_u \cup R_v \cup P(x,y)$ é uma árvore de Steiner que não contém (u,v) e tal que $c(R') \leq c(R)$. □

TESTE 3.6. Bottleneck Degree 3 (BD3) - *Seja u um vértice não-terminal de grau 3, adjacente aos vértices v, w e z. Se*

$$\min\{B(v,w)+B(v,z), B(w,v)+B(w,z), B(z,v)+B(z,w)\} \leq c(u,v)+c(u,w)+c(u,z),$$

então existe uma ASM onde o grau de u é no máximo 2. Portanto, u e suas três arestas adjacentes podem ser substituídas por três arestas: (v,w) com custo $c(u,v)+c(u,w)$, (v,z) com custo $c(u,v)+c(u,z)$ e (w,z) com custo $c(u,w)+c(u,z)$.

TEOREMA 3.2. *O Teste BD3 é válido.*

Prova. Suponha que $B(v,w) + B(v,z) \leq c(u,v) + c(u,w) + c(u,z)$, os outros dois casos são similares. Seja $P_{vw} \in \mathcal{P}(v,w)$ um caminho tal que $SD(P_{vw}) = B(v,w)$, $T(P_{vw}) = \{v,w\} \cup (T \cap P_{vw})$, $P_{vz} \in \mathcal{P}(v,z)$ um caminho tal que $SD(P_{vz}) = B(v,z)$ e $T(P_{vz}) = \{v,z\} \cup (T \cap P_{vz})$. Seja R uma árvore de Steiner usando o vértice u com grau 3. Removendo u e suas arestas adjacentes de R, obtemos sub-árvores R_v, R_w e R_z. Escolha dois vértices x_1 e y_1 de $T(P_{vw})$, consecutivos em P_{vw}, tal que $x_1 \in R_w$ e $y_1 \notin R_w$. Agora escolha dois vértices x_2 e y_2 de $T(P_{vz})$, consecutivos em P_{vz}, tal que (i) se $y_1 \in R_v$ então $x_2 \in R_z$ e $y_2 \notin R_z$, (ii) se $y_1 \in R_z$ então $x_2 \in R_v$ e $y_2 \notin R_v$. Como $c(P_{vw}(x_1,y_1)) \leq B(v,w)$ e $c(P_{vz}(x_2,y_2)) \leq B(v,z)$, $c(P_{vw}(x_1,y_1)) + c(P_{vz}(x_2,y_2)) \leq c(u,v) + c(u,w) + c(u,z)$, portanto $R' = R_v \cup R_w \cup R_z \cup P_{vw}(x_1,y_1) \cup P_{vz}(x_2,y_2)$ é uma árvore de Steiner tal que $c(R') \leq c(R)$. O vértice u tem grau no máximo 2 em R'. $\square$

Duin e Volgenant na verdade introduziram um teste mais geral:

TESTE 3.7. Bottleneck Degree k (BDk) - *Seja u um vértice não-terminal de grau k, adjacente aos vértices $V(u)$. Suponha que para qualquer conjunto $X \subseteq V(u)$, $|X| \geq 3$, a soma dos custos das arestas ligando u e X for maior ou igual ao da árvore geradora de custo mínimo do grafo completo nos vértices X, usando as distâncias de gargalo como custos. Nesse caso, é possível remover o vértice u e suas arestas incidentes e, em seu lugar, colocar arestas ligando cada par de vértices v e w em $V(u)$ com custo $c(u,v) + c(u,w)$.*

A primeira vista, a ação do teste BDk não parece ser vantajosa, pois se um vértice é removido, o número de arestas pode aumentar. Mas, frequentemente muitas das novas arestas criadas são imediatamente removidas pelo teste SD. A efetividade do teste BDk diminui rapidamente com k, enquanto a sua complexidade computacional cresce exponencialmente. Assim, a maioria dos procedimentos de pré-processamento usa o teste BDk apenas para valores pequenos de k. Os próprios Duin e Volgenant [17] se limitaram a $k \leq 4$. Duin [16] usou $k \leq 8$, Lucena e Beasley [45] $k \leq 5$, Koch e Martin [38] $k \leq 3$, Polzin e Vahdati [53] $k \leq 7$.

O teste BDk também é conhecido como *Local Special Distance* ou *Non-Terminal Degree* k. Essa última nomenclatura chama atenção para o fato que os testes simples NTD1 e NTD2 são exatamente os testes BD1 e BD2.

TESTE 3.8. Terminal Distance (TDist) - *Seja $[W, \bar{W}]$ uma partição dos vértices em V tal que os subgrafos induzidos por W e $\bar{W}$ sejam conexos e que tanto W quanto $\bar{W}$ contenham pelo menos um terminal. Seja $\delta(W)$ o corte induzido por essa partição. Se $\delta(W) = \{e\}$, então e é escolhível. Se $|\delta(W)| \geq 2$, sejam $e = \operatorname{argmin}_{e' \in \delta(W)} c(e')$ e $f = \operatorname{argmin}_{f' \in \delta(W) \setminus \{e\}} c(f')$, respectivamente, a menor e* *a segunda menor aresta no corte. Suponha que* $e = (u, v)$ *com* $u \in W$ *e* $v \in \bar{W}$*. Se*

$$\min\{d(u,t_1) \,|\, t_1 \in T \cap W\} + c(e) + \min\{d(v,t_2) \,|\, t_2 \in T \cap \bar{W}\} \ \leq\ c(f)\ ,$$

então e *é escolhível.*

Esse teste é referenciado por alguns autores como *Nearest Special Vertices (NSV)* e por outros como *Short Edges (SE)*. Pode-se notar que o teste TD1 é um caso particular do TDist.

TEOREMA 3.3. *O teste TDist é válido.*

Prova. Se $\delta(W) = \{e\}$, o teste é trivial. Suponha que $|\delta(W)| \geq 2$ e que as condições do teste são satisfeitas. Seja R uma árvore de Steiner que não usa a aresta e. Há exatamente um caminho P em R entre t_1 e t_2. Escolha uma aresta $g \in (P \cap \delta(W))$, $c(g) \geq c(f)$. Removendo g de R, obtemos duas sub-árvores, uma contendo t_1 e a outra contendo t_2. Conectando essas sub-árvores usando um caminho mais curto de t_1 até t_2 passando por e, obtemos uma árvore de Steiner R' que usa e e tal que $c(R') \leq c(R)$. □

Há outros testes baseados em alternativas conhecidos, mas sua efetividade é marginal em relação aos três grupos de teste apresentados aqui: SD, BDk e TDist. De fato, usando-se apenas os testes NTD1, NTD2, SD, BD3 e TDist é possível construir procedimentos de pré-processamento bastante efetivos para muitas instâncias. Por exemplo, em Koch e Martin [38] consegue-se reduzir a instância e18 (a mais difícil da OR-Lib) de 62.500 para 5996 arestas. (As instâncias da OR-Lib consistem de grafos aleatórios com diversas densidades de arestas e terminais e custos inteiros extraídos de uma distribuição uniforme entre 1 e 10. O teste SD remove praticamente todas as arestas longas, digamos com custo maior que 4, e o teste TDist contrai as arestas curtas próximas a terminais.)

Esse mesmo procedimento aplicado a diw0559 apenas consegue uma redução de 7013 para 6883 arestas. Não é difícil entender os motivos desse fracasso: nas instâncias VLSI o custo de uma aresta (u, v) é estritamente menor do que o custo de qualquer caminho entre u e v que não use essa aresta. A não ser que existam terminais próximos a (u, v), e em posições muito favoráveis, o teste SD não tem como funcionar. Raciocínios semelhantes explicam o não funcionamento dos testes BD3 e TDist.

Se ao invés do teste SD usa-se o SDE, já é possível alguma redução de instâncias VLSI. Por exemplo, a instância diw0559 pode ser reduzida a 2694 arestas, conforme mostrado na Figura 3.4. (Ao desenhar instâncias reduzidas, mantemos as coordenadas originais dos vértices e das arestas. Isso facilita a comparação com a instância original. As linhas em negrito indicam arestas já escolhidas para fazer parte da solução.) Na verdade, o teste SDE combinado com os testes NTD1 e NTD2 já é capaz de remover a parte do grafo que está fora da envoltória convexa retilinear minimal dos terminais T que não cruza nenhum buraco. Para entender esse processo basta olhar para o canto inferior esquerdo da instância diw0559 original. O teste NTD2 une os dois segmentos que formam esse canto em uma única aresta (u, v), que é imediatamente removida pelo teste SDE, pois $c(u, v) = B(u, v)^{-(u,v)}$. No caso, aparecem dois novos cantos que também podem ser eliminados pelos testes NTD2 e SDE. O processo continua até que cada um dos cantos tenha um terminal ou seja bloqueado por um buraco.

Essas reduções podem ser significativas para as instâncias VLSI que não têm terminais perto dos cantos e com poucos buracos. Esse é o caso da diw0559, mas não da taq903 que seria reduzida de 10.490 para 8485 arestas. De modo geral, as reduções obtidas com os testes mostrados até agora não são suficientes para ajudar substancialmente na resolução das instâncias VLSI.

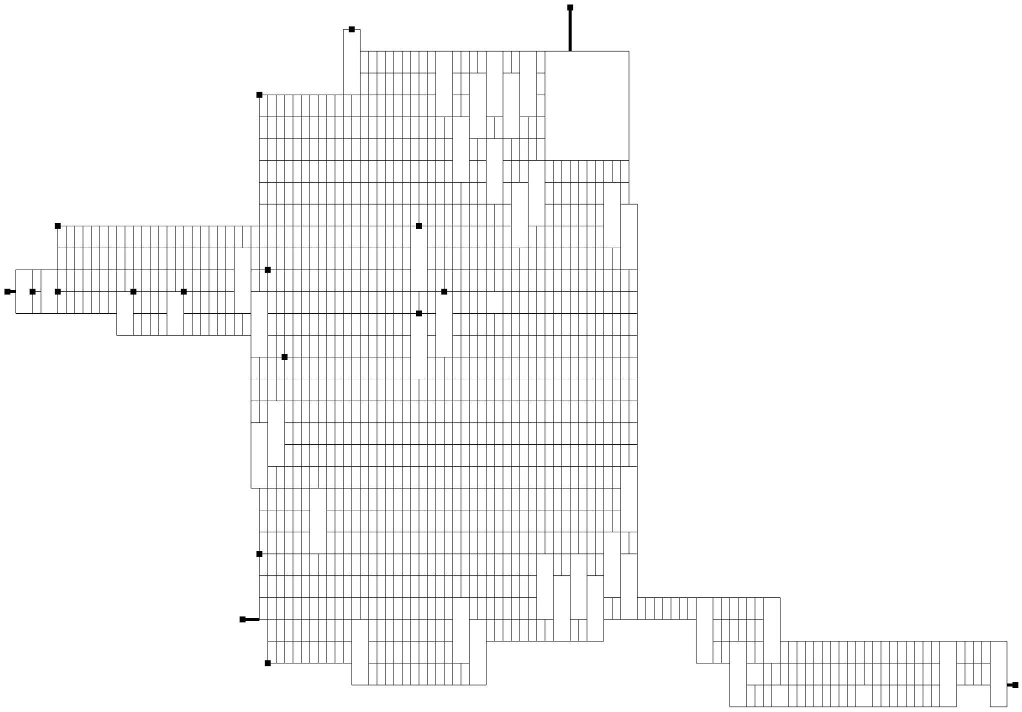

Figura 3.4: diw0559 após os testes SDE, BDk e TDist, $|V| = 1452$, $|E| = 2694$, $|T| = 18$.

3.2.3 Testes baseados em limites

Os testes de redução baseados em limites são aqueles que exploram a relação entre limites inferiores e superiores para a solução ótima do problema. Por exemplo, se com a suposição de que a aresta e está na solução for possível calcular um limite inferior para o problema que seja maior que o limite superior dado por uma solução heurística, fica provado que e é redundante. A título de exemplo, mostramos um teste baseado em limites introduzido por Duin.

TESTE 3.9. *Para cada terminal $t \in T$, defina $V(t)$ como o conjunto do vértices mais próximos de t do que de qualquer outro terminal, e $r(t)$ como o custo do menor caminho entre t e um vértice fora de $V(t)$. Sejam t_1 e t_2 os dois terminais mais próximos de um certo vértice não-terminal u. O valor $d(u,t_1) + d(u,t_2) + \sum_{t \neq t_1 \neq t_2 \in T} r(t)$ é um limite inferior para o custo de todas as árvores de Steiner que usam u. Se esse limite ultrapassar o custo da melhor árvore de Steiner conhecida, o vértice u e suas arestas adjacentes podem ser eliminados.*

Testes baseados em limites podem ser bastante efetivos em instâncias relativamente pequenas ou fáceis, mas infelizmente não são robustos: a medida que as instâncias crescem, cresce também a diferença absoluta entre os limites inferiores e superiores que se consegue calcular em tempo razoável (ou mesmo polinomial). A partir de um certo ponto, tipicamente quando essa diferença passa a ser algumas vezes maior do que o custo médio de uma aresta, os testes deixam de funcionar. Por exemplo, o procedimento de pré-processamento de Polzin e Vahdati [53] se utiliza desse tipo de teste, alguns deles com limites inferiores calculados por *dual ascent*, e consegue boas reduções em várias instâncias VLSI mais simples, como a diw0559. Entretanto, esse pré-processamento já falha totalmente para as instâncias VLSI razoavelmente grandes ou difíceis, como a taq0903. Para instâncias do porte da alue7080 então, não há a menor esperança de que testes baseados em limites obtidos por *dual ascent* possam funcionar.

Em contraste, os testes baseados em alternativas são robustos, no sentido que eles podem funcionar para instâncias arbitrariamente grandes, desde que a estrutura local dessas instâncias não mude. Por exemplo, o algoritmo de pré-processamento que será apresentado nesta tese consegue boas taxas de redução em instâncias VLSI de qualquer tamanho.

Um novo teste de redução por limites introduzido nesta tese será discutido na Seção 4.3.3, sob o nome de *fixação ativa por custos reduzidos*. Não nos pareceu adequado considerar esse teste como parte do *pré-processamento*, afinal de contas o uso de sofisticados e caros mecanismos para se obter os limites inferiores e superiores já faz parte do *processamento* da instância.

3.3 Novos Testes de Redução

Introduzimos dois novos testes de redução para o SPG que generalizam os testes SDE e BD3.

3.3.1 SDE com Expansão

Considere-se o pedaço de instância mostrado na Figura 3.5. A aresta (v_1, u_1) não pode ser eliminada pelo teste SDE, pois $c(v_1, u_1) = 3 < B(v_1, u_1)^{-(v_1, u_1)} = 4$. Mas façamos o seguinte raciocínio. Suponha que a aresta (v_1, u_1) esteja em todas as ASMs (Suposição 1). Seja R uma dessas ASMs. Como u_1 é não-terminal, esse vértice deve ter grau pelo menos 2 em R. Suponha que a aresta (u_1, p) esteja em R (Suposição 2). Removendo-se a aresta (v_1, u_1) de R obtemos duas sub-árvores R_{v_1} e R_{u_1}, e sabemos que $p \in R_{u_1}$. Mas $B(p, v_1) = 2 < c(u_1, v_1) = 3$, o que implica que podemos reconectar R_{v_1} e R_{u_1} usando as arestas (v_1, t) (se t estiver em R_{u_1}) ou (t, p) (se t estiver em R_{v_1}) e assim obter uma nova árvore de Steiner R' tal que $c(R') < c(R)$. Isso é uma contradição e assim fica provado que a Suposição 2 é falsa. Como (u_1, p) não está em R, provamos que (u_1, u_2) está em R. Mas $B(u_2, v_1)^{-(u_1,v_1)} = 5 \leq c(v_1, u_1) + c(u_1, u_2) = 5$. Logo é possível obter uma árvore de Steiner R' que não usa (v_1, u_1) e tal que $c(R') \leq c(R)$. Isso é uma contradição, e assim fica provado que a Suposição 1 é falsa. Ou seja, a aresta (v_1, u_1) é redundante.

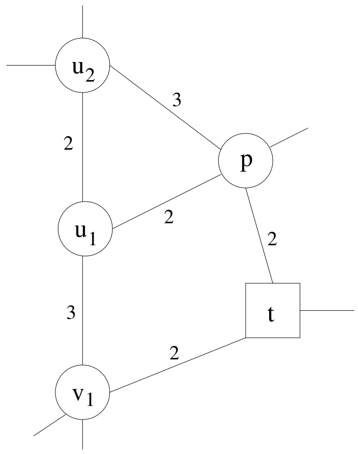

Figura 3.5: Ilustração do teste SDE com Expansão.

O teste SDE com expansão consiste em um procedimento para executar de forma sistemática raciocínios do tipo mostrado acima.

TESTE 3.10. SDE com Expansão - *Se o procedimento SDE com Expansão aplicado numa aresta* $(u_1, v_1) \in E$ *retornar* sucesso*, então* (u_1, v_1) *é redundante.*

```
Procedimento SDE com Expansão
Entrada: aresta (u_1, v_1) em E;
  P ⟵ {(u_1, v_1)};
  Repita {
      expandido ⟵ falso;
      /* Seja P = {(u_m, u_{m-1}), ... , (u_1, v_1), ..., (v_{n-1}, v_n)} o caminho corrente */
      Se (v_n é não-terminal) {
          i ⟵ 0;
          Para cada vértice p ∉ P adjacente a v_n
              Se (B(p, u_n)^{-(u_1,v_1)} > c(P)) {
                  i ⟵ i + 1;
                  v_{n+1} ⟵ p;
              }
          Se (i = 0 ) retorne sucesso;
          Se (i = 1 ) {
              P ⟵ {(u_m, u_{m-1}), ... , (u_1, v_1), ..., (v_{n-1}, v_n), (v_n, v_{n+1})};
              expandido ⟵ verdadeiro;
          }
      }
      /* Seja P = {(u_m, u_{m-1}), ... , (u_1, v_1), ..., (v_{n-1}, v_n)} o caminho corrente */
      Se (u_m é não-terminal) {
          i ⟵ 0;
          Para cada vértice p ∉ P adjacente a u_m
              Se (B(p, v_n)^{-(u_1,v_1)} > c(P)) {
                  i ⟵ i + 1;
                  u_{m+1} ⟵ p;
              }
          Se (i = 0 ) retorne sucesso;
          Se (i = 1 ) {
              P ⟵ {(u_{m+1}, u_m), (u_m, u_{m-1}), ... , (u_1, v_1), ..., (v_{n-1}, v_n)};
              expandido ⟵ verdadeiro;
          }
      }
  } Enquanto (expandido = verdadeiro)
  Se (B(u_m, v_n)^{-(u_1,v_1)} ≤ c(P)) retorne sucesso;
  Senão retorne falha;
```

TEOREMA 3.4. *O Teste SDE com Expansão é válido.*

Prova. Suponha que todas as ASMs contenham o caminho $P = \{(u_m, u_{m-1}), \dots, (u_1, v_1), \dots, (v_{n-1}, v_n)\}$ (inicialmente $P = \{(u_1, v_1)\}$). Seja R uma dessas ASMs. Seja v_n um vértice não-terminal no extremo de P e seja $p \notin P$ um vértice adjacente a v_n. Caso $B(p, v_n)^{-(u_1,v_1)} \leq c(P)$, fica provado que (v_n, p) não pode estar em R, caso contrário, por um raciocínio semelhante ao do Teorema 3.1, seria possível construir uma árvore R' que não usa P e tal que $c(R') \leq c(R)$. Testando essa condição para todos os possíveis vértices p, podemos chegar a uma das seguintes condições.

(a) O contador $i = 0$. Isso significa que nenhuma aresta adjacente a p está em R. Nesse caso fica provado que a suposição é falsa, existe uma ASM que não contém P.

(b) O contador $i = 1$. Fica provado que R também deve conter uma certa aresta $(v_n, p) = (v_n, v_{n+1})$. Dizemos que o caminho P foi *expandido* para $\{(u_m, u_{m-1}), \dots, (u_1, v_1), \dots, (v_{n-1}, v_n), (v_n, v_{n+1})\}$. O ponto é que a suposição de que todas as ASMs contêm P só pode ser verdadeira se todas as ASMs contiverem o novo caminho P expandido.

(c) O contador $i \geq 2$. Não podemos concluir nada.

O mesmo raciocínio é aplicado também ao outro extremo de P, até que não seja possível concluir mais nada. Então aplica-se um teste similar ao SDE, considerando-se todo o caminho P como uma única aresta. Caso $B(u_m, v_n)^{-(u_1,v_1)} \leq c(P)$, fica provado que a suposição que todas as ASMs contêm P é falsa. Remontando a cadeia de suposições, fica provado que a suposição inicial de que todas as ASMs contêm $P = \{(u_1, v_1)\}$ é falsa. □

3.3.2 BD3 com Expansão

Considere-se o pedaço de instância mostrado na Figura 3.6, os custos das arestas são proporcionais ao comprimento das linhas. Não é possível aplicar o teste BD3 sobre o vértice u. Mas façamos o seguinte raciocínio. Suponha que u tenha grau 3 em todas as ASMs (Suposição 1). Seja R uma dessas ASMs. Como v_1 é não-terminal, esse vértice deve ter grau pelo menos 2 em R. Suponha que a aresta (v_1, c) esteja em R (Suposição 2). Removendo-se a aresta (u, v_1) de R, obtemos duas sub-árvores R_u e R_{v_1}, e sabemos que $c \in R_{v_1}$ e $w_1, z_1 \in R_u$. Mas como $c(c, f) + c(f, w_1) = c(u, v_1)$, podemos reconetar R_u e R_{v_1} obtendo uma nova árvore de Steiner R' em que u tem grau 2 e tal que $c(R') = c(R)$. Isso é uma contradição e, portanto, provamos que a Suposição 2 é falsa. Por um raciocínio semelhante mostramos que (v_1, d) também não pode estar em R, logo (v_1, v_2) tem que estar em R. Da mesma forma, $(v_2, b) \notin R$ e, portanto, $(v_2, v_3) \in R$. Por último, descobrimos que $(z_1, d) \notin R$ e $(z_1, z_2) \in R$. Agora podemos considerar os três caminhos $P_v = \{(u, v_1), (v_1, v_2), (v_2, v_3)\}$, $P_z = \{(u, z_1), (z_1, z_2)\}$ e $P_w = \{(u, w_1)\}$ como se fossem três arestas e aplicar o teste BD3.

Como $B(v_3, z_2) + B(z_2, w1) = c(P_v) + c(P_w) + c(P_z)$, por um raciocínio similar ao do Teorema 3.2, concluímos que existe uma árvore de Steiner R' em que u tem grau no máximo 2 e tal que $c(R') = c(R)$. Isso contradiz a Suposição 1. Logo, podemos aplicar a ação do teste BD3 sobre u. Nesse caso particular, as novas arestas (v_1, z_1) e (v_1, w_1) seriam imediatamente removidas pelo teste SDE e o resultado final de todo o processo seria remover a aresta (u, v_1).

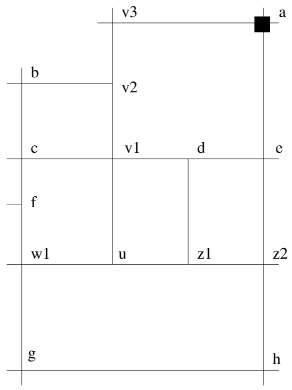

Figura 3.6: Ilustração do teste BD3 com Expansão.

O teste BD3 com expansão consiste em um procedimento para executar de forma sistemática raciocínios do tipo mostrado acima.

TESTE 3.11. BD3 com Expansão - *Se o procedimento BD3 com Expansão aplicado num vértice u de grau 3 retornar* sucesso*, então existe uma ASM em que u tem grau no máximo 2. Logo, as ações do teste BD3 usual podem ser executadas.*

Antes de mostrar esse procedimento, defimos a distância de Steiner gargalo entre os vértices u e v, mas supondo que uma certa aresta (x, y) tem custo zero e que os vértices $\{x, y\}$ podem ser considerados como terminais:

$$B(u,v)^{+(x,y)} = \min \left\{ \begin{array}{c} B(u,v), \\ \max\{B(u,x), B(y,v)\}, \max\{B(u,y), B(x,v)\}, \\ \max\{B(u,x), B(x,v)\}, \max\{B(u,y), B(y,v)\} \end{array} \right\}$$

Recordando a notação, se P é um caminho simples (sem vértices repetidos) e os vértices u e v estão em P, $P(u, v)$ é o subcaminho em P de u até v.

Procedimento BD3 com Expansão

Entrada: vértice u de grau 3 adjacente a v_1, w_1 e z_1;

$P_v \longleftarrow \{(u, v_1)\}$;

$P_w \longleftarrow \{(u, w_1)\}$;

$P_z \longleftarrow \{(u, z_1)\}$;

Repita {

$expandido \longleftarrow falso$;

/* Seja $P_v = \{(u, v_1), \ldots, (v_{n-1}, v_n)\}$ o caminho P_v corrente */

Se (v_n é não-terminal) {

$i \longleftarrow 0$;

Para cada vértice $p \notin P_v$ adjacente a v_n

Se (existirem vértices $w_j \in P_w$ e $z_k \in P_z$ tal que

$$\min \left\{ \begin{array}{l} B(v_n, w_j)^{+(v_n,p)} + B(v_n, z_k)^{+(v_n,p)}, \\ B(w_j, v_n)^{+(v_n,p)} + B(w_j, z_k)^{+(v_n,p)}, \\ B(z_k, v_n)^{+(v_n,p)} + B(z_k, w_j)^{+(v_n,p)} \end{array} \right\} \leq c(P_v) + c(P_w(u, w_j)) + c(P_z(u, z_k));$$

) {

$i \longleftarrow i + 1$;

$v_{n+1} \longleftarrow p$;

}

Se (i = 0) retorne *sucesso*;

Se (i = 1) {

$P_v \longleftarrow \{(u, v_1), \ldots, (v_{n-1}, v_n), (v_n, v_{n+1})\}$;

$expandido \longleftarrow verdadeiro$;

}

}

Tenta expandir P_w;

Tenta expandir P_z;

} Enquanto ($expandido = verdadeiro$)

Se (condição do Teste BD3 considerando P_v, P_w e P_z como arestas) retorne *sucesso*;

Senão retorne *falha*;

TEOREMA 3.5. *O Teste BD3 com Expansão é válido.*

Prova. Suponha que todas as ASMs contenham os caminhos $P_v = \{(u, v_1), \ldots, (v_{n-1}, v_n)\}$,

P_w e P_z (inicialmente todos esses caminhos contêm uma única aresta). Seja R uma dessas ASMs. Seja v_n um vértice não-terminal no extremo de P_v e seja $p \notin P_v$ um vértice adjacente a v_n. O ponto chave da prova é mostrar que se a condição indicada no procedimento for válida, (v_n, p) não pode estar em R.

Suponha que (v_n, p) esteja em R. Seja $S = \langle G, T, c \rangle$ a instância do PSG e seja $S' = \langle G, T \cup \{v_n, p\}, c' \rangle$ uma instância modificada onde $c'(v_n, p) = 0$. A árvore R ainda é uma ASM para S' com custo $c'(R) = c(R) - c(v_n, p)$. Para qualquer par de vértices $x, y \in V$, $B'(x, y) = B(x, y)^{+(v_n, p)}$. Assim, aplicando o raciocínio do Teorema 3.2 em S', considerando P_v, $P_w(u, w_j)$ e $P_z(u, z_k)$ como arestas, pode ser concluído que existe uma árvore de Steiner R' com custo $c'(R') \leq c(R) - c(v_n, p)$ tal que u tem grau no máximo 2. Assim, $R' \cup \{(v_n, p)\}$ é uma árvore de Steiner para S, onde u tem grau no máximo 2 e com custo menor ou igual a $c(R)$. Contradição.

O resto é similar ao que foi feito no Teorema 3.4. Testando essa condição para todos os possíveis vértices p, podemos chegar a uma das seguinte condições.

(a) O contador $i = 0$. Isso significa que nenhuma aresta adjacente a p está em R. Nesse caso fica provado que a suposição é falsa, existe uma ASM que não contém P_v.

(b) O contador $i = 1$. Fica provado que R também deve conter uma certa aresta $(v_n, p) = (v_n, v_{n+1})$. Dizemos que o caminho P_v foi *expandido* para $\{(u, v_1), \ldots, (v_{n-1}, v_n), (v_n, v_{n+1})\}$. A suposição de que todas as ASMs contêm P_v, P_w e P_z só pode ser verdadeira se todas as ASMs também contiverem o novo caminho P_v expandido.

(c) O contador $i \geq 2$. Não podemos concluir nada.

O mesmo raciocínio é aplicado também a P_w e a P_z até que não seja possível concluir mais nada. Então aplica-se um teste similar ao BD3, considerando-se cada um dos três caminhos como uma única aresta. Se as condições desse teste se verificarem, fica provado que a suposição que todas as ASMs contêm P_v, P_w e P_z é falsa. Remontando a cadeia de suposições, fica provado que a suposição inicial de que todas as ASMs contêm as três arestas adjacentes a u é falsa. □

A idéia do teste BD3 com expansão não é absolutamente original. Em 1995, Winter [71] propôs uma versão mais simples desse teste (e que usa propriedades geométricas) nos grafos de grade induzidos pelo problema de Steiner retilinear. Mas não conhecemos nenhum autor que tenha usado testes semelhantes no PSG. Talvez um dos motivos a explicar o "esquecimento" do trabalho de Winter, é que a abordagem do problema de Steiner retilinear através da grade induzida foi amplamente suplantada pelas recentes técnicas baseadas em topologias (ver [69] por exemplo).

3.4 Procedimento de Pré-processamento

Os testes NTD1, NTD2, TDist, SDE com expansão e BD3 com expansão foram empregados na construção de um procedimento de pré-processamento que fosse particularmente efetivo para as instâncias VLSI.

3.4.1 Estratégia de redução

Num certo estágio do pré-processamento, é comum que haja muitas escolhas de reduções disponíveis. Um fato importante, é que o tamanho da instância final obtida, quando mais nenhuma redução é possível, depende dessas escolhas. Em outras palavras, a ordem em que os testes são aplicados faz diferença.

Depois de algumas experiências com as 116 instâncias VLSI da SteinLib, achamos conveniente dividir o pré-processamento em 2 etapas. Em ambas as etapas, os testes NTD1, NTD2, TDist e SDE com Expansão são aplicados em qualquer ordem e a primeira redução válida encontrada é executada. Mas na Etapa I, o teste BD3 com Expansão só é aplicado em uma aresta u, se pelo menos 2 das 3 novas arestas criadas podem ser imediatamente removidas pelo teste SDE (nesse caso, a ação do teste equivale a remover uma aresta adjacente a u). Na Etapa II, não há restrições ao teste BD3 com Expansão.

A questão é que a Etapa I preserva a estrutura de G, que continua sendo um grafo de grade com buracos, exceto por algumas poucas contrações de arestas devidas ao teste TDist. O grau de quase todos os vértices continua sendo no máximo 4. O pequeno grau dos vértices parece ser muito importante para que a expansão funcione adequadamente. Se as reduções da Etapa II são aplicadas diretamente sobre a instância original, o grau de vários vértices cresce rapidamente, o que bloqueia novas reduções.

A Figura 3.7 mostra a instância diw0559 reduzida ao fim da Etapa I. (Como a estrutura de grade está preservada, não é necessário escrever o custo das arestas no desenho, esses custos continuam sendo proporcionais ao tamanho dos segmentos). Na Etapa II, essa instância é ligeiramente reduzida para 164 arestas. A Figura 3.8 mostra a instância taq0903 após a Etapa I. Na Etapa II, essa instância é reduzida para 3034 arestas. Finalmente, na Figura 3.9, aparece a instância alue7080.

3.4.2 Implementação dos testes

A taxa de redução alcançada por um procedimento de pré-processamento depende de quais testes são aplicados e em qual ordem isso é feito. Já o seu tempo de execução depende de como esses testes são implementados, uma questão nada trivial, já que os modernos testes usam condições bastante complexas.

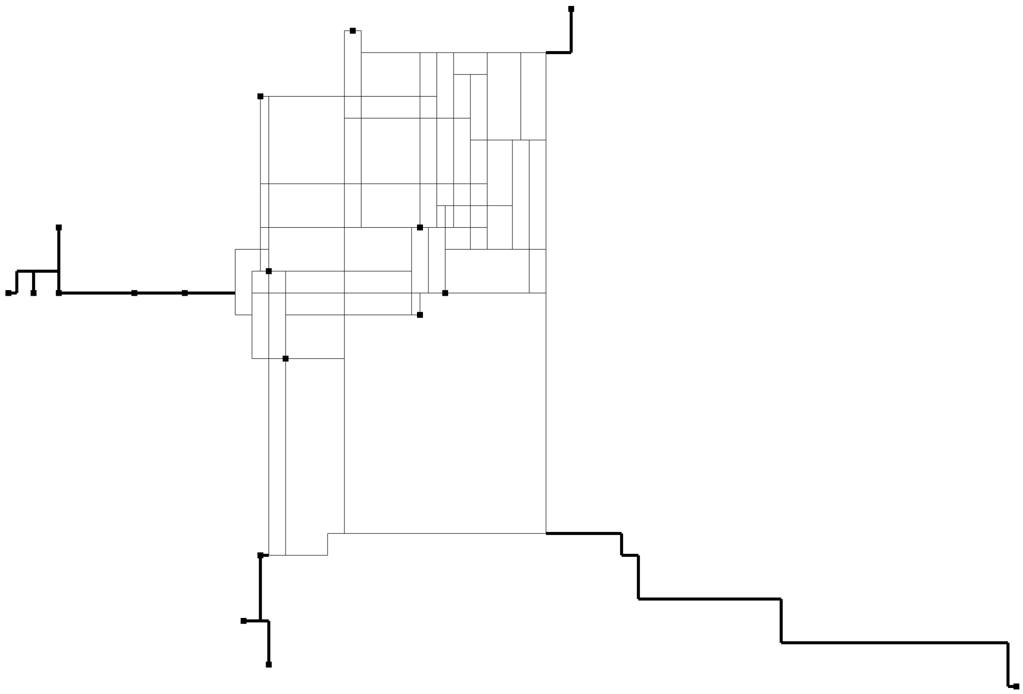

Figura 3.7: diw0559 após a Etapa I do pré-processamento, $|V| = 92$, $|E| = 151$, $|T| = 11$.

Os testes simples NTD1, NTD2 e TD1 podem ser executados em tempo $O(|V|)$. O teste TDist é razoavelmente barato. Seja $T = (V, E')$ uma árvore geradora de custo mínimo do grafo corrente G. Cada aresta $e \in E'$ induz uma partição de V correspondendo aos dois componentes conexos de $(V, E' \setminus \{e\})$. Duin [16] mostrou que apenas essas $|V| - 1$ partições precisam ser consideradas pelo teste TDist. Isso leva a um algoritmo de complexidade $O(|V|^2)$ para identificar todas as arestas satisfazendo a condição do teste TDist.

O custo de se executar os testes SD e BD3 está no cálculo das distâncias de Steiner gargalo. Isso continua sendo verdadeiro para os novos testes com expansão. A abordagem padrão, sugerida por Duin e Volgenant, é construir uma tabela com a distância de Steiner gargalo entre todos os pares de vértices. Isso pode ser feito em tempo $O(|V|^3)$. Qualquer sequência de reduções pelos testes NTD1, NTD2, SD ou BD3, não afeta as distâncias de Steiner entre os vértices que permanecem no grafo. Já uma contração de aresta devida ao teste TDist pode reduzir as distâncias de Steiner gargalo. Isso não é problema, pois nesse caso a tabela continua dando limites superiores para as verdadeiras distâncias de Steiner, o que leva a reduções válidas. De tempos em tempos, a tabela pode ser atualizada. Na verdade, como o tempo $O(|V|^3)$ para se construir uma tabela com as distâncias de gargalo exatas B é considerado exagerado, muitas vezes usa-se um dos vários procedimentos aproximados disponíveis para se obter uma tabela com bons limites superiores para essas distâncias em tempo $O(|V|^2 \cdot \log |V|)$ [16]. É claro que nesse caso, algumas reduções válidas podem ser perdidas.

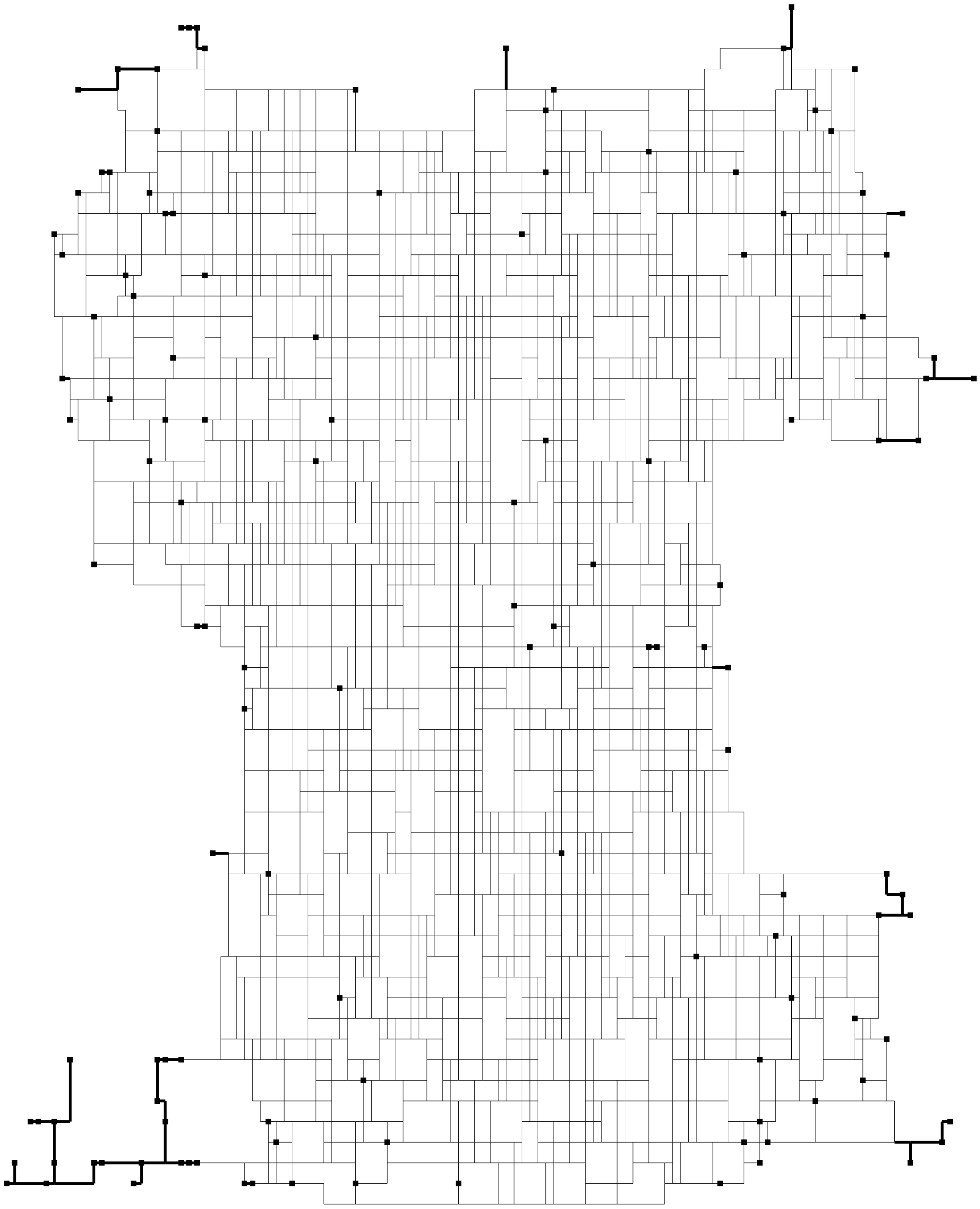

Figura 3.8: taq0903 após a Etapa I do pré-processamento, $|V| = 1995$, $|E| = 3387$, $|T| = 92$.

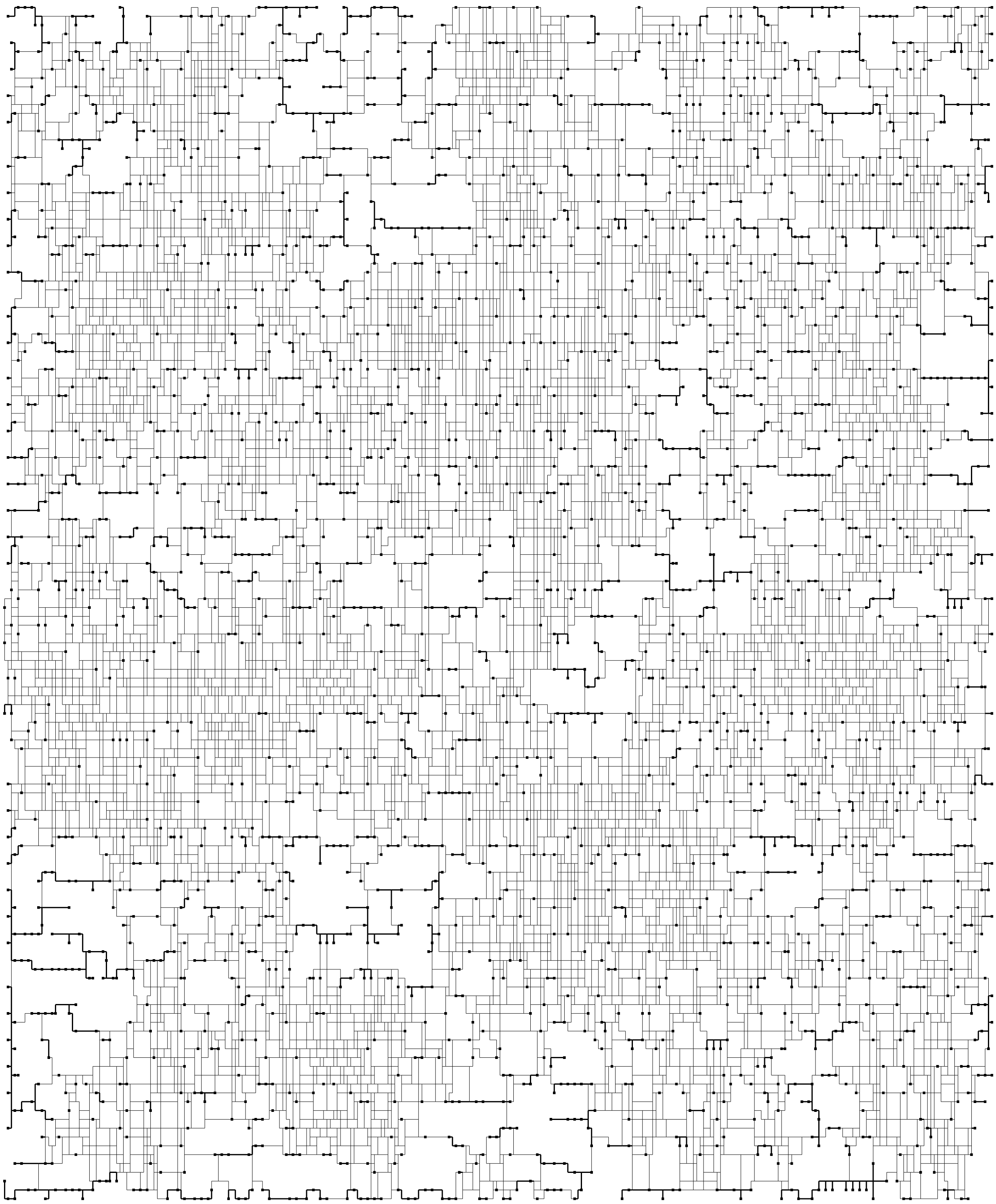

Figura 3.9: alue7080 após a Etapa I do pré-processamento, $|V| = 10.672$, $|E| = 17.645$, $|T| = 1433$.

Essa abordagem de manter uma tabela completa de distâncias gargalo, exatas ou aproximadas, não é conveniente para as instâncias VLSI:

- Os grafos de grade com buracos são muito esparsos, $|V| > |E|/2$. Assim, o tempo de cpu e mesmo a quantidade de memória necessária seria muito grande para as instâncias maiores. Por exemplo, a instância alue7080 tem 34.479 vértices, uma tabela completa de pares de vértices teria 594 milhões de entradas. Em contraste, a instância e18 tem mais ou menos o mesmo número de arestas, mas apenas 2500 vértices.

- Devido a estrutura de custos, é importante usar o teste SDE, que requer distâncias B^{-e}, que não podem ser facilmente obtidas de uma tabela B. Para complicar ainda mais a situação, uma redução feita pelo teste SDE (quando a igualdade ocorre) pode *aumentar* as distâncias de gargalo entre certos pares de vértices [16]. Isso significa que a tabela B teria que ser atualizada após cada aplicação do teste SDE. Isso seria impraticável.

Optamos por calcular distâncias de Steiner gargalo "na hora", a medida que elas são pedidas pelos testes. Aproximamos $B(u, v)$ (e também B^{-e}, B^{+e}) considerando no máximo 2 terminais em cada possível caminho entre u e v:

$$\tilde{B}(u, v) = \min\{d(u, v), \max_{x \in T}\{d(u, x), d(x, v)\}, \max_{x,y \in T}\{d(u, x), d(x, y), d(y, v)\}\}.$$

Esses cálculos repetidos são um tanto caros, mas não são proibitivos porque os testes SDE e BD3, mesmo com expansão, são bastante locais. Isso significa que na maior parte do tempo temos que calcular distâncias entre vértices que estão muito próximos entre si, e apenas um pequeno subgrafo de G precisa ser considerado. Na prática, o custo de calcular cada $\tilde{B}(u, v)$ é quase uma constante. Nossas experiências indicaram que a perda na taxa de redução causada por se usar $\tilde{B}(u, v)$ ao invés de $B(u, v)$ é mínima.

3.5 Experimentos Computacionais

O procedimento de pré-processamento foi implementado em C++ e os tempos medidos numa máquina Sun ULTRA 167 MHz, no sistema operacional Solaris. As 116 instâncias VLSI de teste são encontradas na SteinLib [39]. As colunas nas tabelas 3.1–3.4 têm o seguinte significado. As três primeiras colunas são o tamanho da instância original. As três colunas seguintes são o tamanho da instância reduzida final, após a aplicação das Etapas I e II do pré-processamento. A coluna seguinte dá o tempo de cpu em segundos gasto pelo procedimento. As duas últimas colunas são respectivamente, o valor da solução base (a soma dos custos das arestas escolhidas pelo pré-processamento), e o valor de uma solução ótima da instância. Alguns comentários:

- Foi possível resolver 43 das 116 instâncias apenas com pré-processamento baseado em alternativas. Isso inclui 9 instâncias com mais de 2000 arestas, e até uma instância (dmxa1010) que não pode ser resolvida nem com *branch-and-cut* em Koch e Martin [38]. Outras 48 instâncias foram reduzidas a menos de 10% do seu tamanho original (medido pelo número de arestas). Todas as demais instâncias foram reduzidas a menos de 32% do tamanho original, exceto na alue7065, que só foi reduzida a 38%.

- O tempo gasto pelo procedimento de pré-processamento é razoavelmente pequeno para instâncias da ordem de milhares de arestas, mas já começa a pesar para instâncias da ordem de dezenas de milhares de arestas. Isso talvez cause dificuldades nas instâncias VLSI vindas de aplicações mais modernas, que podem chegar a centenas de milhares de arestas. De qualquer maneira, o pré-processamento contribuiu bastante para que fosse possível resolver as instâncias alue7065, alue7080, alut2610 e alut2625 pela primeira vez, conforme será mostrado no capítulo seguinte.

3.6 Comentários Finais

- Recentemente, em outubro de 2000, tomamos conhecimento de um artigo de Polzin e Vahdati [53] que descreve um procedimento de pré-processamento que implementa boas aproximações de todos os testes clássicos de Duin e Volgenant em $O(|V| \cdot \log |V| + |E|)$, o que leva a excelentes tempos de execução. Em particular, é dito que em [51] foi possível caracterizar exatamente em que situações o teste SDE causa aumento nas distâncias de Steiner gargalo e, nesses casos, atualizar rapidamente as estruturas de dados. É possível que usando essas idéias seja possível reimplementar os novos testes com expansão de forma bem mais eficiente.

- Os novos testes com expansão foram criados tendo em mente o pré-processamento de instâncias VLSI, mas eles também podem ser efetivos para outros tipos de instâncias do PSG, desde que elas sejam *esparsas*. Isso inclusive pode acontecer em instâncias originalmente densas, caso os testes tradicionais sejam suficientes para torná-las esparsas. Por exemplo, a instância e20 da OR-Lib é razoavelmente densa, com 2500 vértices e 62.500 arestas. Os testes NTD1, NTD2, SDE, BD3 e TDist a reduzem a 1257 vértices e 2330 arestas. Introduzindo-se os novos testes SDE e BD3 com expansão a partir desse ponto, a instância é completamente resolvida.

- Os novos testes têm uma filosofia distinta dos testes tradicionais. Ao usar um teste de Duin e Volgenant, o pré-processamento verifica se uma certa condição é satisfeita e, se for o caso, executa a ação correspondente. Naturalmente, existe um teorema, provado no papel, que garante a validade do processo. Já nos testes com expansão, não se sabe *a priori* que condições devem ser satisfeitas. O que se faz é assumir uma certa hipótese inicial e, usando condições semelhantes àquelas usadas nos testes tradicionais, tentar demonstrar que outras hipóteses são ou não compatíveis com ela. Os fatos já demonstrados servem de apoio para novas demonstrações, até que uma das seguintes situações ocorra: (a) Não é possível concluir mais nada. Nesse caso diz-se que o teste "falhou" e a instância permanece inalterada. (b) Chega-se a uma contradição, logo fica provado que a hipótese inicial era falsa e pode-se adotar a ação correspondente. Com um pouco de licença poética, pode-se dizer que o procedimento acabou de "provar um teorema".

De qualquer forma, os procedimentos SDE com expansão e BD3 com expansão estão longe de esgotar os recursos de um "provador de teoremas" nesse contexto. Por exemplo, suponha que o vértice x seja um terminal de grau 2 adjacente aos vértices y e z, e que em certo ponto do procedimento SDE com expansão seja concluído que (x, y) não pertence a uma certa ASM R. Uma implicação, não explorada no procedimento atual, é que (x, z) também deve pertencer a R. Logo z pode ser considerado como um terminal no cálculo das distâncias de Steiner gargalo, o que pode facilitar novas expansões. Além disso, se z é não-terminal, ele deve ter grau 2 em R, logo (x, z) pode ser considerado como um segundo caminho a ser expandido, etc.

Apenas experimentos podem avaliar se as novas reduções obtidas com procedimentos mais sofisticados, valem o aumento nos custos computacionais.

- Na Seção 3.2.3 foi comentado que os testes de redução baseados em limites deixam de funcionar após um certo tamanho de instância, quando a diferença entre os limites inferiores e superiores que se consegue calcular em tempo razoável ultrapassa um certo patamar. Talvez seja possível usar a idéia dos testes baseados em expansão para criar novos testes baseados em limites que funcionem para instâncias maiores.

  Por exemplo, suponha que usamos o teste SDE para mostrar que uma certa aresta e é redundante e que, após algumas expansões, o teste tenha falhado. Mesmo com essa falha, ficou provado que se e está em uma certa ASM R, alguns outros vértices e arestas também devem ou não devem estar em R. Todas essas informações podem ser usadas para obter limites inferiores mais fortes (por exemplo, usando custos reduzidos), e caso esse limite ultrapasse o limite superior conhecido, fica provado que e é redundante.

	Tamanho Original			Tamanho Final			Tempo	Solução	
Instância	$\|V\|$	$\|E\|$	$\|T\|$	$\|V\|$	$\|E\|$	$\|T\|$	(segs.)	Base	Ótima
alue2087	1244	1971	34	40	65	13	2	670	1049
alue2105	1220	1858	34	1	0	1	1	1032	1032
alue3146	3626	5869	64	65	104	26	15	1244	2240
alue5067	3524	5560	68	300	504	38	77	1127	2586
alue5345	5179	8165	68	1100	1860	61	109	229	3507
alue5623	4472	6938	68	768	1307	57	52	365	3413
alue5901	11543	18429	68	1066	1821	59	81	349	3912
alue6179	3372	5213	67	441	752	49	49	645	2452
alue6457	3932	6137	68	793	1356	53	93	526	3057
alue6735	4119	6696	68	340	556	50	46	648	2696
alue6951	2818	4419	67	257	413	46	35	672	2386
alue7065	34046	54841	544	11911	20956	426	7581	2583	23881
alue7066	6405	10454	16	1768	3109	9	300	86	2256
alue7080	34479	55494	2344	8355	14387	1328	11222	18790	62449
alue7229	940	1474	34	1	0	1	1	824	824
alut0787	1160	2089	34	9	13	5	1	903	982
alut0805	966	1666	34	34	54	16	2	492	958
alut1181	3041	5693	64	207	369	39	26	699	2353
alut2010	6104	11011	68	374	641	45	74	743	3307
alut2288	9070	16595	68	1195	2136	60	269	466	3843
alut2566	5021	9055	68	701	1223	62	63	213	3073
alut2610	33901	62816	204	10576	19829	182	6990	810	12239
alut2625	36711	68117	879	11759	21628	763	8832	2410	35459
alut2764	387	626	34	1	0	1	1	640	640
diw0234	5349	10086	25	760	1413	21	221	242	1996
diw0250	353	608	11	1	0	1	1	350	350
diw0260	539	985	12	1	0	1	1	468	468
diw0313	468	822	14	1	0	1	1	397	397
diw0393	212	381	11	1	0	1	1	302	302

Tabela 3.1: Resultados do pré-processamento nas séries alue, alut e diw.

Instância	Tamanho Original $\|V\|$	$\|E\|$	$\|T\|$	Tamanho Final $\|V\|$	$\|E\|$	$\|T\|$	Tempo (segs.)	Solução Base	Ótima
diw0445	1804	3311	33	27	42	10	5	957	1363
diw0459	3636	6789	25	19	30	8	2	1035	1362
diw0460	339	579	13	1	0	1	1	345	345
diw0473	2213	4135	25	14	22	6	2	871	1098
diw0487	2414	4386	25	4	5	3	4	1340	1424
diw0495	938	1655	10	1	0	1	1	616	616
diw0513	918	1684	10	1	0	1	1	604	604
diw0523	1080	2015	10	1	0	1	1	561	561
diw0540	286	465	10	1	0	1	1	374	374
diw0559	3738	7013	18	83	141	11	17	697	1570
diw0778	7231	13727	24	187	337	15	40	900	2173
diw0779	11821	22516	50	2377	4490	39	1390	649	4440
diw0795	3221	5938	10	225	400	10	104	172	1550
diw0801	3023	5575	10	334	605	10	155	106	1587
diw0819	10553	20066	32	1394	2604	26	803	302	3399
diw0820	11749	22384	37	1816	3407	32	1756	428	4167
dmxa0296	233	386	12	1	0	1	1	344	344
dmxa0368	2050	3676	18	47	76	9	2	525	1017
dmxa0454	1848	3286	16	1	0	1	1	914	914
dmxa0628	169	280	10	1	0	1	1	275	275
dmxa0734	663	1154	11	1	0	1	1	506	506
dmxa0848	499	861	16	34	54	11	1	169	594
dmxa0903	632	1087	10	53	90	7	3	160	580
dmxa1010	3983	7108	23	1	0	1	3	1488	1488
dmxa1109	343	559	17	9	13	5	1	293	454
dmxa1200	770	1383	21	29	42	13	1	267	750
dmxa1304	298	503	10	1	0	1	1	311	311
dmxa1516	720	1269	11	1	0	1	1	508	508
dmxa1721	1005	1731	18	4	5	3	1	607	780
dmxa1801	2333	4137	17	310	553	17	40	159	1365

Tabela 3.2: Resultados do pré-processamento nas séries diw e dmxa.

Instância	Tamanho Original $\|V\|$	$\|E\|$	$\|T\|$	Tamanho Final $\|V\|$	$\|E\|$	$\|T\|$	Tempo (segs.)	Solução Base	Ótima
gap1307	342	552	17	1	0	1	1	549	549
gap1413	541	906	10	1	0	1	1	457	457
gap1500	220	374	17	1	0	1	1	254	254
gap1810	429	702	17	1	0	1	1	482	482
gap1904	735	1256	21	7	9	4	4	645	763
gap2007	2039	3548	17	41	70	9	2	611	1104
gap2119	1724	2975	29	1	0	1	3	1244	1244
gap2740	1196	2084	14	13	19	5	1	359	745
gap2800	386	653	12	1	0	1	1	386	386
gap2975	179	293	10	1	0	1	1	245	245
gap3036	346	583	13	28	42	9	1	146	457
gap3100	921	1558	11	16	25	8	1	151	640
gap3128	10393	18043	104	2146	3903	60	769	1501	4292
msm0580	338	541	11	7	9	4	1	367	467
msm0654	1290	2270	10	1	0	1	1	823	823
msm0709	1442	2403	16	1	0	1	2	884	884
msm0920	752	1264	26	7	9	4	3	674	806
msm1008	402	695	11	13	18	6	1	249	494
msm1234	933	1632	13	7	9	4	1	386	550
msm1477	1199	2078	31	12	16	6	2	924	1068
msm1707	278	478	11	1	0	1	1	564	564
msm1844	90	135	10	6	8	4	1	122	188
msm1931	875	1522	10	4	5	3	1	187	604
msm2000	898	1562	10	1	0	1	1	594	594
msm2152	2132	3702	37	191	333	24	13	444	1590
msm2326	418	723	14	9	12	5	1	286	399
msm2492	4045	7094	12	39	61	9	6	937	1459
msm2525	3031	5239	12	11	15	6	3	1129	1290
msm2601	2961	5100	16	178	305	12	27	441	1440

Tabela 3.3: Resultados do pré-processamento nas séries gap and msm.

Instância	Tamanho Original $\|V\|$	$\|E\|$	$\|T\|$	Tamanho Final $\|V\|$	$\|E\|$	$\|T\|$	Tempo (segs.)	Solução Base	Ótima
msm2705	1359	2458	13	4	5	3	1	615	714
msm2802	1709	2963	18	7	11	4	2	805	926
msm2846	3263	5783	89	347	595	58	49	944	3135
msm3277	1704	2991	12	1	0	1	1	869	869
msm3676	957	1554	10	1	0	1	1	607	607
msm3727	4640	8255	21	1	0	1	11	1376	1376
msm3829	4221	7255	12	338	594	10	52	430	1571
msm4038	237	390	11	1	0	1	1	353	353
msm4114	402	690	16	1	0	1	1	393	393
msm4190	391	666	16	1	0	1	1	381	381
msm4224	191	302	11	1	0	1	1	311	311
msm4312	5181	8893	10	1299	2355	10	199	149	2016
msm4414	317	476	11	1	0	1	1	408	408
msm4515	777	1358	13	35	55	8	1	195	630
taq0014	6466	11046	128	1717	3064	82	290	1167	5326
taq0023	572	963	11	37	63	7	1	236	621
taq0365	4186	7074	22	984	1771	21	77	170	1914
taq0377	6836	11715	136	2020	3577	115	319	856	6393
taq0431	1128	1905	13	108	188	10	14	222	897
taq0631	609	932	10	8	11	4	1	361	581
taq0739	837	1438	16	64	105	12	6	153	848
taq0741	712	1217	16	84	140	14	7	62	847
taq0751	1051	1791	16	104	178	14	7	154	939
taq0891	331	560	10	1	0	1	1	319	319
taq0903	6163	10490	130	1704	3034	90	244	788	5099
taq0910	310	514	17	1	0	1	1	370	370
taq0920	122	194	17	1	0	1	1	210	210
taq0978	777	1239	10	1	0	1	1	566	566

Tabela 3.4: Resultados do pré-processamento nas séries msm e taq.

Capítulo 4

Heurísticas Duais em Algoritmos Exatos para o Problema de Steiner em Grafos

O Problema de Steiner em Grafos (PSG) é definido da seguinte maneira. Seja $G = (V, E)$ um grafo conexo com custos inteiros positivos $c(e)$ nas arestas (também denotados como $c(u, v)$ quando $e = (u, v)$) e $T \subseteq V$ um conjunto de vértices, conhecidos como *terminais*. Uma *árvore de Steiner* é uma sub-árvore (V', E') de G tal que $T \subseteq V'$. O PSG é o problema de encontrar uma árvore de Steiner que minimize $\sum_{e \in E'} c(e)$.

4.1 Formulações

Aneja [2] propôs a seguinte formulação do PSG como um problema de programação linear inteira. Para cada aresta $e \in E$, defina uma variável binária x_e tal que $x_e = 1$ se e pertence a uma árvore de Steiner. Seja $[W, \bar{W}]$ uma partição dos vértices em V. Se tanto W quanto $\bar{W}$ tiverem pelo menos um terminal, o corte $\delta(W) = \{(u, v) \in E \mid u \in W, v \in \bar{W}\}$ é chamado de *corte de Steiner*.

$$
(P_c) \left\{
\begin{array}{llll}
\text{Min} & \displaystyle\sum_{e \in E} c_e x_e & & (1) \\
\text{s.t.} & \displaystyle\sum_{e \in \delta(W)} x_e \geq 1 & \forall\, W \text{ definindo um corte de Steiner} & (2) \\
& x_e \geq 0 & \forall\, e \in E & (3) \\
& x_e \text{ Inteiro} & \forall\, e \in E & (4)
\end{array}
\right.
$$

A formulação por cortes (P_c) é a mais simples e natural para o problema. Infelizmente sua relaxação linear costuma ser fraca e essa formulação não leva a bons algoritmos exatos. Na verdade, é provado que o valor dessa relaxação nunca é menor do que a metade da solução ótima inteira (ver Goemans e Williamson [26]). Mas, essa garantia teórica pouco significa na prática, onde diferenças de alguns pontos percentuais já inviabilizam a solução de uma instância. Um modo de reforçar essa formulação é através do estudo dos poliedros induzidos por P_c, de forma a identificar novas desigualdades válidas, se possível definindo facetas. Essa abordagem foi tentada por autores como Ferreira [20], Grötschel e Monma [29], Chopra e Rao [12] e Biha et al. [9]. Os algoritmos de *branch-and-cut* resultantes têm desempenho apenas razoável, há muitas dificuldades na separação das novas desigualdades encontradas.

Os melhores algoritmos exatos conhecidos para o PSG são baseados em uma das três formulações mais sofisticadas que serão mostradas a seguir. Todas elas levam a relaxações lineares com exatamente o mesmo valor, que costuma estar muito próximo ao valor do ótimo inteiro. De fato, não é conhecida nenhuma instância para a qual essa razão seja maior do que $7/8$.

A formulação por multifluxos (P_{mf}) foi introduzida de forma independente por Claus e Maculan [13] e Wong [72] (Beasley [6] propôs uma formulação por multifluxos em grafos não-direcionados). Seja $D = (V, A)$ o grafo direcionado obtido através da substituição de cada aresta $e = (u, v) \in E$ por um par de arcos opostos (u, v) e (v, u) com o mesmo custo de $c(e)$. Escolha um terminal r para ser a *raiz*. Uma *arborescência de Steiner* é uma sub-árvore de D em que existe um caminho orientado de r para todos os demais terminais. Há uma clara correspondência entre uma árvore de Steiner de custo mínimo em G e uma arborescência de Steiner de custo mínimo em D. Para cada aresta $a \in A$, defina uma variável binária x_a tal que $x_a = 1$ se a pertence a uma arborescência de Steiner. Para cada aresta $a \in A$ e para cada terminal $k \in T \setminus \{r\}$, defina uma variável f_a^k representando a quantidade de fluxo do tipo k que passa por um arco a. Seja $\delta^-(u)$ o conjunto dos arcos que entram no vértice u e $\delta^+(u)$ os arcos que saem de u.

$$
(P_{mf}) \begin{cases}
\text{Min} \quad \sum_{a \in A} c_a x_a & (1) \\
\text{s.t.} \quad \sum_{a \in \delta^-(u)} f_a^k - \sum_{a \in \delta^+(u)} f_a^k = 0 \quad \forall\, k \in T \setminus \{r\};\ \forall\, u \in V \setminus \{r, k\} & (2) \\
\qquad \sum_{a \in \delta^-(u)} f_a^k - \sum_{a \in \delta^+(u)} f_a^k = 1 \quad \forall\, k \in T \setminus \{r\};\ u = k & (3) \\
\qquad 0 \le f_a^k \le x_a \le 1 \quad \forall\, a \in A;\ \forall\, k \in T \setminus \{r\} & (4) \\
\qquad x_a \text{ Inteiro} \quad \forall\, a \in A & (5)
\end{cases}
$$

Essa formulação diz que uma unidade de fluxo do tipo k deve sair da raiz r e ir até cada terminal em $k \in T \setminus \{r\}$, e que esses fluxos só podem passar por arcos incluídos na arborescência de

Steiner. Uma característica interessante de P_{mf} é usar um número de variáveis e restrições limitado por um polinômio no tamanho da instância ($O(|E| \cdot |T|)$). Em princípio, é possível resolver a relaxação linear de P_{mf} resolvendo-se um único programa linear, embora isso só tenha um custo computacional aceitável quando $|T|$ é pequeno. Os algoritmos construídos sobre essa formulação usam métodos para aproximar a relaxação linear de P_{mf}. Em Wong [72] e Duin [16] é usada uma heurística de *dual ascent*. Recentemente, Bahiense et al. [4] usaram técnicas lagrangeanas.

A formulação por cortes direcionados (P_{cd}) foi introduzida por Wong [72] e também é definida sobre arborescências de Steiner em D. Seja $[W, \bar{W}]$ uma partição dos vértices em V. Se a raiz r estiver em W e pelo menos algum outro terminal estiver em $\bar{W}$, $\delta^+(W) = \{(u, v) \in A \mid u \in W,\ v \in \bar{W}\}$ é chamado de *corte direcionado de Steiner*. Seja $\mathcal{W}$ o conjunto de todos os subconjuntos de vértices W que definem cortes orientados de Steiner.

$$
(P_{cd}) \left\{
\begin{array}{llll}
\text{Min} & \displaystyle\sum_{a \in A} c_a x_a & & (1) \\
\text{s.t.} & \displaystyle\sum_{a \in \delta^+(W)} x_a \geq 1 & \forall\, W \in \mathcal{W} & (2) \\
 & x_a \geq 0 & \forall\, a \in A & (3) \\
 & x_a \text{ Inteiro} & \forall\, a \in A & (4)
\end{array}
\right.
$$

A equivalência entre as formulações P_{mf} e P_{cd} é uma decorrência do clássico teorema do fluxo máximo–corte mínimo (ver [72] ou [25]). A formulação P_{cd} é usada nos algoritmos de *branch-and-cut* em Koch e Martin [38] e Polzin e Vahdati [53].

A formulação por eliminação de ciclos generalizada (P_{ecg}) foi introduzida de forma independente por Lucena [44], Goemans [25] e Margot at al. [47]. A formulação P_{ecg} opera no grafo original G. Para cada vértice $u \in V$, defina uma variável binária adicional y_u, tal que $y_u = 1$ se u pertence a uma árvore de Steiner. Naturalmente, $y_u = 1$ para todos os vértices terminais u em T. Dado um conjunto de vértices S, defina $E(S)$ como o conjunto das arestas com ambos os extremos em vértices de S.

$$
(P_{ecg}) \begin{cases}
\text{Min} & \sum_{e \in E} c_e x_e & & (1)\\
\text{s.t.} & \sum_{e \in E} x_e = \sum_{u \in V} y_u - 1 & & (2)\\
& \sum_{e \in E(S)} x_e \leq \sum_{u \in S \setminus \{d\}} y_u & \forall\, S \subseteq V,\ \forall d \in S & (3)\\
& 0 \leq y_u \leq 1 & \forall\, u \in V & (4)\\
& 0 \leq x_e \leq 1 & \forall\, e \in E & (5)\\
& x_e \text{ Inteiro} & \forall\, e \in E & (6)
\end{cases}
$$

Uma prova da equivalência entre as formulações P_{ecg} e P_{cd} pode ser encontrada em [25]. A formulação P_{ecg} é usada no algoritmo de *branch-and-cut* em Lucena e Beasley [45].

Existem outras formulações para o PSG ainda mais fortes do que as descritas acima (ver [43] ou [52], por exemplo). Mas essas formulações levam a programas lineares extremamente grandes e ainda não foi possível mostrar que elas são computacionalmente viáveis. Nesta tese optamos trabalhar com a formulação P_{dc}. Em parte, essa escolha se deveu ao fato de que essa formulação já tinha sido testada com bons resultados em [38]. Mas existiu um outro motivo. Como em P_{dc} só existe um tipo de variável e um tipo de restrição, o seu dual é também muito simples e de compreensão mais intuitiva. Isso é conveniente para o desenvolvimento de programas que trabalham com esse dual.

4.2 Heurística de Dual Ascent

Conforme já dito, a relaxação linear das formulações fortes para o PSG costuma produzir limites inferiores muito próximos do valor da solução de uma instância. O ponto crítico dos melhores algoritmos exatos é justamente resolver uma dessas relaxações, um problema que, apesar de polinomial, é computacionalmente muito caro para instâncias grandes. Nesse caso, pode ser interessante a utilização de métodos heurísticos para aproximar o valor dessas relaxações. Um exemplo disso é a heurística de *dual ascent* proposta por Wong [72].

O dual da relaxação linear da formulação P_{cd} é:

$$(D_{cd})\begin{cases} \text{Max} \quad \sum_{W \in \mathcal{W}} \pi_W & (1) \\ \text{s.t.} \quad \sum_{W \in \mathcal{W}: a \in \delta^+(W)} \pi_W \leq c_a \quad \forall a \in A & (2) \\ \pi_W \geq 0 \quad \forall W \in \mathcal{W} & (3), \end{cases}$$

lembrando que $\mathcal{W}$ é o conjunto de todos os subconjuntos de vértices W que definem cortes orientados de Steiner. O valor de qualquer solução viável para D_{cd} é um limite inferior para o valor da solução ótima do PSG. A heurística de *dual ascent* de Wong [72] é um algoritmo combinatório para gerar boas soluções duais para essa formulação.

Seja π uma solução viável para D_{cd}. Definimos o *custo reduzido* de um arco a com respeito a π como:

$$\bar{c}_\pi(a) = c(a) - \sum_{W \in \mathcal{W}: a \in \delta^+(W)} \pi_W \ .$$

Dizemos que um arco está *saturado* se o seu custo reduzido for zero. Seja $D_\pi = (V, A')$ o subgrafo de D contendo apenas os arcos saturados. Uma *componente raiz* R é uma componente fortemente conexa de D_π que contém um terminal, não contém a raiz e tal que não existe um caminho em D_π de um terminal fora de R até R. Seja $W = w(R)$ o conjunto de vértices formado por R e pelos vértices que alcançam R em D_π. O algoritmo de *dual ascent* de Wong é o seguinte:

Dual Ascent

DUALASCENT(π)
  (como *default*, $\pi = \mathbf{0}$);
  Enquanto (existe uma componente raiz R em D_π) {
    $W \longleftarrow w(R)$;
    aumente π_W até que algum arco em $\delta^+(W)$ fique saturado;
  }
  Retorne π;

Ao final desse procedimento, $v(\pi)$, o valor da solução dual π, é um limite inferior para o valor da solução do PSG. Além disso, o grafo D_π contém uma arborescência de Steiner. É possível obter uma boa solução primal para o PSG removendo-se um conjunto de arcos redundantes (arcos cuja remoção não desconecta a raiz de algum terminal) de D_π. Adotamos a sugestão de Goemans e Williamson [26] para que essa remoção seja feita na ordem reversa da saturação. Suponha que os arcos em D_π foram saturados ao longo do procedimento de *dual ascent* na ordem $a_1, \ldots, a_m$. Se a_m desconecta a raiz de algum terminal em D_π, o arco permanece no grafo, senão é removido. O mesmo é feito sucessivamente com $a_{m-1}, \ldots, a_1$, até que só reste uma arborescência de Steiner.

- O algoritmo proposto originalmente por Wong trabalha no dual da formulação P_{mf}, mas a sua interpretação no dual da formulação P_{dc} é imediata.
- Quando $T = V$, esse *dual ascent* torna-se idêntico ao algoritmo exato proposto por Edmonds [18] para o problema da arborescência de custo mínimo.
- Esse algoritmo também se assemelha a heurística de *dual ascent* proposta por Bilde e Krarup [10] e Erlenkotter [19] para o problema de localização de instalações.

O algoritmo de *dual ascent* deixa em aberto o critério de escolha de qual componente raiz deve ser selecionada em cada iteração. Após uma série de experiências, descritas na Seção 4.5.2.1, concluímos que um bom critério prático é selecionar a componente raiz R tal que $\delta^+(w(R))$ tenha a menor cardinalidade. Essas experiências também confirmaram o bom desempenho desse algoritmo em muitas das instâncias da literatura, como já observado em [16] e [53]. Esses últimos autores também utilizam os cortes gerados pelo *dual ascent* na inicialização do primeiro programa linear em um algoritmo de *branch-and-cut*, o que pode acelerar significativamente a sua execução.

4.3 Novas Heurísticas Duais

A heurística de *dual ascent* de Wong pode obter boas aproximações ao valor da relaxação linear de P_{dc} em muitas instâncias. Nesses casos, às vezes é possível obter a solução exata da instância em tempos muito curtos, ou, pelo menos, reduzí-la significativamente através de testes baseados em custos reduzidos. Entretanto, em muitas outras instâncias, as aproximações obtidas não são boas o suficiente para que isso ocorra. Propomos três novas heurísticas duais com o objetivo de aprimorar o *dual ascent* de Wong, permitindo a obtenção de soluções duais aproximadas melhores, ou ainda, de reduções mais significativas.

4.3.1 Dual Scaling

Dada uma solução dual π obtida por *dual ascent*, podemos gerar uma nova solução dual da seguinte forma. Multiplicamos o valor de cada variável dual π_W em π por um fator α, $0 < \alpha < 1$. Isso torna todos os arcos não-saturados. Aplica-se, então, o procedimento de *dual ascent* novamente. Esse processo pode ser repetido algumas vezes.

Dual Scaling

DUALSCALING (π)
  $\pi^* \longleftarrow \pi$;
  Para $it \longleftarrow 1$ até $itmax$ faça {
    $\pi \longleftarrow$ DUALASCENT($\alpha \cdot \pi$);
    Se $(v(\pi) > v(\pi^*))$, então $\pi^* \longleftarrow \pi$;
    Senão retorne π^*;
  }
  Retorne π^*;

Esse simples procedimento algumas vezes é capaz de aumentar significativamente o valor das soluções duais obtidas. Entretanto, sua maior utilidade ocorre na inicialização do primeiro programa linear em algoritmos de *branch-and-cut*. O *dual scaling* fornece uma solução dual π com um maior número de componentes π_W positivos. Esse maior conjunto de cortes leva a programas lineares menos degenerados.

Em nossos experimentos na Seção 4.5 sempre tomamos $\alpha = 0,875$.

4.3.2 Dual Adjustment

Os procedimentos de *dual adjustment* tentam melhorar uma solução dual aproximada, reduzindo-se o valor de algumas variáveis duais para possibilitar o aumento de outras. Um exemplo de tal procedimento é descrito por Fisher e Kedia [22] para o problema de cobertura de conjuntos. Propomos o seguinte procedimento de *dual adjustment* para o PSG.

Dual Adjustment

DUALADJUSTMENT (π)
  $\pi^* \longleftarrow \pi$;
  (Seja x_{inc} a melhor solução primal conhecida, em sua versão orientada)
  Repita {
    Para todo $\pi_W > 0$ tal que $\delta^+(W)$ cruza x_{inc} mais de uma vez, $\pi_W \longleftarrow 0$;
    $\pi \longleftarrow$ DUALASCENT(π);
    Se $(v(\pi) > v(\pi^*))$, então $\pi^* \longleftarrow \pi$;
    Senão retorne π^*;
  }

A motivação desse procedimento está no fato de que, quando o valor da relaxação linear de P_{dc} é igual ao valor do ótimo inteiro, existe uma solução dual ótima π^* e uma solução primal ótima x^* tal que cada corte $\delta^+(W)$ com $\pi^*_W > 0$ cruza x^* exatamente uma vez.

Os experimentos mostraram que esse procedimento melhora bastante a robustez das heurísticas duais. Quando a execução de um *dual ascent* gera uma solução de baixa qualidade, normalmente o *dual adjustment* funciona. Por outro lado, quando a solução fornecida pelo *dual ascent* é quase ótima, o *dual adjustment* não costuma funcionar. Uma aplicação do *dual adjustment* sobre uma solução gerada pelo *dual scaling* quase sempre leva a uma pequena melhoria. Isso pode ser útil em situações onde a diferença entre os limites primais e duais conhecidos é de exatamente uma unidade.

4.3.3 Fixação Ativa por Custos Reduzidos

Uma técnica freqüentemente adotada em programação inteira são os testes de fixação por custos reduzidos. O mais genérico é: dada uma solução dual viável π para uma formulação do problema com valor $v(\pi)$, uma variável com custo reduzido $\bar{c}_\pi(i)$ e uma solução primal de custo z_{inc}, se $v(\pi) + \bar{c}_\pi(i) \geq z_{inc}$, então a variável i pode ser fixada a zero. Existem testes similares que permitem a fixação de variáveis a outros valores.

No caso do PSG, é possível criar testes mais fortes baseados em custos reduzidos.

TESTE 4.1. ([16],[53]) *Seja π uma solução dual viável para a formulação P_{dc} e $\bar{c}_\pi$ os custos reduzidos correspondentes. Dado um arco $a = (u, v)$ no grafo direcionado D, defina $\bar{c}_\pi(P(r, u))$ como o custo do menor caminho da raiz r até u, com respeito aos custos reduzidos, e $\bar{c}_\pi(P(v, t))$ como o custo do menor caminho de v até o terminal mais próximo $t \neq r$, também com respeito aos custos reduzidos. Se $\bar{c}_\pi(a) + \max\{\bar{c}_\pi(P(r, u)), \bar{c}_\pi(P(v, t))\} + v(\pi) \geq z_{inc}$, então o arco a pode ser eliminado.*

Na verdade, quando os caminhos $P(r, u)$ e $P(v, t)$ são disjuntos, pode-se usar a soma de seus custos no lugar do máximo entre eles. Em muitas instâncias, mesmo com soluções não-ótimas π fornecidas por heurísticas duais, é possível obter significativas reduções. Entretanto, em instâncias maiores e mais difíceis isso não acontece. Propomos o seguinte procedimento para melhorar a eficiência da fixação por custos reduzidos.

Fixação Ativa por Custos Reduzidos

FIXAÇÃO ATIVA (π)
$\pi^* \longleftarrow \pi$;
Enquanto houver reduções {
Escolha um arco a;
$\pi' \longleftarrow \{\pi^* \setminus \pi_W, \text{se } a \in \delta^+W\}$;
Insira a em $D_{\pi'}$;
$\pi \longleftarrow$ DUALASCENT(π');
Aplique o teste 4.1 usando π;
Se ($v(\pi) > v(\pi^*)$), então $\pi^* \longleftarrow \pi$;
}
Retorne π^*;

A idéia desse procedimento é alterar a melhor solução dual conhecida π^*, de forma a obter uma nova solução dual π, em relação à qual um certo arco a tenha custo reduzido máximo ($\bar{c}_\pi(a) = c(a)$). Dessa forma, mesmo que $v(\pi)$ seja um pouco menor que $v(\pi^*)$, $\bar{c}_\pi(a)$ pode ser bem maior do que $\bar{c}_{\pi^*}(a)$, o que aumenta muito a chance do arco a ser eliminado. Normalmente, alguns outros arcos, além de a, também são eliminados usando a solução π. Não é raro que durante a execução da fixação ativa obtenha-se uma solução π melhor do que a solução π^* de entrada. Ou seja, esse procedimento também funciona como *dual adjustment*. Uma segunda possibilidade de melhoria de solução ocorre quando a fixação ativa elimina um arco que estava saturado em π^*. Nesse caso, uma nova chamada a DUALASCENT(π^*) pode melhorar a solução dual.

Esse procedimento funciona de forma consistente em quase todas as instâncias, permitindo que significativas fixações ocorram com soluções duais relativamente ruins.

4.4 Algoritmos Exatos

As heurísticas duais descritas nas seções anteriores são suficientes para resolver de forma ótima boa parte das instâncias que aparecem na literatura. Para as demais instâncias, propomos dois algoritmos exatos baseados nas próprias heurísticas duais.

4.4.1 Branch-and-Ascent

O *branch-and-ascent* é um procedimento de enumeração implícita em árvore que se utiliza das heurísticas duais para o cálculo de limites inferiores e superiores. O nó raiz da árvore de enumeração consiste na aplicação repetida dos procedimentos de *dual ascent*, *dual adjustment* e da fixação ativa por custos reduzidos. Quando esses procedimentos deixam de ser efetivos e ainda há uma diferença entre os limites inferiores e superiores obtidos, partimos para a ramificação. Escolhemos um vértice v incidente ao maior número de arcos saturados na solução dual corrente. Em um dos ramos, o vértice v e seus arcos adjacentes são eliminados. No outro ramo, v passa a ser considerado como um terminal. Esse procedimento, descrito em detalhe no quadro abaixo, é executado novamente em cada um desses nós.

```
ProcessaNó(⟨D, c, T⟩, π* )
  Faça {
        π ⟵ DUALASCENT(0);

        ( Se (v(π) > v(π*)) atualize π*;
          Encontre a solução primal x correspondente a π;
          Se (v(x) < v(x_inc)) atualize x_inc (variável global);
          Se (v(π*) ≥ v(x_inc)) retorne;
          Execute o teste 4.1 sobre π; )

        π ⟵ DUALADJUST(π*);
        ( ... )
        π ⟵ FIXAÇÃO ATIVA(π*);
        ( ... )
  } enquanto (v(π*) melhora ou a instância é reduzida em mais de 1%)

  Seja v o vértice não-terminal incidente ao maior número de arcos saturados em π*;
  ProcessaNó ( ⟨D \ {v}, c, T⟩, π* );
  ProcessaNó ( ⟨D, c, T ∪ {v}⟩, π* );
}
```

A notação $\langle D, c, T\rangle$ indica a instância direcionada que é recebida como parâmetro por cada nó. Na raiz, essa é a instância original, nos demais nós, uma instância já alterada pelas reduções ou ramificações. Cada nó também recebe π^*, a melhor solução dual conhecida do nó pai (a raiz recebe $\pi^* = 0$). Essa solução dual do pai sempre é uma solução dual válida para o filho.

A maior dificuldade encontrada pelo *branch-and-ascent* está na escolha do vértice de ramificação: não há garantia que, ao se fazer uma ramificação sobre um certo vértice v, os limites inferiores obtidos nos nós filhos irão melhorar. Na verdade, esse problema também ocorre em algoritmos de *branch-and-bound* baseados em programação linear ou em algoritmos de *branch-and-cut*. Mas parece que as soluções fracionárias obtidas por PL fornecem melhores informações para essa escolha do que as soluções duais obtidas pelas heurísticas.

Uma maneira de remediar esse problema é através da *ramificação forte*. A idéia é escolher um conjunto N com n vértices candidatos para a ramificação e explorar os $2n$ nós, correspondentes a eliminar ou tornar terminal cada um dos vértices v em N. Assim, sabe-se exatamente qual seriam os dois limites inferiores obtidos em cada uma das n possibilidades. Então, pode se escolher o vértice que obtenha os melhores limites inferiores em seus filhos. Os $2n - 2$ nós restantes são descartados. Uma idéia semelhante é usada, por exemplo, nos algoritmos de *branch-and-cut* em [69].

Fazemos ramificação forte tomando N como o conjunto dos n vértices não-terminais incidentes ao maior número de arcos saturados em π^*. O tamanho de $|N| = n$ varia conforme a profundidade do nó: na raiz, $n = 10$, na profundidade 1, $n = 9$, e assim por diante. A partir da profundidade 9, a ramificação é a normal. Essa estratégia é necessária, pois a ramificação forte com grande valor de n pode ser muito cara, e só vale a pena investir pesadamente na escolha de bons vértices de ramificação nas profundidades mais baixas da árvore. Eventuais decisões ruins em ramos profundos da árvore têm pouco impacto no desempenho global do algoritmo. Ao fazer a ramificação forte, escolhemos o vértice v em N que maximiza a soma do limite inferior do seu melhor filho com duas vezes o limite inferior do seu pior filho.

4.4.2 Branch-and-Cut

Propomos um algoritmo *branch-and-cut* para o PSG cuja principal característica é o uso das heurísticas duais para inicializar o primeiro programa linear (PL). Uma idéia similar foi testada com sucesso por Polzin e Vahdati [53], que usam apenas a heurística de *dual ascent* com essa finalidade. Como as soluções geradas pelas heurísticas duais normalmente já estão próximas do valor da relaxação linear, pode haver uma grande aceleração do algoritmo. Mas há uma possível dificuldade: as soluções geradas por *dual ascent* (e eventualmente melhoradas por *dual adjustment* ou fixação ativa) às vezes levam a um PL altamente degenerado, e muitas rodadas de separação podem ser necessárias até que se consiga melhorar os limites duais. A heurística de *dual scaling* é muito útil nesse contexto. Soluções duais vindas dessa heurística contêm um número maior de variáveis positivas, o que leva a mais cortes e a PLs muito menos degenerados. De fato, usando-se *dual scaling* é bastante provável que o primeiro PL já consiga melhorar os limites inferiores.

A escolha do algoritmo de separação também é importante para o desempenho do nosso algoritmo de *branch-and-cut*. A maneira direta de separar desigualdades violadas em $(P_{cd}.2)$ exige achar $|T| - 1$ cortes orientados mínimos entre pares de vértices. Um dos melhores algoritmos para esse problema, na teoria e na prática, é o *highest-label preflow-push*, que tem complexidade O($|V|^2.|A|^{1/2}$) (ver [1]). Em 1994, Hao e Orlin [31] propuseram uma variante desse algoritmo capaz de encontrar o corte mínimo que separa um dado vértice r de todos os demais vértices no grafo, em um único passo. O mais notável é que a complexidade do novo algoritmo continua sendo de O($|V|^2.|A|^{1/2}$). É possível adaptar esse algoritmo para encontrar o corte orientado mínimo que separa r de algum outro terminal $k \in T$ (ou seja, o corte orientado de Steiner mínimo) em um único passo. Essa adaptação, que já tinha sido usada em Koch e Martin [38], foi usada em nosso procedimento de separação. Dado que o algoritmo de Hao–Orlin original já é bastante complexo, aproveitamos o código em linguagem C escrito por G. Skorobohatyj e disponível na internet [60].

Cada chamada feita à adaptação do algoritmo de Hao–Orlin fornece $|T| - 1$ cortes orientados de Steiner, um deles é o corte mínimo. Mas vale a pena testar se cada um desses cortes é violado, e nesse caso, adicioná-lo ao PL. Muitos dos truques computacionais descritos na Seção 2.4 continuam sendo úteis nesse *branch-and-cut* para o PSG:

- Perturbar os custos das variáveis para reduzir a degenerescência nos PLs;
- Buscar cortes com a menor cardinalidade e forçando mais a função objetivo;
- Em cada rodada de separação, chamar o algoritmo de Hao–Orlin mais de uma vez, para achar conjuntos de cortes violados disjuntos.

O último ponto a ser comentado, é que as heurísticas duais também podem ser úteis depois da inicialização do *branch-and-cut*. Ao se resolver instâncias mais difíceis, os algoritmos de *branch-and-cut* podem passar várias rodadas de separação sem obter melhorias significativas nos limites inferiores, devido a problemas relacionados com degenerescência. Nesses casos, é possível usar a solução dual corrente π do PL como entrada para o *dual scaling* ou o *dual adjustment*. Caso essas heurísticas consigam melhorar essa solução dual, os novos cortes obtidos são acrescentados ao PL. Dessa forma, é possível escapar de pontos de degenerescência. Experimentos preliminares indicam que essa idéia funciona bem, mas ainda não foi possível implementá-la de forma automática no nosso código. Nos experimentos descritos na seção seguinte, as heurísticas duais são usadas apenas na inicialização do primeiro PL.

4.5 Experimentos Computacionais

Os resultados computacionais desta seção foram obtidos em um Pentium II 400 MHz, exceto pelos que aparecem nas Tabelas 4.7 e 4.8, que foram obtidos em uma Sun UltraSparc II 167MHz. Os resultados do algoritmo de *branch-and-cut* que aparecem nessas duas tabelas foram obtidos com o resolvedor de programas lineares CPLEX 5.0 [34].

4.5.1 Instâncias de teste

Há três conjuntos de instâncias de teste para o PSG que são freqüentemente usadas para comparar os diferentes algoritmos que aparecem na literatura. Todas essas instâncias estão disponíveis na SteinLib [39].

4.5.1.1 Instâncias OR-Lib

Essas instâncias foram criadas por Beasley em 1990 [8] e são divididas em três séries com 20 instâncias cada: C, D e E. A série C é formada por instâncias com 500 vértices, o número de arestas varia entre 625 e 12.500 e o número de terminais, entre 5 e 250. Na série D, as instâncias têm 1000 vértices, de 1250 a 25.000 arestas e entre 5 e 500 terminais. Finalmente, a série E, com 2500 vértices, contém instâncias com número de arestas entre 3125 e 62.500 e número de terminais variando de 5 a 1250. Em todos os casos, G é um grafo aleatório e os custos das arestas são inteiros extraídos de uma distribuição uniforme no intervalo $[1, 10]$.

Essas instâncias podem ser bastante reduzidas pelos testes tradicionais de Duin e Volgenant [17]. Os nossos experimentos foram realizados com as instâncias já reduzidas após a aplicação dos testes NTD1, NTD2, SDE, SDE com Expansão, BD3 e TDist. As instâncias C20, D20 e E20 são resolvidas apenas com esse pré-processamento.

As instâncias OR-Lib aparecem na maioria dos artigos sobre o PSG produzidos ao longo dos anos 90. Todas elas já podem ser resolvidas de forma relativamente fácil pelos algoritmos mais modernos.

4.5.1.2 Instâncias de Incidência

Essas instâncias foram criadas por Duin em 1993 [16], como exemplos de instâncias que não podiam ser reduzidas por seus próprios testes. Estão divididas, de acordo com o número de vértices, em quatro séries com 100 instâncias cada (I080, I160, I320 e I640). Em cada série, o número de arestas varia entre $3|V|/2$ e $|V| \cdot (|V| - 1)/2$; já o número de terminais varia entre $\log |V|$ e $|V|/4$. Assim como nas instâncias OR-Lib, G é um grafo aleatório em todos os casos. Os custos das arestas são obtidos pela seguinte regra: se uma aresta é incidente a dois terminais, seu custo é um inteiro extraído de uma distribuição normal com média 300 e variância 10 ($N(300, 10)$); se a aresta é incidente a um terminal, seu custo é extraído de $N(200, 10)$; se não for incidente a nenhum terminal, o custo é retirado de $N(100, 10)$.

As instâncias de incidência não são significativamente reduzidas nem pelos novos testes com expansão, os nossos experimentos sempre foram feitos com as instâncias originais. Em cada uma das séries de incidência existem 20 grupos de cinco instâncias muito similares, com o mesmo número de vértices, arestas e terminais. Para evitar um número excessivo de testes, nas Tabelas 4.3 e 4.4 nos utilizamos apenas da primeira instância em cada grupo, totalizando 80 instâncias. O mesmo foi feito em Koch e Martin [38].

De acordo com a SteinLib, em janeiro de 2001 havia 24 instâncias em aberto na série I320 e 42 na série I640. Nesta tese foi possível resolver todas as instâncias da primeira série e 12 das instâncias em aberto da I640.

4.5.1.3 Instâncias VLSI

Essas instâncias correspondem a problemas reais de roteamento de ligações em projeto de circuitos VLSI, e foram disponibilizadas na internet em 1996 por Koch e Martin. Uma descrição detalhada dessas 116 instâncias é feita em [35, 38]. Elas são divididas em 7 séries: alue, alut, diw, dmxa, gap, msm e taq.

As instâncias VLSI não são significativamente reduzidas pelos testes baseados em alternativas tradicionais, mas podem ser bastante reduzidas pelos novos testes apresentados no capítulo anterior. Realizamos os experimentos com as 77 instâncias que não foram totalmente resolvidas por pré-processamento.

De acordo com a SteinLib, em outubro de 2000 havia quatro instâncias em aberto nessa classe (alue7065, alue7080, alut2610 e alut2625). Nesta tese, foi possível resolver todas essas instâncias.

4.5.2 Heurísticas duais puras

4.5.2.1 Critério de escolha da componente raiz

O algoritmo de *dual ascent* deixa em aberto o critério de escolha de qual componente raiz deve ser selecionada em cada iteração. O primeiro experimento realizado foi uma avaliação do desempenho prático de nove possíveis critérios de escolha:

Aleatório: todas as componentes raízes têm igual probabilidade de ser escolhidas;

Primeiro: escolhe-se a componente raiz que contém o terminal de menor índice;

Circular: escolhe-se a componente que contém o terminal com índice seguinte (em ordem circular) ao terminal da última componente raiz escolhida;

MinArcos / MaxArcos: escolhe-se uma componente que induz um corte com o mínimo/máximo número de arcos;

MinSaturados / MaxSaturados: escolhe-se uma componente que induz um corte que satura o mínimo/máximo número de arcos;

Critério	Tempo (s)	Gap (%) média	Gap (%) σ	Gap (%) max	Redução (%)	Instâncias resolvidas
MinArcos	19.14	1.61	2.08	13.37	58.04	62
Circular	14.74	2.19	2.98	18.18	54.58	57
MinSaturados	20.65	2.25	2.93	15.38	53.47	58
Aleatório	18.63	2.36	3.00	14.30	49.85	51
MaxValor	28.03	2.54	3.13	14.11	49.31	52
MaxSaturados	17.37	3.68	4.60	30.77	44.59	35
Primeiro	15.31	5.29	6.26	31.85	39.28	39
MinValor	15.52	5.63	6.24	28.30	33.54	28
MaxArcos	15.89	6.53	7.17	30.86	33.44	29
MinArcos+MinSaturados	20.32	1.68	2.20	15.38	58.68	66

Tabela 4.1: Critérios de *dual ascent* puro (214 instâncias)

MinValor / MaxValor: escolhe-se uma componente que induz um corte cuja variável dual pode ser aumentada por um valor mínimo/máximo.

As colunas da Tabela 4.1 têm os seguintes significados: **Tempo** é a média do tempo em segundos para a execução do *dual ascent* nas 214 instâncias; as colunas **Gap (%)** são estatísticas sobre a diferença percentual entre os limites superiores e inferiores obtidos, a média, o desvio-padrão (σ) e a máxima diferença; a coluna **Redução** apresenta a redução percentual média no número de arcos obtida utilizando-se o teste 4.1 (em instâncias resolvidas a redução é de 100%); a coluna **Instâncias resolvidas** é o número de instâncias em que os limites obtidos foram idênticos. A última linha mostra os resultados usando-se MinArcos como critério principal e MinSaturados como critério de desempate. Esse foi a combinação que obteve os melhores resultados e, portanto, foi escolhida para todos os nossos demais experimentos.

4.5.2.2 Novas heurísticas duais

Repetimos os mesmos experimentos usando, além do *dual ascent*, as novas heurísticas duais. Em todos os casos foi utilizada a fixação ativa por custo reduzido. Para instâncias OR-Lib foi utilizado também o *Dual Scaling*; para instâncias de incidência, empregou-se o *Dual Adjustment*.

As quatro primeiras colunas da Tabela 4.2 detalham os resultados obtidos apenas com o *dual ascent* com critério MinArcos+MinSaturados para cada classe de instâncias. As quatro colunas seguintes são os resultados obtidos com as novas heurísticas duais.

	apenas *dual ascent*				+ heurísticas duais			
classe	tempo (s)	gap (%)	red. (%)	Instâncias resolvidas	tempo (s)	gap (%)	red. (%)	instâncias resolvidas
OR-Lib	0.62	0.82	83.89	37/57	1.61	0.23	89.51	51/57
VLSI	14.74	1.19	47.93	15/77	24.52	0.58	74.70	49/77
Incidência	38.41	2.77	51.01	14/80	106.35	1.55	65.80	44/80
Todas	19.83	1.68	58.66	66/214	49.01	0.85	75.31	144/214

Tabela 4.2: Efeito da utilização das novas heurísticas duais

A tabela mostra que as novas heurísticas duais levam a uma significativa melhora nos resultados e dois terços das instâncias testadas passam a ser resolvidas de forma ótima apenas com essas heurísticas. Quando isso acontece, os tempos de resolução são muito menores do que os tempos de algoritmos de *branch-and-cut*. Por exemplo, três instâncias anteriormente em aberto (I320-223, I320-224 e I320-225) puderam ser resolvidas em menos de um minuto nesse experimento.

4.5.3 Branch-and-Ascent

As tabelas 4.3 e 4.4 apresentam os resultados obtidos pelo algoritmo de *branch-and-ascent* nas 80 instâncias de incidência selecionadas. Mais da metade dessas instâncias são resolvidas na profundidade 0 da enumeração, ou seja, são resolvidas apenas pelas heurísticas duais. Por outro lado, 6 instâncias da série I640 não puderam ser resolvidas. As três primeiras colunas das tabelas são o tamanho das instâncias e **Ótimo** é o valor da sua solução ótima. A coluna **Nós** é o número de nós explorados na árvore de enumeração, **Prof.** é a profundidade máxima atingida na busca e **Tempo** é o tempo total em segundos gasto pelo algoritmo. Finalmente, a coluna **K&M** apresenta os melhores tempos já reportados na literatura sobre essas instâncias, obtidos pelos algoritmos de *branch-and-cut* de Koch & Martin [38], em uma Sun Enterprise 3000. Essa máquina é cerca de 3 vezes mais lenta do que o PC com um Pentium 400 MHz. O resolvedor de programas lineares usado em [38] foi o CPLEX 5.0.

Os resultados obtidos mostram um clara vantagem do algoritmo de *branch-and-ascent*. Acreditamos que isso se deva ao fato de que as heurísticas duais são capazes de obter limites inferiores quase tão bons quanto os fornecidos por programação linear, mas de forma muito mais rápida. Assim, quando o número de nós explorados na árvore de enumeração pelo *branch-and-ascent* não é muito maior que o do *branch-and-cut*, o tempo total é ainda bastante inferior.

Nas instâncias mais difíceis o número de nós explorados pode ser muito grande. Nesses

Instância	$\|V\|$	$\|E\|$	$\|T\|$	Ótimo	Nós	Prof.	Tempo (s)	K&M [38]
I080-001	80	120	6	1787	1	0	0.02	0.2
I080-011	80	350	6	1479	1	0	0.10	0.8
I080-021	80	3160	6	1175	1	0	0.15	5.5
I080-031	80	160	6	1570	1	0	0.01	0.2
I080-041	80	632	6	1276	1	0	0.10	2.5
I080-101	80	120	8	2608	9	4	0.20	0.1
I080-111	80	350	8	2051	1	0	0.34	4.0
I080-121	80	3160	8	1561	1	0	0.50	7.5
I080-131	80	160	8	2284	1	0	0.04	0.4
I080-141	80	632	8	1788	1	0	0.40	3.5
I080-201	80	120	16	4760	1	0	0.02	0.3
I080-211	80	350	16	3631	1	0	0.12	2.3
I080-221	80	3160	16	3158	1	0	0.58	22.1
I080-231	80	160	16	4354	1	0	0.09	0.4
I080-241	80	632	16	3538	25	9	14.83	1097.1
I080-301	80	120	20	5519	1	0	0.02	0.2
I080-311	80	350	20	4554	17	7	3.64	8.6
I080-321	80	3160	20	3932	1	0	0.36	29.8
I080-331	80	160	20	5226	5	2	0.50	1.7
I080-341	80	632	20	4236	3	1	1.12	3.1
I160-001	160	240	7	2490	1	0	0.14	0.6
I160-011	160	812	7	1677	1	0	0.29	1.6
I160-021	160	12720	7	1352	1	0	0.82	53.4
I160-031	160	320	7	2170	1	0	0.05	0.5
I160-041	160	2544	7	1494	1	0	0.40	29.3
I160-101	160	240	12	3859	1	0	0.04	0.8
I160-111	160	812	12	2869	1	0	0.94	2.5
I160-121	160	12720	12	2363	1	0	4.06	808.0
I160-131	160	320	12	3356	1	0	0.08	1.3
I160-141	160	2544	12	2549	1	0	12.05	42.8
I160-201	160	240	24	6923	27	9	1.05	1.9
I160-211	160	812	24	5583	51	8	37.07	1143.5
I160-221	160	12720	24	4729	1	0	5.08	889.1
I160-231	160	320	24	6662	19	6	2.76	2.3
I160-241	160	2544	24	5086	23	9	83.42	21008.6
I160-301	160	240	40	11816	3	1	0.22	1.1
I160-311	160	812	40	9135	231	20	192.86	687.9
I160-321	160	12720	40	7876	1	0	7.61	2959.4
I160-331	160	320	40	10414	11	5	1.17	1.8
I160-341	160	2544	40	8331	283	24	708.06	54500.7

Tabela 4.3: Resultados do *branch-and-ascent* nas séries I080 e I160 e comparação de tempos.

Instância	$\|V\|$	$\|E\|$	$\|T\|$	Ótimo	Nós	Prof.	Tempo (s)	K&M [38]
I320-001	320	480	8	2672	1	0	0.11	1.3
I320-011	320	1845	8	2053	1	0	1.02	25.5
I320-021	320	51040	8	1553	1	0	4.14	1102.0
I320-031	320	640	8	2673	1	0	0.47	10.5
I320-041	320	10208	8	1707	1	0	13.40	434.6
I320-101	320	480	17	5548	1	0	0.17	1.7
I320-111	320	1845	17	4273	15	5	14.87	206.5
I320-121	320	51040	17	3321	1	0	7.54	2595.3
I320-131	320	640	17	5255	5	2	2.12	7.4
I320-141	320	10208	17	3606	11	5	113.45	—
I320-201	320	480	34	10044	15	7	1.28	5.0
I320-211	320	1845	34	8039	355	21	592.92	8274.6
I320-221	320	51040	34	6679	1	0	43.31	—
I320-231	320	640	34	9862	35	8	17.20	39.5
I320-241	320	10208	34	7027	297	61	2567.66	—
I320-301	320	480	80	23279	29	9	6.33	13.3
I320-311	320	1845	80	17945	58549	34	134052.94	—
I320-321	320	51040	80	15648	9	4	687.94	—
I320-331	320	640	80	21517	1425	35	609.36	266.7
I320-341	320	10208	80	16296	13991	69	154091.24	—
I640-001	640	960	9	4033	1	0	0.39	5.2
I640-011	640	4135	9	2392	1	0	2.31	18.8
I640-021	640	204480	9	1749	1	0	29.26	11004.9
I640-031	640	1280	9	3278	1	0	1.07	6.5
I640-041	640	40896	9	1897	1	0	43.00	2214.9
I640-101	640	960	25	8764	13	5	6.09	26.7
I640-111	640	4135	25	6167	393	15	1038.90	13677.5
I640-121	640	204480	25	4906	1	0	200.99	—
I640-131	640	1280	25	8097	11	4	16.15	28.6
I640-141	640	40896	25	—	—	—	—	—
I640-201	640	960	50	16079	9	4	5.29	31.3
I640-211	640	4135	50	—	—	—	—	—
I640-221	640	204480	50	9821	5	2	798.93	—
I640-231	640	1280	50	15014	77	10	145.61	804.1
I640-241	640	40896	50	—	—	—	—	—
I640-301	640	960	160	45005	367	41	277.11	152.3
I640-311	640	4135	160	—	—	—	—	—
I640-321	640	204480	160	—	—	—	—	—
I640-331	640	1280	160	42796	8377	70	7902.88	3817.9
I640-341	640	40896	160	—	—	—	—	—

Tabela 4.4: Resultados do *branch-and-ascent* nas séries I320 e I640 e comparação de tempos.

Instância	$\|V\|$	$\|E\|$	$\|T\|$	Ótimo	Anterior	Nós	Prof.	Tempo (s)
I320-215	320	1845	34	8015	8041	827	15	1310
I320-223	320	51040	34	6695	6695	1	0	42
I320-224	320	51040	34	6694	6694	1	0	37
I320-225	320	51040	34	6691	6691	1	0	43
I320-241	320	10208	34	7027	7027	15	7	109
I320-242	320	10208	34	7072	7092	59	25	543
I320-243	320	10208	34	7044	7058	31	13	324
I320-244	320	10208	34	7078	7078	61	16	570
I320-245	320	10208	34	7046	7046	21	9	253
I320-311	320	1845	80	17945	17945	33529	29	63279
I320-312	320	1845	80	18122	18122	99311	30	165981
I320-313	320	1845	80	17991	17991	44197	39	79373
I320-314	320	1845	80	18088	18163	70390	44	125664
I320-315	320	1845	80	17987	18056	113494	41	192909
I320-321	320	51040	80	15648	15648	5	2	309
I320-322	320	51040	80	15646	15654	11	4	571
I320-323	320	51040	80	15654	15654	3	1	299
I320-324	320	51040	80	15667	15667	21	10	1210
I320-325	320	51040	80	15649	15653	3	2	430
I320-341	320	10208	80	16296	16330	4007	54	44630
I320-342	320	10208	80	16228	16228	117	15	2629
I320-343	320	10208	80	16281	16338	789	42	10354
I320-344	320	10208	80	16295	16316	1549	34	17944
I320-345	320	10208	80	16289	16314	3025	54	30263

Tabela 4.5: Resultados do *branch-and-ascent* com ramificação forte nas 24 instâncias em aberto da série I320.

casos, conseguimos melhorar o desempenho do algoritmo usando-se a ramificação forte. A Tabela 4.5 mostra resultados obtidos nas 24 instâncias anteriormente em aberto na série I320. A coluna **Anterior** são as melhores soluções anteriormente conhecidas para essas instâncias, de acordo com a SteinLib. Também foram usadas soluções primais muito boas, obtidas pelo algoritmo descrito em Ribeiro et al. [55], como limite superior inicial. Nas instâncias mais difíceis há uma significativa melhora nos tempos em relação ao *branch-and-ascent* puro. Isso pode ser verificado comparando-se com os resultados das instâncias I320-241, I320-311, I320-321 e I320-341 nas Tabelas 4.4 e 4.5.

O algoritmo de *branch-and-ascent* tem bom desempenho nas instâncias de incidência e em outros tipos de instâncias com relativamente poucos vértices (muitas arestas não é problema). Mas, esse desempenho se degrada para instâncias com muitos vértices, pois fica dificil encontrar vértices para ramificação que façam os limites superiores aumentarem nos nós filhos. Mesmo a ramificação forte não é suficiente para resolver essa dificuldade. Na tabela 4.6 mostramos resultados do *branch-and-ascent* com as 6 instâncias OR-Lib que não foram resolvidas apenas com pré-processamento e heurísticas duais. As três primeiras colunas dessa tabela são os tamanhos das instâncias após o pré-processamento, os tamanhos originais das instâncias e os tempos de pré-processamento podem ser encontrados na tabela 4.7.

Instância	$\|V\|$	$\|E\|$	$\|T\|$	Ótimo	Nós	Prof.	Tempo (s)
C18	391	1044	50	113	21	6	6.2
D18	813	2284	97	223	3	1	4.2
D19	700	1932	100	310	3	1	0.48
E13	1418	2962	232	1280	809	38	1563
E18	2219	6185	248	564	—	—	—
E19	1156	2757	174	758	—	—	—

Tabela 4.6: Resultados do *branch-and-ascent* com ramificação forte nas instâncias OR-Lib.

4.5.4 Branch-and-Cut

As tabelas 4.7 e 4.8 apresentam os resultados obtidos pelo algoritmo de *branch-and-cut* nas 6 instâncias OR-Lib e nas 28 instâncias VLSI que não foram resolvidas apenas com pré-processamento e heurísticas duais. As três primeiras colunas são o tamanho das instâncias e **Ótimo** é o valor da sua solução ótima. As colunas **Prep** e **B&C** são os tempos em segundos gastos pelo pré-processamento e pelo algoritmo de *branch-and-cut*. Finalmente a coluna **P&V** apresenta os melhores tempos já reportados na literatura sobre essas instâncias, obtidos pelos algoritmos de Polzin e Vahdati [53] em um PC com processador Pentium II 300 MHz. Essa máquina é cerca de 2 vezes mais rápida do que a Sun ULTRA 167 MHz. O resolvedor de programas lineares usado por esses autores também foi o CPLEX 5.0.

	Tamanho				Tempos		
Instância	$\|V\|$	$\|E\|$	$\|T\|$	Ótimo	Prep	B&C	P&V [53]
C18	500	12500	83	113	8.09	5.0	2.99
D18	1000	25000	167	223	38.2	6.4	1.90
D19	1000	25000	310	310	45.2	1.58	1.55
E13	2500	12500	417	1280	104.1	45.3	4.34
E18	2500	62500	417	564	286.4	1747	74.1
E19	2500	62500	625	758	252.8	16.6	14.0

Tabela 4.7: Resultados do *branch-and-cut* na instâncias OR-Lib e comparação de tempos.

Alguns comentários sobre os resultados nas instâncias OR-Lib:

- Nas instâncias OR-Lib, o conjunto de algoritmos, que inclui o pré-processamento, as heurísticas duais e o *branch-and-cut* apresentado nesta tese, tem tempos piores que a "orquestra" descrita em [53][1]. A questão é que esses autores implementam os testes de redução tradicionais de Duin e Volgenant de forma muito eficiente, em tempo $O(|E| + |V| \log |V|)$. Dessa forma, os seus tempos de pré-processamento são desprezíveis nas instâncias OR-Lib, o que não acontece em nossa implementação. Comparando-se apenas os tempos dos algoritmos de *branch-and-cut*, as duas implementações parecem ter qualidade equivalente. A única instância em que isso não acontece é a E18. A explicação nesse caso é que a heurística primal de [53] consegue achar a solução ótima de 564, como a relaxação linear dessa instância é de 563.44, não há necessidade de se fazer ramificação. Rodando-se o nosso *branch-and-cut* com um limite superior inicial de 564, ele também demora menos de 2 minutos.

Alguns comentários sobre os resultados nas instâncias VLSI:

- Na maioria das instâncias VLSI, o conjunto de algoritmos apresentado nesta tese é superior a "orquestra" descrita em [53]. As 4 instâncias realmente grandes só puderam ser resolvidas pela primeira vez usando os procedimentos descritos aqui. Em algumas instâncias médias com poucos terminais os tempos de [53] são melhores. Talvez, esse melhor desempenho seja devido ao uso de testes de redução por limites similares ao Teste 3.9, que funcionam bem para poucos terminais. De qualquer modo, esses testes só funcionam até um certo tamanho de instância.

[1] Esse artigo usa a imagem de uma orquestra pra ilustrar o efeito conjunto de seus vários algoritmos.

	Tamanho				Tempos		
Instância	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert T\rvert$	Ótimo	Prep	B&C	P&V [53]
alue3146	3626	5869	64	2240	15	0.6	22.5
alue5345	5179	8165	68	3507	109	218	1136
alue5623	4472	6938	68	3413	52	86.8	1298
alue5901	11543	18429	68	3912	81	38.7	653
alue6179	3372	5213	67	2452	49	2.7	4.17
alue6457	3932	6137	68	3057	93	8.3	4.49
alue6735	4119	6696	68	2696	46	1.7	18.8
alue6951	2818	4419	67	2386	109	1.8	22.1
alue7065	34046	54841	544	23881	7581	155542	—
alue7066	6405	10454	16	2256	300	5623	1666
alue7080	34479	55494	2344	62449	11222	7254	—
alut0805	1160	2089	34	958	1	0.1	1.79
alut1181	3041	5693	64	2353	26	1.4	358
alut2010	6104	11011	68	3307	74	4.7	29.1
alut2288	9070	16595	68	3843	269	55.8	1078
alut2566	5021	9055	68	3073	63	40.1	604
alut2610	33901	62816	204	12239	6990	1801578*	—
alut2625	36711	68117	879	35459	8832	664416*	—
diw0779	11821	22516	50	4440	1390	314	1298
diw0820	11749	22384	37	4167	1756	228	159
dmxa0368	2050	3676	18	1017	2	0.17	1.12
dmxa1801	1816	3407	32	1365	40	0.83	0.85
gap3128	10393	18043	104	4292	769	13	23.3
msm2846	3263	5783	89	3135	49	10.7	292
msm4312	5181	8893	10	2016	199	51.5	10.6
taq0014	6466	11046	128	5326	290	103	1556
taq0377	6836	11715	136	6393	319	515	6486
taq0903	6163	10490	130	5099	244	690	16056

* Tempos obtidos com a versão do *branch-and-cut* de outubro de 2000.

Tabela 4.8: Resultados do *branch-and-cut* na instâncias VLSI e comparação de tempos.

- Nas instâncias VLSI, fica difícil comparar os algoritmos de *branch-and-cut* “puros”, descontando-se os diferentes testes de redução, pois cada autor reporta resultados sobre instâncias que sofreram graus muito diferentes de redução antes e durante o *branch-and-cut*. Mesmo com essas limitações, nos parece que o nosso *branch-and-cut* é melhor do que o de [53] para instâncias médias ou grandes, exceto as com poucos terminais. Por exemplo, a instância alue7080 reduzida é 37% maior do que a instância taq0903 original, e no entanto é resolvida em menos de $1/4$ do tempo. Há ainda que se considerar que instâncias não reduzidas têm muita “gordura”, que pode ser removida de forma relativamente fácil pela fixação por custo reduzido ao longo do *branch-and-cut*. Por outro lado, na instância alue7066, o nosso algoritmo é mais lento, mesmo descontando o tempo de pré-processamento.

As Figuras 4.1 e 4.2 mostram soluções ótimas das instâncias taq0903 e alue7080. A Figura 4.3 repete o desenho dessa última solução, mas sem o grafo G, para uma melhor visualização da sua geometria no plano.

Um comentário final sobre algoritmos de *branch-and-cut* para o PSG, é que parece estar bem estabelecido que quase sempre é vantajoso inicializar o primeiro programa linear com uma solução gerada por heurísticas duais, pelo menos um *dual ascent* como faz [53]. O não uso de heurísticas duais faz com que os tempos dos algoritmos de *branch-and-cut* em [45] e [38] fiquem menos competitivos. O mesmo talvez aconteça com algoritmos baseados em relaxação lagrangeana, como o de Bahiense et al [4], também é possível inicializar esse tipo de algoritmo com heurísticas duais.

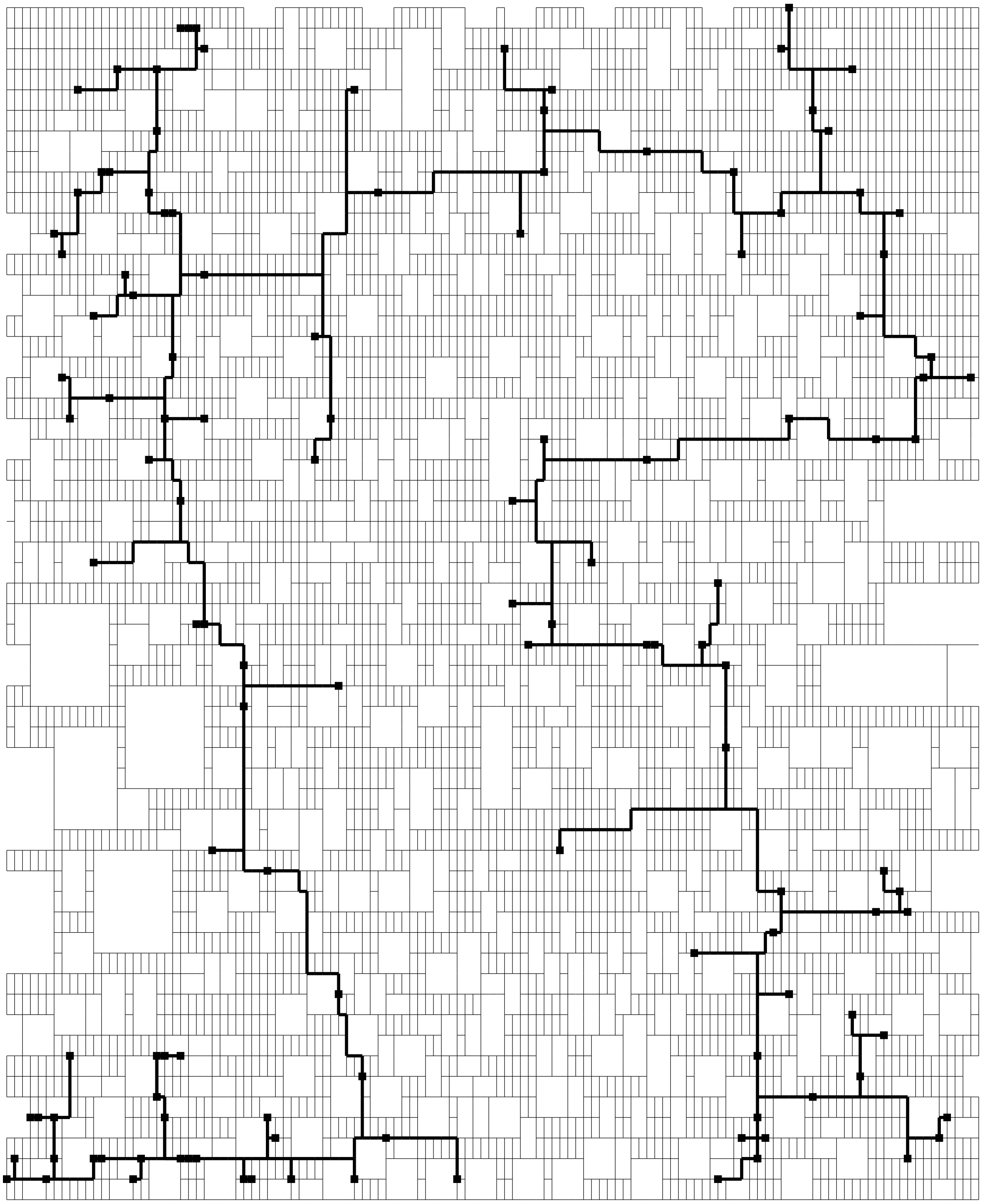

Figura 4.1: Uma solução ótima da taq0903, $|V| = 6163$, $|E| = 10490$, $|T| = 130$.

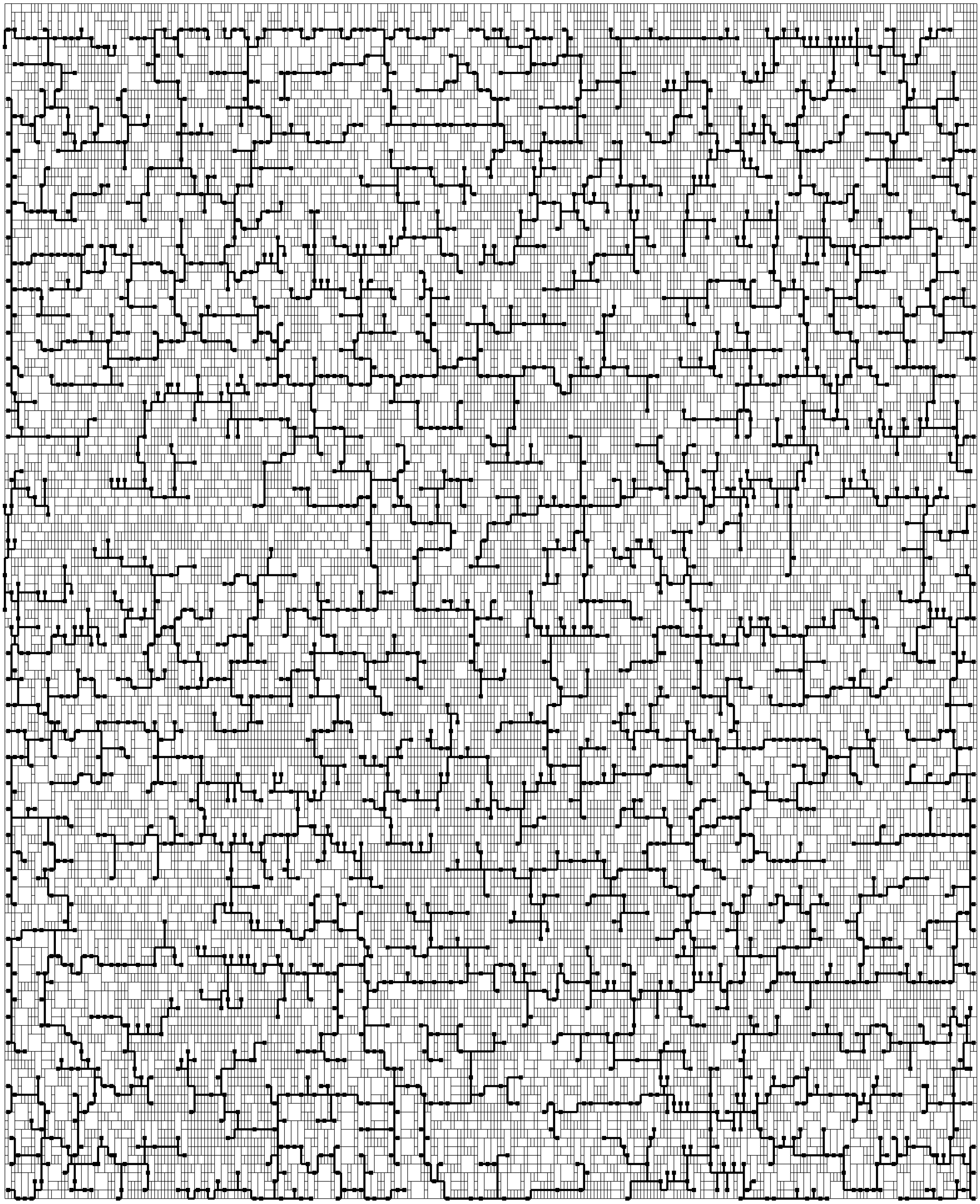

Figura 4.2: Uma solução ótima da alue7080, $|V| = 34.479$, $|E| = 2344$, $|T| = 55494$.

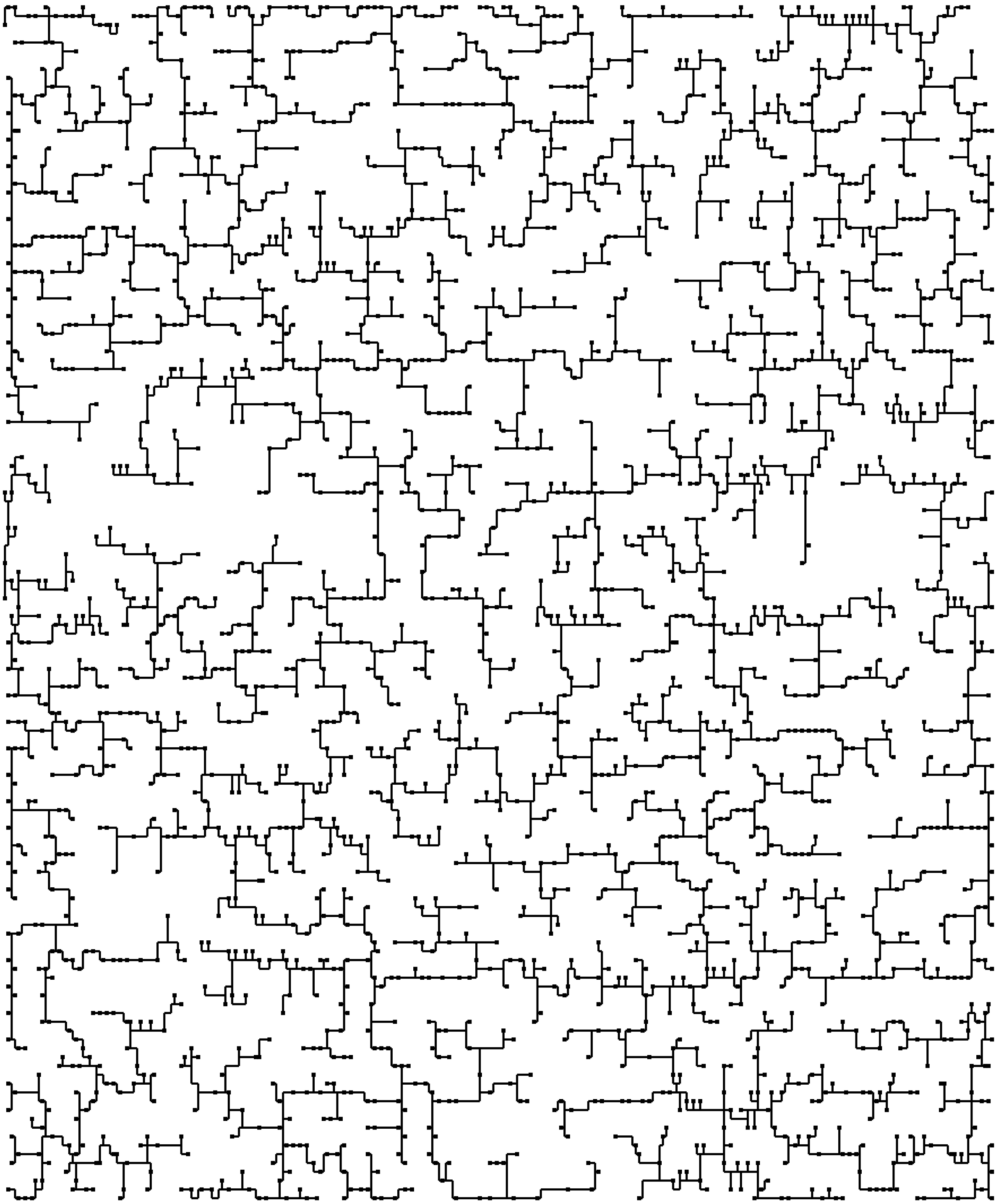

Figura 4.3: Geometria no plano da mesma solução ótima da alue7080.

Capítulo 5

Conclusões e Trabalhos Futuros

Os capítulos anteriores já contém conclusões específicas a cada um dos tópicos tratados nesta tese. A conclusão geral é que atingimos os objetivos propostos. Através da combinação de teoria com prática, do uso de idéias da literatura com novas idéias, foi possível resolver instâncias grandes, tanto no problema do empacotamento de duas árvores de Steiner, como no problema de Steiner em grafos. Vemos um trabalho no futuro imediato sobre esse último problema.

A implementação dos vários algoritmos necessários para a resolução das instâncias mais difíceis do PSG exige um grande esforço de programação, devido a própria sofisticação dos algoritmos em si, ao uso de complexas estruturas de dados e até a necessidade de se fazer interfaces com os pacotes computacionais para programação linear. O mais complexo porém, é a integração de todos esses algoritmos, de modo que os testes de redução, heurísticas primais, heurísticas duais e algoritmos exatos atuem de forma sinérgica. Em colaboração com o aluno de mestrado Renato F. Werneck, todos esses algoritmos foram sendo implementados em um pacote computacional, chamado de Bossa, que já está com 25.000 linhas de código C++.

O principal trabalho futuro que planejamos é a finalização do Bossa *e a sua disponibilização na internet para fins de pesquisa acadêmica.* Dado que o PSG é um dos problemas centrais em otimização combinatória, com muitas aplicações práticas, acreditamos que outros pesquisadores teriam interesse em usar o Bossa em uma das seguintes formas:

- Para resolver instâncias que surjam de aplicações reais, por exemplo, projeto VLSI;
- Como ponto de partida para novos desenvolvimentos para o PSG. Por exemplo, um pesquisador que proponha uma nova heurística primal pode querer usar apenas o nosso pré-processamento. Ou ainda, um pesquisador que crie um novo tipo de corte pode querer testá-lo em nosso algoritmo de *branch-and-cut*.

Entretanto, há algumas tarefas que devem ser feitas antes da disponibilização do Bossa.

- Deveríamos reimplementar os testes de redução tradicionais de Duin e Volgenant, usando as novas técnicas propostas por [53]. As mesmas técnicas também poderiam acelerar bastante os novos testes com expansão. Dessa forma, o Bossa passaria a contar com algoritmos de estado-da-arte também nessa parte, o que o faria obter resultados melhores e mais consistentes em instâncias de todos os tipos e tamanhos.

- A atual implementação do Bossa já se utiliza ao máximo das técnicas de orientação a objeto. Por exemplo, não há nenhuma variável global, todos os dados ficam encapsulados em classes e *templates*. Isso facilitou a depuração dos erros e o reuso de código durante a fase de desenvolvimento. Mas mesmo com esses cuidados, o código atual está um pouco "sujo", parte da sua estrutura ainda reflete uma série de idéias que foram sendo experimentadas ao longo da pesquisa e que foram abandonadas por não funcionarem bem. Esse fenômeno é inevitável quando se faz um programa para pesquisa. Mas agora que os conceitos já estão mais estabilizados, o Bossa deveria sofrer uma "faxina" antes de ser liberado para uso público.

- Deveria haver uma boa documentação do código. Isso seria essencial para que outros pesquisadores possam fazer experiências que exijam sua modificação. Por exemplo, inserir um novo tipo de corte no *branch-and-cut*.

A idéia de divulgar o Bossa veio dos pacotes Concorde [3], desenvolvido para o problema do caixeiro-viajante, e do GeoSteiner [68], para o problema de Steiner euclidiano e retilinear. Ambos os pacotes são hoje referências obrigatórias para todos que estudam esses dois problemas.

Referências Bibliográficas

[1] R. Ahuja, T. Magnanti e J. Orlin. *Network Flows: theory, algorithms, and applications*. Prentice-Hall, New Jersey, 1993.

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

[3] D. Applegate, R. Bixby, V. Chvátal e W. Cook. CONCORDE: A code for solving traveling salesman problems. http://www.keck.caam.rice.edu/concorde.html.

[4] L. Bahiense, F. Barahona e O. Porto. Solving Steiner tree problems in graphs with lagrangean relaxation. Relatório Técnico RC21846, IBM, Yorktown Heights, USA, 2000.

[5] A. Balakrishnan e N. Patel. Problem reduction methods and a tree generation algorithm for the Steiner network problem. *Networks*, 17:65–85, 1987.

[6] J. Beasley. An algorithm for the Steiner problem in graphs. *Networks*, 14:147–159, 1984.

[7] J. Beasley. An SST-based algorithm for the Steiner problem in graphs. *Networks*, 19:1–16, 1989.

[8] J. Beasley. OR-Library: Distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41:1069–1072, 1990. http://mscmga.ms.ic.ac.uk/info.html.

[9] M. Biha, H. Kerevin e A. Mahjoub. Steiner trees and polyhedra. Relatório técnico, Département d'Informatique, Université de Bretagne Occidentale, 1999.

[10] O. Bilde e J. Krarup. Sharp lower bounds and efficient algorithms for the Simple Plant Location Problem. *Annals of Discrete Mathematics*, 1:79–97, 1977.

[11] J. Bondy e U. S. R. Murty. *Graph Theory with Applications*. Macmillan, London, 1976.

[12] S. Chopra e M. Rao. The Steiner problem I: formulations, compositions and extension of facets. *Mathematical Programming*, 64:209–299, 1994.

[13] A. Claus e N. Maculan. Une nouvelle formulation du problème de Steiner sur un graphe. Relatório Técnico 280, Centre de Recherche sur les Transports, Université de Montréal, 1983.

[14] Dash Optimization. *XPRESS-MP Optimiser Subroutine Library Version 12*, 2000.

[15] D. Du, J. Smith e J. Rubinstein, editores. *Advances in Steiner trees*, volume 6 de *Combinatorial Optmization*. Kluwer, 2000.

[16] C. Duin. *Steiner's Problem in Graphs*. Tese de Doutorado, University of Amsterdam, 1994.

[17] C. Duin e A. Volgenant. Reduction tests for the Steiner problem in graphs. *Networks*, 19:549–567, 1989.

[18] J. Edmonds. Optimum branchings. *Journal of Research of the National Bureau of Standards*, B71:233–240, 1967.

[19] D. Erlenkotter. A dual-based procedure for uncapacitaded facility location. *Operations Research*, 26:1590–1602, 1978.

[20] C. Ferreira. *O problema de Steiner em grafos: uma abordagem poliédrica*. Dissertação de Mestrado, IME – Universidade de São Paulo, 1989.

[21] C. Ferreira e Y. Wakabayashi. *Combinatória Poliédrica e Planos-de-corte Faciais*. Texto da 10^a Escola de Computação. IC-Unicamp, Campinas, 1996.

[22] M. Fischer e P. Kedia. Optimal solution of set covering/partitioning problems using dual heuristics. *Management Science*, 36(6):674–688, 1990.

[23] A. Frank. Packing paths, circuits and cuts - a survey. Em *Paths, Flows and VLSI-Layout*, editado por B. Korte, L. Lovász, H. Prömel e A. Schrijver. Springer, Berlin, 1989.

[24] M. Garey e D. Johnson. *Computers and Intractability: A guide to the theory of NP-completeness*. Freeman, San Francisco, 1979.

[25] M. Goemans. The Steiner tree polytope and related polyhedra. *Mathematical Programming*, 63:157–182, 1994.

[26] M. Goemans e D. Williamsom. The primal-dual method for approximation algorithms and its application to network design problem. Em *Approximation algorithms for NP-hard problems*, editado por D. Hochbaum. PWS Publishing Company, 1996.

[27] M. Grötschel, A. Martin e R. Weismantel. Packing Steiner trees: a cutting plane algorithm and computational results. *Mathematical Programming*, 72:125–145, 1996.

[28] M. Grötschel, A. Martin e R. Weismantel. Packing Steiner trees: polyhedral investigations. *Mathematical Programming*, 72:101–123, 1996.

[29] M. Grötschel e C. Monma. Integer polyhedra associated with certain network design problems with connectivity constraints. *SIAM Journal on Discrete Mathematics*, 3:502–523, 1990.

[30] M. Grötschel e M. Padberg. Polyhedral aspects of the travelling salesman problem. Em *The Travelling Salesman Problem*, editado por E. Lawler, J. Lenstra e A. R. Kan. Wiley, New York, 1985.

[31] J. Hao e J. Orlin. A faster algorithm for finding the minimum cut in a directed graph. *Journal of Algorithms*, 17:424–446, 1994.

[32] F. Hwang, D. Richards e P. Winter. *The Steiner tree problem*, volume 53 de *Annals of Discrete Mathematics*. North-Holland, Amsterdam, 1992.

[33] ILOG. *Using the CPLEX Callable Library Version 3*, 1994.

[34] ILOG. *Using the CPLEX Callable Library Version 5*, 1997.

[35] M. Jünger, A. Martin, G. Reinelt e R. Weismantel. Quadratic 0/1 optimization and a decomposition approach for the placement of electronic circuits. *Mathematical Programming*, 63:257–279, 1994.

[36] R. Karp. Reducibility among combinatorial problems. Em *Complexity of Computer Computations*, editado por R. Miller e J. Thatcher, págs. 85–103. Plenum Press, New York, 1972.

[37] M. Kaufmann e K. Mehlhorn. Routing problems in grid graphs. Em *Paths, Flows and VLSI-Layout*, editado por B. Korte, L. Lovász, H. Prömel e A. Schrijver. Springer, Berlin, 1989.

[38] T. Koch e A. Martin. Solving Steiner tree problems in graphs to optimality. *Networks*, 32:207–232, 1996.

[39] T. Koch, A. Martin e S. Voß. SteinLib: An updated library on Steiner tree problems in graphs. Relatório Técnico ZIB-Report 00-37, Konrad-Zuse-Zentrum für Informationstechnik Berlin, Berlin, 2000. http://elib.zib.de/steinlib.

[40] B. Korte, L. Lovász, H. Prömel e A. Schrijver. *Paths, Flows and VLSI-Layout*. Springer, Berlin, 1989.

[41] B. Korte, H. Prömel e A. Steger. Steiner trees in VLSI-layout. Em *Paths, Flows and VLSI-Layout*, editado por B. Korte, L. Lovász, H. Prömel e A. Schrijver. Springer, Berlin, 1989.

[42] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout*. Wiley, Chichester, 1990.

[43] W. Liu. A lower bound for the steiner tree problem in directed graphs. *Networks*, 20:765–778, 1990.

[44] A. Lucena. Tight bounds for the Steiner problem in graphs. Relatório Técnico IRC for Process Systems Engineering, Imperial College, London, 1993.

[45] A. Lucena e J. Beasley. A branch and cut algorithm for the Steiner problem in graphs. *Networks*, 31:39–59, 1998.

[46] N. Maculan. The Steiner problem in graphs. *Annals of Discrete Mathematics*, 31:185–212, 1987.

[47] F. Margot, A. Prodon e T. Liebling. Tree polyhedron on 2-tree. *Mathematical Programming*, 63:183–192, 1994.

[48] G. Nemhauser e L. Wolsey. *Integer and Combinatorial Optimization*. Wiley, New York, 1988.

[49] M. Poggi de Aragão e E. Uchoa. The γ-connected assignment problem. *European Journal of Operational Research*, 118(1):127–138, 1999.

[50] M. Poggi de Aragão, E. Uchoa e R. F. Werneck. Dual heuristics on the exact solution of large Steiner problems. Em *Proceedings of the Brazilian Symposium on Graphs, Algoritms and Combinatorics*, volume 7 de *Electronic Notes in Discrete Mathematics*, Fortaleza, Brasil, 2001.

[51] T. Polzin. *Algorithmen für das Steiner-Problem*. Dissertação de Mestrado, Universität Dortmund, 1997.

[52] T. Polzin e S. Vahdati. A comparison of Steiner tree relaxations. A ser publicado em *Discrete Applied Mathematics*.

[53] T. Polzin e S. Vahdati. Improved algorithms for the Steiner problem in networks. A ser publicado em *Discrete Applied Mathematics*.

[54] J. Reif. Minimum s-t cut of a planar undirected network in $O(n \log^2(n))$ time. *SIAM Journal on Computing*, 12(1):71–81, 1983.

[55] C. C. Ribeiro, E. Uchoa e R. F. Werneck. A hybrid GRASP with perturbations and adaptive path-relinking for the Steiner problem in graphs. Em *Proceedings of the Workshop on Algorithm Engineering as a New Paradigm*, Kyoto University, 2000.

[56] N. Robertson e P. Seymour. An outline of a disjoint paths algoritm. Em *Paths, Flows and VLSI-Layout*, editado por B. Korte, L. Lovász, H. Prömel e A. Schrijver. Springer, Berlin, 1989.

[57] A. Schrijver. Homotopic routing methods. Em *Paths, Flows and VLSI-Layout*, editado por B. Korte, L. Lovász, H. Prömel e A. Schrijver. Springer, Berlin, 1989.

[58] N. Sherwani. *Algorithms for VLSI Physical Design Automation*. Kluwer, Norwell, Massachusetts, terceira edição, 1999.

[59] Y. Shiloach. A polynomial solution to the undirected two paths problem. *Journal of the ACM*, 27(3):445–456, 1980.

[60] G. Skorobohatyj. MATHPROG: A collection of codes for solving various mathematical programming problems. http://elib.zib.de/pub/Packages/mathprog/index.html.

[61] Z. Syed e A. El Gamal. Single layer routing of power and ground networks in integrated circuits. *Journal of Digital Systems*, 6:53–63, 1982.

[62] S. Thienel. *ABACUS - A Branch-And-Cut System*. Tese de Doutorado, Universität zu Köln, 1995.

[63] E. Uchoa e M. Poggi de Aragão. Branch-and-cut for vertex-disjoint packing of two Steiner trees. Em *Anais do IX Congresso Latino-IberoAmericano de Investigação Operativa CLAIO*, Buenos Aires, 1998.

[64] E. Uchoa e M. Poggi de Aragão. Vertex-disjoint packing of two Steiner trees: Polyhedra and branch-and-cut. Em *Proceedings of the 7th Integer Programming and Combinatorial Optmization Conference*, volume 1610 de *Lecture Notes in Computer Science*, págs. 439–452, Graz, 1999.

[65] E. Uchoa e M. Poggi de Aragão. Vertex-disjoint packing of two Steiner trees: Polyhedra and branch-and-cut. *Mathematical Programming*, 90:537–557, 2001.

[66] E. Uchoa, M. Poggi de Aragão e C. C. Ribeiro. Preprocessing Steiner problems from VLSI layout. Relatório Técnico MCC. 32/99, Departamento de Informática, PUC, Rio de Janeiro, 1999.

[67] S. Voss. Steiner's problem in graphs: Heuristic methods. *Discrete Applied Mathematics*, 40:45–72, 1992.

[68] D. Warme, P. Winter e M. Zachariasen. GEOSTEINER: Software for computing Steiner trees. http://www.diku.dk/geosteiner.

[69] D. Warme, P. Winter e M. Zachariasen. Exact algorithms for plane Steiner tree problems: a computational study. Em *Advances in Steiner trees*, editado por D. Du, J. Smith e J. Rubinstein, volume 6 de *Combinatorial Optmization*. Kluwer, 2000.

[70] P. Winter. Steiner problems in networks: A survey. *Networks*, 17:129–167, 1987.

[71] P. Winter. Reductions for the rectlinear Steiner tree problem. *Networks*, 26:187–198, 1995.

[72] R. Wong. A dual ascent approach for Steiner tree problems on a directed graph. *Mathematical Programming*, 28:271–287, 1984.

[73] M. Zachariasen e A. Rohe. Rectilinear group Steiner trees and applications in VLSI design. Relatório Técnico 00906, Institute for Discrete Mathematics, University of Bonn, 2000.