RELATÓRIO FINAL DE INICIAÇÃO CIENTÍFICA

Problema do Caixeiro Viajante com Coleta e Entrega

ALUNO: Fabricio C. Machado
fabcm1@gmail.com

ORIENTADOR: Flávio K. Miyazawa
fkm@ic.unicamp.br

7 de agosto de 2013

Resumo

Neste projeto foram estudados métodos poliédricos para a resolução de problemas combinatórios, em especial o método *branch and cut*. Também foram estudadas diversas classes de desigualdades úteis para a implementação de um programa para a resolução do Problema do Caixeiro Viajante com Coleta e Entrega (PDTSP), um problema desafiante com relação ao tamanho das instâncias resolvidas pelos algoritmos atuais. Os resultados obtidos pelo algoritmo implementado são comparados com os resultados encontrados na literatura recente.

Sumário

1 Introdução

Problemas de otimização, na sua forma geral, tratam de maximizar ou minimizar uma função definida sobre um certo domínio. Nos problemas de otimização combinatória este domínio é finito e, em princípio, bastaria testar todos os elementos em busca de um que seja ótimo. Ainda assim, esta estratégia ingênua costuma ser impraticável, pois o tamanho do domínio pode ser muito grande, mesmo para instâncias de tamanho moderado. Dessa maneira, surge a necessidade de usar técnicas mais elaboradas para encontrar soluções de valor ótimo. Uma das estratégias usadas é associar um politopo ao problema e buscar um sistema de inequações que o descrevam. A subárea de pesquisa que surge, denominada combinatória poliédrica, é um dos temas do estudo deste projeto.

O PROBLEMA DO CAIXEIRO VIAJANTE COM COLETA E ENTREGA (*The Traveling Salesman Problem with Pickup and Delivery - TSPPD*), problema estudado neste projeto, é definido da seguinte forma. Considere um conjunto de n pedidos, cada um composto por um vértice de coleta e um de entrega. Seja $P = \{s_1, ..., s_n\}$ o conjunto dos vértices de coleta e $D = \{t_1, ..., t_n\}$ o conjunto dos vértices de entrega. Vértices Oa e Ob representam a saída e a entrada para o depósito, respectivamente. O TSPPD é definido em um grafo completo e não direcionado $G = (V, E, d)$, onde $V = P \cup D \cup \{Oa, Ob\}$ é o conjunto de vértices, $E = \{\{x, y\} \mid x, y \in V, x \neq y\}$ é o conjunto de arestas e $d(e)$ denota o comprimento (não negativo) das arestas $e \in E$. O objetivo do TSPPD é encontrar um ciclo Hamiltoniano com distância total mínima, começando no vértice Oa e terminando em Ob, sujeito à restrição de precedência de que cada vértice de coleta deve ser visitado antes de seu respectivo vértice de entrega.

O problema TSPPD é NP-difícil, visto que qualquer instância do Problema do Caixeiro Viajante (*Traveling Salesman Problem - TSP*) pode ser transformada polinomialmente em uma instância do TSPPD (para isso, basta substituir um dos vértices do TSP pelos vértices de depósito Oa e Ob e trocar cada outro vértice v_i por um par de vértices s_i e t_i posicionados no mesmo local). Apesar disto, a estratégia de resolução via métodos poliédricos é promissora, pois têm se mostrado eficaz na resolução de instâncias reais de grande porte do TSP (resultados recentes relatam a resolução de instâncias reais com dezenas de milhares de cidades - ver sec. 1.7 de [1]). No entanto, o TSPPD é ainda especialmente difícil do ponto de vista empírico, no artigo de Dumitrescu et al. [3] apenas são resolvidas instâncias com até 35 pedidos e em artigos anteriores apenas instâncias com até 15 pedidos haviam sido resolvidas.

As principais referências usadas no estudo de combinatória poliédrica foram os livros [4] e [8]. Para uma introdução ao TSP e algumas técnicas usadas na fase de separação do algoritmo (especialmente a separação de blossoms), foi usado o livro [1]. Finalmente, a principal referência usada para o estudo de resultados e desigualdes específicas para o PDTSP foi o artigo [3].

No primeiro semestre foi dado ênfase a um estudo teórico de diversos resultados e ferramentas úteis para a combinatória poliédrica. Destacando considerações sobre a complexidade computacional de problemas, propriedades e descrições de poliedros e a equivalência entre otimização e separação.

No segundo semestre foi implementado um algoritmo com uma estratégia *branch-and-cut* para

a resolução exata do TSPPD, usando diversas classes de desigualdades propostas em [3] e outras estratégias descritas no livro [1] para a resolução do TSP. Também foram feitos testes computacionais com as mesmas instâncias usadas no artigo citado.

Na próxima seção será descrito como funciona um método *branch-and-cut*, nas seções 3 e 4 é descrito o poliedro associado ao problema e as classes de cortes usadas no algoritmo e na seção 5 são dados mais detalhes sobre a implementação do algoritmo e os resultados obtidos.

2 O método *Branch-and Cut*

2.1 Problemas de Otimização Combinatória (POC)

É conveniente definirmos formalmente a classe dos PROBLEMAS DE OTIMIZAÇÃO COMBINATÓRIA (POC): Dada uma tripla $(E, c, \mathcal{S})$, onde E é um conjunto finito, c uma função que associa a cada elemento de E um valor real e $\mathcal{S}$ um conjunto de soluções viáveis, formado por subconjuntos de E. Deseja-se encontrar $S \in \mathcal{S}$ que minimiza (ou maximiza) a função objetivo $c(S) := \sum_{e \in S} c(e)$.

Enumerando os elementos de E, podemos associar a S um vetor em $\mathbb{R}^{|E|}$ correspondente ao seu vetor de incidência χ^S definido como: $\chi^S_e = 1$, se $e \in S$ e $\chi^S_e = 0$, caso contrário. Assim, temos $c(S) = c(\chi^S) := \sum_{e \in E} c(e).\chi^S_e$.

Considerando o politopo[1] formado pelo fecho convexo dos vetores de incidência dos elementos de $\mathcal{S}$ e usando o resultado de que seus vértices são precisamente esses vetores de incidência, podemos considerar o problema equivalente de otimização sobre o poliedro. Caso encontremos um sistema de inequações que descrevam o poliedro, obtemos um problema de programação linear. A dificuldade está em obter esse sistema de inequações (o que nem sempre é possível), ou ainda na possibilidade desse sistema ter tamanho exponencial em relação às entradas do problema original.

2.2 Método *Branch and Bound*

Um método *branch and bound* se baseia em enumerar todas as soluções viáveis de um problema em uma estrutura de árvore de decisão e percorrer seus ramos de forma sistemática, evitando ramos que levam a soluções inviáveis ou que não levam a soluções melhores que as já encontradas.

Um exemplo de enumeração das soluções é, para cada aresta $e \in E$, considerar dois subproblemas: aquele em que a aresta está na solução $(x_e = 1)$ e aquele em que a aresta não está $(x_e = 0)$. Assim, poderíamos enumerar todas as possíveis soluções através de uma árvore de decisão binária com $2^{|E|}$ folhas. Entretanto, apesar de comum, essa não é a única estratégia de ramificação possível. O importante é que em cada ramificação o espaço de soluções representado no nó ramificado esteja representado nos seus ramos.

Se conhecermos boas soluções heurísticas (limitantes superiores) e boas relaxações (limitantes inferiores) para o problema, a busca na árvore de decisão pode ser muito mais eficiente. Se em

[1] Ao leitor interessado por uma descrição mais detalhada sobre teoria de poliedros, recomendamos a referência [4].

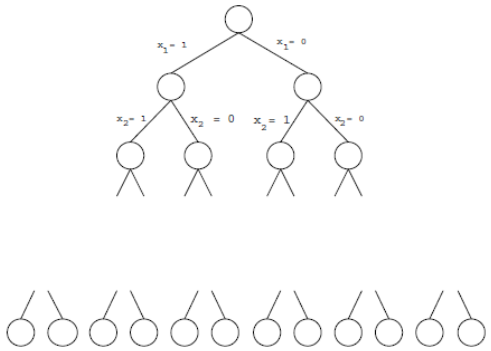

Figura 1: Exemplo de árvore de decisão.

um dado nó dessa árvore o valor do limitante inferior para este nó (e portanto para todos os seus descendentes) for superior ao valor de uma solução heurística conhecida, então a solução ótima do problema não poderá estar em um descendente desse nó, e podemos considerar toda a subárvore a partir desse nó como visitada. Com isso, eliminamos vários ”ramos” da árvore de decisão, e a busca se torna mais eficiente.

2.3 Método de Planos de Corte

Como dito na seção 2.1, estamos considerando um problema de otimização sobre o poliedro P formado pelo fecho convexo dos vetores de incidência dos elementos de $\mathcal{S}$. Caso tenhamos uma descrição do poliedro em termos de desigualdades, este é um problema de programação linear que pode ser resolvido de forma eficiente.

Nos problemas de programação linear inteira[2] (PLI) (como será o caso deste problema) não temos P, mas temos um poliedro Q que é uma aproximação de P, onde $P \subseteq Q$ (conforme será descrito na seção 3.1). Seja y_0 uma solução ótima de Q para a função objetivo. Se $y_0 \in P$, devolvemos o ponto y_0 como solução ótima do problema. Caso $y_0 \notin P$, procuramos por uma desigualdade $ax \leq \gamma$ tal que $P \subseteq \{x \mid ax \leq \gamma\}$ e $ay_0 > \gamma$. Dizemos que esta desigualdade é um *plano de corte* que separa y_0 de P. Neste caso, podemos obter um novo poliedro Q', adicionando esta desigualdade a Q. Em seguida, encontramos uma nova solução ótima de Q' e repetimos o processo até que uma solução ótima de P seja encontrada.

Gomory [5] e Chvátal [2] foram os primeiros a desenvolver métodos automatizados e finitos para se obter planos de corte em problemas de programação linear inteira. Garantindo, assim, que o método termina em um número finito de passos, mas sem obter um limite para o número destes (que pode ser exponencial em relação ao tamanho da entrada).

Um resultado importante é a *equivalência entre otimização e separação*. O problema de, dados um poliedro e um ponto não pertencente a este, encontrar um plano de corte que os separe é chamado de

[2] Ao leitor interessado em uma introdução mais detalhada à problemas de programação linear inteira e métodos *branch and cut*, recomendamos o texto [7], no qual esta seção está baseada.

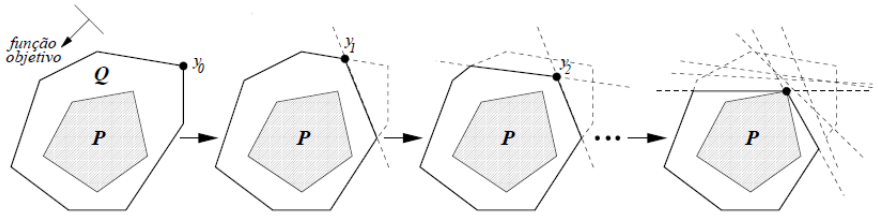

Figura 2: Sequência de inserções de planos de corte.

problema da separação. Conforme descrito em detalhes por Grötschel et al. [6], o método elipsóide é uma redução polinomial da separação para a otimização, de modo que se soubermos resolver este problema de maneira eficiente, poderemos criar um algoritmo polinomial para o problema. A questão é que em muitos casos não é conhecida uma descrição completa das faces de P (e não é esperado que tal descrição seja obtida em um problema NP-difícil), de modo que nem sempre o problema de separação pode ser resolvido em tempo polinomial (os métodos de separação usados neste projeto serão descritos na seção 4).

2.4 Método *Branch and Cut*

Um método *branch and cut* é uma combinação dos dois métodos apresentados. Nesta estratégia, investe-se um grande esforço na geração de planos de corte nos nós da árvore de decisão para limitar seu crescimento e gerar bons limitantes inferiores.

2.5 Um método 2-aproximativo

Considerando a busca na árvore de decisão, é útil já começarmos o algoritmo com uma solução viável obtida através de alguma heurística. Esta solução servirá como limitante superior inicial e poderá ajudar a cortar ramos da árvore mais cedo.

Dado um problema de minimização, dizemos que um algoritmo é uma *k-aproximação* para o problema se, para qualquer instância, o custo c da solução obtida está dentro de um fator k do custo c^* da solução ótima, isto é, se $c/c^* \leq 2$.

Iniciamos o algoritmo deste projeto com um limitante superior inicial gerado pelo seguinte método 2-aproximativo. Primeiro, é encontrada a solução ótima x^*_{TSP} do Problema do Caixeiro Viajante (TSP) sobre o mesmo grafo, isto é, ignorando-se as restrições de precedência. Como as instâncias consideradas no PDTSP são relativamente pequenas em relação às usadas no TSP, esta etapa é resolvida rapidamente. Em seguida, uma solução x viável para o PDTSP é construída, percorrendo-se o ciclo da solução x^*_{TSP} no máximo duas vezes, pulando os vértices já visitados e os vértices t_i de entrega cujo respectivo vértice s_i de coleta ainda não foi visitado.

Assumindo que a função de distância d satisfaça a desigualdade triangular[3], temos que $c(x) \leq$

[3] $\forall u, v, w \in V, d(\{u, v\}) + d(\{v, w\}) \geq d(\{u, w\})$

$2c(x^*_{TSP})$ e como o TSP é uma relaxação do PDTSP, também temos $c(x^*_{TSP}) \leq c(x^*_{PDTSP})$ e portanto $c(x) \leq 2c(x^*_{PDTSP})$, de forma que o método é uma 2-aproximação para problema. Na prática, os resultados obtidos costumam ser bem mais próximos do que o limitante calculado, conforme verificado na seção 5.2.

2.6 Chamadas alternativas da rotina de separação

Um estratégia interessante apresentada na seção 5.10 de [1] aumenta a eficiência das rotinas de separação, especialmente as baseadas em heurísticas, para a geração de planos de corte de maior qualidade.

Dados $\bar{x}$, a melhor solução viável encontrada até o momento pelo algoritmo (limitante superior) e x^*, uma solução fracionária do problema relaxado do nó atual, a estratégia consiste em aplicar as rotinas de separação em:

$$x^{**} = \gamma.\bar{x} + (1-\gamma)x^*,\ \gamma \in (0,1)$$

A idéia é que x^{**} seja um ponto mais próximo do poliedro P do que x^*, mas ainda assim não pertencente a este. Caso a rotina de separação consiga obter um plano de corte, este poderá ter melhor qualidade. Caso não consiga, tenta-se separar x^* normalmente.

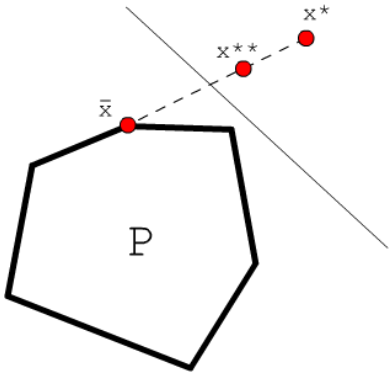

Figura 3: Um corte mais próximo do poliedro.

3 Definição do poliedro

3.1 Formulação

O PDTSP é definido conforme descrito na introdução e uma solução x corresponde ao vetor de incidência das arestas usadas no ciclo, de modo que serão usadas $|E|$ variáveis binárias, cada x_e correspondente à uma aresta $e \in E$. Dado $S \subseteq V$, seja $\delta(S) = \{\{i,j\} \in E \mid i \in S, j \notin S\}$. Se $S = \{i\}$, escreveremos $\delta(i)$ em vez de $\delta(\{i\})$ e dado $E' \subseteq E$, usaremos ainda a notação $x(E') = \sum_{e \in E'} x_e$:

$$minimize \sum_{e \in E} d(e) x_e \tag{1}$$

sujeito a:

$$x_{Oa,Ob} = 1 \tag{2}$$

$$x(\delta(i)) = 2 \qquad \forall i \in V \tag{3}$$

$$x(\delta(S)) \geq 2 \qquad \forall S \subseteq V,\ 3 \leq |S| \leq |V|/2 \tag{4}$$

$$x(\delta(S)) \geq 4 \qquad \forall S \in \mathcal{U} \tag{5}$$

$$x_e \in \{0, 1\} \qquad \forall e \in E \tag{6}$$

Sendo que $\mathcal{U}$ é a coleção dos subconjuntos $S \subseteq V$ satisfazendo $3 \leq |S| \leq |V| - 2$, com $Oa \in S,\ Ob \notin S$ e tal que existe $s_i \in P$ com $s_i \notin S$ e $t_i \in S$.

A restrição (2) faz com que os vértices Oa e Ob apareçam consecutivamente no ciclo. Isso faz com que possamos considerar o ciclo começando em Oa e terminando em Ob. Observe que como essa aresta sempre é usada, ela não precisa ser considerada como uma variável real do modelo. Além disso, o comprimento desta aresta também não é relevante. De fato, nas instâncias consideradas, os vértices Oa e Ob coincidem.

As restrições (3) são restrições de grau, de modo que todo vértice tenha grau 2. As restrições (4) são as restrições de eliminação de subciclos (*subtour elimination constraints - SEC*), que fazem o ciclo ser conexo. As restrições (5) são as restrições de precedência, que garantem que o vértice s_i seja visitado antes do vértice t_i para todo $s_i \in P$.

As restrições (6) são as restrições de integralidade, características dos PLI. Na resolução dos problemas lineares essa restrição é trocada por $0 \leq x_e \leq 1$, gerando um problema relaxado mais facilmente resolvível que, porém, em geral leva a soluções ótimas fracionárias. Tenta-se separar a solução encontrada do poliedro original por algum plano de corte com as técnicas descritas na seção 4. Caso a nova solução ainda seja fracionária e as técnicas falhem (por não conhecermos uma descrição completa, via desigualdades, de P) o algoritmo entra na fase de *branching*, ramificando a árvore de decisão.

Como consequência das restrições de precedência podemos concluir que $x_{\{Oa,t_i\}} = 0$ para todo $t_i \in D$ e $x_{\{Ob,s_i\}} = 0$ para todo $s_i \in P$. Portanto, estas variáveis também não precisam ser efetivamente consideradas no modelo, sendo usadas apenas para facilitar a formulação. Pode-se mostrar (Teorema 1 de [3]) que as equações (2), (3) e as apresentadas neste parágrafo são todas as equações obedecidas por todas as soluções viáveis do problema (as demais sendo apenas combinações lineares destas). Este resultado define a dimensão do poliedro do problema PDTSP.

3.2 Separação de subciclos e restrições de precedência

Observe que existe uma restrição (4) de subciclo para cada subconjunto de vértices (na equação está escrito $|S| \leq |V|/2$, porque se $|S| > |V|/2$ violar a equação, temos que $\bar{S} = V \backslash S$ também viola,

já que $\delta(S) = \delta(\bar{S})$). Existe, portanto, um número exponencial de restrições, o que torna inviável que estas restrições sejam consideradas em um algoritmo de programação linear.

Entretanto, o problema de separação destas desigualdades pode ser resolvido em tempo polinomial. Dado uma solução x, para saber se esta satisfaz todas as desigualdades (4), basta testar se existem pares de vértices u e v separados por um corte $\delta(S)$ tal que:

$$x(\delta(S)) < 2 \qquad u \in S,\ v \in V \backslash S$$

e podemos verificar isto através de um algoritmo de fluxos máximos que encontre um uv-corte mínimo[4]. Desta forma, podemos separar as desigualdades (4) com uma execução de fluxo máximo para cada par de vértices. Na realidade, é possível fazer bem menos execuções, fazendo uso de uma árvore de cortes de Gomory-Hu, que faz uso de apenas $|V| - 1$ chamadas do algoritmo.

A separação das restrições de precedência (5) também pode ser feita de maneira semelhante. Basta considerar os pares de vértices $\{s_i, t_i\}$ e para que o conjunto S satisfaça as exigências de $\mathcal{U}$, basta executar o algoritmo de fluxo máximo em um vetor x' obtido a partir de x, onde o valor de $x_{\{Oa,t_i\}}$ e $x_{\{Ob,s_i\}}$ são substituídos por algum valor alto que garanta que estas arestas não se encontrem no corte mínimo.

A seguir, algumas ilustrações de soluções intermediárias de uma instância (prob15a) do PDTSP. Os vértices do conjunto P estão representados em azul, os vértices do conjunto D em vermelho, $Oa = Ob$ está representado em preto. As arestas com valores fracionários aparecem em vermelho.

Na figura 4 temos, à esquerda, a solução obtida apenas com a restrição (3) e com a restrição (6) relaxada. No centro temos a solução ótima inteira, porém vemos a existência de subciclos. À direita, temos a solução ótima após a inclusão das restrições (4), que corresponde ao problema TSP.

Na figura 5 temos, à esquerda, a solução obtida após a inclusão das restrições (5), porém com a condição (6) novamente relaxada. Essa é a fase mais difícil do algoritmo, pois pôde-se observar que a inclusão das restrições de precedência dificultam muito a convergência para uma solução inteira, o que tornam as desigualdades descritas na seção 4 essenciais para a eficência do algoritmo. À direita, a solução ótima desta instância.

4 Desigualdades válidas

Nesta seção serão descritas outras desigualdades válidas para o poliedro descrito na seção 3.1 e que são usadas na fase de separação do método *branch and cut* implementado.

Dado um conjunto de vértices $S \subseteq V$, usaremos a notação $\pi(S) = \{s_i \in P \mid t_i \in S\}$ e $\sigma(S) = \{t_i \in D \mid s_i \in S\}$. Também denotaremos por $E(S) = \{\{i, j\} \in E \mid i \in S, j \in S\}$ e escreveremos $x(S)$ no lugar de $x(E(S))$.

[4]Uma explicação mais detalhada sobre a relação entre fluxos máximos e cortes mínimos se encontra na monografia escrita na disciplina MS777, disponível em: `http://vigo.ime.unicamp.br/Projeto/2012-2/ms777/ms777_fabricio.pdf`

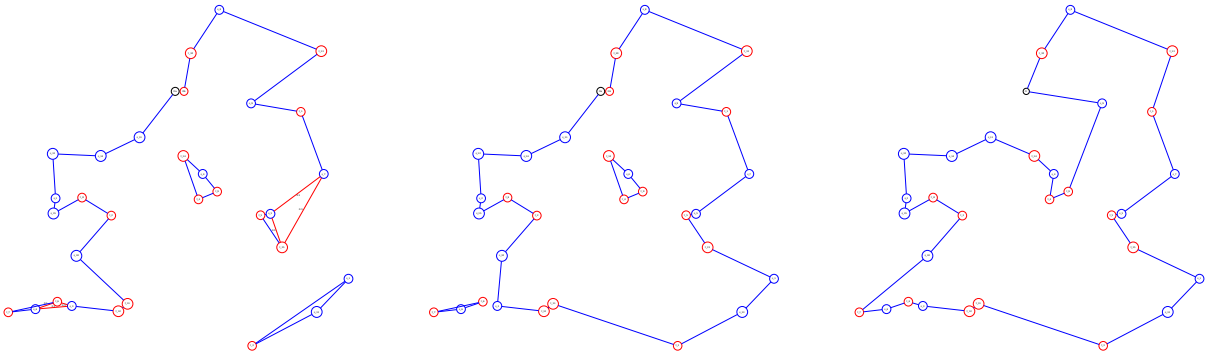

Figura 4: Exemplos de soluções intermediárias sem o uso das restrições de precedência.

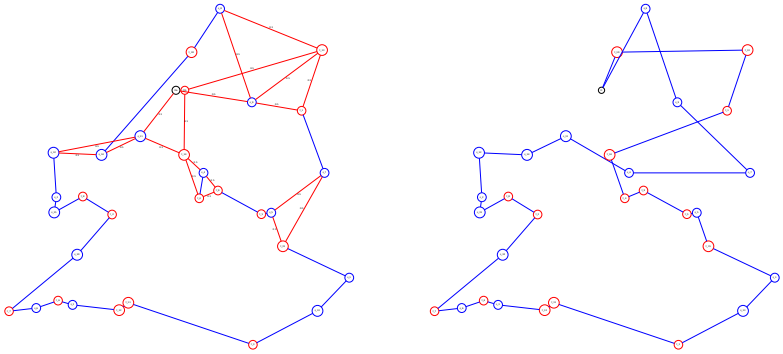

Figura 5: Após a inclusão da restrição de precedência (5).

A seguinte relação será útil para a compreensão das desigualdades. Dado $S \subseteq V$, somando-se a equação (3) para todos os vértices de S e usando o fato de que as arestas com ambos os extremos em S são somadas duas vezes e as arestas com apenas um extremo em S uma, vale:

$$2x(S) + x(\delta(S)) = 2|S|$$

$$|S| - x(S) = \frac{1}{2}x(\delta(S)) \tag{7}$$

4.1 Blossons

As desigualdades blossom não levam em conta as precedências e também são usadas com sucesso na resolução do TSP. Elas são úteis na separação de algumas estruturas fracionárias que surgem nas soluções, após a aplicação das restrições de subciclos, como na figura 6:

Dados $H, T_1, ..., T_k$ subconjuntos de V tais que:

- $|T_i| = 2$, $T_i \cap H \neq \emptyset$ e $T_i \cap (V \backslash H) \neq \emptyset$, $\forall i \in \{1, ..., k\}$.
- $T_1, ..., T_k$ são dois a dois disjuntos.
- k é ímpar e $k \geq 3$.

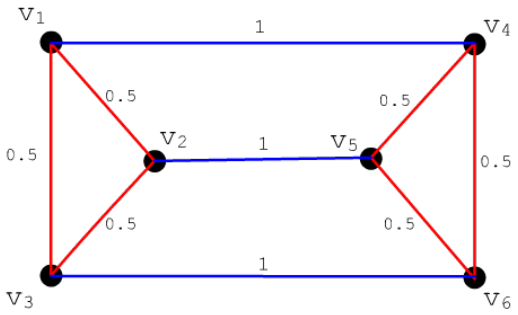

Figura 6: Uma solução fracionária que atende todas as restrições de grau (3) e subciclo (4).

A seguinte desigualdade é válida:

$$x(\delta(H)) + \sum_{i=1}^{k} x(\delta(T_i)) \geq 3k + 1 \tag{8}$$

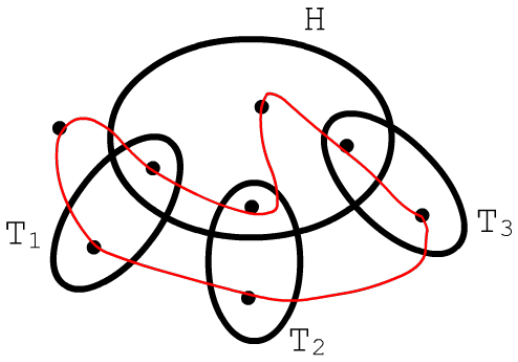

Figura 7: Exemplo de ciclo e conjuntos H, T_1, T_2, T_3.

Para compreender a desigualdade, observe que para cada conjunto T_i, o ciclo pode cruzar a fronteira de H com a aresta contida em T_i ou não. Caso cruze, o ciclo irá cruzar a fronteira de H uma vez e a fronteira de T_i duas vezes no trecho correspondente, contribuindo com 3 unidades no lado esquerdo da desigualdade. Caso não cruze, o ciclo terá que passar por T_i mais de uma vez, cruzando sua fronteira 4 vezes. Disto, resulta: $x(\delta(H)) + \sum_{i=1}^{k} x(\delta(T_i)) \geq 3k$. Entretanto, a soma efetuada no lado esquerdo é um número par, já que um ciclo sempre cruza as fronteiras de conjuntos um número par de vezes e, como k é ímpar, podemos somar mais um ao lado direito.

Olhando de novo a figura 6, observamos que os conjuntos $H = \{v_1, v_2, v_3\}, T_1 = \{v_1, v_4\}, T_2 = \{v_2, v_5\}$ e $T_3 = \{v_3, v_6\}$ geram uma desigualdade violada.

A separação de blossons foi baseada nas seções 7.1 e 7.2 do livro [1]. Defina $\epsilon = 0.3$ e dada uma solução fracionária x, considere o grafo G_ϵ definido sobre os mesmos vértices e formado com as arestas $\{e \in E \mid \epsilon \leq x_e \leq 1 - \epsilon\}$. São encontrados os blocos[5] deste grafo através de um algoritmo

[5]Os blocos são as componentes não separáveis maximais do grafo. Um conjunto de vértices é dito separável (em um grafo sem laços) quando pode ser desconectado do resto de sua componente conexa pela remoção de um único vértice.

baseado em busca em profundidade. Cada bloco será considerado um candidato a conjunto H.

Percorre-se os vértices do bloco em busca de arestas em $\delta(H)$ com $x_e \geq 1 - \epsilon$, estas arestas formarão os conjuntos T. Caso dois conjuntos T se intersectem fora de H, os conjuntos são removidos e a intersecção é adicionada a H. Caso se intersectem dentro de H, os conjuntos também são removidos e a intersecção é retirada de H. Ao final, caso sejam obtidos um número ímpar de conjuntos T, verifica-se se a desigualdade correspondente é violada.

4.2 π- and σ-Inequalities

Dado $S \subseteq V \backslash \{Ob\}$, seja $E_S^\pi = \{\{i,j\} \mid i \in S \backslash \pi(S)\ ,\ j \notin S \cup \pi(S) \cup \{Oa\}\}$. A seguinte desigualdade é válida:

$$x(E_S^\pi) \geq 1 \tag{9}$$

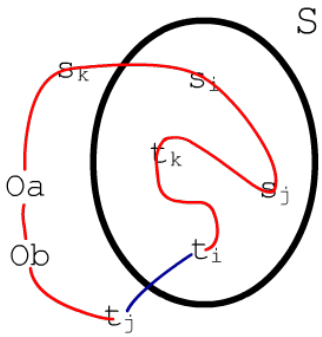

Figura 8: A desigualdade π.

Para compreender esta desigualdade, basta considerar o último vértice visitado em S por um ciclo válido. Este vértice deve pertencer à $S \backslash \pi(S)$ e seu sucessor deve pertencer à $V \backslash (S \cup \pi(S) \cup \{Oa\})$

A desigualdade σ é totalmente análoga, basta considerar o primeiro vértice de S visitado por um subciclo e notar que este vértice deve pertencer à $S \backslash \sigma(S)$ e seu predecessor deve pertencer à $V \backslash (S \cup \sigma(S) \cup \{Ob\})$. Assim, sendo $E_S^\sigma = \{\{i,j\} \mid i \in S \backslash \sigma(S)\ ,\ j \notin S \cup \sigma(S) \cup \{Ob\}\}$, a seguinte desigualdade á válida:

$$x(E_S^\sigma) \geq 1 \tag{10}$$

A separação destas desigualdades foi implementada de forma heurística, com um algoritmo guloso e aleatório. Partindo-se de um conjunto S contendo um vértice inicial (o algoritmo itera sobre todos os vértices de $P \cup D$), vértices v são adicionados iterativamente seguindo a regra:

$$v = argmin_{v \in (P \cup D) \backslash S} \{min\{x(E^\sigma_{S \cup \{v\}}), x(E^\pi_{S \cup \{v\}})\}\}$$

Um ruído aleatório é adicionado às avaliações de $x(E^\sigma_{S \cup \{v\}})$ e $x(E^\pi_{S \cup \{v\}})$ para aumentar a diversidade dos conjuntos gerados. Ainda assim, é frequente que o mesmo conjunto S seja gerado muitas

vezes e que várias desigualdades semelhantes violadas sejam adicionadas. Para evitar isto, foi adicionado a seguinte regra: no decorrer de uma execução da rotina separadora, caso seja encontrado S_1 tal que $x(E^{\sigma}_{S_1}) < 1$ ou $x(E^{\pi}_{S_1}) < 1$, qualquer outro conjunto S_2 encontrado após só terá sua desigualdade adicionada ao modelo se o valor de sua violação for maior que a anteriormente encontrada.

4.3 Lifted Subtour Elimination Constraints (LSEC)

Dado $S \subseteq P \cup D$ tal que existe $s_i \in P \cap S$ e $t_i \in S$, a seguinte desigualdade é válida:

$$x(S) + \sum_{s_j \in S \cap P, t_j \notin S} x_{\{s_i,t_j\}} \leq |S| - 1 \tag{11}$$

Usando a equação (7) no conjunto S, podemos reescrever (11):

$$x(\delta(S)) \geq 2 + \sum_{s_j \in S \cap P, t_j \notin S} 2x_{\{s_i,t_j\}} \tag{12}$$

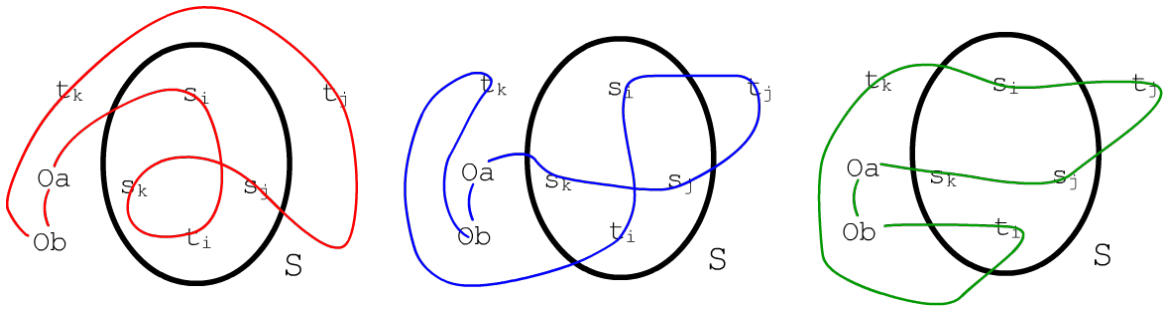

Figura 9: As três possíveis formas de um ciclo passar por s_i.

A parte $x(\delta(S)) \geq 2$ é consequência direta das restrições de subciclos (4). Como o grau de s_i deve ser igual a 2, existem 3 formas de um ciclo passar por s_i: usando nenhuma, uma ou duas das arestas presentes no somatório $\sum_{s_j \in S \cap P, t_j \notin S} 2x_{\{s_i,t_j\}}$, conforme ilustrado na figura 9. Analisando cada um dos casos, vemos que, devido às restrições de precedência, estas arestas forçam o ciclo a passar pela fronteira de S mais vezes.

Para a implementação da separação destas desigualdades, para cada vértice $s_i \in P$, consideramos o conjunto $D_i = \{t_j \in D \backslash \{t_i\} \mid x_{\{s_i,t_j\}} > 0\}$ dos possíveis candidados a vértices t_j a ser considerados no somatório da desigualdade (se $D_i = \emptyset$, não será possível encontrar desigualdades LSEC violadas por x a partir de s_i, já que estas se reduzirão às restrições de subciclo (4)). Para cada subconjunto W de D_i, constrói-se o melhor conjunto S_W tal que $\{s_i, t_i\} \cup \pi(W) \subseteq S_W$ e $W \cap S_W = \emptyset$, isto é:

$$S_W = argmin_{S \subset P \cup D}\{x(\delta(S)) \mid \{s_i, t_i\} \cup \pi(W) \subseteq S \; e \; W \cap S = \emptyset\}.$$

o que pode ser feito de forma semelhante ao efetuado na separação das restrições (4) e (5). Então verifica-se se a desigualdade LSEC correspondente é violada por x.

É importante ressaltar que esta separação proposta possui complexidade exponencial em análise de pior caso, já que são considerados todos os subconjuntos W de D_i. Entretanto, em geral, mesmo em uma solução fracionária, o vértice s_i possui poucos vizinhos na solução x, de modo que não são considerados tantos conjuntos. Além disso, considerando os subconjuntos W em ordem crescente de tamanho, notamos que se $x(\delta(S_W)) \geq 2 + \sum_{t_j \in D_i} 2x_{\{s_i,t_j\}}$, então não precisamos considerar nenhum conjunto $W' \supset W$, pois como estes conjuntos impõem mais restrições na busca de $S_{W'}$, temos que $x(\delta(S_W)) \leq x(\delta(S_{W'}))$ e como $\sum_{t_j \in W'} 2x_{\{s_i,t_j\}} < \sum_{t_j \in D_i} 2x_{\{s_i,t_j\}}$, com certeza não poderá ser encontrada nenhuma desigualdade com W'. Esta estratégia é especialmente útil quando $|W| = 1$, pois neste caso o vértice correspondente pode ser removido de D_i, acelerando a busca.

4.4 Generalized Order Constraints (GOC)

Dados $S_1, ..., S_m \subset P \cup D$ conjuntos mutuamente disjuntos tais que $m \geq 2$ e $S_i \cap \pi(S_{i+1}) \neq \emptyset, \forall i \in \{1, ..., m-1\}$ e $S_m \cap \pi(S_1) \neq \emptyset$. A seguinte desigualdade é válida:

$$\sum_{i=1}^{m} x(S_i) \leq \sum_{i=1}^{m} |S_i| - m - 1 \tag{13}$$

Usando a equação (7) em cada conjunto S_i, podemos reescrever (13):

$$\sum_{i=1}^{m} x(\delta(S_i)) \geq 2m + 2 \tag{14}$$

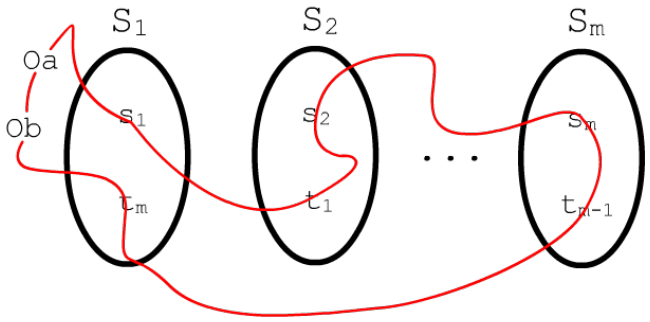

Figura 10: Exemplo de GOC e um ciclo válido.

A parte $\sum_{i=1}^{m} x(\delta(S_i)) \geq 2m$ é consequência direta das restrições de subciclos (4), mas conforme ilustrado na figura 10, devido ao fato de cada conjunto S_i possuir o predecessor de algum vértice de S_{i+1}, um ciclo que respeite as restrições de precedência deverá cruzar a fronteira de algum conjunto pelo menos 4 vezes.

Neste trabalho, apenas foi implementada a separação para o caso $m = 2$. Para cada par de pedidos

$\{s_i, t_i\}$ e $\{s_j, t_j\}$, são calculados os melhores conjuntos S_1 e S_2 com estes pares. Isto é:

$$S_1 = argmin_{S \subset P \cup D}\{x(\delta(S)) \mid \{s_i, t_j\} \subseteq S, \{s_j, t_i\} \cap S = \emptyset\}.$$
$$S_2' = argmin_{S \subset P \cup D}\{x(\delta(S)) \mid \{s_j, t_i\} \subseteq S, \{s_i, t_j\} \cap S = \emptyset\}.$$
$$S_2 = S_2' \backslash (S_1 \cap S_2').$$

Estes são os "melhores" conjuntos com os pares $\{s_i, t_i\}$ e $\{s_j, t_j\}$ pois, usando o fato dos conjuntos calculados terem fronteira mínima, é possível mostrar que $x(\delta(S_2')) = x(\delta(S_2))$ e portanto será encontrada alguma desigualdade violada pela solução x, caso exista alguma do tipo.

5 Resultados

5.1 Detalhes de Implementação

Todos os testes foram executados em um processador Intel(R) Core(TM)2 Quad CPU Q6600 @ 2.40GHz rodando Linux. Os algoritmos foram implementados em C++, o algoritmo *branch and cut* foi implementado usando o resolvedor Gurobi v.5.5 e os algoritmos com grafos com o auxílio da biblioteca Lemon v.1.2.3.

Além das separações das restrições de subciclos e precedência serem aplicadas sobre todas as soluções inteiras geradas pelo algoritmo, foram experimentadas diversas metodologias para a aplicação das demais rotinas de separação (apresentadas na seção 4) nas soluções fracionárias geradas pelo algoritmo. Os melhores resultados foram obtidos aplicando-se todas as seis rotinas de separação nos 100 primeiros nós da árvore de busca gerados e a cada 10 nós, após.

Sempre que as seis rotinas de separação são aplicadas, usa-se a estratégia descrita na seção 2.6, em que a partir da melhor solução viável conhecida, usa-se uma solução fracionária intermediária para a geração de planos de corte de melhor qualidade (caso não seja possível, tenta-se separar a solução fracionária original). Foram testados diversos valores de γ, chegando-se a melhores resultados com $\gamma = 0.1$.

As instâncias são as mesmas usadas por Dumitrescu et al. [3] e foram criadas gerando-se $2n + 1$ pontos de forma aleatória em uma grade $[0, 1000] \times [0, 1000]$, o primeiro ponto correspondendo ao depósito $Oa = Ob$. A função de comprimento d foi calculada como a distância euclideana entre os pontos (arredondada). Foram feitos testes com instâncias de tamanho $n = 5, 10, 15, 20, 25, 30, 35$, com 5 instâncias de cada tamanho nomeadas *probnX*, onde X varia entre a,b,c,d,e.

Nas instâncias de pequeno e médio porte, os resultados apresentados são a média (arredondada) obtida após 10 execuções do algoritmo sobre as mesmas instâncias, usando-se sementes diferentes. Nas instâncias maiores, o programa frequentemente atingiu o tempo limite de execução (definido como 14400s, de maneira semelhante ao feito por Dumitrescu et al. [3]) e apenas uma semente foi usada. Além das rotinas separadoras das desigualdades $-\pi$, $-\sigma$ usarem uma componente aleatória na execução (conforme explicado na seção 4.2), o Gurobi efetua o branching de forma aleatória, causando uma variação nos resultados obtidos.

5.2 Resultados obtidos pelo método aproximativo

Na tabela a seguir, os resultados obtidos pelo método aproximativo descrito na seção 2.5. São exibidos o valor da solução ótima de cada instância c^*, o valor da solução obtida pela aproximação c e a diferença porcentual $gap = 100(c - c^*)/c^*$:

instância	c*	c	gap	instância	c*	c	gap
prob5a	3585	4408	23	prob15b	5391	5731	6,3
prob5b	2565	4237	65,2	prob15c	5008	7223	44,2
prob5c	3787	4694	24,0	prob15d	5566	6733	21,0
prob5d	3128	3947	26,2	prob15e	5229	7374	41,0
prob5e	3123	3217	3,0	prob20a	5698	7122	25,0
prob10a	4896	5682	16,1	prob20b	6213	7132	14,8
prob10b	4490	5306	18,2	prob20c	6200	8228	32,7
prob10c	4070	5481	34,7	prob20d	6106	7446	21,9
prob10d	4551	5386	18,3	prob20e	6465	7596	17,5
prob10e	4874	7133	46,3	prob25c	7095	9115	28,5
prob15a	5150	6434	24,9	prob25e	6754	8343	23,5

Tabela 1. Resultados obtidos pelo método aproximativo.

Observamos que apesar da garantia ser da aproximação ser no máximo 100% maior que a solução ótima, apenas em uma instância (pequena) o valor obtido foi maior que 50%, sendo que na maioria o gap foi bem menor. Apesar da rotina envolver a resolução de um TSP com os mesmos vértices, notou-se que a execução foi bem rápida, levando $3s$ para resolver todas as instâncias.

5.3 Resultados obtidos pelo método exato

A seguir, os resultados obtidos pelo método *branch and cut* implementado. Na tabela, são exibidos o número de sementes usado, o tempo gasto, o número de nós criados na árvore de busca, o número de cortes inserido, e a razão $Gap = 100\underline{c}/\overline{c}$ (onde $\underline{c}$ é o melhor limitante inferior encontrado e $\overline{c}$ o melhor limitante superior). As colunas com * à direita contêm os resultados obtidos por Dumitrescu et al. [3] para comparação.

instância	sementes	tempo (s)	nós	cortes	Gap (%)	tempo* (s)	nós*	cortes*	Gap* (%)
prob5a	10	0	1	22	100	0	1	19	100
prob5b	10	0	1	0	100	0	1	6	100
prob5c	10	0	1	10	100	0	1	6	100
prob5d	10	0	1	0	100	0	1	6	100
prob5e	10	0	1	44	100	0	1	45	100
prob10a	10	1	85	476	100	3	4	134	100
prob10b	10	1	58	428	100	2	1	135	100
prob10c	10	0	19	182	100	0	1	93	100
prob10d	10	1	136	655	100	1	1	93	100
prob10e	10	1	64	523	100	4	1	97	100
prob15a	10	7	310	1302	100	8	6	268	100
prob15b	10	12	793	1663	100	21	45	580	100
prob15c	10	1	9	311	100	0	1	165	100
prob15d	10	5	123	900	100	14	15	390	100
prob15e	10	1	20	668	100	0	1	140	100
prob20a	10	5	34	888	100	12	9	438	100
prob20b	10	56	959	3193	100	20	20	473	100
prob20c	10	17	400	1252	100	19	28	444	100
prob20d	10	14	168	1108	100	17	28	369	100
prob20e	10	85	1401	2256	100	58	115	962	100
prob25a	1	14402	31989	5274	93,6	14400	14377	3244	97,8
prob25b	1	10956	53257	5600	100	3138	7952	2266	100
prob25c	10	346	2410	2894	100	291	878	1255	100
prob25d	1	14403	45163	5283	98,0	14323	21627	3873	100
prob25e	10	861	4878	4222	100	72	125	704	100
prob30a	1	14535	12230	4951	80,9	14400	8269	3238	98,5
prob30b	1	8486	51025	3150	100	2843	4528	2262	100
prob30c	1	14402	24303	5302	99,0	1891	4269	2019	100
prob30d	1	11376	17902	5504	100	573	1783	1372	100
prob30e	1	14415	16803	6285	93,6	14400	10523	3412	99,4
prob35a	1	14435	19300	4376	94,0	2104	3541	1649	100
prob35b	1	14598	9724	6104	73,1	14400	6167	2764	94,8
prob35c	1	14505	5111	6449	78,6	14400	8427	3194	98,9
prob35d	1	14504	7759	6643	72,8	14400	9025	2944	97,2
prob35e	1	14532	8119	6662	77,5	14400	9791	3562	94,6

Tabela 2. Resultados obtidos pelo método *branch and cut*.

Observamos que o programa implementado chegou a ser mais rápido do que o programa usado no artigo de referência nas instâncias de pequeno e médio porte, porém não obteve resultados tão bons nas instâncias maiores. É importante ressaltar que o resolvedor Gurobi e o computador não foram os mesmos usados na implementação do artigo, o que deve ter dado uma vantagem para o programa deste trabalho. De fato, neste trabalho não foram implementadas todas as rotinas de separação e classes de desigualdades propostas no artigo, o que diminuiu a capacidade do algoritmo controlar o crescimento da árvore. Podemos observar isto comparando a quantidade de nós criados nas árvores de busca.

5.4 Comparação entre versões do programa

É interessante comparar o quanto os diversos procedimentos implementados neste projeto contribuem para a perfomance final. Para isto, foram realizados testes com as seguintes versões do programa:

A versão A é a versão completa, usada nos testes da seção anterior. A versão B não possui a estratégia de separação com soluções intermediárias descrita na seção 2.6. A versão C, além de não possuir a estratégia de separação, apenas usa as desigualdades GOC além das de subciclo e precedência. A versão D não usa nenhuma estratégia de separação além das desigualdades de subciclo e precedência.

Na tabela a seguir, os tempos de execução de cada versão na resolução das instâncias de tamanho intermediário. Denotamos por X as execuções que excederam o tempo limite ($14400s$).

instância	**A**	**B**	**C**	**D**
prob15a	7	4	3	603
prob15b	12	53	13	X
prob15c	1	1	1	14
prob15d	5	9	2	168
prob15e	1	1	0	26
prob20a	5	6	4	1887
prob20b	56	26	42	X
prob20c	17	24	39	X
prob20d	14	15	15	X
prob20e	85	118	95	X
prob25a	X	X	X	X
prob25b	10956	9157	X	X
prob25c	346	2444	3619	X
prob25d	X	X	X	X
prob25e	861	863	443	X

Tabela 3. Comparação entre o tempo de execução de diferentes versões do programa.

Observando a coluna da versão D, podemos observar claramente a importância das desigualdades propostas na seção 4 para a melhor performance do algoritmo. Ainda assim, existe uma troca entre o esforço computacional gasto ao se aplicar as rotinas de separação nos nós da árvore de busca e o efeito produzido pela inclusão dos cortes no modelo. Isto explica o sucesso alcançado pela versão C em certas instâncias.

Comparando-se as versões A e B, apesar de A ter ido um pouco pior em prob20b e prob25b, destacamos a grande melhora obtida por A em prob25c, o que chama atenção para o potencial do método descrito na seção 2.6.

6 Conclusão

Com a realização deste projeto foi possível uma iniciação às ferramentas usadas na resolução de problemas de otimização combinatória, em especial aos métodos poliédricos e aos algoritmos *branch and cut*. O estudo do Problema do Caixeiro Viajante com Coleta e Entrega propiciou um exemplo concreto de aplicação destas técnicas e uma experiência com as dificuldades existentes na implementação de um algoritmo eficiente.

Também foi possível comparar os resultados obtidos diretamente com resultados encontrados na literatura recente [3]. Foram obtidos bons resultados nas instâncias pequenas e médias, ainda que não tenha sido possível superar os resultados nas instâncias maiores. Destacamos a estratégia de separação proposta por [1] e descrita na seção 2.6 deste relatório, que demonstra grande potencial para melhorar a eficiência do programa em certas instâncias.

7 Apoio

O autor agradece ao CNPq pelo suporte por meio da concessão de bolsa PIBIC de iniciação científica.

Referências Bibliográficas

[1] D. L. Applegate, R. E. Bixby, V. Chvatal e W. J. Cook, *The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics)*. Princeton University Press, Princeton, NJ, USA, 2007.

[2] Chvátal, V. *Edmonds polytopes and a hierarchy of combinatorial problems*. Discrete Mathematics, 4:185-224, 1973.

[3] I. Dumitrescu, S. Ropke, J.-F. Cordeau e G. Laporte, *The traveling salesman problem with pickup and delivery: polyhedral results and a branch-and-cut algorithm*. Mathematical Programming, V. 121, I. 2, 269-305. 2009.

[4] C.E. Ferreira e Y. Wakabayashi, *Combinatória Poliédrica e Planos-de-Corte Faciais*. X Escola de Computação, Unicamp, 1996.

[5] Gomory, R. *Outline of an algorithm for integer solutions to linear programs*. Bulletin of the American Mathematical Society, 64:275-278. 1958.

[6] M. Grötschel, L. Lovász, A. Schrijver, *The ellipsoid method and its consequences in combinatorial optimization*. Combinatorica 1, 169-197. 1981.

[7] F. K. Miyazawa, *Programação Inteira*, XI Escola Regional de Informática SBC - Paraná, pp. 49-90, 2003.

[8] G.L. Nemhauser e L.A.Wolsey, *Integer and Combinatorial Optimization*. JohnWiley & Sons, 1988.