UNIVERSIDADE FEDERAL FLUMINENSE

HUGO DE OLIVEIRA BARBALHO

ALGORITMO NON-DELAYED RELAX-AND-CUT PARA O PROBLEMA DO CICLO DE STEINER

NITERÓI

2014

UNIVERSIDADE FEDERAL FLUMINENSE

HUGO DE OLIVEIRA BARBALHO

ALGORITMO NON-DELAYED RELAX-AND-CUT PARA O PROBLEMA DO CICLO DE STEINER

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: ALGORITMOS E OTIMIZAÇÃO.

Orientador:
LUIZ SATORU OCHI

Co-orientador:
LUIDI GELABERT SIMONETTI

NITERÓI

2014

HUGO DE OLIVEIRA BARBALHO

ALGORITMO NON-DELAYED RELAX-AND-CUT PARA O PROBLEMA DO CICLO DE STEINER

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: ALGORITMOS E OTIMIZAÇÃO.

Aprovada em FEVEREIRO de 2014.

BANCA EXAMINADORA

Prof. LUIZ SATORU OCHI - ORIENTADOR, UFF

Prof. LUIDI GELABERT SIMONETTI - CO-ORIENTADOR, UFF

Prof. ABÍLIO LUCENA PEREIRA FILHO, UFRJ

Prof. YURI ABITBOL, UFF

Niterói

2014

Aos meus pais e minha namorada Cinthya.

Agradecimentos

Inicialmente, agradeço aos meus pais, pela educação e pelo apoio às minhas escolhas.

Aos meus irmãos Artur e Isabelle.

À minha namorada Cinthya, que esteve ao meu lado em todos os momentos, inclusive na escrita desse trabalho.

Aos meus colegas de faculdade.

Aos Professores com os quais tive aula.

Aos meus orientadores, Prof. Satoru e Prof. Luidi, por contribuirem para esse trabalho e por ajudarem na minha formação acadêmica.

Ao Professor Abilio, que apesar de não ser oficialmente meu orientador, também contribuiu na confecção desse trabalho.

À CAPES e à FAPERJ pelo apoio financeiro.

Resumo

O Problema do Ciclo de Steiner (Steiner Cycle Problem-SCP) é ainda pouco estudado na literatura, mas existem grandes semelhanças com um problema já bem difundido que é o Problema da Árvore de Steiner em Grafos. Aplicações do SCP podem ser encontradas, por exemplo, em projetos de redes confiáveis de telecomunicação e de transporte, onde alguns nós (pontos terminais) devem estar obrigatoriamente na solução e outros (pontos de Steiner) não são necessários mas pode existir uma penalidade para cada nó não escolhido.

Na literatura do problema da árvore de Steiner, Lucena [27] desenvolveu um algoritmo chamado Non Delayed Relax-and-Cut (NDRC), onde um número exponencial de desigualdades são dualizadas durante a relaxação lagrangeana. Para não dualizar todas essas restrições de fato, é usado um algoritmo de separação para escolher apenas aquelas que estejam violadas.

Neste trabalho, é proposto um algoritmo NDRC para o problema do Ciclo de Steiner, assim como um algoritmo heurístico para encontrar soluções viáveis para o problema. Outra contribuição desse trabalho foi a criação de uma heurística de separação de desigualdades *2-matching* para relaxações do tipo *1-tree*. Em seguida, foram criadas instâncias para esse problema, uma vez que, de nosso conhecimento, na literatura, não há registro de instâncias disponíveis.

Os resultados computacionais utilizando essas instâncias mostrou que o algoritmo desenvolvido consegue encontrar soluções primais viáveis que distam, em média, 0,61% das soluções encontradas pelo algoritmo de *branch-and-cut*. Além disso, o algoritmo NDRC também conseguiu bons limites duais.

Palavras-chave: Relaxação Lagrangeana, problema do ciclo de Steiner, NDRC, Heurística.

Abstract

The Steiner Cycle problem (SCP) has so far attracted little attention but is closely related to a widely-studied problem called Steiner Tree Problem (STP). SCP arises in the optimal design of reliable telecommunication and transportation networks. Some nodes (terminal nodes) must be in the network, while the others (Steiner nodes) are not required to e in the cycle but may pay a penalty for being out of it

In the Steiner Tree problem bibliograph, Lucena [27] developed an algorithm named Non Delayed Relax-and-Cut (NDRC) to the STP, in which a exponential number of inequalities are dualized during the lagrangean relaxation. To avoid dualizing all constraints, a separation algorithm chooses only the violated ones.

This work presents a NDRC algorithm to the Steiner Cycle Problem and an heuristic algorithm capable of build a feasible solution to it. Another contribution of this work, is a separation heuristic to *2-matching* inequalities when the *1-tree* relaxation is used for SCP. At last but not least, instances are designed to the Steiner Cycle Problem, since, by what we know of, there are no public instances for this problem.

The computacional results using this instances show that the NDRC algorithm developed in this work was able to find feasible primal solutions which were, on average, 0,61% far from the solution found by the *branch-and-cut* algorithm. No only this, but the algorithm NDRC also find good dual bounds.

Keywords: Lagrangian Heuristic, Steiner Cycle Problem, NDRC, Heuristic.

Sumário

Lista de Abreviaturas e Siglas **ix**

Lista de Figuras **x**

Lista de Tabelas **xi**

1 Introdução **1**

1.1 Definição do tema 1

1.2 Motivação 2

1.3 Objetivos 2

1.4 Revisão Bibliográfica 3

1.4.1 Problema do Caixeiro Viajante 3

1.4.2 Problema da Árvore de Steiner 4

1.4.3 Problema do Ciclo de Steiner 4

1.5 Elementos da Dissertação 4

2 Programação Linear e o Modelo do SCP **5**

2.1 Introdução à Programação Linear 5

2.1.1 Modelagem 5

2.1.1.1 Problema da Mistura 5

2.1.2 Dualidade 7

2.1.3 Programação Inteira 7

2.2 Relaxação 8

2.2.1 Relaxação *1-tree* 8

2.3 Formulação para o SCP 9

3 Relaxação Lagrangeana **11**

3.1 Dual Lagrangeano 12

3.2 Limites e Otimalidade 12

3.2.1 Restrições de Desigualdade 13

3.3 Resolvendo o Dual Lagrangeano 14

4 Algoritmo Non-Delayed Relax and Cut para o SCP **16**

4.1 Algoritmo Non-Delayed Relax and Cut 16

4.1.1 Modificações do Método Subgradiente para o NDRC 16

4.2 Reformulação do SCP 18

4.2.1 Substituição das variáveis 18

4.2.2 SCP reformulado como uma 1-forest no grafo expandido 19

4.2.3 SCP reformulado com uma 1-tree 21

4.3 Algoritmo NDRC para o SCP 23

4.3.1 Identificação das GSECs violadas 25

4.4 Desigualdades *2-matching* 26

4.4.1 Heurística de separação 27

4.5 Algoritmo de fixação de variáveis 28

5 Heurísticas para o SCP **30**

5.1 Heurística de Takahashi & Matsuyama para STP 31

5.2 Heurísticas construtivas para o SCP 31

5.2.1 *Tree-heuristic* 31

5.2.2 *Cycle-heuristic* 32

5.3 Busca Local 35

5.3.1 Reconnect 35
5.3.2 Or-opt 35
5.4 Heurística Lagrangeana 36

6 Resultados Computacionais 37
6.1 Resultados das heurísticas 38
6.2 Resultados do NDRC 39
6.2.1 Qualidade dos limites 40
6.2.2 Fixações de variáveis 41
6.2.3 Impacto das 2-matchings 43

7 Conclusão e Trabalhos Futuros 46

Referências 48

Lista de Abreviaturas e Siglas

TSPLIB	:	Traveling Salesman Problem Library
SM	:	Subgradient Method
SCP	:	Steiner Cycle Problem;
STP	:	Steiner Tree Problem;
TSP	:	Traveling Salesman Problem;
NDRC	:	Non Delayed Relax-and-Cut;
GSEC	:	Generalized SubTour Elimination Constraints;
SEC	:	SubTour Elimination Constraints;

Lista de Figuras

4.1 Grafo auxiliar para $(\bar{x}, \bar{y})$. 19

4.2 Grafo auxiliar para $(\bar{x}, \bar{z})$. 19

4.3 Topologias para as arestas de r. 21

4.4 Uma solução viável para 4.27 no grafo auxiliar. 23

4.5 GSECs violadas. 25

4.6 *Comb* com três *teeth*. 26

6.1 2-Matching para kroA100-25-1. 44

6.2 2-Matching para kroA100-10-1. 45

Lista de Tabelas

6.1 Instâncias para o SCP 38

6.2 Qualidade das soluções encontradas pelas heurísticas 39

6.3 NDRC-*Tree-Heuristic* x NDRC-*Cycle-Heuristic* 40

6.4 Qualidade das soluções duais encontradas pelo NDRC 42

6.5 Qualidade das soluções primais encontradas pelo NDRC 42

6.6 Fixações encontradas pelo NDRC 44

6.7 Impacto das 2-Matchings no NDRC-*Tree-Heuristic* 45

Capítulo 1

Introdução

1.1 Definição do tema

Em muitos problemas de roteamento e escalonamento existe a necessidade de saber a ordem em que os itens de um conjunto serão visitados. Um problema clássico desse tipo é o Problema do Caixeiro Viajante (*Traveling Salesman Problem* - TSP), onde deve-se encontrar a melhor ordem de visitar todas as cidades sem repetição (ver [7]). Porém, em algumas aplicações práticas não é necessário que todas as cidades sejam visitadas. Um problema com essa característica é o Problema do Ciclo de Steiner (*Steiner Cycle Problem* - SCP). O SCP foi proposto por Salazar em [19].

O SCP tem como objetivo encontrar um ciclo de menor custo formado por todos os vértices terminais e por alguns, ou até mesmo nenhum, vértices de Steiner. Esse problema pode ser aplicado em projetos de redes confiáveis de telecomunicação e de transporte, onde algumas cidades (vértices terminais) devem ser visitadas enquanto que outras (vértices de Steiner) são opcionais. Vale ressaltar que, se a distância entre as cidades respeitarem a desigualdade triangular, então caso o grafo de definição do problema seja completo, este corresponderá a um TSP considerando apenas os vértices terminais.

Apesar de pouco estudado na literatura, o SCP tem grandes semelhanças com alguns problemas mais difundidos. Pode-se citar, além do TSP, outros dois problemas: o Problema da Árvore de Steiner (*Steiner Tree Problem* - STP), onde o objetivo é encontrar uma árvore de custo mínimo e não um ciclo (ver [16]); e o Problema do Ciclo Simples (*Simple Cycle Problem*), onde o conjunto de cidades que devem ser obrigatoriamente visitadas é vazio.

Assim como os problemas a que ele se assemelha, o SCP também é um problema de

Otimização Combinatória. Para esse tipo de problema, costumam-se usar métodos heurísticos para encontrar soluções viáveis de boa qualidade, dado que resolvê-lo de forma ótima consumiria um tempo muito grande à medida que o tamanho das instâncias aumentasse. Nesse trabalho, são propostas soluções para o Problema do Ciclo de Steiner.

1.2 Motivação

Do ponto de vista prático, a motivação desse trabalho é resolver o problema de redes confiáveis de telecomunicação e de transporte. Como ressaltado por Gröschel e Monma em [11], a capacidade de sobrevivência é um fator importante em um projeto de rede de telecomunicação. Capacidade de sobrevivência significa que um serviço pode ser recuperado ao conseguir rotas alternativas por outros vértices e ligações da rede quando ocorre uma falha em algum dos vértices ou ligações. Essa é a principal característica que justifica o uso da estrutura de um ciclo e não de uma árvore.

Do ponto de vista teórico, a motivação encontra-se na dificuldade computacional de resolver tal problema de otimização. Essa dificuldade computacional se deve ao fato de que o problema SCP é classificado como um problema *NP-Difícil*, uma vez que ele é uma generalização do TSP. Existem muitos avanços na tentativa de resolver de forma exata os problemas STP e TSP, portanto existe um leque de alternativas para tentar resolver o SCP.

1.3 Objetivos

Os objetivos desse trabalho são:

1. **Modelo para o Problema:**
   Apresentar um modelo e uma relaxação para o Problema do Ciclo de Steiner onde será feita a relaxação Lagrangeana.

2. ***Non Delayed Relax-and-Cut*:**
   Desenvolver um algoritmo *Non Delayed Relax-and-Cut* para o Problema do Ciclo de Steiner.

3. **Heurística Primal:**
   Desenvolver uma heurística primal rápida para ser executada ao longo da relaxação

Lagrangeana e tirar proveito das informações duais. Com isso teremos soluções viáveis para o problema.

4. **Instâncias para o SCP:**
   Criar instâncias para o SCP baseadas nas instâncias do TSP para os testes computacionais.

1.4 Revisão Bibliográfica

Nessa seção, será apresentada uma breve revisão bibliográfica para o Problema do Caixeiro Viajante, para o problema da árvore de Steiner e para o problema do Ciclo de Steiner.

1.4.1 Problema do Caixeiro Viajante

O Problema do Caixeiro Viajante foi estudado no século XVIII por um matemático irlandês chamado *William Rowam Hamilton* e pelo matemático britânico *Thomas Penyngton Kirkman*. Desde então, já foram desenvolvidos diferentes algoritmos heurísticos e exatos para resolver o problema [7].

Existem também muitas formulações para o TSP, para mais detalhes veja [32, 30]. A mais citada é a de Dantzig [5], que foi apresentada em 1954. No ano de 1965, Edmonds [8] apresentou a desigualdade *2-matching*, que depois foi generalizada para desigualdade Comb em [12]. Nos anos 80, foram sugeridos algoritmos de separação para as relaxações dos cortes existentes até então. Tal contribuição, que tem promovido o crescimento da teoria poliedral e de *branch-and-cut*, foi conduzido principalmente em [33, 4, 34, 35, 36, 10]. Seguindo essa linha de pesquisa, Applegate [2] desenvolveu o melhor algoritmo conhecido para resolver o problema do Caixeiro Viajante Simétrico de forma exata, chamado de Concorde. Dessa forma, foi possível resolver 19 das 21 instâncias de tamanho n médio (1000<=n<=2392) da TSPLIB, onde o tempo varia entre 5,7 segundos e 3345,3 segundos. A maior instância que o Concorde conseguiu resolver até o momento surgiu de uma aplicação para VLSI e tem 85.900 vértices [2].

Em [15], foi apresentada por *Held & Karp* uma relaxação para o problema do Caixeiro Viajante cujos limitantes são equivalentes aos encontrados na relaxação linear [18]. Tal relaxação calcula uma árvore geradora mínima para o conjunto de vértices $V \setminus \{1\}$ e depois escolhe as duas menores arestas que liguem o vértice 1 à árvore.

1.4.2 Problema da Árvore de Steiner

O problema da árvore de Steiner, apesar de não ser tão explorado quanto o TSP, também é um problema bem difundido na literatura e dentre os trabalhos mais recentes podemos encontrar soluções muito eficientes desenvolvidas por Lucena [27], onde 48 das 50 instâncias usadas para teste tiveram seu ótimo comprovado. Nesse trabalho, Lucena implementou um algoritmo *Non Delayed Relax-and-Cut* (NDRC), que se mostrou competitivo com as heurísticas e metaheurísticas da literatura [38] e, assim, pode-se enfatizar o benefício de se utilizar informações duais para produzir soluções primais.

1.4.3 Problema do Ciclo de Steiner

O Problema do Ciclo de Steiner surgiu da necessidade de se projetar uma rede ótima confiável de telecomunicação e transporte [21]. Nessa rede, alguns vértices (terminais) devem estar obrigatoriamente presentes e outros (vértices de Steiner) são opcionais. Na literatura, é possível encontrar resultados teóricos realizados por Salazar [19]. Nesse trabalho, é feita uma análise da estrutura poliedral associada ao SCP apresentando ainda dois procedimentos de fortalecimaneto destas desigualdades, conhecido como *lifting*.

1.5 Elementos da Dissertação

Esse trabalho está organizado da seguintes maneira. No Capítulo 2, é apresentado um modelo para o SCP e uma introdução sobre Programação Linear. No Capítulo 3, é introduzido o conceito de relaxação Lagrangeana, assim como os algoritmos que são usados para resolvê-la. O algoritmo *Non Delayed Relax-and-Cut* que foi implementado por Lucena [27] e a adaptação do método subgradiente para resolve-lo são discutidos no Capítulo 4. Ainda no Capítulo 4, é apresentado um algoritmo *Non Delayed Relax-and-Cut* para o SCP. O Capítulo 5 exibe um algoritmo heurístico para encontrar soluções viáveis para o problema. O Capítulo 6 apresenta os resultados computacionais. E por último, no Capítulo 7, são feitas as considerações finais.

Capítulo 2

Programação Linear e o Modelo do SCP

Esse capítulo é dedicado a apresentar uma introdução de programação linear e a apresentar um modelo para problema do ciclo de Steiner.

2.1 Introdução à Programação Linear

Um modelo de programação linear tem como objetivo expressar um problema real de forma matemática, utilizando apenas restrições e função objetivo lineares. No seu caso mais básico, é composto por: variáveis de decisão, restrições e uma função objetivo. Muitas situações práticas podem ser representadas por modelos desse tipo.

2.1.1 Modelagem

Para apresentar o conceito de modelagem, foi escolhido fazer um modelo para o clássico problema da mistura.

2.1.1.1 Problema da Mistura

Um problema clássico de programação linear é o problema da mistura, onde o objetivo é produzir uma substância a partir da combinação de outras, mas com o detalhe de que isso deve ser feito com o menor custo possível e as restrições dependem da aplicação do problema que se deseja resolver. Por exemplo, se o produto final fosse uma ração composta de osso, soja e peixe, onde cada um desses itens têm uma quantidade de ferro e cálcio por kilograma, e um preço por kilograma associado, então uma possível restrição considerada seria um valor mínimo de ferro e cálcio por kilograma da ração.

Nesse problema da mistura temos as seguintes constantes, ou seja, aqueles valores que são a entrada do problema: P_o, P_s e P_p como sendo o preço por kilograma do osso, da soja e do peixe, respectivamente; Ca_o, Ca_s e Ca_p como sendo a quantidade de cálcio por kilograma de osso, de soja e de peixe, respectivamente; e F_o, F_s e F_p como sendo a quantidade de ferro por kilograma de osso, de soja e de peixe, respectivamente.

Depois de identificar quais são as constantes do problema, o passo seguinte é definir quais são as variáveis de decisão. Para esse problema, pode-se definir x_o, x_s e x_p como sendo as variáveis que informam quantos kilogramas de osso, soja e peixe estarão na ração, respectivamente. Além disso, é preciso saber qual é o domínio dessas variáveis, ou seja, quais são os possíveis valores a serem atribuídos a elas. Para as nossas variáveis, o domínio é $\mathbb{R}_+$, por se tratar de kilogramas.

No terceiro passo, deve-se escolher as equações que representam de forma matemática o problema real, ou seja, as restrições do modelo. Como dito anteriormente, nesse problema existem as restrições de valor mínimo de ferro e cálcio, que podem ser expressas da seguinte forma:

$$Ca_o x_o + Ca_s x_s + Ca_p x_p \geq Ca_{min} \tag{2.1}$$

$$F_o x_o + F_s x_s + F_p x_p \geq F_{min} \tag{2.2}$$

onde Ca_{min} é a quantidade mínima de cálcio por kilograma e F_{min} é a quantidade mínima de ferro por kilograma. Como não se pode ter essas substâncias com pesos negativos, deve-se adicionar as restrições de não negatividade, para garantir o domínio das variáveis:

$$x_o \geq 0 \tag{2.3}$$

$$x_s \geq 0 \tag{2.4}$$

$$x_p \geq 0 \tag{2.5}$$

No último passo da modelagem, a função objetiva é formulada. Como o objetivo é encontrar a ração mais barata, então o custo de produção deve ser minimizado. Sendo assim, a função objetivo é:

$$minimizar \quad P_o x_o + P_s x_s + P_p x_p \tag{2.6}$$

E assim, tem-se o modelo de programação linear capaz de resolver o problema da mistura.

Apesar do uso de modelos para definir problemas de forma matemática existir há mais tempo, o ano de 1947 foi um marco na área de otimização linear, pois nesse ano

foi introduzido o método simplex [6]. Algum tempo depois, em 1984 [22], ocorreu outro marco importante na área, que foi a criação do método de pontos interiores. Esses dois métodos são até hoje as principais ferramentas para resolução de modelos de programação linear.

2.1.2 Dualidade

Usando a programação linear, outras perguntas em relação ao problema podem ser respondidas, além de apenas encontrar a melhor solução. No problema da mistura, que citamos anteriormente, poderíamos querer saber o quanto a variação do valor nutricional mínimo altera o custo final da produção da ração. Esse tipo de análise corresponde a ver o problema sob outro ponto de vista e que é chamada de problema dual.

2.1.3 Programação Inteira

Para outros problemas, normalmente mais complexos, o modelo deve ser linear inteiro, ou seja, algumas variáveis de decisão são discretas, por exemplo, se a variável representasse o número de pessoas. Para resolver esse tipo de problema, outras técnicas de otimização são necessárias, pois o simplex resolve apenas modelos lineares. Os métodos mais bem-sucedidos para resolver problemas de programação inteira são baseados em enumeração implícita, como o *branch-and-bound*, e de planos de corte. A combinação desses dois métodos deu origem ao *branch-and-cut*, na década de 1980.

Um conceito fundamental aplicado por esses métodos de otimização é a relaxação linear, que substitui no problema de programação inteira as variáveis $x \in \mathbb{Z}_+^n$ por $x \in \mathbb{R}_+^n$ e assim, tem-se um problema de programação linear. Sendo assim, para um problema de programação inteira qualquer:

$$z = \min\{c^T x : Ax \leq b, x \in \mathbb{Z}_+^n\} \tag{P}$$

sua relaxação linear será:

$$\underline{z} = \min\{c^T x : Ax \leq b, x \in \mathbb{R}_+^n\} \tag{PL}$$

E como toda relaxação, o valor encontrado na relaxação linear é um limite inferior para um problema de minimização, isto é, $\underline{z} \leq z$.

2.2 Relaxação

Relaxação Linear é apenas um tipo de relaxação. Suponha o seguinte problema de programação inteira:

$$z_{PI} = \min\{cx : x \in S\} \tag{PI}$$

Definição 2.1. *Uma relaxação de PI é um problema de minimização definido como*

$$z_R = \min\{z_r(x) : x \in S_R\} \tag{R}$$

com as duas propriedades a seguir:

1. $S \subseteq S_R$
2. $z_r(x) \leq cx, \forall x \in S.$

A ideia da relaxação é encontrar um problema mais simples de resolver, para que então seja possível encontrar limites duais[1]. Note que, para encontrar um limite válido, o problema (R) deve ser resolvido de forma ótima. Esse limite pode ser usado em algoritmos de enumeração implícita como o *branch-and-bound*.

2.2.1 Relaxação *1-tree*

Uma conhecida relaxação para o problema do Caixeiro Viajante é a *1-tree*. Criada em 1970, por Held & Karp [15], essa relaxação consegue encontrar limites tão bons quanto os limites encontrados pela relaxação linear (ver [18]).

Seja $G(V, E)$ um grafo não direcionado com um conjunto de vértices V, um conjunto de arestas E e um custo c_e associado à cada elemento $e \in E$. Ainda, o conjunto de arestas com que têm uma extremidade em i é representado por $\delta(i)$. Dessa forma, a região poliedral $\Re_{TSP}$ para o problema do Caixeiro Viajante pode ser definida como:

$$\sum_{e \in \delta(i)} x_e = 2, \quad \forall i \in V \tag{2.7}$$

$$\sum_{e \in E(S)} x_e \leq |S| - 1, \quad \forall S \subset V \tag{2.8}$$

$$0 \leq x_e \leq 1, \quad \forall e \in E \tag{2.9}$$

[1] Limite Inferior para problemas de minimização e Limite Superior para problemas de maximização

Onde as restrições (2.7) garatem que todos os vértices em V terão grau 2. O outro conjunto de restrições (2.8), chamado de SEC (*Subtour Elimination Constraint*), tem como objetivo garantir a conexidade das soluções viáveis.

A formulação para o TSP pode ser definida como:

$$min\{\sum_{e \in E} c_e x_e \ : x \in \Re_{TSP} \cap \mathbb{Z}^{|E|}\} \tag{2.10}$$

A relaxação *1-tree* consiste em escolher um vértice $r \in V$ para ser removido temporariamente da solução. Feito isso, é calculada a árvore geradora[2] de custo mínimo para o grafo $G' = (E \setminus \delta(r), V \setminus \{r\})$. Para os vértices de G', as igualdades (2.7) são relaxadas, mas as desigualdades (2.8) estão sendo respeitadas.

Em seguida, as duas arestas mais baratas de $\delta(r)$ são escolhidas para ligar o vértice r à árvore. Se ao final da relaxação todos os vértices tiverem grau 2, então foi encontrada uma solução ótima para o problema (2.10).

2.3 Formulação para o SCP

Seja $G = (V, E, T)$ um grafo não direcionado finito com um conjunto de vértices V, um conjunto de arestas E e um conjunto de vértices terminais $T \subseteq V$. Existe um custo $\{c_e : e \in E\}$ associado às arestas de G e os vértices não terminais são representados por $N = V \setminus T$. Ainda, o conjunto de arestas que têm uma das extremidades em i será representado por $\delta(i)$. O ciclo de Steiner é um ciclo que visita todos os vértices em T e pode ainda visitar vértices não terminais, os quais são chamados de vértices de Steiner. O objetivo do SCP é encontrar o ciclo de Steiner com menor custo.

Suponha que sejam conhecidos, a priori, quais são os vértices não terminais que irão aparecer no ciclo de Steiner ótimo. Então, o novo problema que teríamos para resolver seria o TSP. Baseado nesse fato, foi feita a formulação para o SCP.

Associado à cada aresta, existe uma variável x_e. Se $x_e = 1$, então a aresta e está presente no ciclo de Steiner. Caso contrário, se $x_e = 0$, a aresta e não está no ciclo. Além disso, também existe uma variável y_i associada à cada vértice $i \in V$. A variável y_i recebe valor 1 se o vértice i estiver presente no ciclo, e valor 0, caso contrário. Seja $\Re_0$ a seguinte

[2] Árvore geradora de um grafo G é um subgrafo $G" = (V, E")$ que contém todos os vértices de G, não contém ciclos e $E" \subseteq E$

região poliedral:

$$\sum_{e \in \delta(i)} x_e = 2y_i, \quad \forall i \in N \tag{2.11}$$

$$\sum_{e \in \delta(i)} x_e = 2, \quad \forall i \in T \tag{2.12}$$

$$\sum_{e \in E(S)} x_e \leq |S \cap T| + \sum_{i \in S \cap N} y_i - 1, \quad \forall S \subset V, S \cap T \neq \emptyset, S \setminus N \neq T \tag{2.13}$$

$$\sum_{e \in E(S)} x_e \leq \sum_{i \in S \setminus \{j\}} y_i, \quad \forall j \in S, S \subset V, S \cap T = \emptyset \tag{2.14}$$

$$y_i = 1, \quad \forall i \in T \tag{2.15}$$

$$0 \leq x_e \leq 1, \quad \forall e \in E \tag{2.16}$$

$$0 \leq y_i \leq 1, \quad \forall i \in V \tag{2.17}$$

Podemos definir a formulação para o SCP como sendo:

$$min\{\sum_{e \in E} c_e x_e \ : (x, y) \in \Re_0 \cap (\mathbb{Z}^{|E|} \text{ x } \mathbb{Z}^{|V|})\} \tag{2.18}$$

Sabendo que os vértices que estarão presentes no ciclo de Steiner devem formar um ciclo simples[3], deve-se adicionar as Equações (2.11) e (2.12). A Equação (2.11) define que os vértices não terminais tenham grau 2, caso eles estejam na solução, ou grau 0, caso contrário. Na Equação (2.12), os vértices sempre devem ter grau 2, uma vez que nesse conjunto de restrições os vértices envolvidos são terminais.

As desigualdades (2.13) e (2.14) são generalizações das restrições de eliminação de subciclo (*Subtour Elimination Constraints* - SECs) sugeridas por Dantzig[5] para o TSP. Elas são chamadas de restrições de eliminação de subciclo generalizada (*Generalized Sub-Tour Elimitation Constraints* - GSECs) e definem que uma solução viável para o poliedro (2.11)-(2.17) não pode formar ciclo, exceto nos casos onde todos os terminais estão presentes em S.

[3]Um ciclo em que cada um de seus vértices é visitado uma única vez

Capítulo 3

Relaxação Lagrangeana

Nesse capítulo serão apresentados alguns conceitos de relaxação lagrangeana. Foi visto no capítulo anterior que existem algoritmos, como o simplex, capazes de resolver um problema quando ele é de programação linear. Vimos também que o problema pode se tornar mais complicado quando as variáveis são inteiras e assim temos que recorrer à outras técnicas de otimização. Nas aplicações práticas mais interessantes, os problemas de otimização são *NP-Difíceis*, isto é, não é conhecido, para eles, um algoritmo em que a complexidade cresça polinomialmente em função dos dados de entrada do problema. Nesses casos, é interessante investigar algoritmos que garatam fatores de aproximação para as soluções encontradas, ou simplesmente, algoritmos que obtêm limitantes para o valor da solução ótima. Considere o seguinte problema de otimização, onde $f : \mathbb{R}^n \to \mathbb{R}$ e $S \subseteq \mathbb{R}$:

$$\begin{array}{ll} \text{Minimizar} & f(x) \\ \text{sujeito a} & x \in S \end{array}$$

Uma relaxação para o problema anterior seria:

$$\begin{array}{ll} \text{Minimizar} & f_R(x) \\ \text{sujeito a} & x \in S_R \end{array}$$

onde $f_R : \mathbb{R}^n \to \mathbb{R}$ e $f_R(x) \leq f(x)$ para qualquer $x \in S$ e $S \subseteq S_R$. Sendo assim, pode-se ver que a solução ótima f_R^* dessa relaxação é um limite inferior para problema inicial. No Capítulo 2, o problema em questão tinha $S = \Re_0 \cap \{0,1\}^n$, onde $\Re_0 \subseteq \mathbb{R}^n$ e a relaxação linear corresponde a considerar $S_R = \Re_0$.

3.1 Dual Lagrangeano

Considere o seguinte problema de programação linear:

$$\begin{aligned} \text{Minimizar} \quad & c^T x \\ \text{sujeito a} \quad & Ax = b \\ & x \in X. \end{aligned} \tag{3.1}$$

O procedimento da relaxação Lagrangeana consiste em relaxar as restrições levando-as à função objetivo com um multiplicador μ associado, chamado de *multiplicador Lagrangeano*. Como resultado desse procedimento, é obtido um problema com a seguinte forma:

$$\begin{aligned} \text{Minimizar} \quad & c^T x + \mu^T (Ax - b) \\ \text{sujeito a} \quad & x \in X. \end{aligned} \tag{3.2}$$

que é chamado de *relaxação Lagrangeana* ou *subproblema Lagrangeano* e a função

$$L(\mu) = \min\{c^T x + \mu^T (Ax - b) | x \in X\} \tag{3.3}$$

é chamada de *função Lagrangeana*.

Lemma 3.1 (Princípio dos limites de Lagrange)**.** *Para qualquer multiplicador Lagrangeano μ, o valor da função Lagrangeana $L(\mu)$ é um limite inferior para o valor ótimo da função objetivo z^* do problema (3.1).*

Prova. Como $Ax = b$ para toda solução viável x de (3.1), então pode-se afirmar que $z^* = \min\{c^T x | Ax = b, x \in X\} = \min\{c^T x + \mu^T (Ax - b) | Ax = b, x \in X\}$, para qualquer *multiplicador Lagrangeano* μ. Já que remover a restrição Ax=b da segunda formulação não pode aumentar o valor da função objetivo, então ela só pode diminuir ou permanecer igual. Por tanto, $z^* \geq \min\{c^T x + \mu^T (Ax - b) | x \in X\} = L(\mu)$. □

3.2 Limites e Otimalidade

Para encontrar o melhor limite da relaxação Lagrangeana, deve-se resolver o seguinte problema de otimização:

$$L^* = \max_{\mu} L(\mu) \tag{3.4}$$

que é chamado de problema *Dual Lagrangeano* do problema original (3.1). Do lemma (3.1) tem-se a seguinte implicação imediata.

Propriedade 3.2 (Dualidade Fraca)**.** *A solução ótima L^* do problema dual Lagrangeano (3.4) é um limite inferior para o valor z^* da solução ótima do problema (3.1).*

Corolário 3.3 (Teste de Otimalidade)**.** *Seja μ um multiplicador Lagrangeano. Se x for um solução viável para (3.1) satisfazendo $L(\mu) = c^T x$, então:*

- *$L(\mu)$ é uma solução ótima do problema Lagrangeano dual, isto é, $L^* = L(\mu)$, e*
- *x é uma solução ótima do problema primal (original) (3.1)*

Como indicado pela propriedade anterior e seu corolário, o Princípio dos limites de Lagrange implica em uma vantagem na relaxação Lagrangeana. Ela fornece um certificado para a otimilidade de uma solução viável para o problema primal (3.1).

Mesmo quando $L(\mu) \leq c^T x$, é possível estimar o quanto uma solução viável dista da solução ótima. Se $\frac{(c^T x - L(\mu))}{L(\mu)} \leq 0,05$, por exemplo, então podemos concluir que a solução viável x não dista mais que 5% do ótimo.

3.2.1 Restrições de Desigualdade

No problema (3.1), as restrições consideradas mais difíceis eram de igualdade. Na prática, problemas são modelados com desigualdades. Portanto, considere o seguinte problema:

$$\begin{aligned} \text{Minimizar} \quad & c^T x \\ \text{sujeito a} \quad & Ax \leq b \\ & x \geq 0. \end{aligned} \tag{3.5}$$

Nesse caso, os multiplicadores Lagrangeanos só admitem valores não-negativos. Sendo assim, o problema Lagrangeano dual fica:

$$L^* = \max_{\mu \geq 0} L(\mu) \tag{3.6}$$

Com esse tipo de restrição, o Princípio dos limites de Lagrange (3.1) e a propriedade da dualidade fraca (3.2) continuam valendo [1]. Porém, um vetor x pode não ser uma solução primal ótima mesmo que x seja primal viável e tenha atingido à solução ótima do problema dual Lagrangeano $L^* = L(\mu)$ para algum $\mu \geq 0$. Com isso, o teste de otimalidade (Corolário) deve ser alterado para:

Propriedade 3.4. *Se $L(\mu)$ é o valor atingido por um vetor x tal que*

- *x é uma solução viável do problema primal (3.5), e ainda*

- *x satisfaz as condições das folgas completares $\mu^T(Ax - b) = 0$*

então, $L(\mu)$ é o valor ótimo do problema dual Lagrangeano (3.6) e x é uma solução ótima para o problema primal (3.5).

3.3 Resolvendo o Dual Lagrangeano

Nessa seção, é descrito um método para aproximar o dual Lagrangeano

$$L^* = \max_{\mu} L(\mu) \tag{3.7}$$

da função Lagrangeana que relaxa a função $Ax = b$, dada por:

$$\begin{aligned} \text{Minimizar} \quad & c^T x \\ \text{sujeito a} \quad & Ax = b \\ & x \geq 0. \end{aligned}$$

Como mencionado anteriormente, as restrições dualizadas são consideradas difíceis, e feita as dualizações dessas restrições, o problema que resta é supostamente mais fácil de otimizar. Por esse motivo, fixando um μ é possível calcular o valor de $L(\mu)$ e uma solução correspondente $x \in X$.

Note que a função Lagrangeana é o envelope inferior para o conjunto dos hiperplanos $\{c^T x + \mu^T(Ax - b) | x \in X\}$, portanto, $L(\mu)$ é uma função côncava. Isto é, seria equivalente a resolver o seguinte problema de programação linear:

$$\begin{aligned} \text{Maximizar} \quad & w \\ \text{sujeito a} \quad & w \leq c^T x + \mu^T(Ax - b) \\ & x \in X, \mu \in \mathbb{R}^n. \end{aligned}$$

Normalmente, o número de restrições desse problema é exponencial, então é usado o método do gradiente descendente para calcular um valor aproximado para o dual Lagrangeano. De forma mais precisa, dada uma função côncava $f : \mathbb{R}^n \to \mathbb{R}$, um vetor x é um subgradiente de f se, para todo $y \in \mathbb{R}^n$, $f(y) \leq f(x) + g^T(y - x)$. A função f é diferenciável em x se e somente se f admitir um único subgradiente g^T em x. Se L fosse diferenciável, seria necessário apenas usar o método do gradiente descendente para convergir em direção ao valor ótimo. Porém, L não é diferenciável por se tratar de uma função linear dividida em pedaços. Portanto, é computada uma seqüência de $(\mu_k)_{k \in \mathbb{N}}$, tal que $L(\mu_k)$ converge para a solução ótima, ao longo do método subgradiente a seguir:

Algoritmo 3.1 Subgradiente

1. Faça $k = 0$ e escolha um $\mu_0 \in \mathbb{R}^n$;
2. Calcule $L(\mu_k)$ e um vetor $x_k \in X$ onde esse valor é obtido;
3. Escolha um subgradiente $g_k = Ax_k - b$ da função L em μ_k;
4. Se $g_k = 0$, então pare, a solução ótima é $L(\mu_k)$;
5. Calcule $\mu^i_{k+1} = \mu^i_k + \theta^T_k g^i_k$;
6. Incremente k e volte ao passo 2.

Na prática, para conseguir calcular o μ^i_{k+1} na linha 5, o tamanho do passo θ_k é dado pela fórmula:

$$\theta_k = \frac{LS - L(\mu_k)}{||Ax_k - b||^2} \tag{3.8}$$

onde LS é um limite superior para a solução ótima que queremos calcular.

No caso onde são dualizadas restrições do tipo $Ax \leq b$, devemos fazer algumas pequenas alterações no algoritmo Subgradiente de modo que $\mu_k \geq 0$, para todo $k \geq 0$. Para isso, no passo 5, a i-ésima coordenada de μ_{k+1} deve receber o maior valor entre 0 e a i-ésima coordenada de $\mu_k + \theta^T_k g_k$.

Capítulo 4

Algoritmo Non-Delayed Relax and Cut para o SCP

Nesse capítulo, será apresentado o algoritmo Non Delayed Relax and Cut e uma proposta para resolver o problema do Ciclo de Steiner usando esse algoritmo. Nessa proposta temos uma reformulação e uma relaxação Lagrangeana para o problema.

4.1 Algoritmo Non-Delayed Relax and Cut

O algoritmo NDRC em [25, 26] é baseado no uso do método subgradiente (SM) descrito no Capítulo 3. Nesse trabalho, nós seguimos as ideias apresentadas em [25, 26, 27] para aplicar o NDRC ao problema do ciclo de Steiner.

4.1.1 Modificações do Método Subgradiente para o NDRC

Em uma iteração k do SM, as desigualdades do tipo

$$\{a_i x \leq b_i : i = 1, 2, .., m\} \tag{4.1}$$

podem ser classificadas em três conjuntos. O primeiro contém as desigualdades que são violadas na solução corrente $\overline{x}^k$. No segundo, estão as desigualdades com multiplicadores diferentes de zero. Note que é possível ter elementos em comum entre esses conjuntos que acabaram de ser definidos. E por último, o conjunto que contém as outras desigualdades. Esses três conjuntos serão denominados, respectivamente, como: conjunto *Currently Violated Active* (CA(k)), conjunto *Previously Violated Active* (PA(k)) e *Currently Inactive* (CI(k)).

Para a relaxação Lagrangeana tradicional, ou seja, quando o número de desigualdades dualizadas m cresce polinomialmente em função do tamanho do problema, Beasley[4] sugere que faça, como boa prática de convergência do SM, $g_k^i = 0$ arbitrariamente sempre que $g_k^i < 0$ e $\mu_k^i = 0$, para $i \in \{1, 2, .., m\}$. Em nosso contexto, todas as desigualdades do conjunto CI(k) estão sujeitas à essa alteração.

Existem duas observações a serem feitas para justificar o uso da prática sugerida por Beasley. A primeira é, independentemente das alterações sugeridas, os multiplicadores das desigualdades em CI(k) não irão mudar, uma vez que no subgradiente descrito no Capítulo 3, o Passo 5 para as desigualdades (4.1) é dado por:

$$\mu_{k+1}^i = \max\{0, \mu_k^i + \theta_k^T g_k^i\} \tag{4.2}$$

Sendo assim, na iteração corrente do SM, as desigualdades em CI(k) não irão contribuir diretamente nos custos Lagrangeanos. Mas por outro lado, eles iriam fazer uma grande diferença no calculo do θ_k e esse fato nos leva à segunda observação. Normalmente, para a aplicação descrita anteriormente, o número de casos com subgradiente estritamente negativo associado ao conjunto CI(k) tende a ser grande. Se todos fossem usados de forma explícita em (3.8), então o tamanho do passo θ_k seria extremamente pequeno, deixando assim os multiplicadores praticamente inalterados. Desse modo, é esperado ocorrer problemas na convergência do algoritmo (3.1).

Seguindo a sugestão de Beasley, conseguimos lidar de forma adequada com o número expressivo de desigualdades do conjunto CI(k). Porém, ainda podemos ter problemas com um possível número grande de desigualdades no conjunto CA(k)\PA(k). Esse é o conjunto de novas desigualdades que serão dualizadas de fato, isto é, seus multiplicadores são diferentes de zero ao final da iteração k.

Suponha que exista um número muito grande de desigualdes em CA(k)\PA(k). Essas são as desigualdades violadas na solução do problema dual Lagrangeano $L(\mu_k)$ e têm o zero como valor para seus multiplicadores. Normalmente, essas desigualdades podem ser particionadas em subconjuntos, por exemplo, um conjunto de vértices associado a um grafo. Em seguida, de acordo com algum critério, uma desigualdade maximal irá existir dentre esses subconjuntos. Com o objetivo de evitar penalizar a mesma variável diferentes vezes, apenas uma desigualdade maximal é dualizada por subconjunto de desigualdades. Fora essas desigualdades, todas as outras em CA(k)\PA(k) terão seu valor de subgradiente alterado para zero arbitrariamente, passando então a pertencer ao conjunto CI(k).

4.2 Reformulação do SCP

Para aplicação da relaxação Lagrangeana, costuma-se escolher algumas restrições para serem descartadas, de forma que o problema resultante seja de fácil resolução. Idealmente, esse novo problema deve ser capaz de encontrar limites de boa qualidade para o problema original (3.1) (no nosso caso, para o SCP). A formulação (2.18), a princípio, não parece ter uma estrutura com essa característica. Porém, veremos a seguir que essa estrutura está escondida na mesma. Primeiramente, as variáveis binárias $\{y_i : i \in V \setminus T\}$ devem ser substituídas pelo seus complementos em 1. Em seguida, o grafo $G = (V, E, T)$ é expandido com a adição de um novo vértice artificial, concomitantemente à adição de algumas arestas incidentes a esse vértice.

4.2.1 Substituição das variáveis

Para implementar o primeiro dos dois passos descritos anteriormente, as variáveis $\{y_i : i \in V \setminus T\}$ são substituídas pelas novas variáveis $\{z_i = 1 - y_i, \forall i \in V \setminus T\}$. Com essa substituição denotamos o poliedro $\Re_0$ por poliedro $\Re_1$

$$\sum_{e \in \delta(i)} x_e + 2z_i = 2, \quad \forall i \in N \tag{4.3}$$

$$\sum_{e \in \delta(i)} x_e = 2, \quad \forall i \in T \tag{4.4}$$

$$\sum_{e \in E(S)} x_e + \sum_{i \in S \cap N} z_i \leq |S| - 1, \quad \forall S \subset V, S \cap T \neq \emptyset, S \setminus N \neq T \tag{4.5}$$

$$\sum_{e \in E(S)} x_e + \sum_{i \in S \setminus \{j\}} z_i \leq |S| - 1, \quad \forall j \in S, S \subset V, S \cap T = \emptyset \tag{4.6}$$

$$z_i = 0, \quad \forall i \in T \tag{4.7}$$

$$0 \leq x_e \leq 1, \quad \forall e \in E \tag{4.8}$$

$$0 \leq z_i \leq 1, \quad \forall i \in V \tag{4.9}$$

e reformulamos o SCP como

$$min\{\sum_{e \in E} c_e x_e \ : (x, z) \in \Re_1 \cap (\mathbb{Z}^{|E|} \text{ x } \mathbb{Z}^{|V|})\} \tag{4.10}$$

Dessa forma, as GSECs (4.5) e (4.6) são expressas como um claro lifting das SECs

comuns. No poliedro $\Re_1$, as desigualdades (4.6) são necessárias apenas no caso onde existe alguma aresta com custo negativo.

4.2.2 SCP reformulado como uma 1-forest no grafo expandido

Agora vamos reinterpretar o significado das variáveis $\{z_i : i \in V \setminus T\}$ assumindo que um vértice artificial $n+1$ foi adicionado ao grafo $G(V, E, T)$ e que cada variável z_i, $i \in V \setminus T$, representa uma aresta que liga o vértice i ao vértice artificial $n+1$. O grafo proveniente dessa expansão é dado por $G' = (V', E', T)$, onde $V' = V \cup \{n+1\}$ e $E' = E \cup \{(i, n+1) : i \in V \setminus T\}$. Dessa forma, G é um subgrafo de G'.

Seja $(\bar{x}, \bar{y})$ um vetor pertencente à um ciclo de Steiner de G e consequentemente uma solução viável inteira de $\Re_0$. Seja também $\bar{G} = (V, \bar{E}, T)$ um grafo auxiliar de $(\bar{x}, \bar{y})$, ou seja, com um conjunto de arestas $\bar{E} \subseteq E$ correspondente a todos os valores não nulos de $\bar{x}$. Os pesos das arestas do grafo $\bar{G}$, como representado pela Figura 4.1, são dados por $\{\bar{x}_e : e \in \bar{E}\}$. Na Figura 4.1, os círculos preenchidos representam os vértices terminais e o círculo com a letra r dentro é o vértice terminal escolhido para ser a raiz. Para toda solução viável de $\Re_0$, deve existir uma solução viável associada em $\Re_1$ e vice versa. Dessa forma, $(\bar{x}, \bar{z})$ é dada como uma solução associada para $(\bar{x}, \bar{y})$. Assim como $\bar{G}' = (V', \bar{E}', T)$ é o grafo auxiliar para $(\bar{x}, \bar{z})$. O grafo auxiliar G' aparece na Figura 4.2, onde o círculo com maior raio representa o vértice artificial $n+1$.

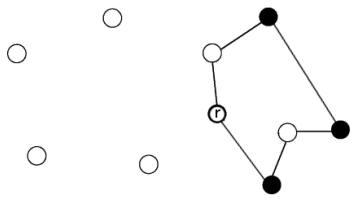


Figura 4.1: Grafo auxiliar para $(\bar{x}, \bar{y})$.

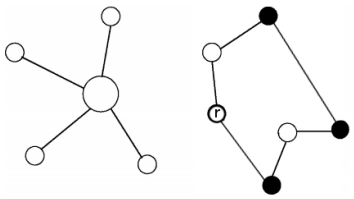


Figura 4.2: Grafo auxiliar para $(\bar{x}, \bar{z})$.

Dado um vetor solução $(\bar{x}, \bar{y})$ para o SCP, não é difícil notar que existem apenas duas topologias possíveis para a solução associada a ela. Se todos os N vértices estiverem no ciclo de Steiner, então não irá existir aresta alguma ligada ao vértice $n+1$ e assim

haverá uma componente com apenas o vértice $n+1$ e uma outra componente com todos os vértices. No outro caso, quando há pelo menos um $\bar{y}_i = 0$ para algum $i \in N$,o grafo $\bar{G}'$ terá duas components conexas disjuntas: o ciclo de Steiner de G e uma *estrela* centralizada em $n+1$ induzida pelas arestas com $\bar{z}_i = 1$. Nos dois casos, o grafo auxiliar $\bar{G}'$ terá uma árvore e um ciclo com exatamente $\bar{E}' = |V'| - 1 = |V|$ arestas. Com isso, temos o tipo de estrutura que estavamos procurando.

Dualizando as restrições de grau (4.3) e (4.4), e usando a relaxação de *Held & Karp* [15], também conhecida como *1-tree*, é possível fazer a relaxação Lagrangeana da mesma maneira que foi feito em [27]. Fazendo isso, tem-se o seguinte poliedro $\Re_{1\_rlx}$, que é uma relaxação do poliedro $\Re_1$:

$$\sum_{e \in E \setminus \delta(r)} x_e + \sum_{i \in N} z_i = |V| - 2, \tag{4.11}$$

$$\sum_{e \in \delta(r)} x_e = 2, \tag{4.12}$$

$$\sum_{e \in E(S)} x_e \leq |S| - 1, \quad \forall S \subset V \setminus \{r\} \tag{4.13}$$

$$\sum_{e \in E(S)} x_e + \sum_{i \in S \cap N} z_i \leq |S| - 1, \quad S \subset V' \setminus \{r\}, n+1 \in S \tag{4.14}$$

$$z_i = 0, \quad \forall i \in T \tag{4.15}$$

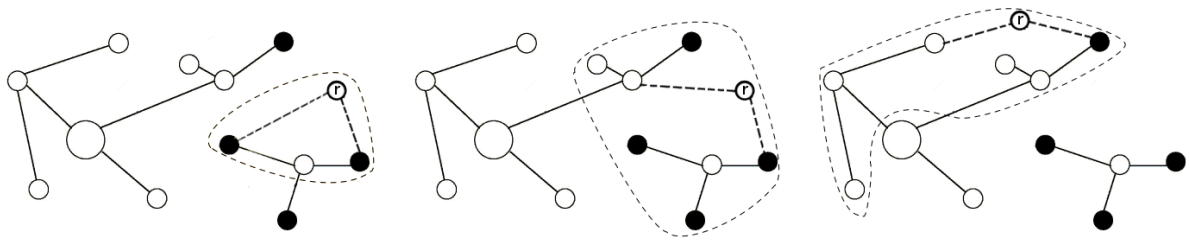
$$0 \leq x_e \leq 1, \quad \forall e \in E \tag{4.16}$$

$$0 \leq z_i \leq 1, \quad \forall i \in V \tag{4.17}$$

Assim o problema Lagrangeano a ser resolvido será:

$$min\{\sum_{e \in E} c'_e x_e + \sum_{i \in N} c'_i z_i \ : (x,z) \in \Re_{1\_rlx} \cap (\mathbb{Z}^{|E|} \text{ x } \mathbb{Z}^{|V|})\} \tag{4.18}$$

Note que as GSECs (4.5) e (4.6) foram substituidas pelas SECs (4.13) e (4.14) e agora temos um problema mais simples de ser resolvido. Isto é, sobre os custos Lagrageanos $\{c'_e : e \in E'\}$ das arestas resultantes, um limite inferior para o SCP é dado por uma floresta induzida pelos vértices em $V' \setminus \{r\}$ de menor custo com exatamente $|V' \setminus \{r\}| - 2$ arestas, além da soma das duas arestas com menores custos de r. Essa floresta pode ser encontrada forçando o algoritmo de Kruskal [23] para árvore geradora mínima a parar quando tiver selecionado $|V' \setminus \{r\}| - 2$ arestas de G'.



(a) Topologia 1 (b) Topologia 2 (c) Topologia 3

Figura 4.3: Topologias para as arestas de r.

Com essa estrutura, também é simples encontrar as GSECs violadas, tudo o que deve ser feito é analisar a topologia da componente conexa que contém o vértice $n+1$. Se ela for uma *estrela* centralizada em $n+1$ ou se o vértice $n+1$ estiver isolado, então não temos GSECs violadas. Caso contrário, se uma aresta $(i, n+1)$, para $i \notin T$, aparecer na solução com uma ou mais arestas em $\delta(i) \subset E$, então existe uma GSEC violada.

Outras GSECs ainda podem ser encontradas ao escolher as arestas do vértice r. Existem três possíveis topologias nessa ocasião. Na primeira, as duas arestas podem ser ligadas à componente que representa a solução e nesse caso a GSEC é composta pelos vértices no ciclo que é formado com a adição dessas arestas, se não estiverem todos os terminais nesse ciclo (ver Figura 4.3a). Na segunda, uma aresta é ligada à cada componente e, assim, a GSEC é composta por todos os vértices da componente da solução e todos os vértices da subárvore da componente $n+1$ a que r está ligada (ver Figura 4.3b). Na última, as duas arestas são ligadas à componente do vértice $n+1$ e, assim, a GSEC é formada por todos os vértices das subárvores de $n+1$ a que são ligadas mais o vértice r, mesmo que estejam indo para a mesma subárvore (ver Figura 4.3c).

4.2.3 SCP reformulado com uma 1-tree

G' sendo uma *1-florest* com $|V'|-1$ arestas é uma estrutura muito atraente para trabalhar. Porém, trabalhar com uma árvore geradora parece ser ainda mais interessante, uma vez isso irá garantir pelo menos um vértice terminal na componente que representa a solução.

Assuma que uma aresta com custo infinitamente pequeno seja criada ligando o vértice $n+1$ a um vértice terminal qualquer de $T \setminus \{r\}$. Esse vértice será chamado de q e a aresta artificial de $(q, n+1)$. O grafo resultante é $G_q = (V', E_q, T)$, onde $E_q = E' \cup \{(q, n+1)\}$.

Se todas as arestas de G tiverem custos não negativos, uma *1-tree* de G_q com a aresta $(q, n+1)$ irá obrigatoriamente existir. Portanto, essa árvore geradora irá fornecer um limite inferior válido para o SCP definido em G. Também é possível conseguir um limite

inferior válido calculando uma *1-tree* para o grafo G_q contendo, obrigatoriamente, a aresta $(q, n+1)$, ainda que algumas arestas de G tenham custos negativos. Sendo assim, de um modo ou de outro pode-se obter um limite Lagrangeano inferior válido para o SCP usando *1-tree* de G_q.

Para apresentar a reformulação do SCP baseada na ideia descrita, considere $N_q = (V \setminus T) \cup \{q\}$. Sejam ainda as variáveis com valores reais $\{x_e : e \in E\}$, $\{z_i : i \in N_q\}$ e o poliedro $\Re_2$ dado por

$$\sum_{e \in E \setminus \delta(r)} x_e + \sum_{i \in N_q} z_i = |V| - 1 \tag{4.19}$$

$$\sum_{e \in \delta(r)} x_e = 2 \tag{4.20}$$

$$\sum_{e \in E(S)} x_e \leq |S| - 1, \quad \forall S \subset V \setminus \{r\} \tag{4.21}$$

$$\sum_{e \in E(S)} x_e + \sum_{i \in S \cap N_q} z_i \leq |S| - 1, \quad \forall S \subset V' \setminus \{r\}, n+1 \in S \tag{4.22}$$

$$z_i = 0, \quad \forall i \in T \setminus \{q\} \tag{4.23}$$

$$0 \leq x_e \leq 1, \quad \forall e \in E \tag{4.24}$$

$$0 \leq z_i \leq 1, \quad \forall i \in N_q \tag{4.25}$$

$$z_q = 1, \tag{4.26}$$

A reformulação do SCP como uma *1-tree* fica sendo

$$min\{\sum_{e \in E} c_e x_e \ : (x, z) \in \Re_2 \cap (\mathbb{Z}^{|E|} \text{ x } \mathbb{Z}^{|N_q|})\} \tag{4.27}$$

Uma solução ótima para (4.27) é uma *1-tree* que necessariamente contém a aresta $(q, n+1)$. Além disso, não devem existir GSECs violadas, ou seja, os dois casos devem acontecer:

1. se a aresta $(i, n+1)$ pertence à solução, então nenhuma outra aresta incidente a i pertence à solução, para todo $i \in V \setminus T$;

2. as duas arestas incidentes à raiz r que pertecem à solução vão estar ligadas a dois vértices da componente de q.

Garantidas essas condições, podemos afirmar que: se a aresta $(q, n+1)$ for removida e

todos os vértices da componente q tiverem grau dois, então essa componente forma uma solução ótima para o SCP (veja Figura 4.4).

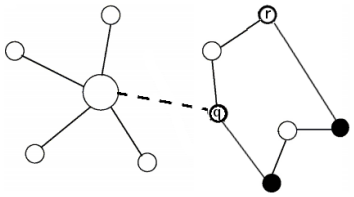


Figura 4.4: Uma solução viável para 4.27 no grafo auxiliar.

4.3 Algoritmo NDRC para o SCP

O algoritmo é baseado na reformulação (4.27) que é dada sobre o grafo expandido $G_q = (V', E_q, T)$. Considere o poliedro $\Re_3$ definido por:

$$\sum_{e \in E \setminus \delta(r)} x_e + \sum_{i \in N_q} z_i = |V| - 1 \tag{4.28}$$

$$\sum_{e \in \delta(i)} x_e + 2z_i = 2, \quad \forall i \in V \setminus T \tag{4.29}$$

$$\sum_{e \in \delta(i)} x_e = 2, \quad \forall i \in T \setminus \{r\} \tag{4.30}$$

$$\sum_{e \in \delta(r)} x_e = 2 \tag{4.31}$$

$$\sum_{e \in E(S)} x_e + \sum_{i \in S \setminus T} z_i \leq |S| - 1, \quad \forall S \subset V, S \cap T \neq \emptyset, S \setminus N \neq T \tag{4.32}$$

$$\sum_{e \in E(S)} x_e + \sum_{i \in S \setminus \{j\}} z_i \leq |S| - 1, \quad \forall j \in S, S \subset V, S \cap T = \emptyset \tag{4.33}$$

$$z_i = 0, \quad \forall i \in T \setminus \{q\} \tag{4.34}$$

$$0 \leq x_e \leq 1, \quad \forall e \in E \tag{4.35}$$

$$0 \leq z_i \leq 1, \quad \forall i \in N_q \tag{4.36}$$

$$z_q = 1, \tag{4.37}$$

A reformulação utilizada é dada por

$$min\{\sum_{e \in E} c_e x_e \ : (x, z) \in \Re_3 \cap (\mathbb{Z}^{|E|} \text{ x } \mathbb{Z}^{|N_q|})\} \tag{4.38}$$

Assuma que as GSECs (4.32) e (4.33) e a restrições de grau (4.29) e (4.30) sejam dualizadas da forma Lagrangeana. Considere também $\{c'_e : e \in E\}$ e $\{d'_i : i \in N_q\}$ os custos Lagrangeanos modificados associados respectivamente a $\{x_e : e \in E\}$ e $\{z_i : i \in N_q\}$. Assim, tem-se o novo poliedro $\Re_{3\_rlx}$ que é uma relaxação do poliedro $\Re_3$ e é definido como:

$$\sum_{e \in E \setminus \delta(r)} x_e + \sum_{i \in N_q} z_i = |V| - 1 \tag{4.39}$$

$$\sum_{e \in \delta(i)} x_e + 2z_i = 2, \quad \forall i \in V \setminus T \tag{4.40}$$

$$\sum_{e \in \delta(i)} x_e = 2, \quad \forall i \in T \setminus \{r\} \tag{4.41}$$

$$\sum_{e \in \delta(r)} x_e = 2 \tag{4.42}$$

$$\sum_{e \in E(S)} x_e \leq |S| - 1, \quad \forall S \subset V \setminus \{r\} \tag{4.43}$$

$$\sum_{e \in E(S)} x_e + \sum_{i \in S \cap N_q} z_i \leq |S| - 1, \quad \forall S \subset V' \setminus \{r\}, n+1 \in S \tag{4.44}$$

$$z_i = 0, \quad \forall i \in T \setminus \{q\} \tag{4.45}$$

$$0 \leq x_e \leq 1, \quad \forall e \in E \tag{4.46}$$

$$0 \leq z_i \leq 1, \quad \forall i \in N_q \tag{4.47}$$

$$z_q = 1, \tag{4.48}$$

Onde o problema resultante é dado por

$$min\{\sum_{e \in E} c'_e x_e + \sum_{i \in N_q} d'_i z_i \ : (x, z) \in \Re_{3\_rlx} \cap (\mathbb{Z}^{|E|} \text{ x } \mathbb{Z}^{|N_q|})\} \tag{4.49}$$

O problema Lagrangeano resultante (4.49) é trivialmente resolvido ao se escolher as $|V|$ arestas mais baratas, se não fossem adicionadas as SECs (4.43) e (4.44). Porém, por análise empírica, podemos notar que o tempo de convergência é menor, para encontrar o limite dual para o problema (4.38), quando o problema Lagrangeano é uma árvore geradora mínima (Minimum Spanning Tree - MST) associada ao grafo G_q. Dessa forma, optamos pela segunda alternativa, isto é, todas as SECs (4.43) e (4.44) são adicionadas ao problema Lagrangeano.

Para a implementação do algoritmo NDRC nesse trabalho, os subgradientes das $|V \setminus$

$\{r\}|$ restrições são calculados por simples inspeções. O procedimento para encontrar as GSECs violadas na solução $(\bar{x}, \bar{z})$ é explicado a seguir, mesmo nos casos onde não são necessárias.

4.3.1 Identificação das GSECs violadas

Essa identificação pode ser dividida em duas etapas. A primeira etapa consiste em encontrar as GSECs violadas sem considerar as arestas que são ligadas ao vértice r. A segunda etapa encontra as GSECs que envolvem o vértice r.

No primeiro caso, o procedimento de separação é o mesmo usado para o STP em [27]. Dada uma árvore geradora do grafo G_q (Figura 4.5), podemos obter as GSECs violadas identificando todos os vértices de algumas subárvores da árvore geradora de G_q. Devemos procurar por GSECs violadas em todas as subárvores provenientes do vértice $n+1$ (veja Figura 4.5), e como única exceção para essa regra temos o vértice q.

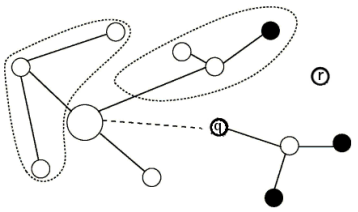


Figura 4.5: GSECs violadas.

Para subárvores que originam GSECs (4.32) (veja a subárvore circulada que tem um vértice terminal na Figura 4.5), dualizamos apenas as desigualdades violadas maximais (em termos de números de vértices). Não é difícil perceber que existem diferentes GSECs (4.32) violadas quando o número de vértices da subárvore é maior ou igual a 3. Foi escolhido dualizar apenas as desigualdades maximais pois isso evita penalizar demasiadamente a mesma variável. Além disso, são elas que dão mais impacto na convergência, uma vez que englobam um maior número de vértices.

Para subárvores que originam GSECs (4.33), a situação é um pouco mais complicada. Assuma um conjunto de vértices S formado pela GSEC (4.33) maximal. Dessa forma, podemos dizer que o número de GSECs violadas induzidas por S é $|S|-1$. Como exemplo, suponha uma GSEC (4.33) violada e seu conjunto $S = \{k, l, m\}$, $k, l, m \in V \setminus T$. Sem perder a generalidade, suponha ainda que as arestas $(k, n+1)$, (k, l) e (k, m) pertençam à árvore geradora G_q. Nesse caso, duas GSECs (4.33) maximais estarão violadas, e são

elas:

$$\sum_{e \in E(S)} x_e + \sum_{i \in S \setminus \{l\}} z_i \leq |S| - 1 \tag{4.50}$$

e

$$\sum_{e \in E(S)} x_e + \sum_{i \in S \setminus \{m\}} z_i \leq |S| - 1 \tag{4.51}$$

Cada uma dessas duas GSECs vão estar violadas por exatamente uma unidade. Em casos mais genéricos, dualizar todas as $|S| - 1$ desigualdades poderia aumentar o tempo de execução, uma vez que seria necessário gerenciar mais desigualdades quando $|S|$ fosse muito grande. Podemos notar também que o mesmo grupo de $|S| - 1$ desigualdades está definido sob o mesmo conjunto de variáveis, e assim, uma variável pode ser penalizada excessivamente. Da mesma forma como é feito em [27], iremos dualizar apenas as desigualdades *surrogates* obtidas somando todas as $|S| - 1$ desigualdades maximais violadas e dividindo o resultado por $|S| - 1$.

4.4 Desigualdades *2-matching*

Para o problema do Caixeiro Viajante (TSP), existem muitas desigualdades válidas estudadas; veja Junger Reinelt & Rinaldi [20] e Naddef [29]. Inicialmente, estamos apenas interessados nas desigualdades *comb* de Grutschel & Padberg [12, 13]. Elas podem ser definidas como: seja $t \geq 3$ um número ímpar, $H \subset V$ e $T_j \subset V$, para $j = 1, .., t$, tal que $T_j \cap H \neq \emptyset$ e $T_j \setminus H \neq \emptyset$, para $j = 1, ..., t$, e ainda T_j conjuntos disjuntos de vértices (Veja Figura 4.6). A desigualdade *comb* é dada por:

$$\sum_{e \in E(H)} x_e + \sum_{j=0}^{t} \sum_{e \in E(T_j)} x_e \leq |H| + \sum_{j=0}^{t} |T_j| - \left\lceil \frac{3t}{2} \right\rceil \tag{4.52}$$

O conjunto H é chamado de *handle* e T_j são chamados de *teeth*.

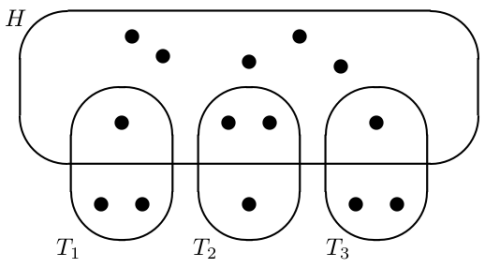


Figura 4.6: *Comb* com três *teeth*.

As desigualdades *combs* induzem facetas para o TSP quando o número de vértice é maior que 5 [12, 13]. Chvátal [3] provou que, para $|T_j \cap H| = 1$ a desigualdade continua

válida, e por esse motivo, às vezes ela é chamada de *Chvátal comb*. Se além disso, tivermos também $|T_j \setminus H| = 1$ para todo $j = 1, .., t$, então teremos a clássica desigualdade *2-matching* de Edmonds [8]. Nesse trabalho, é usada apenas a desigualdade *2-matching*:

$$\sum_{e \in E(H)} x_e + \sum_{e \in E(F)} x_e \leq |H| - \left\lfloor \frac{t}{2} \right\rfloor \tag{4.53}$$

onde $F = T_1 \cup T_2 \cup ... \cup T_t \subset \delta(H)$.

4.4.1 Heurística de separação

É de boa prática que um algoritmo de separação seja executado em tempo polinominal. Em 1982, Padberg & Rao [37] desenvolveram o primeiro algoritmo de separação polinomial para as desigualdades *2-matching*. Esse algoritmo, porém, é complicado de implementar e consome um grande tempo computacional ao tentar encontrar o corte ímpar de menor custo em um grafo $G = (V, E)$ expandido. Recentemente, Letchford, Reinelt & Theis [24] desenvolveram um algoritmo mais simples e mais rápido que usa o algoritmo de fluxo máximo *pre-flow push* de Goldberg & Tarjan [9] como um subprocedimento e tem uma complexidade de $O(|V||E|^2 log(\frac{|V|^2}{|E|}))$.

Além desses, existem outros estudos sobre algoritmos de separação para *comb* e *2-matching*, mas todos eles procuram por desigualdades violadas em soluções onde existem algumas arestas e tal que $0 < x_e < 1$. Como a nossa relaxação para o problema do ciclo de Steiner encontra soluções onde $x_e \in \{0, 1\}$ e diante da conhecida dificuldade computacional de separar de forma exata as *2-matching*, foi desenvolvida uma heurística para encontrar desigualdades violadas em uma relaxação do tipo *1-tree* que vale inclusive para o TSP.

A heurística é dividida basicamente em duas etapas:

1. Encontrar o diâmetro da árvore[1] para formar o conjunto *handle* H.
2. Determinar os vértices que irão formar os conjuntos *teeth* T.

Essa heurística pode ser implementada de forma eficiente se algumas informações forem computadas ao longo do algoritmo da *1-tree*. No algoritmo do Prim para encontrar a árvore geradora mínima, cada nova aresta (i, j) escolhida para entrar na solução irá ligar um novo vértice j à árvore T até então formada. Dessa forma, podemos dizer então que

[1] O mais longo caminho dentre cada par de vértices da árvore

a distância entre esses vértices é $d(i,j) = 1$ e que o conjunto de arestas nesse caminho é $path(i,j) = \{(i,j)\}$. Além disso, devemos também computar a distância e o caminho entre o novo vértice j e todos os outros $k \in T \setminus \{i\}$ da seguinte forma: $d(k,j) = d(k,i)+1$ e $path(k,j) = path(k,i) \cup \{(i,j)\}$. Considerando que o algoritmo Prim encontra a árvore geradora mínima em $O(n^2)$, calcular *path* para todo par de vértice durante a execução dele não irá mudar a sua complexidade assintótica. Por fim, temos que calcular a distância e o caminho para o vértice raiz r. Esse vértice em especial tem dois caminhos e sendo assim iremos guardar o maior deles. Considere as arestas de r na solução como sendo (r,a) e (r,b), com isso temos a distância $d(r,j) = 1+\max\{d(a,j), d(b,j)\}$ para todo vértice $j \in T$ e seu caminho é $path(r,j) = \{r,j\} \cup path(a,j)$ ou $path(r,j) = \{r,j\} \cup path(b,j)$, depende de qual caminho é maior.

A segunda etapa da heurística de separação é a fase de escolha dos t *teeth*. Para nossa heurística, tentamos incluir o maior número de *teeth* possíveis, satisfazendo as seguintes condições:

1. t é impar;
2. $|T_j \cap H| = 1, \forall j = 1, .., t$;
3. $|T_j \setminus H| = 1, \forall j = 1, .., t$;
4. $(T_1 \cap T_2) \cup (T_2 \cap T_3) \cup ... \cup (T_{t-1} \cap T_t) = \emptyset$.

É possível fazer essa escolha em $O(n)$, se guardarmos os vizinhos de cada vértice na *1-tree*. Sendo assim, como já teremos calculado o primeiro passo da heurística de separação *1-tree* sem aumentar a complexidade assintótica do algoritmo da *1-tree*, obter as *2-matching* terá uma complexidade $O(1+n)$=$O(n)$.

Para aplicar essa heurística à relaxação que fizemos para o SCP, devemos lembrar que existe um vértice $n+1$ e uma aresta $(q, n+1)$ artificiais, onde q é um vértice terminal qualquer diferente de r. Sendo assim, nenhum caminho *path* pode passar pela aresta $(q, n+1)$ e para isso $d(q, n+1) = -|V| - 1$. Logo o maior caminho nunca irá passar por essa aresta.

4.5 Algoritmo de fixação de variáveis

O TSP é conhecido por ser um difícil problema de programação inteira 0-1. Dessa forma, é importante eliminar aquelas variáveis que vão ter valor 0 ou 1 em qualquer solução ótima.

Em [40], é descrito um método capaz de encontrar um grande número dessas variáveis. Esse método avalia trocas de arestas aplicadas à uma *1-tree* mínima e as relaciona com um limite superior conhecido. Ele pode ser usado antes de qualquer algoritmo para o TSP ou em qualquer algoritmo baseado na relaxação *1-tree*, que é o nosso caso.

Considere, nessa seção, T como sendo uma *1-tree* geradora mínima do grafo $G = (V, E)$ e T_{ab}^{+} uma *1-tree* mínima proveniente de T ao fixar a aresta (a, b) dentro da solução. Seja ainda T_{ab}^{-} uma *1-tree* mínima proveniente de T ao fixar a aresta (a, b) fora da solução e ub um limite superior conhecido. As arestas são divididas em dois tipos, aquelas que tem alguma extremidade em r e as que não tem.

Para o primeiro caso, onde a aresta tem uma extremidade em r, suponha que as arestas (r, o) e (r, p) estejam na *1-tree* e considere uma função $l(T)$ que calcula o custo da *1-tree* T. Dessa forma temos que:

1. se $l(T_{ro}^{-}) = l(T) - c_{ro} + \min\{c_{rj} : (r, j) \in E, j \neq p, o\} > ub$, então a variável x_{ro} pode ter seu valor fixado em 1;

2. se $l(T_{rp}^{-}) = l(T) - c_{rp} + \min\{c_{rj} : (r, j) \in E, j \neq p, o\} > ub$, então a variável x_{rp} pode ter seu valor fixado em 1;

3. e para cada aresta $(r, j) \notin E(T)$, se $l(T_{rj}^{+}) = l(T) + c_{rj} - \max\{c_{ro}, c_{rp}\} > ub$, então a variável x_{rj} pode ter seu valor fixado em 0.

O segundo caso, onde a aresta não tem extremidades em r, é um pouco mais complicado. Para facilitar a explicação, vamos definir o *fundamental path* da aresta $(p, o) \notin T$ como sendo o conjunto de arestas do único caminho que existe entre p e o na árvore T. Definimos também o *fundamental cutset* da aresta $(a, b) \in T$ como sendo o conjunto de arestas do grafo G que ligam as duas componentes conexas remanescentes ao se remover (a, b) de T. Sendo assim, para uma aresta $(a, b) \in T$, a árvore T_{ab}^{-} será obtida trocando (a, b) pela aresta de menor custo do seu *fundamental cutset*, denotado por (t_{ab}, s_{ab}). Para uma aresta $(p, o) \notin T$, a árvore T_{po}^{+} será obtida trocando (p, o) pela aresta de maior custo do seu *fundamental path*, denotado por (t_{po}, s_{po}). Para esses casos, o teste de fixação fica sendo:

1. Para a aresta $(a, b) \in T$: se $l(T_{ab}^{-}) = l(T) - c_{ab} + c(t_{ab}, s_{ab}) > ub$, então a variável x_{ab} pode ter seu valor fixado em 1;

2. Para a aresta $(p, o) \notin T$: se $l(T_{po}^{+}) = l(T) + c_{po} - c(t_{po}, s_{po}) > ub$, então a variável x_{po} pode ter seu valor fixado em 0;

Capítulo 5

Heurísticas para o SCP

Encontrar um ciclo de Steiner de boa qualidade não é uma tarefa simples. Podemos comparar com o problema da árvore de Steiner, por exemplo, fazendo a seguinte análise: se fosse possível saber quais vértices vão pertencer à solução ótima, então o STP seria reduzido ao problema da árvore geradora mínima, enquanto que para o SCP o novo problema seria o TSP. Com isso desenvolvemos duas heurísticas para construir soluções de boa qualidade para o SCP e também um procedimento de refinamento que é executado após cada uma delas. As duas heurísticas são baseadas na heurística de Takahashi & Matsuyama [39] para o STP. Nas descrições dos algoritmos desse capítulo serão utilizadas as seguintes funções:

1. $\boldsymbol{Prim(V, E)}$: calcula a árvore geradora mínima para o grafo $G(V, E)$;
2. $\boldsymbol{Dijkstra(a, V, E)}$: calcula o caminho mínimo de $a \in V$ para todos os demais vértices de V no grafo $G(V, E)$;
3. $\boldsymbol{Dijkstra(a, b, V, E)}$: calcula o caminho mínimo entre a e b no grafo $G(V, E)$;
4. $\boldsymbol{Tsp\_Heuristic(V, E)}$: executa uma heurística do TSP no grafo $G(V, E)$;
5. $\boldsymbol{Cost(S)}$: calcula o custo da solução S.

$Tsp\_Heuristic(V, E)$ é um algoritmo guloso que ordenar as arestas, para que então as de menores custos sejam inseridas na solução sem que as restrições do TSP sejam violadas. Isso se repete até que $|V|$ arestas sejam adicionadas.

5.1 Heurística de Takahashi & Matsuyama para STP

Em 1980, Takahashi & Matsuyama propuseram um algoritmo aproximativo que encontra uma solução viável para o problema da árvore de Steiner em $O(|T||V|^2)$ ([17]). No pior caso desse algoritmo, o custo da solução encontrada dista $2(1 - \frac{1}{|T|})$ do ótimo ([17]).

O algoritmo, descrito em pseudo-código no Algoritmo 5.1, inicia com uma solução vazia na linha 1. Na linha 2, um vértice terminal qualquer t é escolhido para ser adicionado à árvore na linha 3. Em seguida, na linha 5, procura-se pelo vértice terminal mais próximo da árvore. Na linha 6, a árvore é atualizada com as arestas e vértices do caminho encontrado. Esse processo é repetido até que todos os vértices terminais estejam presentes na árvore.

Algoritmo 5.1 Takahashi_&_Matsuyama(V,E,T)

1. $S \leftarrow \emptyset$;
2. Escolha um vértice $t \in T$;
3. $V(S) \leftarrow \{t\}$;
4. **Enquanto** $(T \setminus V(S)) \neq \emptyset$ **faça**:
5. Escolha o vértice $t \in (T \setminus V(S))$ mais próximo de S;
6. Adicione esse caminho à solução S;
7. **Fim-Enquanto**
8. **Retorne** S.

5.2 Heurísticas construtivas para o SCP

Descrevemos agora nossas heurísticas para o SCP. Como indicado anteriormente, estas se baseiam na heurística de Takahashi & Matsuyama para STP.

5.2.1 *Tree-heuristic*

Essa heurística é dividida em duas etapas. A primeira é selecionar os vértices que vão fazer parte do ciclo de Steiner e a segunda é escolher as arestas do ciclo.

Na primeira etapa, como descrito no Algoritmo 5.2, é usada a heurística do Takahashi & Matsuyama [39] para encontrar uma árvore de Steiner na linha 1. O conjunto H dos vértices que vão pertencer à solução final é inicializado na linha 2. Na linha 3, é encontrada uma árvore geradora mínima no subgrafo induzido pelos vértices pertencentes a essa árvore de Steiner S. Enquanto essa árvore geradora mínima tiver como folha algum vértice não terminal i, então esse vértice é removido de H na linha 5. Na linha 6, uma nova árvore geradora mínima é encontrada para o subgrafo induzido por H. Ao final desse loop, o conjunto H irá conter todos os vértices que vão pertencer ao ciclo de Steiner final.

Na segunda etapa, os vértices em H provenientes da etapa anterior são utilizados para encontrar um ciclo simples aplicando uma heurística do TSP (linha 8). Com isso, teremos uma solução viável para o problema do ciclo de Steiner.

Algoritmo 5.2 Tree-Heuristic(V,E,T)

1. $S \leftarrow$ Takahashi_&_Matsuyama(V,E,T);
2. $H \leftarrow V(S)$;
3. $A_{geradora} \leftarrow Prim(E, H)$;
4. **Enquanto** Houver um $i \notin T$ como folha em $A_{geradora}$ **faça**:
5. $H = H \setminus \{i\}$;
6. $A_{geradora} \leftarrow Prim(E, H)$;
7. **Fim-Enquanto**
8. **Retorne** $Tsp\_Heuristic(E, H)$.

5.2.2 *Cycle-heuristic*

Essa heurística é muito parecida com a do Takahashi & Matsuyama. Como descrito no Algoritmo 5.3, a heurística inicia escolhendo um vértice terminal s que é adicionado ao ciclo. Em seguida, na linha 3, é computado o caminho mínimo dele para todos os outros terminais e escolhe-se o vértice $t \in T \setminus \{s\}$ que esteja mais próximo de s. Esse caminho é adicionado à solução, na linha 4.

Nos passos seguintes, calcula-se: (1) o menor caminho de s para todos os vértices que não estejam em S, na linha 6; (2) na linha 7, o menor caminho de t para todos os vértices

fora da solução S. Na linha 8, é escolhido o vértice terminal w mais próximo de s. Em seguida, na linha 9, o vértice terminal j mais próximo de t. Se (s, w) for o menor caminho, então esse caminho é adicionado à solução S na linha 11 e s passa a ser w na linha 12. Caso contrário, se (t, j) for o menor caminho, então esse caminho é adicionado à solução na linha 14 e t passa a ser j na linha 15.

O loop das linhas 5-17 se repete enquanto houver um vértice terminal fora da solução S. Quando concluído, na linha 18, os vértices s e t são ligados pelo menor caminho que existe entre eles passando apenas por vértices que não estejam na solução S. E assim temos uma solução viável S para o ciclo de Steiner.

Algoritmo 5.3 Cycle-Heuristic(V,E,T)

1. $S \leftarrow \emptyset$;
2. Escolha um vértice $s \in T$;
3. Escolha um vértice $t \in T$ mais próximo de s, onde $t \neq s$;
4. Adicione esse caminho à S;
5. **Enquanto** $T \setminus V(S) \neq \emptyset$ **faça**:
6. $D_s \leftarrow Dijkstra(s, V, E \setminus V(S))$;
7. $D_t \leftarrow Dijkstra(t, V, E \setminus V(S))$;
8. $min_s \leftarrow \min\{D_s(s, w) : w \in V \setminus V(S), i \in T, w \neq t\}$;
9. $min_t \leftarrow \min\{D_s(t, j) : j \in V \setminus V(S), j \in T, j \neq s\}$;
10. **Se** $min_s \leq min_t$ **faça**
11. $S \leftarrow S \cup D_s(s, w)$;
12. $s \leftarrow w$;
13. **Se não**
14. $S \leftarrow S \cup D_t(t, j)$;
15. $t \leftarrow j$;
16. **Fim-Se**
17. **Fim-Enquanto**
18. Calcule o menor caminho de s a t passando apenas por vértices em $V \setminus V(S)$;
19. Adicione as arestas desse caminho à S;
20. **Retorne** S.

5.3 Busca Local

Nessa seção iremos apresentar duas buscas locais. A primeira delas, a *reconnect*, é proposta nesse trabalho para o problema do ciclo de Steiner. A segunda, é uma conhecida busca local baseada em troca de arestas para o TSP.

5.3.1 Reconnect

Essa buscal local recebe um ciclo de Steiner como solução inicial e tem como objetivo encontrar caminhos mais baratos para ligar dois vértices terminais consecutivos no ciclo. Para facilitar a explicação, suponha $cpath(i, j)$ como sendo o conjunto de vértices no caminho entre um par de vértices terminais (i, j) consecutivos na solução.

O procedimento, detalhado no Algoritmo 5.4, faz para cada par de vértices terminais consecutivos (s, t) na solução: o cálculo do menor caminho entre esses dois vértices no subgrafo induzido pelos vértices que ligam o vértice s ao vértice t na solução ($cpath(s,t)$) e os vértices que não pertencem à solução (linha 3); e na linha 4 atualiza o caminho entre esses dois vértices. A quantidade de operações desse procedimento é de $O(|T||N|log|N|)$, lembrando que $N = V \setminus T$.

Algoritmo 5.4 Reconnect(S,V,E,T)

1. **Para-todo** Par de vértices terminais (s, t) consecutivos em S **faça**:
2. $H \leftarrow (V \setminus V(S)) \cup cpath(s, t)$;
3. $D_{(s,t)} \leftarrow Dijkstra(s, t, E(H), H)$;
4. $S \leftarrow (S \setminus cpath(s, t)) \cup D_{(s,t)}$;
5. **Fim-Para-todo**
6. **Retorne** S.

5.3.2 Or-opt

Um dos métodos mais conhecidos de troca de arestas para o TSP, chamado de Or-opt, foi criado por Or [31]. Esse método visa encontrar um ciclo melhor que o original trocando as arestas de três vértices consecutivos de posição até que nenhuma melhora seja encontrada.

Esse processo é então repetido para dois vértices consecutivos e depois com apenas um vértice. Por exemplo, seja a solução original com os seguintes vértices ligados na ordem dada $s = (1, 2, 3, 4, 5, 6, 1)$ e que o conjunto de vértices escolhidos para serem feita as trocas foi $(2, 3, 4)$. Dessa forma, as soluções vizinhas de s serão $\{(1, 5, 2, 3, 4, 6, 1), (1, 5, 6, 2, 3, 4, 1)\}$. Cada iteração do Or-opt requer $O(n^2)$ operações.

5.4 Heurística Lagrangeana

Nossa heurística Lagrangeana utiliza uma das construções apresentadas (*Tree-Heuristic* ou *Cycle-Heuristic*) para produzir uma solução inicial. Em seguida, aplica-se as duas buscas locais de forma consecutiva (Reconnect e Or-opt respectivamente) enquanto houver uma melhora na solução (Ver Algoritmo 5.5).

Essa heurística é executada ao longo do método subgradiente quando ocorre uma melhora no limite dual ou quando ficar 100 iterações sem ser executada. Para construir a solução inicial, as arestas tem seus custos modificados de acordo com a solução dual corrente $(\bar{x}, \bar{y})$, que são dados por: $\{c_e(1 - \bar{x}_e) : e \in E\}$. Já as buscas locais são aplicadas sobre os custos originais do grafo.

Algoritmo 5.5 H_Lagrangeana(V,E,T)

1. $S \leftarrow Tree\text{-}Heuristic(V, E, T)$ ou $Cycle\text{-}Heuristic(V, E, T)$;
2. $custo \leftarrow Cost(S)$;
3. $tmp\_custo \leftarrow custo$;
4. **Faça**;
5. $custo \leftarrow tmp\_custo$;
6. $S \leftarrow Reconnect(S, V, E, T)$;
7. $S \leftarrow Or(S, V, E)$;
8. $tmp\_custo \leftarrow Cost(S)$;
9. **Enquanto** $tmp\_custo < custo$;
10. **Retorne** S.

Capítulo 6

Resultados Computacionais

Esse capítulo apresenta os resultados computacionais para o algoritmo NDRC e para as heurísticas, discutidos, respectivamente, nos Capítulos 4 e 5.

Toda implementação foi realizada em C++ e compilada com g++ 4.4.6. Os testes foram executados em uma maquina com processador intel core i7 quarta geração de 16GB de memória RAM.

Para avaliar a qualidade das heurísticas e ter uma comparação para o NDRC, foi feita uma adaptação do algoritmo *branch-and-cut* para o problema do ciclo Simples [28]. Para isso, foi imposto que cada vértice terminal estivesse obrigatoriamente presente na solução. O *solver* utilizado foi o Xpress Optimizer 64-bit v24.01.04, com prioridade para expandir os *branchs* das variáveis y e o *pool* com as desigualdades foi gerenciado pelo próprio Xpress. Para encontrar as GSECs violadas foi utilizado o algoritmo de Hao e Orlin [14] de corte mínimo.

As instâncias utilizadas foram criadas com base nas já existentes na biblioteca TSPLIB para o problema do Caixeiro Viajante (TSP). As distâncias entre os vértices das instâncias do TSP utilizadas satisfazem a desigualdade triangular. Essa adaptação foi feita ligando, de forma aleatória, alguns vértices dessas instâncias do TSP. O processo de adaptação inicia escolhendo de forma aleatória os vértices terminais. Em seguida, a aresta de cada par de vértice terminal é eliminada do grafo com uma probabilidade de 70%. Na etapa seguinte, calcula-se k árvores geradoras mínimas sem arestas em comum, para que então suas arestas sejam fixadas na instância. Por último, as arestas com pelo menos um vértice não terminal na extremidade, que ainda não esteja na instância, são adicionadas com probabilidade de $\frac{|T|}{|V|}$, onde V é o conjunto de vértices e T o conjunto de vértices terminais. A Tabela 6.1 mostra as instâncias criadas para realizar os testes computacionais.

Instâncias	$\|V\|$	$\|E\|$	$\|T\|$	Densidade
tsp-kroA100-10-1	100	1113	10	22,48 %
tsp-kroA100-10-10	100	1130	10	22,83 %
tsp-kroA100-10-100	100	1144	10	23,11 %
tsp-kroA100-25-1	100	1632	25	32,97 %
tsp-kroA100-25-10	100	1689	25	34,12 %
tsp-kroA100-25-100	100	1720	25	34,75 %
tsp-kroA100-40-1	100	2086	40	42,14 %
tsp-kroA100-40-10	100	2131	40	43,05 %
tsp-kroA100-40-100	100	2123	40	42,89 %
tsp-kroA100-75-1	100	1948	75	39,35 %
tsp-kroA100-75-10	100	1978	75	39,96 %
tsp-kroA100-75-100	100	1988	75	40,16 %
tsp-pcb442-100-1	442	23759	100	24,38 %
tsp-pcb442-100-10	442	23795	100	24,41 %
tsp-pcb442-100-100	442	23862	100	24,48 %
tsp-pcb442-190-1	442	35977	190	36,91 %
tsp-pcb442-190-10	442	35940	190	36,87 %
tsp-pcb442-190-100	442	36158	190	37,09 %
tsp-pcb442-350-1	442	30625	350	31,42 %
tsp-pcb442-350-10	442	30460	350	31,25 %
tsp-pcb442-350-100	442	30619	350	31,41 %

Tabela 6.1: Instâncias para o SCP

6.1 Resultados das heurísticas

Inicialmente, vamos comparar a performance de cada uma das heurísticas Lagrangeanas usando os custos originais do grafo.

Na Tabela 6.2, são apresentados resultados de uma execução de cada heurística e o resultado do *branch-and-cut* para cada instância. A primeira coluna apresenta o nome da instância. A seguir, temos 2 blocos de colunas para as heurísticas e um bloco para o algoritmo *branch-and-cut*. Os blocos de colunas para as heurísticas trazem, o valor da solução encontrada, o tempo de CPU gasto e o gap percentual para o valor ótimo. O bloco para o algoritmo de *branch-and-cut* traz o valor da solução ótima e o tempo de CPU para encontra-la. O *branch-and-cut* foi limitado em 1 hora, e assim, as instâncias onde o custo é representado por '-' significa que nenhuma solução primal viável foi encontrada no tempo estipulado e a instância onde o custo é seguido por '*' significa que não foi provada a otimalidade.

Como podem notar, a heurística *Cycle-Heuristic* conseguiu encontrar soluções melhores que a heurística *Tree-Heuristic* com os custos originais em 16 das 21 instâncias. Para as instâncias com 442 vértices, não foram encontrados limites primais usando o *branch-and-cut* no tempo estipulado.

Apesar dos altos valores dos *gaps* (em média 13,43 % para *Cycle-Heuristic* e 17,00

% para *Tree-Heuristic*), é necessário lembrar que a prioridade da heurística é ser rápida, pois ela será executada em algumas iterações do algoritmo NDRC, vide Seção 5.4. Na Tabela 6.2, nota-se o curto tempo de execução necessário para as heurísticas conseguirem encontrar uma solução viável de boa qualidade, ainda que o pior resultado esteja a 33,09% da solução ótima, como é o caso da heurística *Tree-Heuristic* para a instância tsp-kroA100-10-10. Nas instâncias menores, a heurística *Cycle-Heuristic* foi um pouco mais lenta. O mesmo resultado não acontece nas instâncias maiores, onde o comportamento se inverte. Em média, a *Cycle-Heuristic* executou em 0,93 segundos, enquanto que a *Tree-Heuristic* executou em 1,46 segundos.

	Cycle-Heuristic			Tree-Heuristic			Branch-and-Cut	
Instâncias	Custo	Tempo(s)	Gap	Custo	Tempo(s)	Gap	Custo	Tempo(s)
tsp-kroA100-10-1	11486	0,01	14,06 %	11486	0,01	14,06 %	9871	1,95
tsp-kroA100-10-10	**10415**	0,01	17,40 %	12857	0,01	33,09 %	8603	5,06
tsp-kroA100-10-100	10193	0,01	0,68 %	10193	0,00	0,68 %	10124	19,21
tsp-kroA100-25-1	**17144**	0,02	12,73 %	21241	0,03	29,57 %	14961	3,88
tsp-kroA100-25-10	18592	0,03	17,42 %	**17437**	0,02	11,95 %	15354	8,80
tsp-kroA100-25-100	18751	0,03	15,72 %	**17352**	0,02	8,92 %	15804	27,26
tsp-kroA100-40-1	22899	0,03	15,26 %	**21475**	0,03	9,64 %	19405	22,73
tsp-kroA100-40-10	**22851**	0,03	16,92 %	26291	0,02	27,79 %	18984	7,47
tsp-kroA100-40-100	**22733**	0,02	13,22 %	23243	0,03	15,13 %	19727	28,23
tsp-kroA100-75-1	**33948**	0,04	11,85 %	34340	0,03	12,85 %	29926	1,71
tsp-kroA100-75-10	**33107**	0,04	11,88 %	36847	0,04	20,82 %	29175	0,47
tsp-kroA100-75-100	**34521**	0,12	15,01 %	36274	0,03	19,12 %	29340	1,20
tsp-pcb442-100-1	**32136**	2,61	-	40493	3,27	-	-	-
tsp-pcb442-100-10	**34028**	2,49	-	48077	4,68	-	-	-
tsp-pcb442-100-100	**36288**	0,92	-	43238	2,66	-	-	-
tsp-pcb442-190-1	**50657**	2,12	-	56589	3,92	-	-	-
tsp-pcb442-190-10	**52227**	2,31	-	61135	2,92	-	-	-
tsp-pcb442-190-100	**55529**	1,76	-	62960	3,88	-	-	-
tsp-pcb442-350-1	**85929**	1,74	-	92565	3,19	-	-	-
tsp-pcb442-350-10	**98086**	2,45	-	100052	2,92	-	-	-
tsp-pcb442-350-100	**84872**	2,72	12,42 %	90002	3,04	17,41 %	74329*	3613,40
Média		0,93	13,43 %		1,46	16,97 %		

Tabela 6.2: Qualidade das soluções encontradas pelas heurísticas

6.2 Resultados do NDRC

Nessa seção, são comparadas as performances das heurísticas utilizando o grafo com seus custos modificados, como descrito na Subseção 5.4. Os algoritmos NDRC-*Tree-Heuristic* e NDRC-*Cycle-Heuristic* executam ao longo do método subgradiente as heurísticas *Tree-Heuristic* e *Cycle-Heuristic*, respectivamente. Em ambos, foi utilizada a mesma reformulação (4.38) com a dualização das restrições (4.32), (4.33) em forma de surrogates e as *2-matching* (4.53).

O número máximo de iterações para o NDRC foi fixado em 6000. Já o parâmetro inicial $\alpha = 2$ é reduzido em 40% a cada 300 iterações sem encontrar um limite dual melhor. Esses valores foram escolhidos com base em testes empíricos.

A comparação entre esses dois algoritmos é apresentada na Tabela 6.3, onde é possível ver que o algoritmo NDRC-*Tree-Heuristic* conseguiu encontrar *gaps* menores ou iguais ao algoritmo NDRC-*Cycle-Heuristic* em 17 das 21 instâncias. Essa melhora dos *gaps* foi fruto, principalmente, da diminuição do limite primal. Isso mostra que a heurística *Tree-Heuristic* conseguiu se beneficiar mais com uso das informações duais fornecidas pelo algoritmo NDRC levando a melhores resultados. O limite dual foi ligeiramente melhor usando o *Cycle-Heuristic*, conseguindo limite melhor em 5 instâncias e limite pior em 2 instâncias. O tempo teve o mesmo comportamento da tabela das heurísticas, ou seja, o NDRC-*Cycle-Heuristic* executou mais rápido em 16 das 21 instâncias e em média rodou em 470,68 segundo, enquanto que o NDRC-*Tree-Heuristic*, em média, rodou em 500,91 segundos.

	NDRC-*Tree-Heuristic*				NDRC-*Cycle-Heuristic*			
Instâncias	Primal	Dual	Tempo	Gap	Primal	Dual	Tempo	Gap
tsp-kroA100-10-1	9871	9870,96	8,99	OPT	9871	9871,01	11,55	OPT
tsp-kroA100-10-10	8603	8602,96	3,76	OPT	8603	8602,97	3,62	OPT
tsp-kroA100-10-100	10124	10124	3,95	OPT	10124	10124	3,81	OPT
tsp-kroA100-25-1	14961	14961	2,65	OPT	14961	14961	4,51	OPT
tsp-kroA100-25-10	15595	15225,4	20,41	2,37 %	**15354**	15239,5	17,51	**0,75 %**
tsp-kroA100-25-100	15804	15782	16,99	0,14 %	15804	15782	14,57	0,14 %
tsp-kroA100-40-1	19432	19357	20,52	0,39 %	19432	19357	18,35	0,39 %
tsp-kroA100-40-10	**18984**	18975,7	23,43	**0,04 %**	19320	18975,5	23,21	1,78 %
tsp-kroA100-40-100	**19739**	19676,7	20,72	**0,32 %**	19777	19676,8	19,81	0,51 %
tsp-kroA100-75-1	**30306**	29861,5	28,58	**1,47 %**	30598	29861,5	26,07	2,41 %
tsp-kroA100-75-10	**29370**	29165,5	24,46	**0,70 %**	29437	29165,5	21,22	0,92 %
tsp-kroA100-75-100	**29597**	29286,5	29,54	**1,05 %**	30162	29286,5	27,58	2,90 %
tsp-pcb442-100-1	28487	28189,7	685,06	1,04 %	**28422**	28198,8	588,13	**0,79 %**
tsp-pcb442-100-10	**30333**	28335,1	718,9	**6,59 %**	31594	28377,4	720,46	10,18 %
tsp-pcb442-100-100	**31390**	28931,2	713,37	**7,83 %**	32002	28910,9	658,8	9,66 %
tsp-pcb442-190-1	**45218**	42839,4	789,44	**5,26 %**	45642	42846,5	644,01	6,12 %
tsp-pcb442-190-10	47975	44395,2	803,19	7,46%	**47858**	44390,9	645,65	**7,24 %**
tsp-pcb442-190-100	**48287**	45702	831,77	**5,35 %**	48640	45706,8	629,37	6,03 %
tsp-pcb442-350-1	78187	74806,5	1707,82	4,32 %	**78003**	74806,5	1745,88	**4,10 %**
tsp-pcb442-350-10	**84190**	80507,6	2713,44	**4,37 %**	84574	80507,9	2668,96	4,81 %
tsp-pcb442-350-100	**76932**	73638,3	1352,14	**4,28 %**	77641	73638,1	1391,12	5,16 %
Média			500,91	3,12 %			470,68	3,76 %

Tabela 6.3: NDRC-*Tree-Heuristic* x NDRC-*Cycle-Heuristic*

6.2.1 Qualidade dos limites

Para mostrar a qualidade dos limites encontrados pelo algoritmo NDRC, é feita uma comparação com os limites duais encontrados pela Relaxação Linear, dada pela formulação

(2.18), e com os limites primais encontrados no *branch-and-cut*. A versão do NDRC utilizada nessa comparação foi a NDRC-*Tree-Heuristic*, uma vez que ela conseguiu *gaps* melhores em 81% das instâncias.

Na Tabela 6.4, pode-se notar que os limites duais encontrados pelo NDRC-*Tree-Heuristic* são iguais à Relaxação Linear para as instâncias do grupo *tsp-kroA100*, exceto na instância *tsp-kroA100-25-10*, onde teve um pequeno *gap* de 0,84% em relação à Relaxação Linear. Para esse grupo não houve ganho no tempo computacional, na verdade, em 7 instâncias o NDRC foi mais lento.

Para o segundo grupo de instância, o *tsp-pcb442*, o algoritmo de Relaxação Linear teve mais dificuldade para encontrar o valor da relaxação linear e por isso seu tempo de execução foi limitado em no máximo 1 hora. Apenas em uma das nove instâncias desse grupo o algoritmo NDRC-*Tree-Heuristic* teve o limite dual pior que o limite dual do algoritmo de Relaxação Linear limitado em uma hora e esse *gap* foi de apenas 0,84%. Nas instâncias *tsp-pcb442-350-10* e *tsp-pcb442-350-100* o algoritmo de Relaxação Linear não excedeu o tempo limite de 1 hora e dessa forma, ele convergiu mais rápido.

Na Tabela 6.5, pode-se ver que o método proposto conseguiu atingir bons limites primais também. Para o grupo *tsp-kroA100*, foi possível encontrar 6 valores ótimos com o algoritmo NDRC-*Tree-Heuristic* e o nos outros casos o maior *gap* para o ótimo foi de 1,55% na instância *tsp-kroA100-25-10*. No segundo grupo, *tsp-pcb442*, apenas a instância *tsp-pcb442-350-100* conseguiu encontrar um limite primal antes de ser abortado com 3613,40 segundos de execução. Nessa instância, o *gap* para o NDRC-*Tree-Heuristic* foi de 3,38%.

Vale ressaltar que, apesar do tempo do branch-and-cut ser menor que a heurística Lagrangeana em metade das instâncias do grupo *tsp-kroA100*, podemos perceber que o método proposto é capaz de conseguir bons limites e isso é importante para instâncias que o *branch-and-cut* não consegue encontrar nem a relaxação linear. Pode-se notar uma diferença no tempo do algoritmo de Relaxação Linear e do *branch-and-cut* das Tabelas 6.4 e 6.5. Isso se deve ao preprocessamento realizado pelo Xpress ao executar o *branch-and-cut*.

6.2.2 Fixações de variáveis

Além dos bons limites encontrados pelo método NDRC, também foi possível realizar fixações de variáveis para o problema. Na Tabela 6.6, é apresentada a quantidade de fixações de arestas e de vértices realizadas ao longo do NDRC-*Tree-Heuristic* e do NDRC-*Cycle-*

	NDRC-*Tree-Heuristic*		Relaxação Linear		
Instâncias	Dual	Tempo(s)	Dual	Tempo(s)	Gap
tsp-kroA100-10-1	9870,96	8,99	9871,00	4,64	0,00 %
tsp-kroA100-10-10	8602,96	3,76	8603,00	5,70	0,00 %
tsp-kroA100-10-100	10124,00	3,95	10124,00	25,22	0,00 %
tsp-kroA100-25-1	14961,00	2,65	14961,00	4,70	0,00 %
tsp-kroA100-25-10	15225,40	20,41	15354,00	8,28	0,84 %
tsp-kroA100-25-100	15782,00	16,99	15782,00	38,30	0,00 %
tsp-kroA100-40-1	19357,00	20,52	19357,00	15,77	0,00 %
tsp-kroA100-40-10	18975,70	23,43	18975,50	10,80	0,00 %
tsp-kroA100-40-100	19676,70	20,72	19676,75	35,98	0,00 %
tsp-kroA100-75-1	29861,50	28,58	29861,50	0,58	0,00 %
tsp-kroA100-75-10	29165,50	24,46	29165,50	0,34	0,00 %
tsp-kroA100-75-100	29286,50	29,54	29286,50	0,28	0,00 %
tsp-pcb442-100-1	28189,70	685,06	27403,25	-	-2,87 %
tsp-pcb442-100-10	28335,10	718,90	28576,41	-	0,84 %
tsp-pcb442-100-100	28931,20	713,37	27509,39	-	-5,17 %
tsp-pcb442-190-1	42839,40	789,44	42688,86	-	-0,35 %
tsp-pcb442-190-10	44395,20	803,19	43618,37	-	-1,78 %
tsp-pcb442-190-100	45702,00	831,77	44854,84	-	-1,89 %
tsp-pcb442-350-1	74806,50	1707,82	74790,57	-	-0,02 %
tsp-pcb442-350-10	80507,60	2713,44	80509,17	86,66	0,00 %
tsp-pcb442-350-100	73638,30	1352,14	73641,88	126,15	0,00 %

Tabela 6.4: Qualidade das soluções duais encontradas pelo NDRC

	NDRC-*Tree-Heuristic*		Branch-and-Cut		
Instâncias	Primal	Tempo(s)	Primal	Tempo(s)	Gap
tsp-kroA100-10-1	9871	8,99	9871	1,95	0,00 %
tsp-kroA100-10-10	8603	3,76	8603	5,06	0,00 %
tsp-kroA100-10-100	10124	3,95	10124	19,21	0,00 %
tsp-kroA100-25-1	14961	2,65	14961	3,88	0,00 %
tsp-kroA100-25-10	15595	20,41	15354	8,80	1,55 %
tsp-kroA100-25-100	15804	16,99	15804	27,26	0,00 %
tsp-kroA100-40-1	19432	20,52	19405	22,73	0,14 %
tsp-kroA100-40-10	18984	23,43	18984	7,47	0,00 %
tsp-kroA100-40-100	19739	20,72	19727	28,23	0,06 %
tsp-kroA100-75-1	30306	28,58	29926	1,71	1,25 %
tsp-kroA100-75-10	29370	24,46	29175	0,47	0,66 %
tsp-kroA100-75-100	29597	29,54	29340	1,20	0,87 %
tsp-pcb442-100-1	28487	685,06	-	-	-
tsp-pcb442-100-10	30333	718,90	-	-	-
tsp-pcb442-100-100	31390	713,37	-	-	-
tsp-pcb442-190-1	45218	789,44	-	-	-
tsp-pcb442-190-10	47975	803,19	-	-	-
tsp-pcb442-190-100	48287	831,77	-	-	-
tsp-pcb442-350-1	78187	1707,82	-	-	-
tsp-pcb442-350-10	84190	2713,44	-	-	-
tsp-pcb442-350-100	76932	1352,14	74329	3613,40	3,38 %

Tabela 6.5: Qualidade das soluções primais encontradas pelo NDRC

Heuristic usando o algoritmo de fixação descrito no Capítulo 4. As primeiras 4 colunas correspondem à quantidade pertencentual de variáveis fixadas no algoritmo NDRC-*Tree-Heuristic* e as últimas 4 colunas utilizando o algoritmo NDRC-*Cycle-Heuristic*.

Para o primeiro caso, $x_e = 1$, o algoritmo NDRC-*Tree-Heuristic* fixou mais variáveis em 9 das 12 instâncias do grupo *tsp-kroA100*. Nenhum dos dois algoritmos conseguiu fazer esse tipo de fixação para o segundo grupo de instâncias.

Para o segundo caso, $x_e = 0$, o algoritmo NDRC-*Tree-Heuristic* fixou mais variáveis em 7 das 12 instâncias do grupo *tsp-kroA100*. Já para o segundo grupo, ele superou o algoritmo NDRC-*Cycle-Heuristic* em 6 das 9 instâncias.

Para o terceiro caso, $y_i = 1$, o algoritmo NDRC-*Tree-Heuristic* fixou mais variáveis em 5 instâncias, enquanto que o NDRC-*Cycle-Heuristic* foi melhor em apenas 2. Em uma instância ocorreu um empate e para as outras nenhuma variável y_i foi fixada com valor 1.

Para o último caso, $y_i = 0$, o algoritmo NDRC-*Tree-Heuristic* fixou mais variáveis em 3 instâncias, enquanto que o NDRC-*Cycle-Heuristic* foi melhor em 4. Em uma instância empatou e para as outras, nenhuma variável y_i foi fixada com valor 0.

Esses números mostram quanto o algoritmo NDRC-*Tree-Heuristic* foi superior em relação ao algoritmo NDRC-*Cycle-Heuristic*. O algoritmo NDRC-*Tree-Heuristic* também teve um melhor desempenho em termos de número total de fixações quando comparado ao algoritmo NDRC-*Cycle-Heuristic*. Na última linha de todas as colunas, tem-se o percentual total de variáveis fixadas e podemos ver que o NDRC-*Tree-Heuristic* conseguiu mais fixações em todas elas. O mesmo pode ser visto nas médias.

6.2.3 Impacto das 2-matchings

Usando a heurística de separação proposta no Capítulo 4 para encontrar desigualdades *2-matchings* violadas, foi possível acelerar a convergência do método subgradiente. Nas Figuras 6.1 e 6.2, é possível observar esse fenômeno, onde a linha correspondente ao subgradiente com *2-matching*(linha mais clara) cresce, no eixo vertical, mais rápido à medida que a iteração aumenta. Além disso, para a instância kroA100-25-1, a versão com *2-matching* converge na iteração 708, enquanto que a versão sem *2-matching* converge apenas na iteração 1175. A instância kroA100-25-1 também teve esse mesmo comportamento, a versão com *2-matching* converge na iteração 3745, enquanto que a versão sem *2-matching* converge apenas na iteração 4782.

Analisando todas as instâncias pela Tabela 6.7, pode-se ver que a versão do NDRC usando as desigualdades *2-matchings* teve um desempenho melhor. Na média, ele teve *gap* de 3,12%, enquanto que o NDRC sem as *2-matchings* teve *gap* de 3,40%. Além disso, conseguiu 9 *gaps* melhores e 6 *gaps* piores. Ao utilizar as *2-matchings*, o NDRC teve um

	NDRC-*Tree-Heuristic*				NDRC-*Cycle-Heuristic*			
Instâncias	$x_e = 1$	$x_e = 0$	$y_i = 1$	$y_i = 0$	$x_e = 1$	$x_e = 0$	$y_i = 1$	$y_i = 0$
tsp-kroA100-10-1	**3,86%**	**93,35%**	**2,00%**	**36,00%**	2,88%	93,26%	1,00%	25,00%
tsp-kroA100-10-10	**2,48%**	93,63%	1,00%	**24,00%**	2,21%	**93,89%**	**2,00%**	17,00%
tsp-kroA100-10-100	1,49%	**92,31%**	0,00%	13,00%	**1,66%**	92,05%	**2,00%**	**15,00%**
tsp-kroA100-25-1	**2,39%**	**94,55%**	**8,00%**	5,00%	2,08%	94,00%	7,00%	5,00%
tsp-kroA100-25-10	0,00%	55,12%	0,00%	0,00%	**0,12%**	**72,82%**	0,00%	**1,00%**
tsp-kroA100-25-100	0,52%	83,90%	2,00%	2,00%	**0,70%**	**87,79%**	2,00%	**5,00%**
tsp-kroA100-40-1	**0,53%**	84,37%	**1,00%**	2,00%	0,43%	**84,42%**	0,00%	**3,00%**
tsp-kroA100-40-10	**1,55%**	**90,66%**	**8,00%**	**9,00%**	0,00%	70,30%	0,00%	0,00%
tsp-kroA100-40-100	**0,57%**	**84,88%**	**3,00%**	0,00%	0,38%	81,72%	1,00%	0,00%
tsp-kroA100-75-1	**0,21%**	**67,81%**	0,00%	0,00%	0,05%	57,29%	0,00%	0,00%
tsp-kroA100-75-10	**0,61%**	75,38%	0,00%	0,00%	0,46%	**79,12%**	0,00%	0,00%
tsp-kroA100-75-100	**0,35%**	**73,94%**	0,00%	0,00%	0,00%	55,48%	0,00%	0,00%
tsp-pcb442-100-1	0,00%	83,03%	0,00%	0,00%	0,00%	**84,86%**	0,00%	0,00%
tsp-pcb442-100-10	0,00%	**25,96%**	0,00%	0,00%	0,00%	1,29%	0,00%	0,00%
tsp-pcb442-100-100	0,00%	**13,40%**	0,00%	0,00%	0,00%	2,28%	0,00%	0,00%
tsp-pcb442-190-1	0,00%	**15,60%**	0,00%	0,00%	0,00%	5,93%	0,00%	0,00%
tsp-pcb442-190-10	0,00%	0,38%	0,00%	0,00%	0,00%	**0,68%**	0,00%	0,00%
tsp-pcb442-190-100	0,00%	**11,46%**	0,00%	0,00%	0,00%	4,66%	0,00%	0,00%
tsp-pcb442-350-1	0,00%	1,04%	0,00%	0,00%	0,00%	**1,90%**	0,00%	0,00%
tsp-pcb442-350-10	0,00%	**0,15%**	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
tsp-pcb442-350-100	0,00%	**1,85%**	0,00%	0,00%	0,00%	0,04%	0,00%	0,00%
Soma	**0,07%**	**19,45%**	**0,48%**	**1,76%**	0,05%	14,35%	0,29%	1,37%
Média	0,69%	54,42%	1,19%	4,33%	0,52%	50,66%	0,71%	3,38%

Tabela 6.6: Fixações encontradas pelo NDRC

pequeno aumento no tempo computacional de 3,36 % na média.

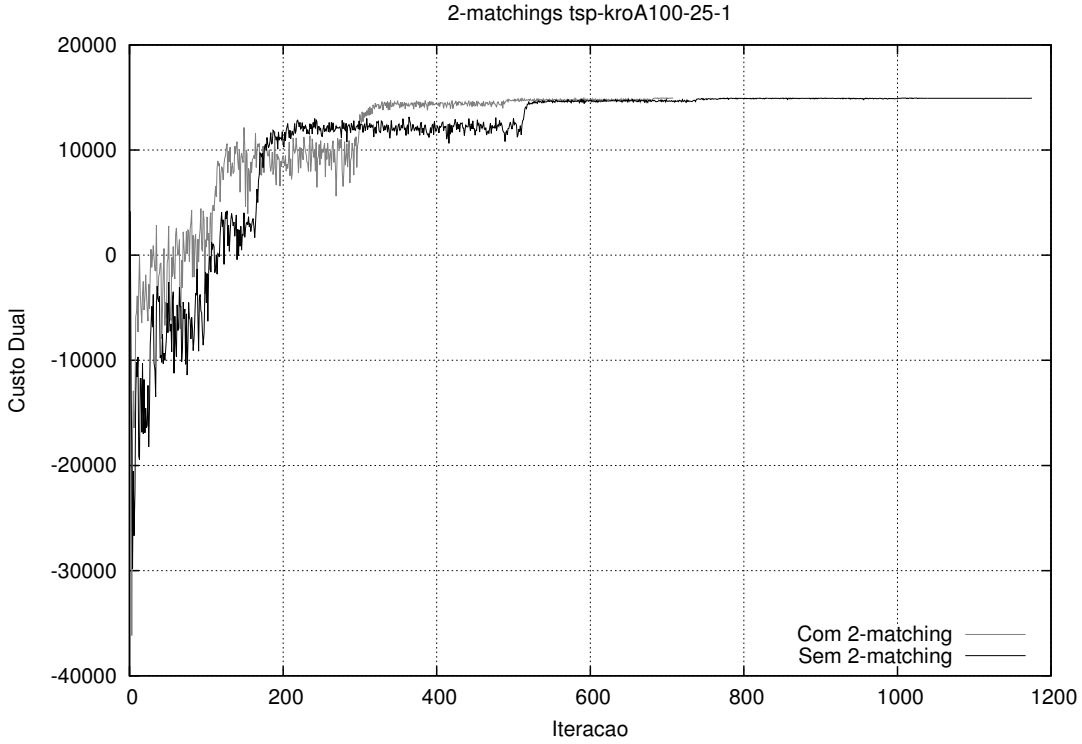


Figura 6.1: 2-Matching para kroA100-25-1.

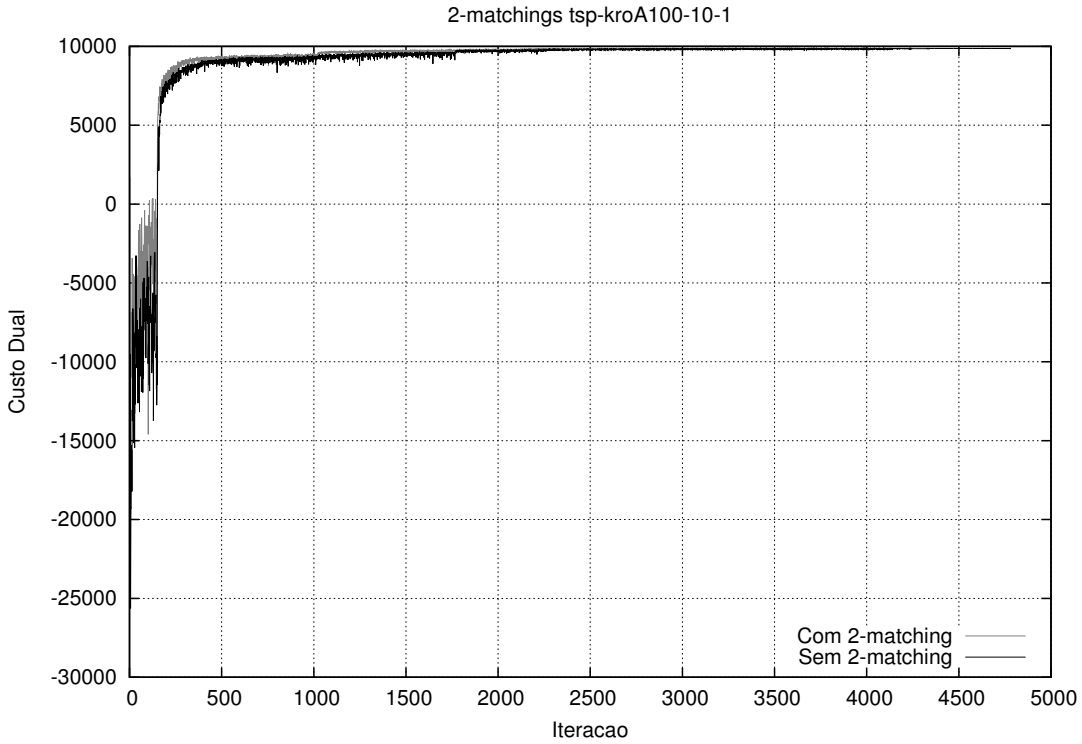


Figura 6.2: 2-Matching para kroA100-10-1.

	NDRC-*Tree-Heuristic* Sem 2-Matching				NDRC-*Tree-Heuristic* Com 2-matching			
Instâncias	Primal	Dual	Tempo	Gap	Primal	Dual	Tempo	Gap
tsp-kroA100-10-1	9871	9870,92	10,43	OPT	9871	9870,96	8,99	OPT
tsp-kroA100-10-10	8603	8602,91	3,97	OPT	8603	8602,96	3,76	OPT
tsp-kroA100-10-100	10124	10124,00	4,71	OPT	10124	10124	3,95	OPT
tsp-kroA100-25-1	14961	14961,00	3,57	OPT	14961	14961	2,65	OPT
tsp-kroA100-25-10	15529	15213,90	17,94	**2,03 %**	15595	15225,4	20,41	2,37 %
tsp-kroA100-25-100	15804	15782,00	14,68	0,14 %	15804	15782	16,99	0,14 %
tsp-kroA100-40-1	19432	19357,00	19,02	0,39 %	19432	19357	20,52	0,39 %
tsp-kroA100-40-10	19322	18975,60	21,50	1,79 %	18984	18975,7	23,43	**0,04 %**
tsp-kroA100-40-100	19736	19676,80	18,16	**0,30 %**	19739	19676,7	20,72	0,32 %
tsp-kroA100-75-1	30397	29861,50	23,69	1,76 %	30306	29861,5	28,58	**1,47 %**
tsp-kroA100-75-10	29252	29165,50	16,40	**0,30 %**	29370	29165,5	24,46	0,70 %
tsp-kroA100-75-100	29830	29286,50	20,78	1,82 %	29597	29286,5	29,54	**1,05 %**
tsp-pcb442-100-1	28352	28200,50	695,67	**0,53 %**	28487	28189,7	685,06	1,04 %
tsp-pcb442-100-10	30726	28314,70	759,27	7,85 %	30333	28335,1	718,9	**6,59 %**
tsp-pcb442-100-100	32023	28918,10	737,82	9,70 %	31390	28931,2	713,37	**7,83 %**
tsp-pcb442-190-1	45195	42846,00	797,19	**5,20 %**	45218	42839,4	789,44	5,26 %
tsp-pcb442-190-10	48080	44391,50	821,76	7,67 %	47975	44395,2	803,19	**7,46 %**
tsp-pcb442-190-100	48457	45714,00	874,76	5,66 %	48287	45702	831,77	**5,35 %**
tsp-pcb442-350-1	78224	74806,70	1381,55	4,37 %	78187	74806,5	1707,82	**4,32 %**
tsp-pcb442-350-10	83717	80509,00	2697,77	**3,83 %**	84190	80507,6	2713,44	4,37 %
tsp-pcb442-350-100	77091	73639,60	1188,44	4,48 %	76932	73638,3	1352,14	**4,28 %**
Média			482,34	3,40 %			500,91	**3,12 %**

Tabela 6.7: Impacto das 2-Matchings no NDRC-*Tree-Heuristic*

Capítulo 7

Conclusão e Trabalhos Futuros

Nesse trabalho foi desenvolvido um algoritmo NonDelayed-Relax-And-Cut para o problema do ciclo de Steiner. Esse problema tem como objetivo encontrar um ciclo de custo mínimo onde alguns vértices são obrigatórios e outros são opcionais. Como aplicação, tem-se projetos de redes confiáveis de telecomunicação e de transporte.

Primeiro, foi introduzida uma formulação para o problema do ciclo de Steiner. Em seguida, foi apresentada uma reformulação para o problema SCP de modo a conseguir desenvolver uma heurística Lagrangeana capaz de conseguir bons limites duais e primais.

Usando a reformulação proposta, foi desenvolvido um algoritmo Non Delayed Relax and Cut para o SCP, e com isso foi possível mostrar a qualidade dos limites duais. Para conseguir os limites primais, foram propostas duas heurísticas e uma buscal local. Quando executadas ao longo do subgradiente (com os custos modificados) conseguem encontrar bons limites primais. A *Tree-Heuristic* foi a que conseguiu os melhores resultados.

Uma outra contribuição, foi a criação de uma heurística de separação para desigualdades do tipo *2-matching* quando a solução da relaxação é uma *1-tree*. Essa heurística pode ser aplicada tanto para o SCP quanto para o TSP em baixo tempo computacional.

Por fim, para testar a eficiência da relaxação apresentada, foram criadas instâncias para o SCP, uma vez que não foram encontrados resultados computacionais para esse problema na literatura.

Com o algoritmo NDRC desenvolvido nesse trabalho foi possível encontrar o valor da relaxação linear em 13 das 14 instâncias as quais o algoritmo da Relaxação Linear convergiu. Nas outras 7 instâncias, o NDRC conseguiu um limite dual e primal melhor que o algoritmo da Relaxação Linear com limitação de tempo de 1 hora em 6 instâncias. A heurística de separação para as desigualdades *2-matchings* também mostrou-se eficiente,

pois ela fez o método subgradiente convergir mais rápido.

Com o número de fixações, o bom limite primal e os cortes encontrados ao longo do subgradiente, acredita-se que isso possa diminuir o tempo do algoritmo *branch-and-cut* utilizando essas informações em um *warm-start*.

Além disso, ainda existe a possibilidade de fazer a relaxação do SCP usando o problema de Assigment. Teoricamente o limite é o mesmo, mas como a solução relaxada será formada por ciclos isso pode ter um impacto positivo na qualidade das soluções primais.

Referências

[1] Ahuja, R. K.; Magnanti, T. L.; Orlin, J. B. Network flows: theory, algorithms, and applications.

[2] Applegate D.L., Bixby R.E., C. V. e. C. W. The traveling salesman problem - a computational study, 2006.

[3] Chvátal, V. Edmonds polytopes and weakly hamiltonian graphs. *Mathematical Programming 5*, 1 (1973), 29–40.

[4] Crowder, H.; Padberg, M. Solving large-scale symmetric traveling salesman problems to optimality. *Management Science 26* (1980), 495–509.

[5] Dantzig, G.B.; Fulkerson, D.; Johnson, S. Solution of a large-scale traveling salesman problem. *Operations Research 2* (1954), 393–410.

[6] Dantzig, G. B. Reminiscences about the origins of linear programming. *Operations Research Letters 1*, 2 (1982), 43–48.

[7] Davendra, D. Traveling salesman problem, theory and applications.

[8] Edmonds, J. Maximum matching and a polyhedron with 0, 1-vertices. *J Res Natl Bur Stan 69B* (1965), 125–130.

[9] Goldberg, A. V.; Tarjan, R. E. A new approach to the maximum-flow problem. *Journal of the ACM (JACM) 35*, 4 (1988), 921–940.

[10] Grotschel, M.; Holland, O. Solution of large-scale symmetric traveling salesman problems. *Mathematical Programming 51* (1991), 141–202.

[11] Grötschel, M.; Monma, C. L. Integer polyhedra arising from certain network design problems with connectivity constraints. *SIAM Journal on Discrete Mathematics 3*, 4 (1990), 502–523.

[12] Grötschel, M.; Padberg, M. W. On the symmetric travelling salesman problem i: Inequalities. *Mathematical Programming 16*, 1 (1979), 265–280.

[13] Grötschel, M.; Padberg, M. W. On the symmetric travelling salesman problem ii: Lifting theorems and facets. *Mathematical Programming 16*, 1 (1979), 281–302.

[14] Hao, J.; Orlin, J. B. A faster algorithm for finding the minimum cut in a directed graph. *Journal of Algorithms 17*, 3 (1994), 424–446.

[15] Held, M.; Karp, R. The traveling-salesman problem and minimum spaning tress. *Operations Research*, 17 (1970), 1138–1162.

[16] Hwang, F.; Richards, D. S. Steiner tree problems. *Networks 22*, 1 (1992), 55–89.

[17] Hwang, F. K.; Richards, D. S.; Winter, P. *The Steiner tree problem*. Elsevier, 1992.

[18] Johnson, D.S; McGeoch, L.; Rothberg, E. Asymptotic experimental analysis for the held- karp traveling salesman bound. *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms* (1996), 341–350.

[19] Juan-José; Salazar-González. The steiner cycle polytope. *European Journal of Operational Research*, 147 (2003), 671–679.

[20] Jünger, M.; Reinelt, G.; Rinaldi, G. The traveling salesman problem. *Handbooks in Operations Research and Management Science 7* (1995), 225–330.

[21] K. Steiglits, P. Weiner, D. K. The design of minimum cost survivable networks. *IEEE Transactions and Circuit Theory*, 16 (1969), 455–460.

[22] Karmarkar, N. A new polynomial-time algorithm for linear programming. *Combinatorica 4*, 4 (1984), 373–395.

[23] Kruskal, J. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms 7* (1956), 48–50.

[24] Letchford, A. N.; Reinelt, G.; Theis, D. O. A faster exact separation algorithm for blossom inequalities. In *Integer programming and combinatorial optimization*. Springer, 2004, pp. 196–205.

[25] Lucena, A. Steiner problem in graphs: lagrangean relaxation and cutting planes. *COAL Bulletin*, 21 (1992), 2–8.

[26] Lucena, A. Tight bounds for the steiner problem in graphs, 1993.

[27] Lucena, A. Non delayed relax-and-cut algorithms. *Annals of Operations Research*, 140 (2005), 375–410.

[28] Lucena, A.; Salles da Cunha, A.; Simonetti, L. A new formulation and computational results for the simple cycle problem. *Electronic Notes in Discrete Mathematics 44* (2013), 83–88.

[29] Naddef, D. Polyhedral theory and branch-and-cut algorithms for the symmetric tsp. in g. gutin & a. punnen (eds.). In *The traveling salesman problem and its variations*. Kluwer Academic Publisher, 2002, pp. 29–116.

[30] O¨ncan, T.; Altinel, I.; Laporte, G. A comparative analysis of several asymmetric traveling salesman problem formulations. *Computers & Operations Research 36* (2009), 637–654.

[31] Or, I. Traveling salesman-type combinatorial problems and their relation to the logistics of regional blood banking. *Ph.D. Thesis, Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL* (1976).

[32] Orman, A.; Williams, H. A survey of different integer programming formulations of the travelling salesman problem. *Optimisation, Econometric and Financial Analysis Advances in Computational Management Science Springer: Berlin, Heidelberg* (2006), 91–104.

[33] Padberg, M.; Hong, S. On the symmetric travelling salesman problem: A computational study. *Mathematical Programming Study 12* (1980), 78–107.

[34] Padberg, M.; M, G. Polyhedral computations. *The traveling salesman problem* (1985), 307–360.

[35] Padberg, M.; Rinaldi, G. Optimization of a 532-city symmetric traveling salesman problem by branch and cut. *Operations Research Letters 6*, 1 (1987), 1–7.

[36] Padberg, M.; Rinaldi, G. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review 33* (1991), 60–100.

[37] Padberg, M. W.; Rao, M. Odd minimum cut-sets and b-matchings. *Math. Oper. Res 7* (1982), 67–80.

[38] Ribeiro, C.C., E. U. e. R. W. A hybrid grasp with perturbations for the steiner problem in graphs. *INFORMS Journal on Computing*, 14 (2002), 228–246.

[39] Takahashi, H.; Matsuyama, A. An approximate solution for the steiner problem in graphs. *Math. Japonica 24*, 6 (1980), 573–577.

[40] Volgenant, T.; Jonker, R. The symmetric traveling salesman problem and edge exchanges in minimal l-trees. *European Journal of Operational Research 2* (1983), 394–403.