UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS GRADUAÇÃO EM ENGENHARIA DE PRODUÇÃO

WALTON PEREIRA COUTINHO

UM ALGORITMO *BRANCH-AND-BOUND* PARA O PROBLEMA DO CAIXEIRO VIAJANTE SUFICIENTEMENTE PRÓXIMO

JOÃO PESSOA
2014

WALTON PEREIRA COUTINHO

UM ALGORITMO *BRANCH-AND-BOUND* PARA O PROBLEMA DO CAIXEIRO VIAJANTE SUFICIENTEMENTE PRÓXIMO

Dissertação apresentada ao curso de Pós-Graduação em Engenharia de Produção da Universidade Federal da Paraíba, como requisito final para obtenção do Grau de Mestre. Área de Concentração: Pesquisa Operacional.

Orientador: Prof. Dr. Roberto Quirino do Nascimento

Co-orientador: Prof. Dr. Anand Subramanian

João Pessoa

2014

WALTON PEREIRA COUTINHO

UM ALGORITMO *BRANCH-AND-BOUND* PARA O PROBLEMA DO CAIXEIRO VIAJANTE SUFICIENTEMENTE PRÓXIMO

Dissertação apresentada ao curso de Pós-Graduação em Engenharia de Produção da Universidade Federal da Paraíba, como requisito final para obtenção do Grau de Mestre. Área de Concentração: Pesquisa Operacional.

Aprovada em 13 de Fevereiro de 2014.

BANCA EXAMINADORA

Prof. Dr. Roberto Quirino do Nascimento — Orientador
Departamento de Computação Científica — UFPB

Prof. Dr. Anand Subramanian — Co-orientador
Departamento de Engenharia de Produção — UFPB

Prof. Dr. Lucídio dos Anjos Formiga Cabral — Examinador Interno
Departamento de Computação Científica — UFPB

Prof. Dr. Artur Alves Pessoa — Examinador Externo
Departamento de Engenharia de Produção — UFF

João Pessoa
2014

Aos meus pais Cosme Nivaldo Coutinho da Silva e Marluce Albertino Pereira Coutinho.

As coisas encobertas são para o Senhor, nosso Deus; porém as reveladas são para nós e para nossos filhos, para sempre, para cumprirmos todas as palavras desta lei. (Dt.29:29)

AGRADECIMENTOS

Primeiramente gostaria de agradecer a Deus pela sua graça, todo o sustento e todos os livramentos que recebi durante esta jornada.

Aos meus pais, por terem se esforçado ao máximo, com sabedoria, privando-se diversas vezes de suas próprias necessidades para me proporcionar a melhor educação possível. Estes sim, eu considero como os verdadeiros mestres.

À toda minha família: novamente aos meus pais Nivaldo e Marluce; às minhas irmãs Kylma e Ilka; pela ligação fraterna que temos e pela ajuda e paciência nas horas necessárias; à minha Lidiane, pelo seu amor, companheirismo, cumplicidade, sabedoria e por ser tão bonita por dentro e por fora. Agradeço ainda às minhas tias/madrinhas Elza França e Christina Albertino por sempre me apoiarem quando precisei.

Aos meus comparsas Allan, Emerson e Jefferson, pelos momentos de descontração e a "zuada" que fazemos com a banda. Ao pastor Anderson e todos os meus irmãos da IECT, por terem orado por mim.

Agradeço especialmente aos meus orientadores Prof. Roberto Quirino e o Prof. Anand Subramanian, e ao meu orientador à distância Prof. Artur Pessoa. A estes, dedico as palavras de Isaac Newton quando disse "Se enxerguei mais longe, foi porque me apoiei sobre os ombros de gigantes".

Aos meus amigos Raphael e Artur Kramer e Luciano Costa pela presença constante e ajuda crucial nos momentos em que precisei.

Aos colegas do mestrado, Fernanda, Léo, Emanoela, Íris, Tálita, Vitória, Elaine e Valtânia, em especial, pelo exemplo de dedicação e superação.

Ao pessoal do LabMeQA, Yuri, Acioli, Cathaline, Gracinha, Vitor, Bia, Katyanne, Teobaldo e Felipe, pela zueira e parceria nos congressos.

Aos Professores Lucídio Cabral, Luciano Costa e Marcel Góis por terem contribuído indiretamente para esta conquista.

Aos professores e funcionários do DEP e do PPGEP da UFPB, em especial Cláudia Gohr, Luiz Bueno, Maria Silene, Ana Araujo, Neto Davi e Nildo.

Finalmente à todos os que direta e indiretamente me auxiliaram nesta conquista.

RESUMO

Esta pesquisa trata do Problema do Caixeiro Viajante Suficientemente Próximo, uma variante do Problema do Caixeiro Viajante que possui diversas aplicações em logística. No Problema do Caixeiro Viajante Suficientemente Próximo, ao invés de visitar o próprio vértice (cliente), o caixeiro deve visitar uma região especifica contendo este vértice. Para resolver este problema, é proposto um algoritmo exato, simples e efetivo, baseado em *branch-and-bound* e Programação Cônica de Segunda Ordem. O algoritmo proposto foi testado em 824 instâncias sugeridas na literatura. Soluções ótimas foram obtidas para instâncias com até mil vértices. Foram consideradas instâncias nos espaços bi e tridimensional.

Palavras-chave: problema do caixeiro viajante suficientemente próximo; *branch-and-bound*; programação cônica de segunda ordem; logística.

ABSTRACT

This research deals with the Close-Enough Traveling Salesman Problem, a variant of the Traveling Salesman Problem wich has several applicatios in logistics. In the Close-Enough Traveling Salesman Problem, rather than visiting the vertex (customer) itself, the salesman must visit a specific region containing such vertex. To solve this problem, we propose a simple yet effective exact algorithm, based on Branch-and-Bound and Second Order Cone Programming. The proposed algorithm was tested in 824 instances suggested in the literature. Optimal solutions are obtained for open problems with up to a thousand vertices. We consider both instances in the two- and three-dimensional space.

Keywords: close-enough traveling salesman problem; branch-and-bound; second order cone programming; logistics.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIAÇÕES

AMR	*Automatic Meter Reading*
AP	*Arc-based Partitioning*
BeFS	*Best-First Search*
BFS	*Breadth-First Search*
CEARP	*Close-Enough Arc Routing Problem*
CETSP	*Close-Enough Traveling Salesman Problem*
DFS	*Depth-First Search*
GP	*Grid-based Partitioning*
MINLP	*Mixed-Integer Nonlinear Programming*
MIP	*Mixed-Integer Programming*
ORRP	*Optimal Robot Routing Problem*
QCQP	*Quadratically Constrained Quadratic Programming*
RFID	*Radio-Frequency Identification*
SOCP	*Second Order Cone Programming*
TSP	*Traveling Salesman Problem*
TSPN	*Traveling Salesman Problem With Neighborhoods*
UAV	*Unmanned Aerial Vehicle*

1 INTRODUÇÃO

1.1 DEFINIÇÃO DO TEMA E DO PROBLEMA DE PESQUISA

O Problema do Caixeiro Viajante (TSP, do inglês *Traveling Salesman Problem*) é um dos problemas mais conhecidos no ramo da Pesquisa Operacional. O TSP consiste em determinar um Caminho Hamiltoniano de comprimento mínimo sobre um grafo (orientado ou não) de modo que todos os vértices do grafo sejam visitados pelo caixeiro. De modo semelhante, o Problema do Caixeiro Viajante Euclidiano (ETSP, do inglês *Euclidean Traveling Salesman Problem*) consiste em encontrar um *tour* de comprimento mínimo sobre um conjunto de pontos em um espaço Euclidiano.

O Problema do Caixeiro Viajante Suficientemente Próximo (CETSP, do inglês *Close-Enough Traveling Salesman Problem*) é uma generalização do TSP e pode ser visto, de acordo com Mennell (2009), como um caso particular de três outros problemas relacionados com o TSP: O Problema do Caixeiro Viajante com Vizinhanças (TSPN, do inglês *Traveling Salesman Problem with Neiborhoods*) (Arkin e Hassin, 1994), o Problema do Caixeiro Viajante Generalizado (Silberholz e Golden, 2007) e o *Covering Tour Problem* (Gendreau et al, 1997). O CETSP é um problema $\mathcal{NP}$-*Hard* uma vez que ele generaliza o TSP, para o caso quando $r_i \to 0 \; \forall \; i \in V$, que é um problema conhecidamente $\mathcal{NP}$-*Hard*.

No CETSP, ao invés de visitar a exata localização do vértice como no ETSP, o caixeiro deve visitar uma região específica que contenha tal vértice. Neste trabalho, considera-se as regiões de cobertura como círculos. Esta é uma hipótese clássica na literatura do CETSP. Desta forma, seja V o conjunto de vértices, então um vértice $i \in V = \{1, \ldots, n\}$ é considerado coberto se o caixeiro passar através do disco D_i com raio r_i contendo o vértice

i ou pelo menos tocar na borda deste disco. Em um espaço tridimensional as regiões são consideradas como esferas e, de modo semelhante, os vértices são considerados cobertos se o caixeiro passa através de ou pelo menos toca a fronteira de suas respectivas esferas. A Figura 1.1 ilustra um exemplo de uma solução viável para o CETSP em um espaço bidimensional, de modo análogo a Figura 1.2 mostra uma solução viável para o CETSP no espaço tridimensional.

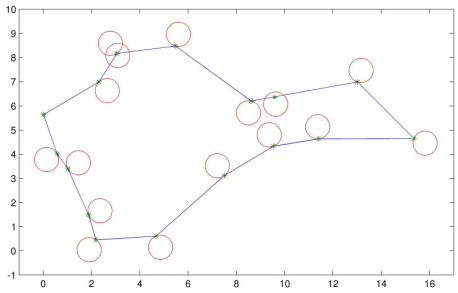

Figura 1.1: Solução Viável para o CETSP no $\mathbb{R}^2$

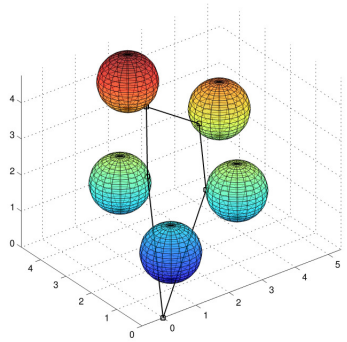

Figura 1.2: Solução Viável para o CETSP no $\mathbb{R}^3$

Diversas heurísticas foram desenvolvidas para encontrar soluções sub-ótimas para o CETSP. Em seu trabalho recente, Behdani e Smith (2013) destaca que existe uma carência de algoritmos exatos para a solução do CETSP. Mennell et al (2011) destaca ainda que mesmo desenvolver bons *lower bounds* para o CETSP é uma tarefa não trivial. Até onde se sabe este trabalho é o primeiro a propor um método que produz soluções ótimas para o CETSP. Mais especificamente, propõe-se um algoritmo *branch-and-bound* combinatório, simples e ao mesmo tempo efetivo, no qual o subproblema é baseado em uma formulação de Programação Cônica de Segunda Ordem (SOCP, do inglês *Second-Order Cone Programming*) (Lobo et al, 1998; Alizadeh e Goldfarb, 2003; Boyd e Vandenberghe, 2004), capaz de resolver instancias com até mil vértices, dependendo de suas características, em espaços Euclidianos de duas ou três dimensões.

1.2 JUSTIFICATIVA

O TSP é um problema clássico da literatura que possui aplicações diretas na Engenharia de Produção, como por exemplo em logística. O CETSP também possui diversas aplica-

ções em problemas reais. Por exemplo, através do uso de etiquetas de Identificação Por Radiofrequência (RFID, do inglês *Radio Frequency Identification*) conectadas a medidores físicos pode-se codificar o número do medidor e sua respectiva leitura em sinais digitais. Desta forma, um veículo equipado com um Sistema de Leitura Automática (AMR, do inglês *Automatic Meter Reading*) poderá coletar os dados dos medidores a partir de uma certa distância. Portanto, em um contexto de AMR, o veículo não necessita visitar a exata localização de cada cliente, mas apenas passar a uma certa distância (raio). (Gulczynski et al, 2006; Dong et al, 2007).

Veículos Aéreos Não Tripulados (UAVs, do inglês do inglês *Unmanned Aerial Vehicles*), também conhecidos como *drones*, são aeronaves que não precisam de uma tripulação a bordo. UAVs podem ser controlados através de comandos automatizados a partir de um computador remoto e/ou por um controle remoto operado por um piloto à distância. O propósito principal para o uso de UAVs veio a partir de operações militares especiais, onde a presença de uma tripulação humana torna-se muito perigosa para ser considerada.

No entanto, UAVs não são usados apenas em operações militares. Diversas aplicações civís surgem a cada dia, como exemplos do uso de UAVs pode se citar operações de reconhecimento aéreo, detecção aérea de incêndios, rastreamento de embarcações em alto-mar, entrega de suprimentos ou mercadorias, monitoramento geográfico, vigilância de tubulações, pesquisas meteorológicas, dentre outras. Se o UAV é equipado com um sensor, este veículo será capaz de operar a partir de uma certa distância do seu alvo. Outro exemplo ocorre quando o UAV necessita apenas "soltar" sua carga tão próximo quanto possível da localização do alvo, como em operações militares. Estes são exemplos de aplicações que podem ser modeladas como um CETSP.

A maioria dos trabalhos relacionados ao CETSP apresentam algoritmos capazes de fornecer soluções sub-ótimas em tempo computacional competitivo. Por outro lado, apenas uma abordagem exata pode ser encontrada na literatura até então. Recentemente, Behdani e Smith (2013) apresentaram uma abordagem exata baseada em dois esquemas de discretização. Em seu trabalho, os autores apresentam limites inferiores e superiores para a solução ótima do CETSP. Os experimentos são conduzidos em instancias com até 21 vértices.

Tabela 1.1: Algoritmos para o CETSP e problemas relacionados disponíveis na literatura

Autor	**Ano**	**Abordagem**
Dumitrescu e Mitchell	2003	Algoritmos aproximativos *constant-factor* e PTAS
Gulczynski et. al.	2006	Diversas Heurísticas
Dong et.al.	2007	Heurísticas baseadas em clusterização e na envoltória convexa
Yuan et. al.	2007	Heurísticas para Otimização Global
Shuttleworth et. al.	2008	Diversas Heurísticas
Mennell et. al.	2011	Heurística baseada em Zonas de Steiner
Behdani e Smith	2013	Discretização + Programação Inteira

De acordo com a Tabela 1.1, pode-se observar a carência de algoritmos exatos capazes de fornecer soluções ótimas para o CETSP. Estes trabalhos serão discutidos com mais detalhes na revisão da literatura. Devido à relevância e aplicabilidade deste problema em casos reais e a abordagem proposta para resolução, justifica-se a realização desta pesquisa.

1.3 OBJETIVOS

1.3.1 Objetivo Geral

Propor um algoritmo *branch-and-bound* combinatório para o Problema do Caixeiro Viajante Suficientemente Próximo.

1.3.2 Objetivos Específicos

- Realizar um levantamento da literatura sobre o CETSP.
- Implementar a formulação do sub-problema que será resolvido à cada nó durante a execução do *branch-and-bound*.
- Selecionar uma estratégia para construir uma relaxação inicial para o CETSP.
- Desenvolver um método para checagem da viabilidade das soluções geradas durante a execução do algoritmo.
- Definir uma ou mais regras de *branching* para o *branch-and-bound*.
- Selecionar uma estratégia de *branching* para o *branch-and-bound*.
- Testar e validar o algoritmo proposto em instâncias da literatura.
- Comparar os resultados obtidos com outros métodos disponíveis na literatura.

1.4 ESTRUTURA DO TRABALHO

O restante deste trabalho está organizado da seguinte maneira:

- No Capítulo 2 apresenta-se uma revisão da literatura envolvendo o CETSP e conceitos básicos envolvendo métodos *branch-and-bound* e SOCP
- No Capítulo 3 apresenta-se o algoritmo *branch-and-bound*.
- No Capítulo 4 apresenta-se os resultados dos experimentos computacionais realizados.
- No Capítulo 5 apresenta-se as conclusões e trabalhos futuros da pesquisa.

2 REVISÃO BIBLIOGRÁFICA E FUNDAMENTAÇÃO TEÓRICA

2.1 PROBLEMA DO CAIXEIRO VIAJANTE SUFICIENTEMENTE PRÓXIMO

2.1.1 Definição Formal

O CETSP pode ser formalmente descrito da seguinte maneira. Considere um conjunto de pontos $V = \{0, \ldots, n\}$ no espaço bi-dimensional com coordenadas $(\bar{x}_i, \bar{y}_i),\ i = 0, \ldots, n$. Cada vértice i é envolto por um círculo D_i de raio r_i. Assume-se ainda que $(\bar{x}_i, \bar{y}_i) \neq (\bar{x}_j, \bar{y}_j),\ \forall\ i, j \in V,\ i \neq j$, ou seja, não há sobreposição entre dois vértices distintos. O problema consiste em determinar o valor das coordenadas dos *hitting points* $(x_i, y_i) \in \mathbb{R}^2, i = 0, \ldots, n$, também conhecidos por *representative points* (Mennell, 2009), e uma sequência $\mathcal{S} = \{k_0, k_1, \ldots, k_n\}$ de tal forma que o *tour* sobre estas coordenadas forme um Ciclo Hamiltoniano de comprimento mínimo e $(x_i, y_i) \in \mathcal{D}_i,\ \forall\ i \in V$.

No exemplo da figura 2.1 o *tour* toca o circulo associado ao vértice 1 e 3 apenas uma vez, enquanto que no vértice 2 o tour "corta"o círculo associado a esse vértice infinitas vezes. Neste trabalho denota-se o vértice $i = 0$ como o depósito e assume-se que $\mathcal{D}_0$ é um círculo com $r_0 = 0$. Esta definição pode ser facilmente estendida para o caso tridimensional utilizando-se três coordenadas para representar os vértices, como $(\bar{x}_i, \bar{y}_i, \bar{z}_i)$, ao invés de duas. A Figura 2.2 ilustra uma solução viável para o CETSP.

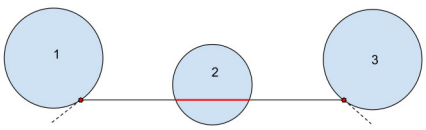

Figura 2.1: Ilustração do conceito de *hitting points*

Quando existe apenas um único *hitting point* associado a um vértice, então este ponto é denominado um *turn point*. Behdani e Smith (2013) demonstram que o número total de *turn points* em uma solução viável para o CETSP pode ser menor ou igual a n.

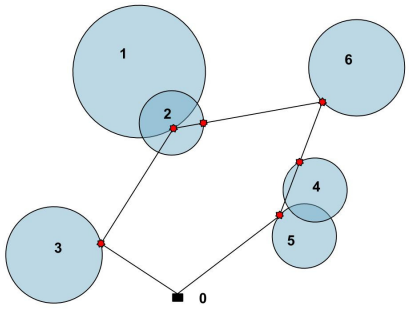

Figura 2.2: Ilustração de uma solução viável para o problema do caixeiro viajante suficientemente próximo

2.1.2 Trabalhos Relacionados

Com o objetivo de obter soluções aproximadas, diversas heurísticas foram desenvolvidas para o CETSP. Gulczynski et al (2006) and Dong et al (2007) propuseram diversos métodos heurísticos para o caso quando todos os discos possuem o mesmo raio. Ambos os métodos foram baseados no conceito de "super nós". Um conjunto viável S de super nós é definido como um conjunto de pontos no $\mathbb{R}^2$, incluindo o depósito, tal que cada vértice, $v_i \in V$, $i = 1, ..., n$, está a pelo menos r unidades distante de qualquer ponto em S. As heurísticas foram desenvolvidas em três fases distintas: (i) geração do conjunto de super nós; (ii) construção de um *tour* sobre o conjunto de super nós; e por fim, movimentação dos super nós na tentativa de melhoria do *tour*. Embora estas heurísticas compartilhem a mesma estrutura, os procedimentos adotados em cada fase diferem uns dos outros nos dois trabalhos.

Yuan et al (2007) trata o Problema de Roteamento de Robôs (ORRP, do inglês *Optimal Robot Routing Problem*) como um TSPN no qual os conjuntos compactos que cobrem os vértices são considerados círculos disjuntos com um determinado raio. O ORRP consiste em determinar uma rota ótima para um robô móvel operando em uma rede de sensores sem fio de tal modo que este robô possa coletar dados de todos os sensores minimizando a distancia total percorrida. Portanto, este problema pode ser modelado como um CETSP

no qual os discos representando as áreas de ação de cada sensor são disjuntos. Os autores propuseram um algoritmo em duas fases para o TSPN, decompondo o problema original em um problema combinatório e um problema contínuo. O primeiro com o objetivo de determinar um *tour* ótimo sobre o conjunto de vértices original utilizando um algoritmo exato. Enquanto que o segundo problema consiste em encontrar as cordenadas dos *hitting points* que minimizam o comprimento do *tour* gerado pela primeira fase utilizando um algoritmo genético.

Baseado nos resultados preliminares de Mennell (2009), Mennell et al (2011) apresenta um heurística denominada *Steiner Zone Heuristic* baseada em três fases. A primeira fase é chamada Redução do Grafo, que o número total de vértices é reduzido a um número menor de Zonas de Steiner. Uma Zona de Steiner é definida como a interseção entre uma ou mais circunferências cujo grau é determinado pelo número de circunferências que se interceptam, a Figura 2.3 mostra um exemplo com 3 vértices, 3 Zonas de Steiner de grau 1, três de grau 2 e uma de grau 3. A segunda fase é chamada *Tour Finding*, onde um *tour* do TSP é construído sobre um conjunto de super nós selecionados arbitrariamente de cada Zona de Steiner. Na última fase, chamada *Tour Improvement*, métodos exatos e/ou heurísticos são aplicados no sentido de redução do comprimento do *tour*.

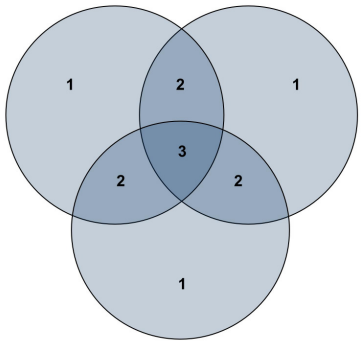

Figura 2.3: Todas as possíveis Zonas de Steiner para três vértices que se interceptam

Mennell (2009) apresenta uma formulação baseada em Programação Não-Linear Inteira Mista (MINLP, do inglês*Mixed Integer Nonlinear Programming*) para o CETSP. As variáveis de decisão são as coordenadas dos *hitting points* $(x_i, y_i) \in \mathbb{R}^2$ e as variáveis $e_{ij} \in \{0, 1\}\ \forall\ i, j = 0, \ldots, n$, onde $e_{ij} = 1$ se o ponto i precede o ponto j na sequência e $e_{ij} = 0$ caso contrário. Os dados do problema são as posições dos vértices $(\bar{x}_i, \bar{y}_i),\ i = 0, \ldots, n$

e seus respectivos raios, denotados por $r_i, i = 0, \ldots, n$, e um número muito pequeno, denotado por $\epsilon \in \mathbb{R}$, representando um limite inferior para o comprimento de cada distância entre dois *hitting points*.

No modelo representado pelas equações (2.1 - 2.8) a função objetivo minimiza o somatório das distâncias entre os pontos pertencentes ao ciclo. As restrições (2.2) garantem que todos os pontos sejam cobertos, as restrições (2.3) limitam o tamanho mínimo de cada aresta ao número ϵ, as restrições de grau (2.4 - 2.5) e de *subtour* (2.6) são as provenientes do TSP clássico, por fim as expressões (2.7) e (2.8) definem os *bounds* sobre as variáveis. O autor relata que as restrições (2.3) foram adicionadas ao modelo com o intuito de melhorar o desempenho computacional do *solver* de MINLP utilizado.

$$\min \quad \sum_{i=0}^{n}\sum_{j=0}^{n} e_{ij}\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{2.1}$$

$$\text{s.a} \quad (x_i - \bar{x}_i)^2 + (y_i - \bar{y}_i)^2 \leq r_i^2, \qquad i = 1, \ldots, n. \tag{2.2}$$

$$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \geq \epsilon \qquad i, j = 0, \ldots, n. \tag{2.3}$$

$$\sum_{i=0}^{n} e_{ij} = 1 \qquad j = 0, \ldots, n. \tag{2.4}$$

$$\sum_{j=0}^{n} e_{ij} = 1 \qquad i = 0, \ldots, n. \tag{2.5}$$

$$\sum_{i,j \in S'}^{n} e_{ij} \leq |S'| - 1 \qquad \forall\, S' \subset \{0, \ldots, n\}, 2 \leq |S'| \leq n. \tag{2.6}$$

$$e_{ij} \in \{0, 1\}\ i, j = 0, \ldots, n. \tag{2.7}$$

$$(x_i, y_i) \text{ livres } \ i, j = 0, \ldots, n. \tag{2.8}$$

Apesar da existência de métodos na literatura e também *solvers* disponíveis para resolver problemas de programação inteira não linear, tentar resolver este modelo pode tornar-se impraticável, para instâncias não triviais, do ponto de vista computacional.

Behdani e Smith (2013) propuseram dois diferentes esquemas de discretização para aproximar as regiões de cobertura contínuas. A primeira foi denominada *Grid-based Partitioning* (GP) e aproxima regiões de corbetura de forma arbitrária por retângulos. A segunda foi denominada *Arc-based Partitioning* (AP), útil apenas para regiões circulares,

e discretiza a borda dos circulos em um número finito de possíveis *hitting points*. As Figuras 2.4 e 2.5 mostram exemplos dos dois tipos de partição definidos. As definições a seguir fornecem suporte para os modelos de Programação Inteira Mista (MIP, do inglês *Mixed Integer Programming*) apresentados pelos autores.

De maneira formal, Behdani e Smith (2013) fornecem a Definição 2.1.1 para um conjunto de partições. Denota-se i_0 como $\{i \in V | i = 0\}$ e $conv(.)$ como a envoltória convexa de um conjunto:

Definição 2.1.1. *Um conjunto* $C = \{C_0, \ldots, C_p\}$ *é chamado uma partição do plano bidimensional para o CETSP se:*

- $C_0 = i_0$
- C_i *é um conjunto compacto não vazio para todo* $1 \leq i, j \leq p$
- $conv(C_i \cap C_j)$ *é vazia para todo* $1 \leq i, j \leq p$
- $(\cup_{i=1}^{p} C_i) \supseteq (\cup_{v \in V} B_v)$*, onde é a borda do conjunto* D_v*.*

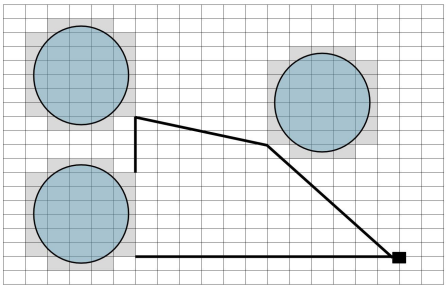

Figura 2.4: Ilustração do GP e uma solução viável para essa relaxação

Cada célula no primeiro esquema de particionamento pode ser representada pela tupla $(a_i, b_i, w_{1i}, w_{2i})$, onde (a_i, b_i) denotam vértice inferior esquerdo e w_{1i} e w_{2i} a altura e largura, respectivamente. Desse modo o vértice superior direito é representado por $(a_i + w_{1i}, b_i + w_{2i})$. No segundo esquema de particionamento um arco é representado por $(a_i, b_i, r_i, \alpha_{1i}, \alpha_{2i})$, onde (a_i, b_i) são os centros das circunferências de raio r_i, as quantidades α_{1i} e α_{2i} representam os ângulos inicial e final de cada arco.

A distância entre duas partições distintas é denotada por l_{ij}. O cálculo de l_{ij} é baseado na Definição 2.1.2:

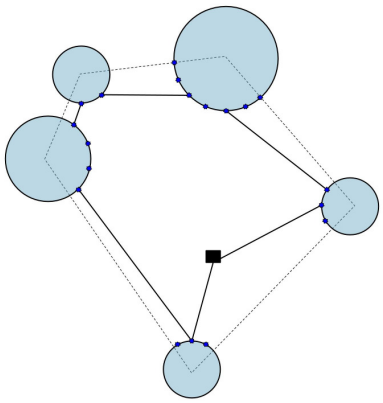

Figura 2.5: Ilustração do AP e uma solução viável para essa relaxação

Definição 2.1.2. *Duas células C_i e C_j são denominadas regularmente localizadas se ambas condições valem:*

- *(i)* $a_i + w_{1i} \leq a_j$ *ou (ii)* $a_i \geq a_j + w_{1j}$*;*
- *(iii)* $b_i + w_{2i} \leq b_j$ *ou (iv)* $b_i \geq b_j + w_{2j}$*;*

A Figura 2.6 mostra um exemplo de como a posição entre duas células distintas definem se elas são *regularmente localizadas*. Neste caso, uma célula j não obedeceria a Definição 2.1.2 se esta estiver localizada nas regiões I ou II (hachuradas). Se $a_i + w_{1i} < a_j$ ou $a_i < a_j + w_{1j}$ então $l_{ij} = max\{b_i - b_j - w_{2j}, b_j - b_i - w_{2i}\}$. Ou se $b_i + w_{2i} > b_j$ ou $b_i < b_j + w_{2j}$ então $l_{ij} = max\{a_i - a_j - w_{1j}, a_j - a_i - w_{1i}\}$.

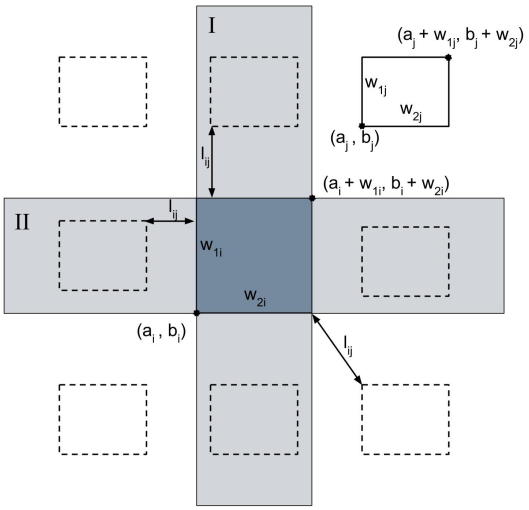

Figura 2.6: Ilustração mostrando as possíveis posições entre duas células no caso do GP

Por outro lado, se duas células C_i e C_j obedecem a Definição 2.1.2, pode-se mostrar que a distancia entre estas duas células é dada pelo comprimento do menor segmento de linha que liga dois vértices distintos pertencentes a elas. No caso do AP, esta distância pode ser previamente calculada como a menor distância entre dois vértices $l_{ij} = c_{ij}^{\min} = \|(\bar{x_j}, \bar{y_j}) - (\bar{x_i}, \bar{y_i})\| - (r_j + r_i)$, onde $i, j \in V$, e armazenada em uma matriz bidimensional.

Apresenta-se aqui um modelo baseado em MIP proposto por Behdani e Smith (2013) que fornece um *lower bound* para uma solução ótima do CETSP (Equações (2.9 - 2.15)). Considere $C = \{C_0, \ldots, C_p\}$ uma partição do plano bidimensional para o CETSP e $C^{\min} = \{l_{ij}\}$. O conjunto $N(v) = \{1 \leq i \leq p | C_i \cap S_v \neq \emptyset\}$. As variáveis de decisão são $x_{ij} \in \{0, 1\}$, que é igual 1 se C_i precede C_j no *tour* e y_i que indica se C_i é visitado ou não.

$$\min \quad \sum_{i=0}^{p} \sum_{j=0}^{p} l_{ij} x_{ij} \tag{2.9}$$

$$\text{s.a} \quad \sum_{j=0}^{p} x_{ij} = \sum_{j=0}^{p} x_{ji}, \qquad i = 0, \ldots, p. \tag{2.10}$$

$$y_i = \sum_{j=0}^{p} x_{ij}, \qquad i = 0, \ldots, p. \tag{2.11}$$

$$\sum_{i \in N(v)} y_i \geq 1, \qquad \forall\, v \in V \tag{2.12}$$

$$\sum_{i \in S'} \sum_{j \notin S'} x_{ij} \geq y_u, \qquad \forall\, S' \subset \{1, \ldots, p\} | 2 \leq |S'| \leq |C| - 2 \text{ e } u \in S' \tag{2.13}$$

$$x_{ij} \in \{0, 1\} \forall\, i, j = 0, \ldots, p. \tag{2.14}$$

$$0 \leq y_i \leq, \forall\, i = 1, \ldots, p,\ y_0 = 1. \tag{2.15}$$

Neste modelo, a função objetivo linear (2.9) minimiza a distância total percorrida no *tour*. As restrições (2.10) obrigam que o número total de arestas entrando em uma célula restrições seja igual ao número total de arestas que saem dessa célula. As restrições (2.11) fazem o *link* entre as variáveis de decisão do modelo. As restrições (2.12) garantem que todos os vértices serão atendidos. As restrições (2.13) são restrições tradicionais de *subtour*. Por fim, as restrições (2.14) e (2.15) definem os *bounds* sobre as variáveis.

Além deste os autores também desenvolveram outros dois modelos, bem mais complexos, também baseados em MIP e uma Decomposição de Benders. Como resultado

Behdani e Smith (2013) conseguiram fornecer *lower bounds* e *upper bounds* para instancias com no máximo 21 vértices. A principal característica dessas abordagens é que quanto maior o tamanho do conjunto C melhor a qualidade dos *bounds* e consequentemente as soluções aproximam-se mais da solução ótima para o CETSP. Por outro lado, altos níveis de particionamento tornam o método computacionalmente proibitivo.

Recentemente Hà et al (2013) estudou uma variante para o CETSP chamada Problema de Roteamento de Arcos Suficientemente Próximo (CEARP, do inglês Close Enough Arc Routing Problem). No CEARP considera-se um grafo direcionado predefinido $G = (V, A)$, onde $V = \{v_0, v_1, \ldots, v_{n-1}\}$ é o conjunto de vértices e $A = \{(v_i, v_j) \mid v_i, v_j \in V\}$ o conjunto de arcos que conectam estes vértices, e outro conjunto de vértices no $\mathbb{R}^2$, representando os clientes que devem ser cobertos denotado por $W = \{w_1, w_2, \ldots, w_l\}$. O CEARP consiste em encontrar um ciclo Hamiltoniano de custo mínimo sobre o grafo G tal que todos os clientes em W sejam cobertos, ou seja, estejam a uma certa distância de qualquer arco pertencente ao ciclo. Este problema se encaixa perfeitamente no contexto de AMR, e foi originalmente chamado *CETSP over a street network* por Shuttleworth et al (2008), uma vez que em sua abordagem os arcos do grafo representaram o conjunto de ruas.

2.2 CONCEITOS BÁSICOS SOBRE ALGORITMOS *BRANCH-AND-BOUND*

Os algoritmos *branch-and-bound* têm sua origem no trabalho de Land e Doig (1960) que propuseram um algoritmo para resolver problemas de programação inteira. Já o termo *branch-and-bound* foi introduzido por Little et al (1963), quando em seu trabalho os autores apresentaram um algoritmo para o TSP capaz de resolver instâncias com até 40 vértices.

As definições a seguir podem ser encontradas em Wolsey (1998). Um algoritmo *branch-and-bound* nada mais é do que uma enumeração implícita de todas as possíveis soluções para um problema de natureza combinatória. O uso de *bounds* para a função a ser otimizada combinado com o valor da melhor solução corrente é o que permite ao algoritmo apenas analisar algumas partes do espaço de soluções.

Genericamente um algoritmo *branch-and-bound* funciona a partir da subdivisão do

espaço de busca do problema original em subproblemas ("nós"), o *status* de uma solução corrente em relação ao espaço de soluções é descrito por um *pool* de soluções ainda não exploradas ("nós abertos") e a solução incumbente, que é a melhor solução encontrada até então. Inicialmente, assumindo-se o caso de minimização, existe apenas um subconjunto de soluções, que é o espaço inteiro, e a melhor solução até então tem custo igual a ∞. Os nós são gerados dinamicamente durante a execução do algoritmo. Uma iteração do algoritmo tem, de modo geral, três fases: Seleção do nó à explorar, cálculo dos *bounds*, e o *branching*.

A árvore é desenvolvida dinamicamente durante a busca e consiste inicialmente do "nó raiz". Deve-se destacar a importância de uma boa solução inicial viável, muitas vezes produzida por uma heurística, para inicializar o custo da melhor solução até então. A cada iteração do algoritmo, um nó é escolhido do *pool* de nós abertos, segundo uma estratégia de *branching* pré-estabelecida. A busca em largura (BFS - *Breadth-first Search*) pesquisa os nós na ordem em que foram criados, ou seja, realiza a busca nível após nível na árvore. A busca em profundidade (DFS - *Depth-first Search*) pesquisa os nós mais profundos primeiro. Por fim, a busca pelo melhor *bound* (BeFS - *Best-first Search*) pesquisa os nós na ordem do menor *bound* existente.

Se o algoritmo for "ansioso" (do inglês - *Eager branch-and-bound*), então um *branching* (ramificação do nó) é realizado no nó corrente. O *branching* consiste em criar dois ou mais nós filhos através da adição de restrições ao subproblema do nó pai, dessa forma o subespaço de soluções é dividido em subespaços menores. Um *lower bound* é então calculado para cada filho através da solução de seus respectivos subproblemas. Caso esta solução seja viável seu valor é comparado com o valor da solução incumbente. Se ele exceder o valor da solução incumbente o nó é descartado ("podado"), uma vez que nenhuma solução viável pode ser melhor que a incumbente. Se a solução é inviável e seu valor é menor do que o valor da solução incumbente então o nó filho correspondente é adicionado ao *pool* de nós abertos.

Se a estratégia de seleção "preguiçosa" é utilizada (do inglês - *Lazy branch-and-bound*) então a ordem entre calcular os *bounds* e fazer o *branching* se altera.

2.3 PROGRAMAÇÃO CÔNICA DE SEGUNDA ORDEM

2.3.1 Forma Geral

Os conceitos aqui apresentados foram baseados nos trabalhos de Lobo et al (1998) e Alizadeh e Goldfarb (2003) e serão úteis para o entendimento do método de resolução do subproblema associado ao algoritmo *branch-and-bound* proposto neste trabalho. O subproblema consiste em determinar a solução ótima do CETSP dada uma sequência parcial formada apenas por um subconjunto de vértices da instância original.

Em um SOCP na forma primal padrão uma função linear é minimizada, sujeita à restrições formadas pelo produto cartesiano de cones de segunda ordem. Tal problema pode ser definido pelas equações (2.16 - 2.18):

$$\min \quad \boldsymbol{f}^\top \boldsymbol{x} \tag{2.16}$$

$$\text{s.a} \quad \|A_i\boldsymbol{x} + \boldsymbol{b}_i\| \leq \boldsymbol{c}_i^\top \boldsymbol{x} + d_i, \quad i = 1, ..., m \tag{2.17}$$

$$F\boldsymbol{x} = \boldsymbol{g} \tag{2.18}$$

Em que $\boldsymbol{x} \in \mathbb{R}^n$ é a variável de decisão e os parâmetros do problema são $\boldsymbol{f} \in \mathbb{R}^n$, $A_i \in \mathbb{R}^{n_i \times n}$, $\boldsymbol{b}_i \in \mathbb{R}^{n_i}$, $\boldsymbol{c}_i \in \mathbb{R}^n$, $d_i \in \mathbb{R}$, $F \in \mathbb{R}^{p \times n}$ e $\boldsymbol{g} \in \mathbb{R}^p$. A restrição (2.17) é chamada de restrição cônica de segunda ordem, uma vez que é requerido que o par $(A_i\boldsymbol{x} + \boldsymbol{b}_i; \boldsymbol{c}_i^\top \boldsymbol{x} + d_i)$ pertença ao cone de segunda ordem de dimensão $n_i + 1, i = 1, \ldots, m$.

Uma vez que a função objetivo (2.16) é linear e as restrições (2.17) e (2.18) são convexas então todo SOCP é convexo.

Restrições cônicas de segunda ordem podem ser usadas para representar várias outras restrições de classe convexa em diversos problemas de otimização. Por exemplo, quando $A_i = 0$, para todo $i = 1, ..., m$, o SOCP é reduzido a um problema de programação linear da forma (2.19-2.21), em que $\alpha_i \in \mathbb{R}$ tal que $\alpha_i = \|\boldsymbol{b}_i\| - d_i$:

$$\min \quad \boldsymbol{f}^\top \boldsymbol{x} \tag{2.19}$$

$$\text{s.a} \quad \boldsymbol{c}_i^\top \boldsymbol{x} \geq \alpha_i \quad i = 1, ..., m \tag{2.20}$$

$$F\boldsymbol{x} = \boldsymbol{g} \tag{2.21}$$

Outro caso especial ocorre quando $\boldsymbol{c}_i = 0 \; \forall \; i = 1, \ldots, m$, na equação (2.17), neste caso a i-ésima restrição cônica de segunda ordem se reduz à $\|A_i\boldsymbol{x} + \boldsymbol{b}_i\| \leq d_i$ que é equivalente à uma restrição quadrática.

O dual do problema definido por (2.16 - 2.18) pode ser escrito como nas equações (2.22 - 2.24):

$$\max \quad \sum_{i=1}^{m} (\boldsymbol{b}_i^\top \boldsymbol{u}_i - d_i v_i) \tag{2.22}$$

$$\text{s.a} \quad \sum_{i=1}^{m} (A_i^\top \boldsymbol{u}_i - \boldsymbol{c}_i v_i) + \boldsymbol{f} = 0 \tag{2.23}$$

$$v_i \geq \|\boldsymbol{u}_i\| \; , \; i = 1, \ldots, m. \tag{2.24}$$

Onde as variáveis de decisão são $\boldsymbol{u}_i \in \mathbb{R}^{n_i}$ e $v_i \in \mathbb{R}, i = 1, \ldots, m$, e os dados do problema são $\boldsymbol{f} \in \mathbb{R}^n, A_i \in \mathbb{R}^{n_i \times n}, \boldsymbol{b}_i \in \mathbb{R}^{n_i}$ e $d_i \in \mathbb{R}, i = 1, \ldots, m$. Observa-se que as restrições cônicas (2.24) trazem implicitamente a definição do cone dual $\mathcal{Q}_n^*$ associado às restrições cônicas (2.17).

Problemas quadráticos de otimização podem ser transformados em problemas cônicos de maneira direta. Os trabalhos de Alizadeh e Goldfarb (2003) e Lobo et al (1998) apresentam a transformação de problemas quadráticos sob restrições lineares. Exemplos de outras classes de problemas que também podem ser reformulados podem ser encontrados em Alizadeh e Goldfarb (2003), Boyd e Vandenberghe (2004) e Alzalg (2012).

2.3.2 Reformulação de problemas quadráticos sob restrições quadráticas

De uma maneira geral Problemas de Otimização Quadrática Sob Restrições Quadráticas (QCQP, do inglês *Quadratically Constrained Quadratic Programming*) podem ser resolvidos como SOCPs. Basta notar que um QCQP pode ser expresso como a minimização de uma função linear sob restrições quadráticas como nas equações (2.25 - 2.26):

$$\min \quad \boldsymbol{c}^\top \boldsymbol{x} \tag{2.25}$$

$$\text{s.a} \quad q_i(\boldsymbol{x}) \stackrel{def}{=} \boldsymbol{x}^\top B_i^\top B_i \boldsymbol{x} + \boldsymbol{a}_i^\top \boldsymbol{x} + \beta_i \leq 0, \; \forall \; i = 1, \ldots, m. \tag{2.26}$$

Em que $\boldsymbol{x} \in \mathbb{R}^n$ é a variável de decisão e os parâmetros do problema são $\boldsymbol{c} \in \mathbb{R}^n$, $B_i \in$

$\mathbb{R}^{n\times n}$ tal que $B_i \succeq 0$, $\boldsymbol{a}_i \in \mathbb{R}^n$, $\boldsymbol{c}_i \in \mathbb{R}^n$, $d_i \in \mathbb{R}$, $F \in \mathbb{R}^{p\times n}$ e $\boldsymbol{g} \in \mathbb{R}^p$.

Desta forma, o modelo pode ser expresso pelo SOCP descrito pelas equações (2.27 - 2.30):

$$\min \quad \boldsymbol{c}^\top \boldsymbol{x} \tag{2.27}$$

$$\text{s.a} \quad \|\bar{\boldsymbol{u}}_i\| \leq u_{0i},\ \forall\, i = 1, \ldots, m \tag{2.28}$$

$$\bar{\boldsymbol{u}}_i = \left(B_i\boldsymbol{x};\ \frac{\boldsymbol{a}_i^\top \boldsymbol{x} + \beta_i + 1}{2}\right),\ \forall\, i = 1, \ldots, m \tag{2.29}$$

$$u_{0i} = \frac{1 - \boldsymbol{a}_i^\top \boldsymbol{x} - \beta_i}{2},\ \forall\, i = 1, \ldots, m \tag{2.30}$$

Uma verificação simples dessa reformulação pode ser realizada da seguinte maneira:

$$\|\bar{\boldsymbol{u}}\| \leq u_0 \iff \tag{2.31}$$

$$\left(B\boldsymbol{x};\ \frac{\boldsymbol{a}^\top \boldsymbol{x} + \beta + 1}{2}\right)^\top \left(B\boldsymbol{x};\ \frac{\boldsymbol{a}^\top \boldsymbol{x} + \beta + 1}{2}\right) \leq \left(\frac{1 - \boldsymbol{a}^\top \boldsymbol{x} - \beta}{2}\right)^2 \iff \tag{2.32}$$

$$\boldsymbol{x}^\top B^\top B\boldsymbol{x} + \left(\frac{\boldsymbol{a}^\top \boldsymbol{x} + \beta + 1}{2}\right)^2 \leq \left(\frac{1 - \boldsymbol{a}^\top \boldsymbol{x} - \beta}{2}\right)^2 \iff \tag{2.33}$$

$$\boldsymbol{x}^\top B^\top B\boldsymbol{x} + \left(\frac{(\boldsymbol{a}^\top \boldsymbol{x} + \beta + 1)^2 + (1 - \boldsymbol{a}^\top \boldsymbol{x} - \beta)^2}{4}\right) \leq 0 \iff \tag{2.34}$$

$$\boldsymbol{x}^\top B^\top B\boldsymbol{x} + \frac{4\boldsymbol{a}^\top \boldsymbol{x} + 4\beta}{4} \leq 0 \iff \tag{2.35}$$

$$\boldsymbol{x}^\top B^\top B\boldsymbol{x} + \boldsymbol{a}^\top \boldsymbol{x} + \beta \leq 0 \tag{2.36}$$

3 UM ALGORITMO *BRANCH-AND-BOUND* PARA O CETSP

3.1 DESCRIÇÃO DO MÉTODO

A ideia geral do método é dividir o problema original de determinar uma sequência $\mathcal{S} = \{k_0, k_1, \ldots, k_n\}$ e o valor das coordenadas $(x_i, y_i) \in \mathbb{R}^2, i = 0, \ldots, n$, em dois subproblemas tratados de maneira diferente. O problema principal para o *branch-and-bound* será o de determinar uma sequência ótima para um subconjunto de vértices da instância original, este problema é claramente combinatório. O subproblema resolvido a cada nó da árvore será o de determinar o valor das coordenadas dos *hitting points*, este por sua vez é um problema contínuo.

De modo geral o método foi desenvolvido da seguinte maneira. Cada nó da árvore estará associado a um *tour* parcial ótimo que visita apenas um subconjunto de vértices em uma dada sequência. No nó raiz, o algoritmo escolhe três vértices para gerar uma sequência inicial (Seção §3.3). Uma vez que existem apenas três vértices envolvidos e os custos são simétricos, a ordem desta sequência não irá afetar o custo da solução do nó raiz. Portanto, este *tour* parcial é uma relaxação válida do problema original. O problema de encontrar o valor das coordenadas dos *hitting points*, dada uma predeterminada ordem, pode ser formulado como um SOCP (Seção §3.2). Se a solução associada ao nó raiz for viável, ou seja, todos os vértices forem cobertos, então esta solução será ótima e o problema estará resolvido. Caso contrário, o algoritmo realiza o *branching* em três subproblemas, onde em cada um deles, um vértice que não pertence ao *tour* é inserido em diferentes posições (Seção §3.5). Um determinado nó da árvore é podado se o valor de seu *lower bound* for

maior ou igual ao melhor *upper bound* ou se sua solução for viável. Caso contrário, o algoritmo realiza o *branching* sobre este nó, utilizando a mesma lógica aplicada ao nó raiz.

A figura 3.1 mostra uma ilustração do método para uma instância com 7 vértices. O conjunto de vértices não cobertos é representado por uma lista ao lado de seus respectivos nós, os números marcados em vermelho representam os vértices escolhidos para formar a próxima subsequência. Neste caso, a relaxação encontrada no nó raiz foi $0 \rightarrow 3 \rightarrow 6 \rightarrow 0$. No próximo passo, o vértice 1 foi selecionado para ser inserido em cada posição possível na sequência do nó pai, resultando em três nós filho. Quando o vértice 1 é inserido na primeira ou segunda posição do *tour*, pode-se verificar que o nó 4 é coberto, no entanto o mesmo não acontece quando 1 é inserido na terceira posição. Este exemplo particular, mostra uma possível árvore associada a essa instância com 7 vértices. Podas são desconsideradas para simplificar o exemplo. Como indicado na Figura 3.1, a solução ótima é $0 \rightarrow 3 \rightarrow 1 \rightarrow 6 \rightarrow 5 \rightarrow 0$.

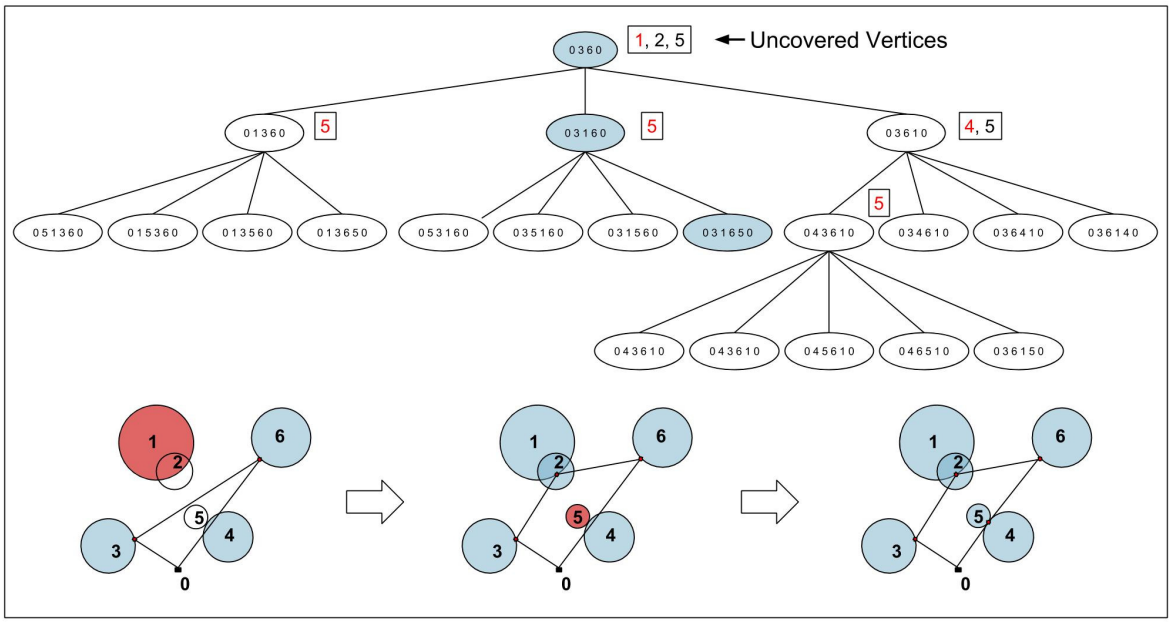

Figura 3.1: Exemplo do algoritmo proposto para uma instância com 7 vértices

O número máximo de nós em um dado nível da árvore é dado por $\frac{(l+2)!}{2}$. No pior caso, o número total de nós corresponde a $\sum_{l=0}^{n} \frac{(l+2)!}{2} = \mathcal{O}(n+2)!$.

3.2 SUBPROBLEMA CÔNICO DE SEGUNDA ORDEM

Tome $\mathcal{S} = \{k_0, k_1, \ldots, k_q\},\ q < n$, como qualquer subsequência encontrada durante a execução do algoritmo. Os subproblemas do *branch-and-bound* consistem em encontrar os valores das coordenadas dos *hitting points*, $(x_{k_j}, y_{k_j}), i \in \mathcal{S}, j = 0, \ldots, q$, de modo que o comprimento deste *tour* parcial seja minimizado. Os dados do problema são r_k e $(\bar{x}_k, \bar{y}_k), k \in \mathcal{S}$, que são os valores dos raios e as coordenadas de cada vértice pertencente a sequência parcial $\mathcal{S}$:

$$\min \quad \sum_{j=1}^{n} \left(\sqrt{(x_{k_{j-1}} - x_{k_j})^2 + (y_{k_{j-1}} - y_{k_j})^2} \right) \tag{3.1}$$

$$+ \sqrt{(x_{k_q} - x_{k_0})^2 + (y_{k_q} - y_{k_0})^2}$$

$$\text{s.a} \quad (x_j - \bar{x}_j)^2 + (y_j - \bar{y}_j)^2 \leq r_j^2, \qquad j = 1, \ldots, q. \tag{3.2}$$

$$(x_{k_j}, y_{k_j)}, k \in \mathcal{S}, j = 1, \ldots, q, \text{ livres} \tag{3.3}$$

Neste modelo, a função objetivo (3.1) minimiza a distância total percorrida. As restrições (3.2) não permitem que os raios dos vértices $j = 1, \ldots, q$, sejam violados e por fim, as restrições (3.3) definem os *bounds* sobre as variáveis.

Como visto na Seção §2.3, este subproblema pode ser convertido em um SOCP utilizando o mesmo raciocínio. O SOCP definido pelas Equações (3.4 - 3.12) segue a formulação apresentada por Mennell (2009). Logo, assumindo que $i_{-1} = i_q$:

$$\min \quad \sum_{k=0}^{q} z_k \tag{3.4}$$

$$\text{s.t.} \quad w_k = x_{i_{k-1}} - x_{i_k}, \qquad k = 0, \ldots, q. \tag{3.5}$$

$$u_k = y_{i_{k-1}} - y_{i_k}, \qquad k = 0, \ldots, q. \tag{3.6}$$

$$s_k = \bar{x}_k - x_k, \qquad k = 0, \ldots, q. \tag{3.7}$$

$$t_k = \bar{y}_k - y_k, \qquad k = 0, \ldots, q. \tag{3.8}$$

$$z_k^2 \geq w_k^2 + u_k^2, \qquad k = 0, \ldots, q. \tag{3.9}$$

$$s_k^2 + t_k^2 \leq r_k^2, \qquad k = 0, \ldots, q. \tag{3.10}$$

$$z_k \geq 0; w_k, u_k, s_k, t_k \text{ livres} \qquad k = 0, \ldots, q. \tag{3.11}$$

$$x_i, y_i \text{ livres} \quad i = 0, \ldots, q. \tag{3.12}$$

Nesta formulação a função objetivo (3.4) é linear. As variáveis $z_k, k = 0, \ldots, q$, representam a distância entre vértices subsequentes na sequência parcial $\mathcal{S}$. As restrições cônicas (3.9) definem o comprimento da aresta que conecta vértices subsequentes da sequência $\mathcal{S}$. As restrições cônicas (3.10) garantem que cada *hitting point* não ultrapassará a borda da circunferência que cobre seu respectivo vértice. As variáveis auxiliares w, u, s e t são definidas nas restrições lineares (3.5 -3.10). As expressões (3.11) e (3.12) definem os limites nas variáveis.

Sabe-se que SOCPs podem ser resolvidos em tempo polinomial (Andersen et al, 2003). Além do mais, alguns conhecidos softwares de otimização como CPLEX, GUROBI e MOSEK são agora capazes de resolver esta importante classe de problemas.

Deve-se destacar que este modelo pode ser facilmente adaptado para subsequências geradas para cada nó da árvore de *branch-and-bound* apenas adicionando-se elementos ao conjunto $\mathcal{S}$. Por exemplo, se a sequência parcial do nó raiz é $\mathcal{S}_{root} = \{0, 5, 8\}$ e o próximo vértice a ser adicionado na sequência é o vértice 6 então as próximas subsequências geradas serão $\mathcal{S}_{child1} = \{0, 6, 5, 8\}$, $\mathcal{S}_{child2} = \{0, 5, 6, 8\}$ e $\mathcal{S}_{child3} = \{0, 5, 8, 6\}$.

3.3 RELAXAÇÃO DA RAIZ

Os três vértices que formam a sequência do nó raiz são selecionados da seguinte maneira. O primeiro vértice é o nó $i_0 = 0$, ou seja, o depósito, cujo raio é igual a zero. O próximo vértice a ser escolhido é o mais distante do depósito. O terceiro vértice a ser inserido é o que produz o maior *lower bound* no valor da solução do nó raiz. Mais especificamente, o algoritmo resolve um SOCP para cada candidato restante e seleciona o vértice associado à sequência que produz a melhor relaxação.

A figura 3.2 ilustra um exemplo envolvendo 7 vértices. Pode-se observar que o

depósito é o primeiro vértice selecionado, seguido pelo vértice 6 e depois o vértice 3, cujo critério de inserção é o descrito logo acima.

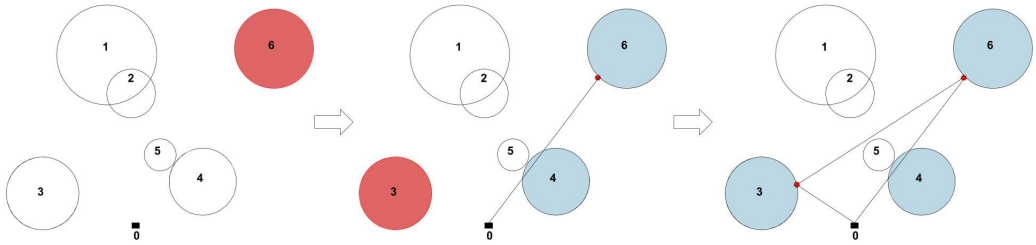

Figura 3.2: Ilustração do procedimento para encontrar uma relaxação válida na raiz

3.4 CHECAGEM DA VIABILIDADE

Assumindo-se $\hat{V}$ como um conjunto de vértices não cobertos por uma sequência parcial durante a execução do algoritmo e sendo $\hat{d}_i$ a distância entre um vértice $i \in \hat{V}$ e a aresta desta sequência parcial mais próxima de i. Suponha que c e $\overline{p_1 p_2}$ são um vértice e uma aresta de uma solução parcial nestas condições. Deseja-se determinar as coordenadas de um ponto $p \in \overline{p_1 p_2}$ que minimiza a distância entre $\overline{p_1 p_2}$ e c.

Lema 3.4.1. *A distância mínima $\hat{d}$ entre c e $\overline{p_1 p_2}$ pode ser calculada pela resolução do problema de otimização definido pela Equação (3.13):*

$$\hat{d} = \min_{0 \leq \theta \leq 1} \|(1-\theta)p_1 + \theta p_2 - c\| \tag{3.13}$$

Cuja solução analítica do problema irrestrito é dada por $\theta = -\frac{(p_2-p_1)^\top (p_1-c)}{\|p_2-p_1\|^2}$. Portanto a distância mínima pode ser calculada pela Equação (3.14)*:*

$$\hat{d} = \begin{cases} \|p_1 - c\|, & \text{if} \quad \theta = 0 \\ \|p - c\|, & \text{if} \quad 0 < \theta < 1, \ \text{where } p = (1-\theta)p_1 + \theta p_2 \\ \|p_2 - c\|, & \text{if} \quad \theta = 1 \end{cases} \tag{3.14}$$

Para checar a viabilidade de uma solução parcial, deve-se calcular o valor de $\hat{d}$ para cada vértice não coberto e para cada aresta existentes nesta solução. Se o número de vértices não cobertos é dado por u e o número de arestas é dado por v então esta

verificação toma $\mathcal{O}(uv)$ operações. Deve-se destacar que esta abordagem permite o cálculo para instâncias tanto no $\mathbb{R}^2$ quanto no $\mathbb{R}^3$.

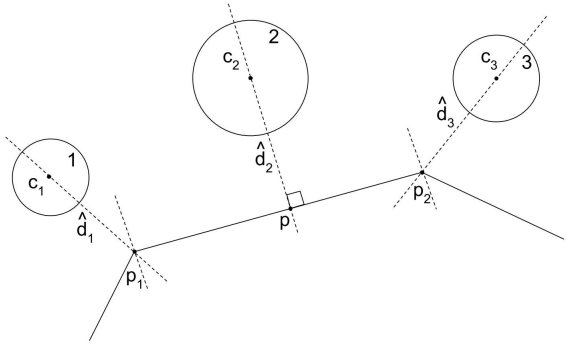

Figura 3.3: Ilustração envolvendo três vértices e uma aresta em uma solução parcial para o CETSP

A figura 3.3 mostra uma ilustração para um caso com três vértices e uma aresta. Para o vértice 1, $\theta = 0$, logo a distância mínima entre c e $\overline{p_1 p_2}$ é igual a distância entre c e p_1. Para o vértice 2, $0 < \theta < 1$, logo a distância mínima entre c e $\overline{p_1 p_2}$ é igual a distância entre c e p. Finalmente, para o vértice 3, θ é igual a 1, logo a distância mínima entre c e $\overline{p_1 p_2}$ é igual a distância entre c e p_2.

3.5 REGRAS DE *BRANCHING*

Primeiramente o algoritmo realiza uma verificação na instância:

Se todos os vértices possuem o mesmo raio: O algoritmo calcula o valor de $\hat{d}_k$ para todo $k \in \hat{V}$. Depois, o máximo valor entre todos os $\hat{d}_k$ é determinado, ou seja, calcula-se $\max_{k \in \hat{V}}\{\hat{d}_k\}$, então o vértice k correspondente é selecionado para ser inserido na solução parcial do nó filho. A Figura 3.4 ilustra um exemplo de como a seleção dos vértices ocorre. Neste caso, dada uma solução parcial $0 \to 3 \to 5 \to 0$. Nota-se que o vértice 4 está coberto, no entanto, os vértices 1, 2 não estão. Portanto $\hat{V} = \{1, 2\}$. Pode-se observar que $\hat{d}_1 > \hat{d}_2$. Dessa forma, o vértice 1 é o vértice associado à $\max_{k \in \hat{V}}\{\hat{d}_k\}$.

Se os vértices possuem diferentes raios: O algoritmo calcula o valor de γ_k para todo $k \in \hat{V}$. O escalar γ_k é definido como uma estimativa do acréscimo do *lower bound* se o vértice k fosse inserido entre seus dois vizinhos mais próximos pertencentes à sequência.

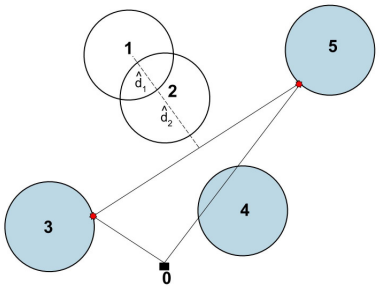

Figura 3.4: Ilustração da primeira regra de *branching*

Para computar γ_k primeiro calcula-se o valor das coordenadas do ponto $\tilde{p}_k$ na borda de D_k que minimiza a distância entre este ponto e sua aresta mais próxima $\overline{p_i p_j}$ na sequência. Então γ_k pode ser calculado como $\gamma_k = \tilde{e}_i^k + \tilde{e}_j^k - e_{ij}$, onde $\tilde{e}_i^k$ é a distância entre $\tilde{p}_k$ e p_i, $\tilde{e}_j^k$ é a distância entre $\tilde{p}_k$ e p_j e por fim, e_{ij} é o comprimento da aresta $\overline{p_i p_j}$. A figura 3.5 mostra um exemplo deste procedimento, sendo $\hat{V} = \{1, 2\}$. Portanto $\gamma_1 = \tilde{e}_1^1 + \tilde{e}_2^1 - e_{34}$ e $\gamma_2 = \tilde{e}_1^2 + \tilde{e}_2^2 - e_{40}$. Suponha que $\gamma_2 > \gamma_1$, como sugere a figura, portanto, segundo esta regra de *branching*, o vértice 2 será próximo a compor a sequência.

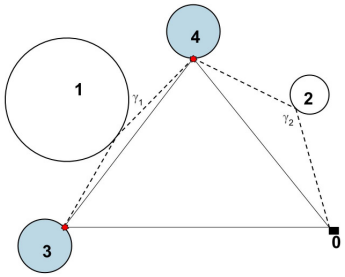

Figura 3.5: Ilustração da segunda regra de *branching*

4 RESULTADOS

4.1 EXPERIMENTOS COMPUTACIONAIS

O algoritmo *branch-and-bound* foi implementado em linguagem C++, os testes foram realizados em um computador Intel Core i7, com 3.40 GHz e 16 GB de RAM no sistema operacional Linux Mint 13. Para resolver os subproblemas Cônicos de Segunda Ordem foi utilizado o *solver* CPLEX (2013), versão 12.4, com o modo paralelo desativado e utilizando apenas uma única *thread*. As ilustrações das soluções geradas para a instâncias 3D foram geradas utilizando a rotina *BUBBLEPLOT3* (Bodin, 2009) implementada em MATLAB. Foi estipulado um tempo máximo de 4 horas para a execução do algoritmo em cada instância.

O desempenho do algoritmo foi testado em 824 instâncias fornecidas nos trabalhos de Mennell (2009) e Behdani e Smith (2013). A dificuldade de cada instância foi mensurada a partir da definição de *overlap* dada por Mennell (2009). O *overlap* é definido como a razão entre a média dos raios de todos os vértices e o maior lado (l_{max}) do retângulo que envolve todos os vértices e seus raios, ou seja, $overlap = \frac{(\sum_{i=0}^{n} r_i)/n}{l_{max}}$.

Mennell (2009) apresenta a seguinte classificação para suas instâncias: Seis problemas cujos nomes começam por *team* mais a instância *bonus1000* foram denominadas ***Team Problems***. Os problemas cujos nomes começam por *rotatingDiamonds*, *bubbles*, *concentricCircles* mais a instância *chaoSigleDep* foram englobadas no subconjunto denominado ***Geometric Problems***. As instâncias *d493*, *dsj1000*, *kroD100*, *lin318*, *pcb442*, *rat195* e *rd400* foram criadas a partir da TSPLIB e foram chamadas ***TSPLIB Problems***.

As instâncias dos grupos ***Team Problems*** e ***Geometric Problems*** foram geradas com $r_i = r, i = 0, \ldots, n$ possuem taxas de *overlap* variadas cujos valores não foram específicados. As instâncias do grupo ***TSPLIB Problems*** geradas com três tamanhos de raio diferentes, portanto com três taxas de *overlap* diferentes, a saber 0.02, 0.1 e 0.3. Apenas as instâncias dos grupos ***Team Problems*** e ***TSPLIB Problems*** possuem versões em 3D. Mennell (2009) disponibilizou também dois conjuntos de instâncias denominados ***Team Random Radius Problems*** e ***TSPLIB Random Radius Problems***, com versões em 2D e 3D, cujos raios de cada vértice foram gerados aleatoriamente de modo que $r_i \neq r_j \; \forall \; i, j \in V$.

Behdani e Smith (2013) disponibilizaram 240 problemas bidimensionais com 7, 9, 11, 13, 15, 17, 19 e 21 vértices. Estas instâncias foram geradas de acordo com o seguinte procedimento. A posição $(\bar{x}_i, \bar{y}_i), i \in V$ de cada vértice e do depósito i_0 foram escolhidas aleatoriamente em um espaço limitado de 16 unidades de comprimento e 10 unidades de largura. Nestas instâncias, a área cobertura de todos os foi definida como uma circunferência de raio r. Em seus experimentos os autores utilizaram três tamanhos de raio diferentes, a saber 0.25, 0.5 e 1.0, obtendo três grupos distintos com *overlaps* 0.015, 0.03 e 0.06 respectivamente.

4.2 SELEÇÃO DA ESTRATÉGIA DE *BRANCHING*

Com o intuito de selecionar a melhor estratégia de *branching* para o algoritmo proposto, foram testadas três alternativas conhecidas na literatura: DFS, BFS e BeFS. Para testar o desempenho de cada estratégia considerou-se as chamadas *Geometric Instances*, com 20 instâncias bidimensionais, fornecidas por Mennell (2009). Para estes testes preliminares um tempo limite de 2 horas foi estipulado para a execução em cada instância.

A Tabela 4.1 mostra um resumo dos resultados encontrados para cada estratégia de *branching*. Nesta tabela a coluna **#Opt** representa o número de ótimos encontrados pelo algoritmo utilizando cada estratégia. A coluna **LB.*** representa a média aritmética dos *lower bounds* encontrados nas instâncias onde não encontrou-se uma solução ótima. A coluna **Tam. Árvore** representa a média do tamanho da árvore para todas as instâncias.

A coluna **Gap*(%)** representa o *gap* médio em relação ao melhor *upper bound* conhecido para as instâncias onde o algoritmo não foi capaz de encontrar uma solução ótima. Por fim, a coluna **CPU T(s)**** representa o tempo médio pra encontrar a solução ótima nas instâncias onde o algoritmo foi capaz de fazê-lo.

Pode-se observar que a estratégia BeFS obteve os melhores resultados dentro do tempo limite estabelecido. Portanto, esta estratégia foi adotada nos demais experimentos apresentados neste trabalho.

Tabela 4.1: Resultados do experimento para escolha da estrategia de *branching*

Estratégia	**#Opt.**	**LB***	**Tam. Árvore**	**Gap* (%)**	**CPU T** (s)**
BeFS	6	746.016	523529	22.627	188.36
BFS	6	652.148	1100302	31.716	1222.62
DFS	6	428.432	668770	56.311	242.97

4.3 RESULTADOS OBTIDOS NAS INSTÂNCIAS DE Behdani e Smith (2013)

A Tabela 4.2 mostra um resumo dos resultados encontrados para as instâncias de Behdani e Smith (2013). Nesta tabela a coluna **Grupo** apresenta os 8 grupos de instâncias divididos em três subgrupos diferentes (*overlap* = 0.015, 0.03, 0.06). A coluna **Tamanho** mostra a quantidade de instâncias em cada grupo. A coluna **#Opt.** mostra o número de ótimos encontrados em cada grupo. A coluna **Arv.** mostra a média do tamanho da árvore na execução do *branch-and-bound* para todas as instâncias. Por fim, a coluna **CPU T(s)** mostra a média dos tempos de execução.

A Figura 4.1 mostra uma ilustração das soluções encontradas para a instância CETSP-6-19. Nesta figura, pode-se verificar que o aumento dos raios dos vértices torna os problemas mais fáceis de resolver para o algoritmo *branch-and-bound*. A Tabela 4.2 corrobora esta afirmação visto que a medida que o *overlap* aumenta diminuem a média dos tamanhos da árvore e a média dos tempos de execução.

Nos testes realizados observou-se que o algoritmo *branch-and-bound* é até centenas de vezes mais rápido computacionalmente do que o algoritmo proposto por Behdani e Smith (2013) nas maiores instâncias. Além do mais, todas as instâncias foram resolvidas

Tabela 4.2: Resumo dos resultados para as instâncias de Behdani e Smith

Grupo	Tamanho	#Opt.	Árv.	CPU T(s)
$r = 0.25$ e $overlap = 0.0156$				
CETSP-06	30	30	26	0.029
CETSP-08	30	30	51	0.062
CETSP-10	30	30	86	0.107
CETSP-12	30	30	139	0.189
CETSP-14	30	30	219	0.308
CETSP-16	30	30	443	0.673
CETSP-18	30	30	544	0.839
CETSP-20	30	30	1016	1.656
$r = 0.50$ e $overlap = 0.0313$				
CETSP-06	30	30	19	0.020
CETSP-08	30	30	39	0.047
CETSP-10	30	30	66	0.085
CETSP-12	30	30	92	0.125
CETSP-14	30	30	144	0.204
CETSP-16	30	30	222	0.334
CETSP-18	30	30	331	0.525
CETSP-20	30	30	437	0.719
$r = 1.0$ e $overlap = 0.0625$				
CETSP-06	30	30	13	0.017
CETSP-08	30	30	23	0.030
CETSP-10	30	30	38	0.049
CETSP-12	30	30	51	0.066
CETSP-14	30	30	71	0.100
CETSP-16	30	30	95	0.144
CETSP-18	30	30	117	0.176
CETSP-20	30	30	163	0.261

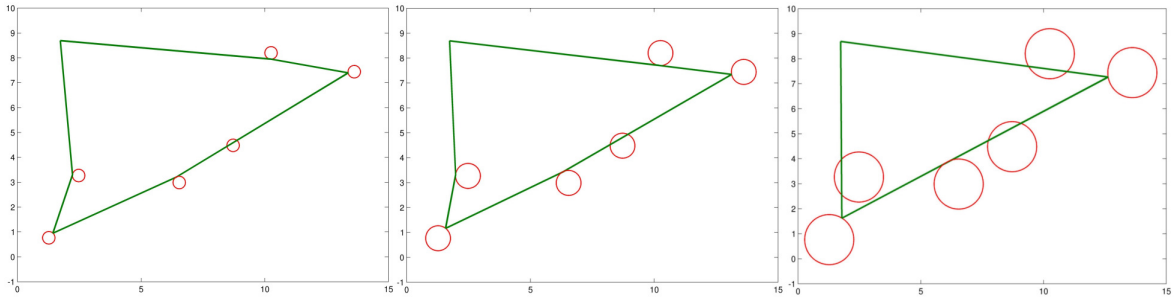

Figura 4.1: Resultado para a instância CETSP-6-19, $r = 0.25, 0.50$ e 1.0, respectivamente.

na otimalidade pelo *branch-and-bound*. Logo não realizou-se uma comparação direta com os resultados encontrados por Behdani e Smith (2013).

Os resultados completos obtidos pelo algoritmo *branch-and-bound* para as instâncias propostas por Behdani e Smith (2013) podem ser encontrados no Apêndice A.

4.4 RESULTADOS OBTIDOS NAS INSTÂNCIAS 2D DE Mennell (2009)

Os resultados encontrados para as instâncias 2D de Mennell (2009) podem ser encontrados na Tabela 4.3. Nesta tabela a primeira coluna mostra o nome de cada instância. A

coluna $|V|$ mostra o tamanho das instâncias. Os *upper bounds* e *lower bounds* encontrados no trabalho de Mennell (2009) podem ser visualizados nas 3ª e 4ª colunas. A coluna **Opt.** mostra o custo das soluções ótimas encontradas pelo algoritmo proposto. Quando o algoritmo não foi capaz de encontrar uma solução ótima para uma determinada instância, a respectiva linha é preenchida com "-". A sexta coluna mostra os *lower bounds* encontrados em cada instância. A coluna **Árv.** mostra o tamanho da árvore para cada instância. A coluna **Gap (%)** mostra o valor do *gap* entre o *lower bound* encontrado e o melhor *upper bound* disponível ou o custo de uma solução ótima, ou seja, $gap = (UB - LB)/UB,\ UB = min\{\mathbf{UB}; \mathbf{Opt.}\}$.

Pode-se observar que o algoritmo *branch-and-bound* foi capaz de encontrar soluções ótimas para 22 de 62 instâncias bidimensionais disponíveis. As instâncias para a quais a solução encontrada supera o valor do melhor *upper bound* conhecido apresentam o valor da respectiva linha em **Opt.** destacado em negrito. Verifica-se ainda que todos os *lower bounds* encontrados pelo algoritmo *branch-and-bound* superaram os *lower bounds* encontrados na literatura.

Percebe-se que a medida que o valor do *overlap* aumenta o algoritmo torna-se mais capaz de encontrar soluções ótimas em tempos computacionais competitivos.

A Figura 4.2 mostra uma comparação entre a solução obtida pela heurística proposta por Mennell (2009) e a solução encontrada pelo algoritmo proposto neste trabalho. O método *branch-and-bound* explora a propriedade relacionada ao número de *turn points*, ou seja, para um determinado número n de vértices que devem ser atendidos, o número de vértices necessários para formar uma solução ótima q é tal que $q \leq n$. As Figuras (4.4 - 4.9) mostram as soluções encontradas para algumas instâncias.

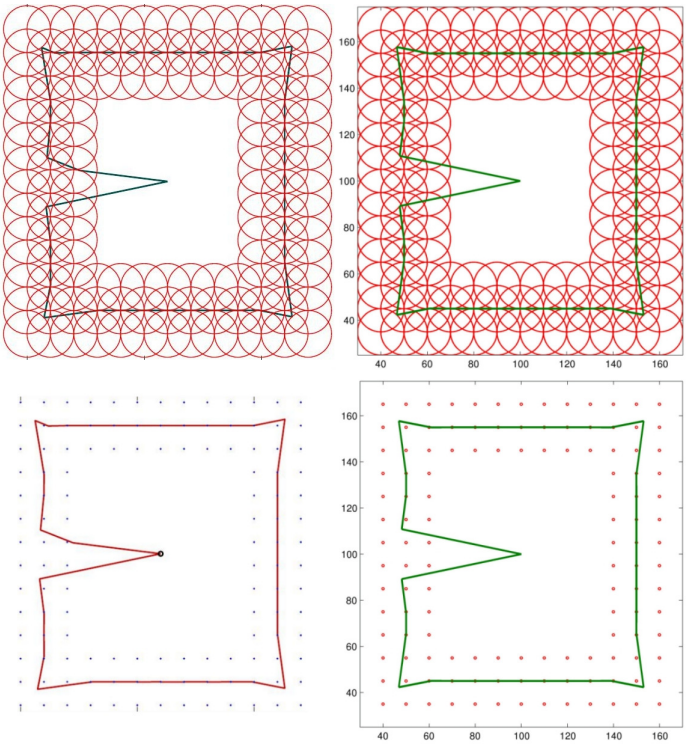

Figura 4.2: *bubbles*3: À esquerda a solução encontrada por Mennell (2009), custo = 530.733. À direita a solução do *branch-and-bound*, custo = 529.955.

4.5 RESULTADOS OBTIDOS NAS INSTÂNCIAS 3D DE Mennell (2009)

A Tabela 4.4 apresenta os resultados obtidos para as instâncias 3D de Mennell (2009). Nesta tabela, o significado das colunas permanece o mesmo já explicado na seção anterior. Pode-se observar que o algoritmo *branch-and-bound* foi capaz de encontrar soluções ótimas para 10 de 42 instâncias tridimensionais disponíveis. As instâncias para as quais a solução encontrada supera o valor do melhor *upper bound* conhecido apresentam o valor da respectiva linha em **Opt.** destacado em **negrito**. Pode-se verificar ainda que todos os *lower bounds* encontrados pelo algoritmo *branch-and-bound* superaram os *lower bounds* encontrados na literatura.

A Figura 4.3 mostra uma comparação entre a solução obtida pela heurística proposta por Mennell (2009) e a solução encontrada pelo algoritmo proposto neste trabalho. As Figuras (4.10 - 4.15) mostram as soluções encontradas para algumas instâncias.

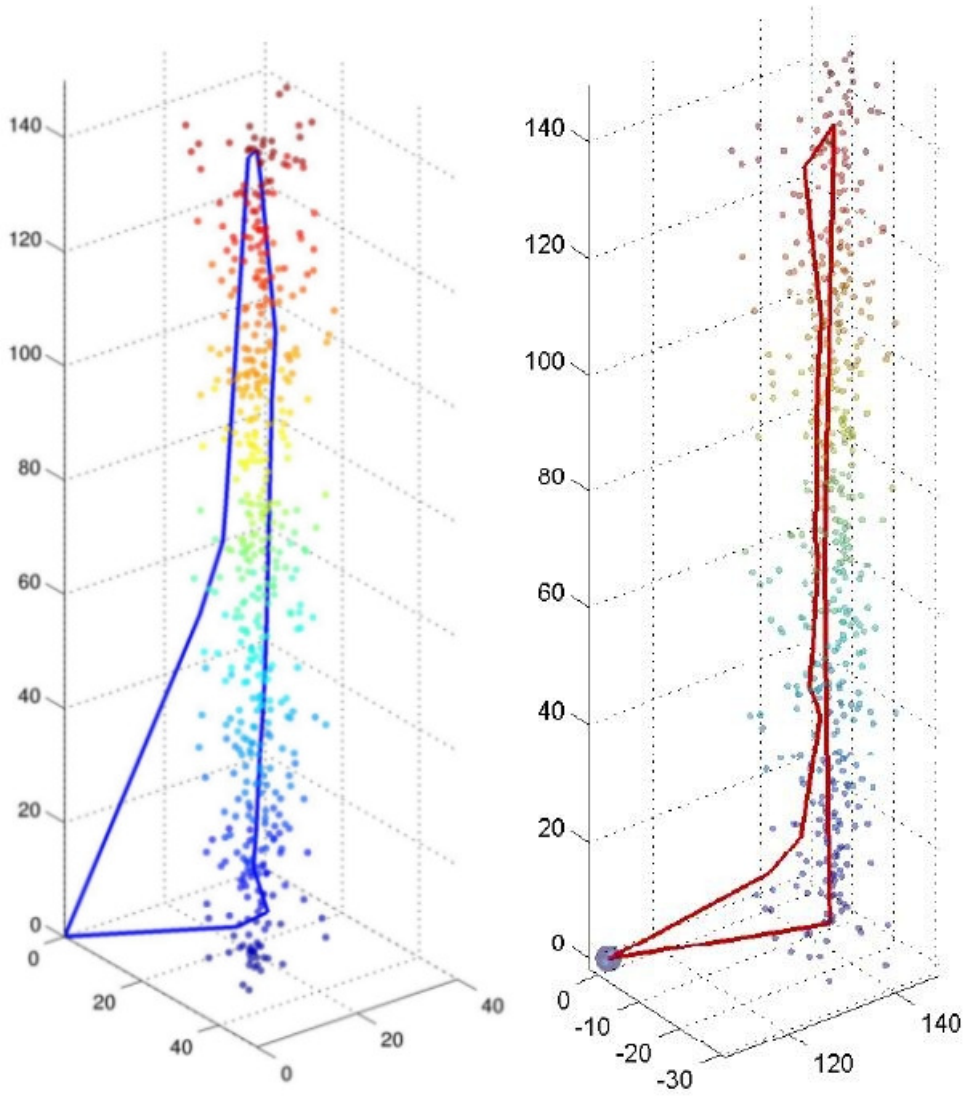

Figura 4.3: *d*493: À esquerda a solução do *branch-and-bound*, custo = 325.207; À direita a solução encontrada por Mennell (2009), custo = 335.592

Tabela 4.3: Resultados para instâncias 2D de Mennell

Instância	$\|V\|$	**Mennell (2009)**		***Branch-and-Bound***				
		UB	**LB**	**Opt.**	**LB**	**Árv.**	**Gap (%)**	**CPU T(s)**
***Overlap*: 0.02**								
d493	493	202.793	57.460	-	**146.331**	859216	27.842	14400.17
dsj1000	1000	935.743	100.190	-	**559.114**	585631	40.249	14400.24
kroD100	100	159.046	78.830	-	**142.872**	1784161	10.169	14400.16
lin318	318	2863.366	977.430	-	**1990.902**	933499	30.470	14400.08
pcb442	442	323.034	32.510	-	**185.847**	842855	42.468	14400.03
rat195	195	158.785	28.200	-	**108.104**	1108939	31.918	14400.04
rd400	400	1033.414	284.030	-	**567.190**	891890	45.115	14400.27
***Overlap*: 0.1**								
d493	493	101.735	31.730	**100.721**	**100.721**	8343	0.000	53.28
dsj1000	1000	376.099	10.170	-	**373.730**	718104	0.630	14400.51
kroD100	100	89.668	0.000	89.668	**89.668**	798	0.000	1.86
lin318	318	1408.482	0.000	**1394.626**	**1394.626**	1118666	0.000	8541.19
pcb442	442	147.244	1.870	-	**137.448**	1383948	6.653	14400.09
rat195	195	68.083	0.000	**67.991**	**67.991**	4735	0.000	17.32
rd400	400	466.102	0.000	-	**432.798**	1436555	7.145	14400.03
Overlap: 0.3								
d493	493	69.758	16.750	69.758	**69.758**	3	0.000	0.32
dsj1000	1000	199.948	0.000	199.948	**199.948**	17	0.000	0.75
kroD100	100	58.541	0.000	58.541	**58.541**	3	0.000	0.07
lin318	318	765.964	0.000	765.964	**765.964**	7	0.000	0.24
pcb442	442	83.537	0.000	83.537	**83.537**	3	0.000	0.31
rat195	195	45.702	0.000	45.702	**45.702**	3	0.000	0.13
rd400	400	224.839	0.000	224.839	**224.839**	24	0.000	0.33
Varias taxas de *overlap*								
bonus1000	1001	402.470	0.000	-	**359.383**	949465	10.706	14400.02
bubbles1	37	349.135	60.620	349.135	**349.135**	66	0.000	0.10
bubbles2	77	428.279	60.620	428.279	**428.279**	121	0.000	0.22
bubbles3	127	530.733	60.620	**529.955**	**529.955**	31995	0.000	193.12
bubbles4	185	829.888	60.620	-	**690.578**	1502931	16.787	14400.01
bubbles5	251	1062.335	60.620	-	**851.822**	1181671	19.816	14400.28
bubbles6	325	1383.139	60.620	-	**993.981**	1025324	28.136	14400.15
bubbles7	407	1720.214	60.620	-	**1123.522**	918674	34.687	14400.19
bubbles8	497	2101.373	60.620	-	**1252.715**	894857	40.386	14400.28
bubbles9	595	2426.274	60.620	-	**1374.407**	765301	43.353	14400.33
chaoSingleDep	201	1039.610	439.260	-	**1000.151**	1760829	3.796	14400.09
concentricCircles1	17	53.158	14.000	53.158	**53.158**	2943	0.000	5.18
concentricCircles2	37	153.132	43.590	-	**149.868**	2699245	2.131	14400.03
concentricCircles3	61	271.076	115.280	-	**247.624**	2093145	8.651	14400.18
concentricCircles4	105	454.457	161.110	-	**358.887**	1320268	21.030	14400.06
concentricCircles5	149	645.381	249.980	-	**459.411**	1112726	28.815	14400.16
rotatingDiamonds1	21	32.389	6.200	32.389	**32.389**	59	0.000	0.09
rotatingDiamonds2	61	140.477	13.870	140.477	**140.477**	274967	0.000	730.37
rotatingDiamonds3	181	380.882	27.870	-	**348.608**	1612881	8.474	14400.07
rotatingDiamonds4	321	770.660	35.570	-	**593.350**	969457	23.008	14400.03
rotatingDiamonds5	681	1510.752	69.570	-	**1106.577**	777682	26.753	14400.23
team1_100	101	307.337	33.520	307.337	**307.337**	2805	0.000	9.61
team2_200	201	246.683	0.000	246.683	**246.683**	187	0.000	0.72
team3_300	301	466.241	8.750	-	**447.534**	1713428	4.012	14400.03
team4_400	401	680.211	20.590	-	**507.302**	938101	25.420	14400.10
team5_499	500	702.823	231.210	-	**524.589**	833603	25.360	14400.05
team6_500	501	225.216	0.000	225.216	**225.216**	24	0.000	0.43
Raios Arbitrários								
bonus1000rdmRad	1001	987.114	-	-	**506.131**	656095	48.726	14400.22
d493rdmRad	493	140.120	-	-	**125.312**	1195835	10.568	14400.14
dsj1000rdmRad	1000	653.128	-	-	**509.740**	798273	21.954	14400.30
kroD100rdmRad	100	141.835	-	-	**136.620**	2293903	3.677	14400.12
lin318rdmRad	318	2080.574	-	-	**1807.681**	1349251	13.116	14400.00
pcb442rdmRad	442	235.188	-	-	**175.834**	966808	25.237	14400.23
rat195rdmRad	195	68.224	-	68.224	**68.224**	1628	0.000	5.16
rd400rdmRad	400	1252.380	-	-	**571.482**	776842	54.368	14400.29
team1_100rdmRad	101	388.537	-	388.537	**388.537**	86565	0.000	269.31
team2_200rdmRad	201	622.738	-	-	**488.182**	1098493	21.607	14400.12
team3_300rdmRad	301	381.828	-	**378.087**	**378.087**	132111	0.000	682.39
team4_400rdmRad	401	1011.772	-	-	**549.907**	870580	45.649	14400.32
team5_499rdmRad	500	454.327	-	-	**442.637**	1145471	2.573	14400.36
team6_500rdmRad	501	666.149	-	-	**489.612**	939270	26.501	14400.08

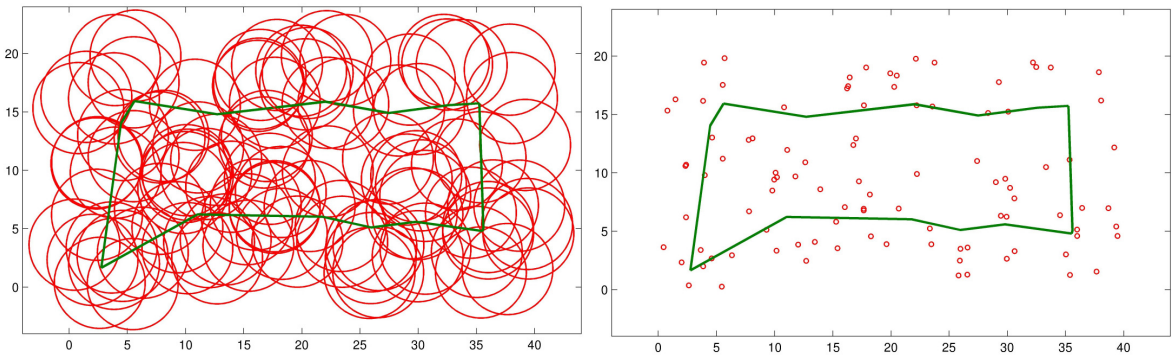

Figura 4.4: Solução para $kroD100$, Raios iguais, 2D, $|V| = 100$, Custo = 89.668

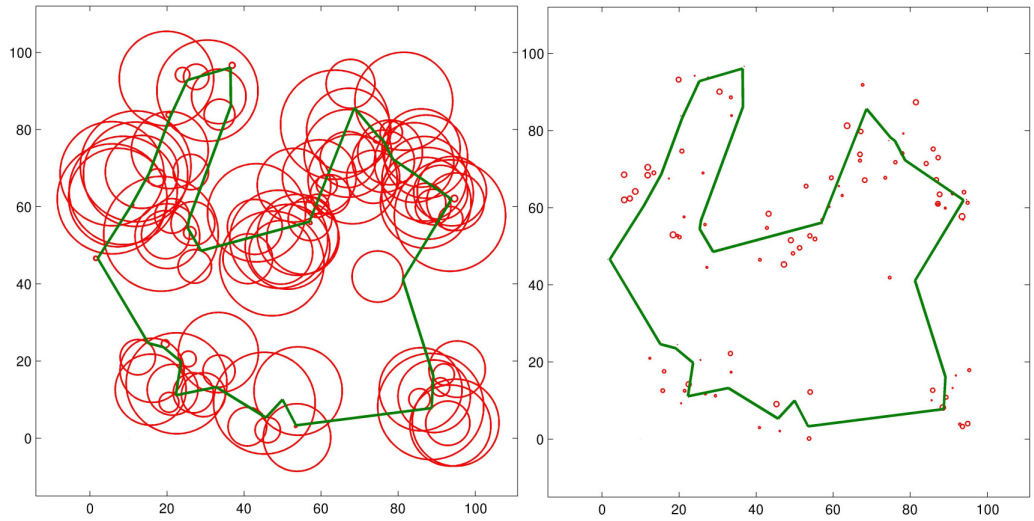

Figura 4.5: Solução para $team1\_100rdmRad$, Raios Arbitrários, 2D, $|V| = 101$, Custo = 388.537

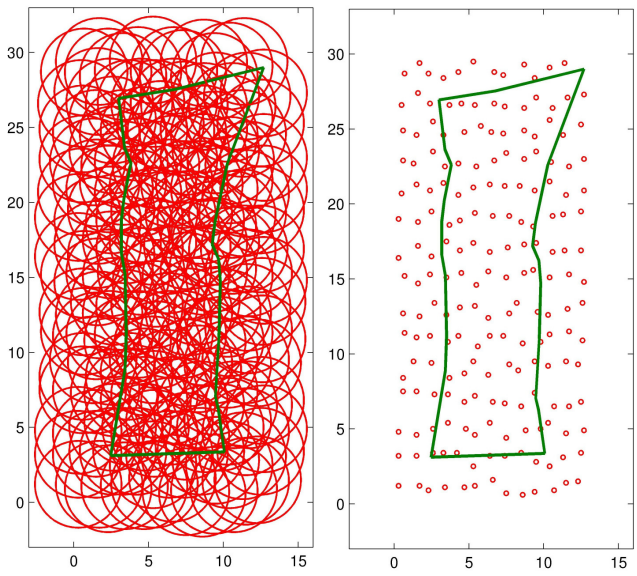

Figura 4.6: Solução para $rat195$, Raios Iguais, 2D, Tamanho = 195, Custo = 67.991

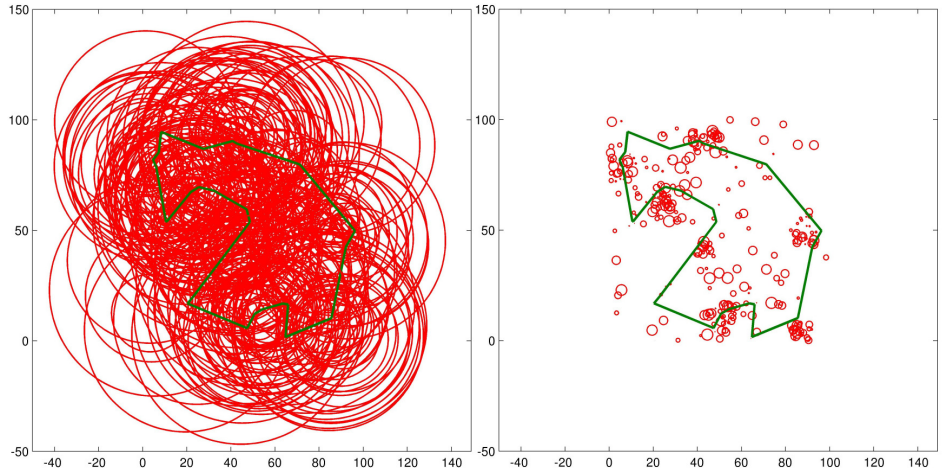

Figura 4.7: Solução para $team3\_300rdmRad$, Raios Arbitrários, 2D, $|V| = 301$, Custo = 378.087

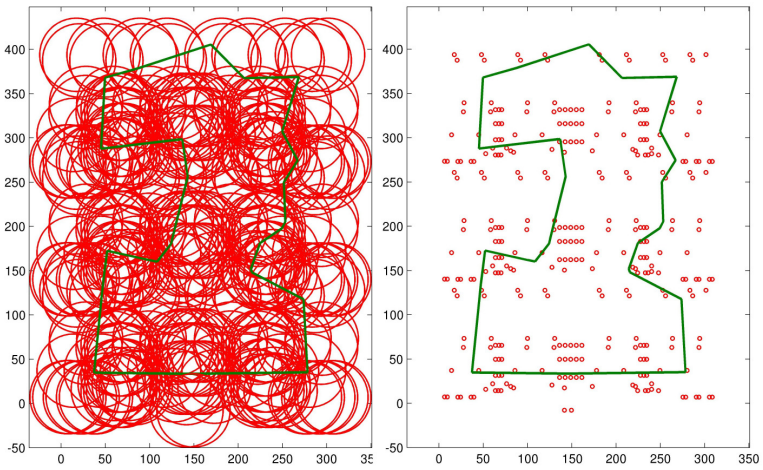

Figura 4.8: Solução para $lin318$, Raios Iguais, 2D, $|V| = 318$, Custo = 1394.626

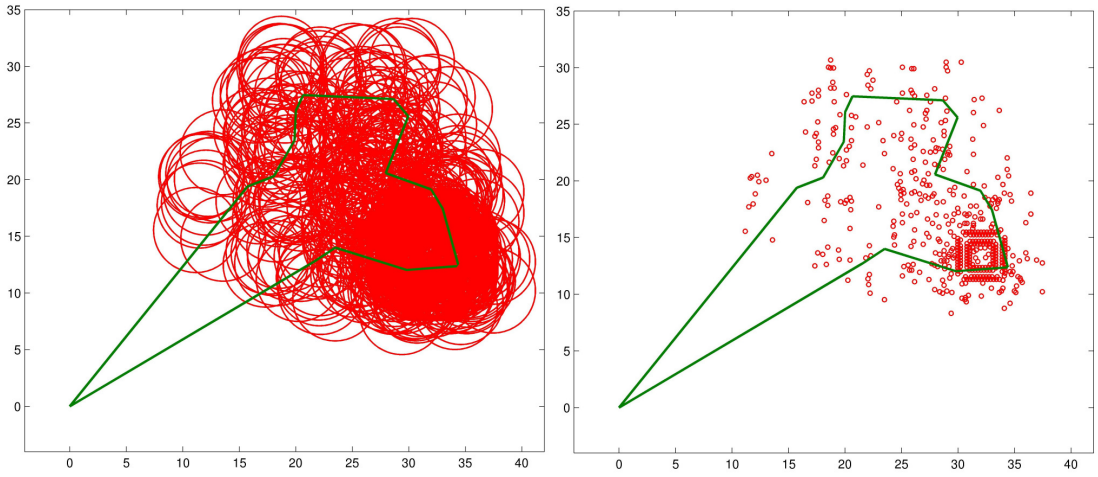

Figura 4.9: Solução para $d493$, Raios Iguais, 2D, $|V| = 493$, Custo = 100.721

Tabela 4.4: Resultados para instâncias 3D de Mennell

Instância	$\|V\|$	**Mennell (2009)**		***Branch-and-Bound***				
		UB	**LB**	**Opt.**	**LB**	**Árv.**	**Gap (%)**	**CPU T(s)**
Overlap ratio: 0.02								
d493	493	1353.137	-	-	**469.783**	642868	65.282	14400.01
dsj1000	1000	3147.865	-	-	**751.510**	494579	76.126	14400.05
kroD100	100	202.021	-	-	**148.231**	1183277	26.626	14400.12
lin318	318	3044.270	-	-	**1994.372**	934859	34.488	14400.14
pcb442	442	404.490	-	-	**186.381**	725462	53.922	14400.18
rat195	195	291.258	-	-	**126.493**	978598	56.570	14400.06
rd400	400	3218.198	-	-	**868.188**	753967	73.023	14400.12
Overlap ratio: 0.1								
d493	493	665.056	-	**-**	**421.164**	827584	36.672	14400.18
dsj1000	1000	1021.252	-	-	**602.987**	644546	40.956	14400.31
kroD100	100	91.669	-	**91.663**	**91.663**	2407	0.000	7.51
lin318	318	1443.427	-	**-**	**1398.254**	1588841	3.130	14400.12
pcb442	442	154.810	-	-	**137.951**	1209228	10.890	14400.13
rat195	195	112.405	-	**-**	**88.721**	1325338	21.070	14400.13
rd400	400	1552.723	-	-	**752.423**	851059	51.542	14400.24
Overlap ratio: 0.3								
d493	493	335.592	-	**325.207**	**325.207**	5064	0.000	31.11
dsj1000	1000	270.399	-	**267.751**	**267.751**	2472	0.000	25.07
kroD100	100	58.926	-	58.926	**58.926**	4	0.000	0.08
lin318	318	766.831	-	766.831	**766.831**	8	0.000	0.24
pcb442	442	83.722	-	83.722	**83.722**	4	0.000	0.33
rat195	195	47.889	-	47.889	**47.889**	4	0.000	0.14
rd400	400	539.954	-	-	**450.720**	507442	16.526	14400.40
Varied overlap ratios								
bonus1000	1001	941.348	-	-	**472.559**	655479	49.800	14400.35
team1_100	101	820.727	-	-	**690.298**	1197716	15.892	14400.22
team2_200	201	283.238	-	**273.383**	**273.383**	86492	0.000	557.94
team3_300	301	1484.411	-	-	**762.683**	930319	48.620	14400.12
team4_400	401	753.813	-	-	**509.803**	894994	32.370	14400.30
team5_499	500	1924.527	-	-	**705.633**	665171	63.335	14400.20
team6_500	501	236.964	-	**230.923**	**230.923**	73	0.000	0.67
Arbitrary Radius Problems								
bonus1000rdmRad	1001	2689.413	-	-	**578.638**	512678	78.485	14400.04
d493rdmRad	493	761.065	-	-	**438.701**	858509	42.357	14400.06
dsj1000rdmRad	1000	2074.844	-	-	**696.289**	656155	66.441	14400.05
kroD100rdmRad	100	171.568	-	-	**137.765**	1350550	19.702	14400.13
lin318rdmRad	318	2189.426	-	-	**1806.783**	1200921	17.477	14400.10
pcb442rdmRad	442	258.404	-	-	**177.231**	962861	31.413	14400.09
rat195rdmRad	195	84.470	-	**82.105**	**82.105**	152636	0.000	790.53
rd400rdmRad	400	3592.601	-	-	**876.280**	766218	75.609	14400.06
team1_100rdmRad	101	907.593	-	-	**726.685**	1137079	19.933	14400.26
team2_200rdmRad	201	1055.948	-	-	**525.310**	1027597	50.252	14400.20
team3_300rdmRad	301	1053.380	-	**-**	**676.184**	1004637	35.808	14400.10
team4_400rdmRad	401	1276.896	-	-	**551.046**	793139	56.845	14400.08
team5_499rdmRad	500	840.477	-	-	**599.741**	940015	28.643	14400.28
team6_500rdmRad	501	1076.352	-	-	**507.122**	870473	52.885	14400.12

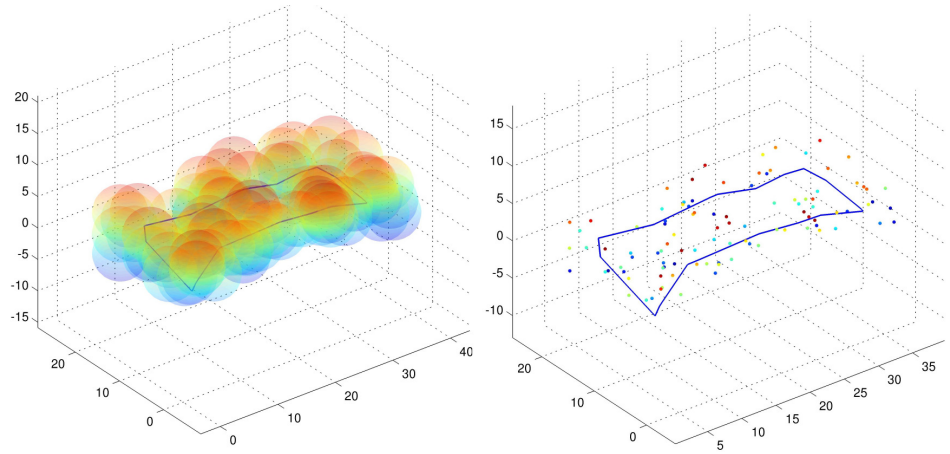

Figura 4.10: Solução para $kroD100$, Raios iguais, 3D, $|V| = 100$, Custo = 91.663

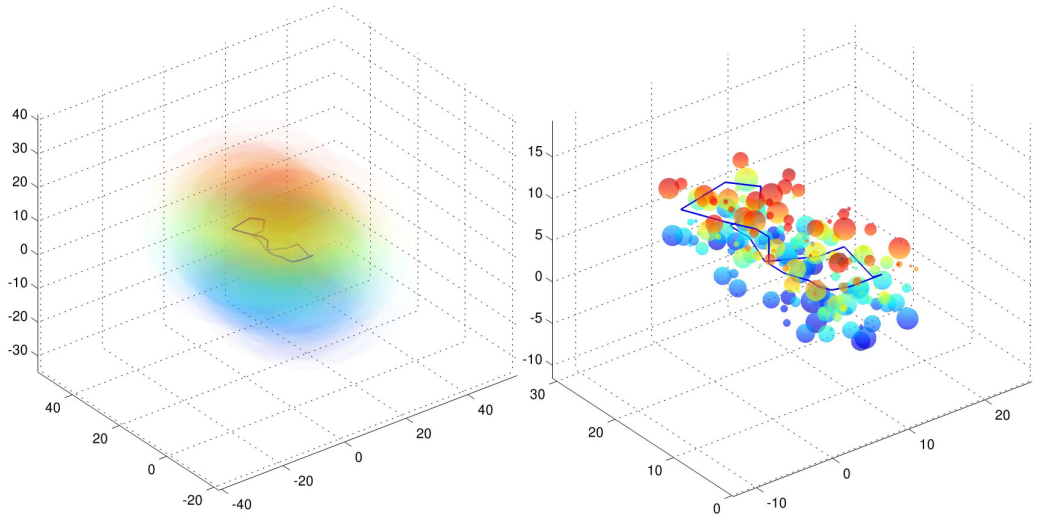

Figura 4.11: Solução para $rat195rdmRad$, Raios arbitrários, 3D, $|V| = 195$, Custo = 82.105

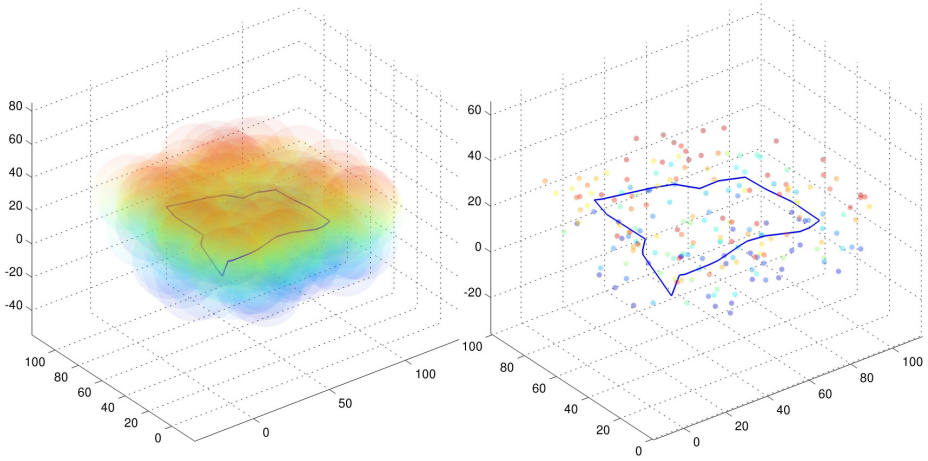

Figura 4.12: Solução para $team2\_200$, Raios iguais, 3D, $|V| = 201$, Custo = 273.383

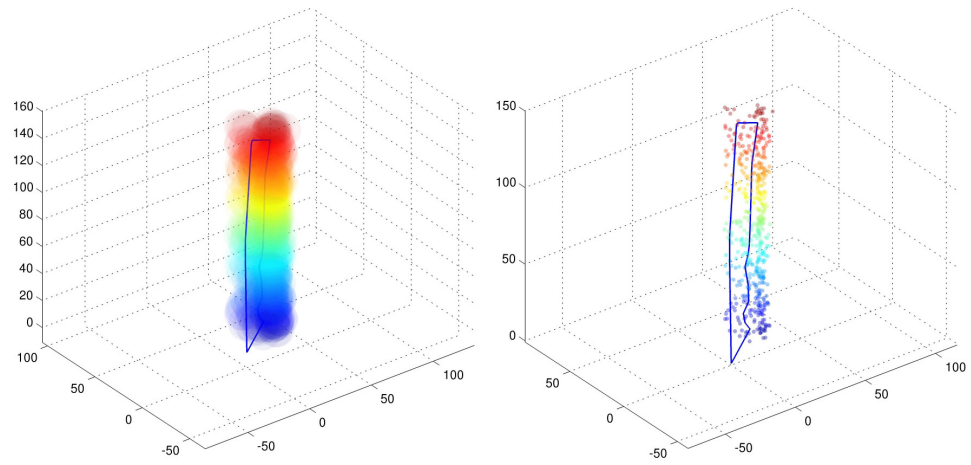

Figura 4.13: Solução para $d493$, Raios iguais, 3D, $|V| = 493$, Custo = 325.207

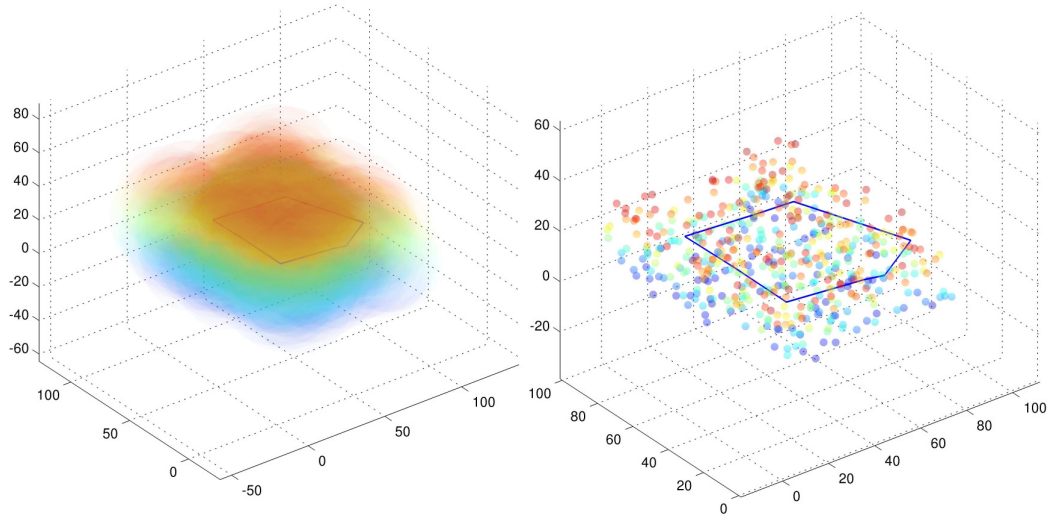

Figura 4.14: Solução para $team6\_500$, Raios iguais, 3D, $|V| = 501$, Custo = 230.923

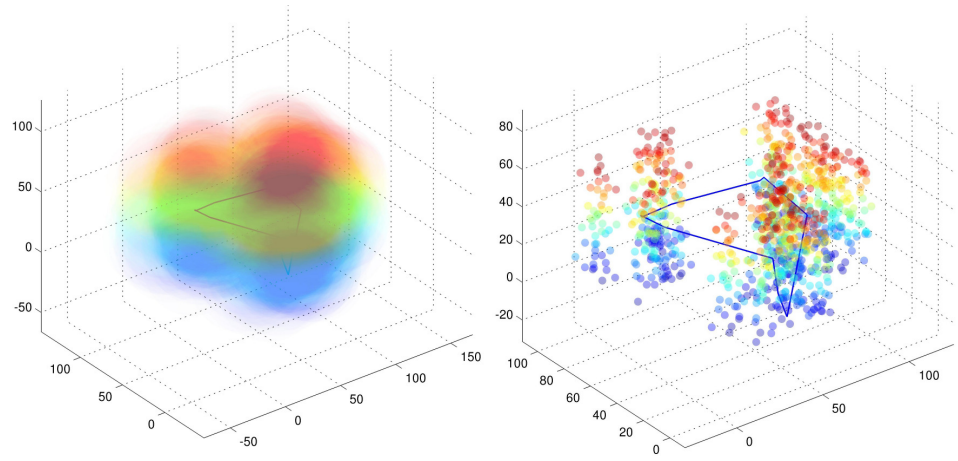

Figura 4.15: Solução para $dsj1000$, Raios iguais, 3D, $|V| = 1000$, Custo = 267.751

5 CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho apresentou um algoritmo *branch-and-bound* para o CETSP, onde o subproblema resolvido em cada nó consiste em um SOCP. O algoritmo proposto foi testado em 824 instâncias disponíveis na literatura. Deste total, 720 instâncias bidimensionais foram propostas por Behdani e Smith (2013) e 62 instâncias bidimensionais e 42 tridimensionais foram propostas por Mennell (2009). O algoritmo foi capaz de encontrar no total a solução ótima para 752 instâncias. Todas as instâncias propostas por Behdani e Smith (2013) foram resolvidas na otimalidade. Das instâncias bidimensionais propostas por Mennell (2009) o algoritmo resolveu 22 na otimalidade e melhorou todos os *lower bounds* encontrados na literatura. No caso tridimensional 10 instâncias foram resolvidas e todos os *lower bounds* também foram melhorados.

No caso bidimensional o algoritmo foi capaz de encontrar bons *lower bounds* para as instâncias com uma taxa de overlap de 0,1. Bons *lower bounds* são importantes por exemplo, para a avaliação de heurísticas.

Por outro lado o método mostrou-se também bastante ineficiente nas instâncias que apresentavam uma baixa taxa de *overlap*. Este resultado contrasta com os resultados obtidos em Behdani e Smith (2013), uma vez que o método proposto pelos autores tem um melhor desempenho para instâncias pequenas e com baixas taxas de *overlap*.

No geral, pode-se observar que a performance do algoritmo torna-se bastante competitiva a medida que a taxa de *overlap* aumenta. Em termos práticos, esta relação está diretamente ligada ao avanço tecnológico. Por exemplo, melhores transmissores e receptores *wireless* tendem a aumentar o raio de ação (raio de cobertura) de equipamentos que

utilizam esta tecnologia, como no caso da AMR ou outras aplicações encontradas na literatura, ou seja a medida que a tecnologia avança os tamanhos dos raios de cada vértice tendem a aumentar.

Conforme o esperado, os *lower bounds* encontrados para as instâncias com baixo *overlap* não são satisfatórios. Neste sentido, como trabalhos futuros, pode-se desenvolver estratégias para melhorar a relaxação da raiz e os *lower bounds* encontrados durante a execução do algoritmo. Além destas abordagens focadas no método *branch-and-bound* outras abordagem podem também ser desenvolvidas, como algoritmos de planos de corte e baseados em programação inteira.

REFERÊNCIAS BIBLIOGRÁFICAS

ALIZADEH, F.; GOLDFARB, D. Second-order cone programming. *Mathematical Programming*, v. 95, p. 3–51, 2003.

ALZALG, B. M. Stochastic second-order cone programming: Applications models. *APPLIED MATHEMATICAL MODELLING*, v. 36, n. 10, p. 5122–5134. ISSN 0307-904X. doi: 10.1016/j.apm.2011.12.053, 2012.

ANDERSEN, E.; ROOS, C.; TERLAKY, T. On implementing a primal-dual interior-point method for conic quadratic optimization. *Mathematical Programming*, v. 95, n. 2, p. 249–277, 2003.

ARKIN, E.; HASSIN, R. Approximation Algorithms For The Geometric Covering Salesman Problem. *Discrete Applied Mathematics*, v. 55, n. 3, p. 197–218. ISSN 0166-218X, 1994.

BEHDANI, B.; SMITH, J. C. An Integer-Programming-Based Approach to the Close-Enough Traveling Salesman Problem. *INFORMS Journal On Computing*. Forthcoming, 2013.

BODIN, P. BUBBLEPLOT3: A simple 3D bubbleplot. `http://www.mathworks.com/matlabcentral/fileexchange/8231-bubbleplot3`, 2009.

BOYD, S.; VANDENBERGHE, L. *Convex Optimization*. Cambridge University Press, 2004.

CPLEX. IBM ILOG CPLEX. URL `http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer`. 12.04, 2013.

DONG, J.; YANG, N.; CHEN, M. Heuristic Approaches for a TSP Variant: The Automatic Meter Reading Shortest Tour Problem. *Extending the Horizons: Advances in Computing, Optimization, and Decision Technologies*, volume 37, p. 145–163. Springer US, 2007.

GENDREAU, M.; LAPORTE, G.; SEMET, F. The Covering Tour Problem. *Operations Research*, v. 45, n. 4, p. 568–576, 1997.

GULCZYNSKI, D.; HEATH, J.; PRICE, C. The Close Enough Traveling Salesman Problem: A Discussion of Several Heuristics. *Perspectives in Operations Research*, volume 36, p. 271–283. Springer US, 2006.

HÀ, M. H.; BOSTEL, N.; LANGEVIN, A.; ROUSSEAU, L.-M. Solving the close-enough arc routing problem. *Networks*. ISSN 1097-0037. Forthcoming, 2013.

LAND, A. H.; DOIG, A. G. An Automatic Method for Solving Discrete Programming Problems. JÜNGER, M.; LIEBLING, T. M.; NADDEF, D.; NEMHAUSER, G. L.; PULLEYBLANK, W. R.; REINELT, G.; RINALDI, G.; WOLSEY, L. A. (Eds.), *50 Years of Integer Programming 1958-2008*, p. 105–132. Springer Berlin Heidelberg, 1960.

LITTLE, J. D. C.; MURTY, K. G.; SWEENEY, D. W.; KAREL, C. An Algorithm for the Traveling Salesman Problem. *Operations Research*, v. 11, n. 6, p. 972–989, 1963.

LOBO, M.; VANDENBERGHE, L.; BOYD, S.; LEBRET, H. Applications of second-order cone programming. *Linear Algebra and Its Applications*, v. 284, n. 1-3, p. 193–228, 1998.

MENNELL, W. *Heuristics for solving three routing problems: Close-Enough Traveling Salesman Problem, Close-Enough Vehicle Routing Problem, Sequence-Dependent Team Orienteering Problem*. PhD thesis, University of Maryland, College Park, 2009.

MENNELL, W.; GOLDEN, B.; WASIL, E. A Steiner-Zone Heuristic for Solving the Close-Enough Traveling Salesman Problem . *Operations Research, Computing, and Homeland Defense*. 12th INFORMS Computing Society Conference (ICS2011), 2011.

SHUTTLEWORTH, R.; GOLDEN, B.; SMITH, S.; WASIL, E. Advances in Meter Reading: Heuristic Solution of the Close Enough Traveling Salesman Problem over a

Street Network. GOLDEN, B.; RAGHAVAN, S.; WASIL, E. (Eds.), *The Vehicle Routing Problem: Latest Advances and New Challenges*, volume 43 of *Operations Research/Computer Science Interfaces*, p. 487–501. Springer US, 2008.

SILBERHOLZ, J.; GOLDEN, B. The Generalized Traveling Salesman Problem: A New Genetic Algorithm Approach. *Extending the Horizons: Advances in Computing, Optimization, and Decision Technologies*, volume 37 of *Operations Research/Computer Science Interfaces Series*, p. 165–181. Springer US, 2007.

WOLSEY, L. A. *Integer Programming*. Wiley-Interscience, 1998.

YUAN, B.; ORLOWSKA, M.; SADIQ, S. On the optimal robot routing problem in wireless sensor networks. *IEEE Transactions on Knowledge and Data Engineering*, v. 19, n. 9, p. 1252–1261. ISSN 1041-4347, 2007.

Apêndices

Apêndice A

Soluções ótimas para as instâncias propostas por Behdani e Smith (2013)

Tabela A.1: Resultados para as instâncias de Behdani $|V| = 7, 9, 11, 13$, $r = 0.25$

Instância	$\|V\|$	Opt.	Arv.	CPU T(s)
CETSP-6-0	7	28.051	33	0.04
CETSP-6-1	7	33.862	33	0.04
CETSP-6-2	7	27.641	21	0.02
CETSP-6-3	7	25.392	33	0.03
CETSP-6-4	7	36.758	21	0.02
CETSP-6-5	7	22.151	16	0.02
CETSP-6-6	7	27.903	27	0.03
CETSP-6-7	7	34.763	16	0.02
CETSP-6-8	7	24.165	33	0.04
CETSP-6-9	7	28.473	27	0.03
CETSP-6-10	7	34.718	21	0.02
CETSP-6-11	7	31.599	27	0.03
CETSP-6-12	7	33.352	33	0.04
CETSP-6-13	7	26.710	21	0.02
CETSP-6-14	7	28.679	33	0.04
CETSP-6-15	7	34.118	16	0.02
CETSP-6-16	7	23.635	33	0.03
CETSP-6-17	7	22.834	16	0.02
CETSP-6-18	7	28.359	21	0.03
CETSP-6-19	7	33.198	21	0.02
CETSP-6-20	7	34.851	32	0.04
CETSP-6-21	7	32.084	33	0.04
CETSP-6-22	7	27.746	21	0.02
CETSP-6-23	7	29.058	33	0.04
CETSP-6-24	7	33.151	21	0.02
CETSP-6-25	7	30.280	21	0.02
CETSP-6-26	7	30.264	33	0.04
CETSP-6-27	7	30.783	32	0.04
CETSP-6-28	7	35.940	21	0.02
CETSP-6-29	7	41.384	33	0.03
CETSP-8-0	9	32.069	63	0.08
CETSP-8-1	9	35.697	52	0.06
CETSP-8-2	9	25.439	27	0.03
CETSP-8-3	9	37.387	63	0.08
CETSP-8-4	9	29.294	47	0.06
CETSP-8-5	9	33.514	47	0.06
CETSP-8-6	9	25.672	61	0.07
CETSP-8-7	9	30.022	39	0.04
CETSP-8-8	9	33.499	47	0.06
CETSP-8-9	9	38.936	47	0.06
CETSP-8-10	9	30.246	21	0.02
CETSP-8-11	9	29.826	47	0.06
CETSP-8-12	9	33.321	33	0.04
CETSP-8-13	9	30.726	45	0.05
CETSP-8-14	9	36.420	68	0.08
CETSP-8-15	9	32.921	59	0.07
CETSP-8-16	9	36.022	47	0.06
CETSP-8-17	9	31.365	40	0.05
CETSP-8-18	9	32.128	33	0.04
CETSP-8-19	9	35.199	62	0.08
CETSP-8-20	9	30.768	47	0.06
CETSP-8-21	9	33.503	55	0.07
CETSP-8-22	9	41.536	63	0.07
CETSP-8-23	9	34.153	63	0.08
CETSP-8-24	9	33.788	63	0.08
CETSP-8-25	9	39.793	68	0.08
CETSP-8-26	9	37.755	55	0.06
CETSP-8-27	9	35.772	68	0.09
CETSP-8-28	9	31.315	52	0.07
CETSP-8-29	9	34.974	47	0.05

Instância	$\|V\|$	Opt.	Arv.	CPU T(s)
CETSP-10-0	11	37.648	87	0.11
CETSP-10-1	11	37.380	81	0.10
CETSP-10-2	11	46.101	95	0.12
CETSP-10-3	11	31.958	80	0.10
CETSP-10-4	11	35.161	80	0.10
CETSP-10-5	11	36.031	96	0.12
CETSP-10-6	11	36.469	84	0.09
CETSP-10-7	11	35.346	61	0.07
CETSP-10-8	11	34.333	46	0.06
CETSP-10-9	11	30.707	79	0.10
CETSP-10-10	11	35.215	77	0.10
CETSP-10-11	11	31.131	68	0.08
CETSP-10-12	11	36.423	99	0.13
CETSP-10-13	11	36.101	55	0.07
CETSP-10-14	11	34.273	85	0.11
CETSP-10-15	11	32.977	33	0.04
CETSP-10-16	11	31.408	63	0.08
CETSP-10-17	11	36.172	101	0.13
CETSP-10-18	11	39.150	33	0.04
CETSP-10-19	11	40.899	110	0.14
CETSP-10-20	11	33.825	96	0.12
CETSP-10-21	11	41.348	62	0.08
CETSP-10-22	11	38.272	114	0.15
CETSP-10-23	11	35.299	68	0.08
CETSP-10-24	11	40.596	194	0.24
CETSP-10-25	11	37.139	88	0.11
CETSP-10-26	11	39.277	211	0.26
CETSP-10-27	11	32.420	99	0.12
CETSP-10-28	11	37.996	81	0.10
CETSP-10-29	11	35.405	63	0.07
CETSP-12-0	13	39.338	142	0.20
CETSP-12-1	13	36.202	127	0.18
CETSP-12-2	13	47.604	124	0.16
CETSP-12-3	13	35.264	122	0.18
CETSP-12-4	13	35.417	101	0.13
CETSP-12-5	13	41.358	337	0.46
CETSP-12-6	13	37.860	101	0.14
CETSP-12-7	13	36.973	61	0.08
CETSP-12-8	13	40.990	156	0.21
CETSP-12-9	13	34.931	95	0.12
CETSP-12-10	13	42.983	124	0.16
CETSP-12-11	13	37.885	101	0.13
CETSP-12-12	13	34.975	101	0.13
CETSP-12-13	13	39.387	166	0.22
CETSP-12-14	13	35.921	119	0.16
CETSP-12-15	13	42.698	72	0.09
CETSP-12-16	13	40.948	160	0.23
CETSP-12-17	13	37.094	145	0.21
CETSP-12-18	13	40.735	122	0.17
CETSP-12-19	13	40.487	177	0.24
CETSP-12-20	13	35.198	147	0.21
CETSP-12-21	13	41.530	241	0.33
CETSP-12-22	13	39.740	270	0.37
CETSP-12-23	13	35.212	99	0.13
CETSP-12-24	13	38.687	145	0.19
CETSP-12-25	13	40.777	159	0.21
CETSP-12-26	13	34.894	122	0.18
CETSP-12-27	13	36.297	138	0.20
CETSP-12-28	13	37.256	114	0.15
CETSP-12-29	13	37.005	79	0.10

Tabela A.2: Resultados para as instâncias de Behdani $|V| = 15, 17, 19, 21$, $r = 0.25$

Instância	$\|V\|$	**Opt.**	**Arv.**	**CPU T(s)**
CETSP-14-0	15	40.445	209	0.32
CETSP-14-1	15	41.751	231	0.31
CETSP-14-2	15	40.154	388	0.62
CETSP-14-3	15	36.239	123	0.18
CETSP-14-4	15	41.032	454	0.57
CETSP-14-5	15	36.336	166	0.25
CETSP-14-6	15	36.549	93	0.12
CETSP-14-7	15	42.216	210	0.29
CETSP-14-8	15	37.416	168	0.23
CETSP-14-9	15	37.945	99	0.13
CETSP-14-10	15	37.478	170	0.25
CETSP-14-11	15	39.180	200	0.28
CETSP-14-12	15	37.982	214	0.33
CETSP-14-13	15	44.277	137	0.19
CETSP-14-14	15	43.238	210	0.33
CETSP-14-15	15	41.829	165	0.26
CETSP-14-16	15	45.761	226	0.30
CETSP-14-17	15	36.990	461	0.64
CETSP-14-18	15	42.912	563	0.79
CETSP-14-19	15	40.263	421	0.60
CETSP-14-20	15	36.329	147	0.20
CETSP-14-21	15	38.699	146	0.19
CETSP-14-22	15	42.718	307	0.42
CETSP-14-23	15	40.349	102	0.14
CETSP-14-24	15	41.446	137	0.18
CETSP-14-25	15	37.190	100	0.14
CETSP-14-26	15	43.857	192	0.28
CETSP-14-27	15	38.837	215	0.30
CETSP-14-28	15	39.600	186	0.22
CETSP-14-29	15	35.514	118	0.17
CETSP-16-0	17	40.875	261	0.40
CETSP-16-1	17	46.802	140	0.20
CETSP-16-2	17	38.612	286	0.44
CETSP-16-3	17	43.401	213	0.32
CETSP-16-4	17	39.596	344	0.57
CETSP-16-5	17	38.984	186	0.27
CETSP-16-6	17	43.818	432	0.65
CETSP-16-7	17	40.548	588	0.83
CETSP-16-8	17	39.123	196	0.31
CETSP-16-9	17	35.423	170	0.22
CETSP-16-10	17	37.432	236	0.36
CETSP-16-11	17	41.860	336	0.54
CETSP-16-12	17	52.277	374	0.55
CETSP-16-13	17	42.703	198	0.32
CETSP-16-14	17	46.434	288	0.39
CETSP-16-15	17	41.872	1517	2.47
CETSP-16-16	17	41.846	831	1.30
CETSP-16-17	17	39.106	404	0.64
CETSP-16-18	17	45.459	962	1.35
CETSP-16-19	17	43.594	546	0.78
CETSP-16-20	17	46.199	340	0.48
CETSP-16-21	17	41.393	171	0.25
CETSP-16-22	17	38.608	165	0.26
CETSP-16-23	17	43.711	344	0.55
CETSP-16-24	17	40.266	2007	3.00
CETSP-16-25	17	40.601	481	0.76
CETSP-16-26	17	41.155	247	0.36
CETSP-16-27	17	45.308	320	0.51
CETSP-16-28	17	44.838	306	0.44
CETSP-16-29	17	43.588	397	0.66

Instância	$\|V\|$	**Opt.**	**Arv.**	**CPU T(s)**
CETSP-18-0	19	41.322	362	0.60
CETSP-18-1	19	48.079	250	0.35
CETSP-18-2	19	41.234	445	0.72
CETSP-18-3	19	44.681	1259	1.94
CETSP-18-4	19	44.842	238	0.40
CETSP-18-5	19	40.794	211	0.33
CETSP-18-6	19	40.294	798	1.10
CETSP-18-7	19	38.894	173	0.28
CETSP-18-8	19	35.898	140	0.19
CETSP-18-9	19	37.798	264	0.44
CETSP-18-10	19	44.962	295	0.46
CETSP-18-11	19	50.117	722	1.18
CETSP-18-12	19	44.023	354	0.55
CETSP-18-13	19	43.899	507	0.78
CETSP-18-14	19	45.635	1331	2.05
CETSP-18-15	19	42.191	475	0.69
CETSP-18-16	19	45.936	907	1.36
CETSP-18-17	19	45.855	1068	1.66
CETSP-18-18	19	46.274	319	0.48
CETSP-18-19	19	43.434	290	0.44
CETSP-18-20	19	43.548	270	0.38
CETSP-18-21	19	38.735	1246	1.96
CETSP-18-22	19	40.608	291	0.40
CETSP-18-23	19	41.802	284	0.39
CETSP-18-24	19	47.271	1011	1.66
CETSP-18-25	19	45.342	272	0.44
CETSP-18-26	19	47.429	404	0.65
CETSP-18-27	19	43.228	381	0.65
CETSP-18-28	19	42.901	684	1.11
CETSP-18-29	19	46.983	1066	1.53
CETSP-20-0	21	41.663	480	0.84
CETSP-20-1	21	48.582	808	1.30
CETSP-20-2	21	39.706	318	0.56
CETSP-20-3	21	52.071	3147	5.32
CETSP-20-4	21	42.449	541	0.88
CETSP-20-5	21	46.030	479	0.72
CETSP-20-6	21	47.124	996	1.52
CETSP-20-7	21	42.608	313	0.54
CETSP-20-8	21	41.654	712	1.12
CETSP-20-9	21	46.386	419	0.70
CETSP-20-10	21	46.776	1393	2.30
CETSP-20-11	21	46.903	497	0.86
CETSP-20-12	21	43.134	1173	2.01
CETSP-20-13	21	45.156	1678	2.77
CETSP-20-14	21	43.382	930	1.41
CETSP-20-15	21	43.001	541	0.88
CETSP-20-16	21	46.982	917	1.45
CETSP-20-17	21	44.101	356	0.54
CETSP-20-18	21	46.090	397	0.66
CETSP-20-19	21	44.669	3832	6.12
CETSP-20-20	21	47.551	2155	3.52
CETSP-20-21	21	43.062	392	0.62
CETSP-20-22	21	46.051	740	1.22
CETSP-20-23	21	48.816	2133	3.60
CETSP-20-24	21	47.472	413	0.73
CETSP-20-25	21	46.423	1691	2.65
CETSP-20-26	21	48.101	955	1.44
CETSP-20-27	21	45.339	472	0.75
CETSP-20-28	21	48.392	1365	2.29
CETSP-20-29	21	42.662	245	0.37

Tabela A.3: Resultados para as instâncias de Behdani $|V| = 7, 9, 11, 13$, $r = 0.50$

Instância	$\|V\|$	**Opt.**	**Arv.**	**CPU T(s)**
CETSP-6-0	7	27.164	21	0.02
CETSP-6-1	7	32.438	16	0.02
CETSP-6-2	7	26.227	16	0.02
CETSP-6-3	7	23.935	21	0.02
CETSP-6-4	7	35.406	21	0.02
CETSP-6-5	7	20.845	21	0.02
CETSP-6-6	7	26.733	11	0.01
CETSP-6-7	7	33.762	7	0.01
CETSP-6-8	7	23.139	16	0.02
CETSP-6-9	7	26.858	21	0.02
CETSP-6-10	7	33.643	3	0.00
CETSP-6-11	7	30.132	27	0.03
CETSP-6-12	7	31.772	21	0.02
CETSP-6-13	7	25.305	21	0.02
CETSP-6-14	7	27.151	16	0.02
CETSP-6-15	7	33.025	16	0.02
CETSP-6-16	7	22.020	16	0.02
CETSP-6-17	7	21.703	11	0.01
CETSP-6-18	7	27.068	21	0.02
CETSP-6-19	7	32.157	21	0.02
CETSP-6-20	7	33.252	32	0.04
CETSP-6-21	7	30.580	33	0.04
CETSP-6-22	7	26.557	21	0.02
CETSP-6-23	7	28.093	21	0.02
CETSP-6-24	7	31.733	21	0.02
CETSP-6-25	7	29.335	7	0.01
CETSP-6-26	7	28.663	21	0.02
CETSP-6-27	7	28.979	32	0.03
CETSP-6-28	7	34.603	11	0.01
CETSP-6-29	7	39.621	21	0.02
CETSP-8-0	9	30.944	16	0.02
CETSP-8-1	9	34.073	47	0.06
CETSP-8-2	9	24.006	33	0.04
CETSP-8-3	9	35.591	47	0.06
CETSP-8-4	9	27.767	47	0.06
CETSP-8-5	9	31.987	27	0.03
CETSP-8-6	9	24.419	39	0.05
CETSP-8-7	9	27.828	40	0.06
CETSP-8-8	9	31.971	21	0.03
CETSP-8-9	9	37.380	33	0.04
CETSP-8-10	9	28.967	21	0.02
CETSP-8-11	9	28.288	47	0.06
CETSP-8-12	9	31.521	33	0.04
CETSP-8-13	9	28.872	33	0.04
CETSP-8-14	9	34.467	68	0.08
CETSP-8-15	9	31.669	47	0.06
CETSP-8-16	9	34.394	33	0.04
CETSP-8-17	9	30.064	21	0.02
CETSP-8-18	9	30.813	33	0.04
CETSP-8-19	9	33.528	63	0.08
CETSP-8-20	9	29.178	33	0.04
CETSP-8-21	9	31.785	33	0.04
CETSP-8-22	9	40.239	16	0.01
CETSP-8-23	9	32.256	47	0.05
CETSP-8-24	9	32.332	47	0.06
CETSP-8-25	9	38.023	46	0.06
CETSP-8-26	9	35.871	47	0.05
CETSP-8-27	9	34.183	52	0.06
CETSP-8-28	9	29.566	47	0.06
CETSP-8-29	9	33.382	47	0.06

Instância	$\|V\|$	**Opt.**	**Arv.**	**CPU T(s)**
CETSP-10-0	11	36.226	63	0.08
CETSP-10-1	11	36.068	55	0.08
CETSP-10-2	11	43.585	88	0.11
CETSP-10-3	11	29.474	62	0.08
CETSP-10-4	11	32.983	63	0.08
CETSP-10-5	11	33.773	81	0.11
CETSP-10-6	11	33.972	60	0.07
CETSP-10-7	11	33.713	45	0.06
CETSP-10-8	11	32.692	47	0.06
CETSP-10-9	11	29.165	79	0.10
CETSP-10-10	11	33.006	63	0.08
CETSP-10-11	11	29.322	63	0.08
CETSP-10-12	11	34.276	101	0.14
CETSP-10-13	11	34.601	33	0.04
CETSP-10-14	11	32.403	33	0.04
CETSP-10-15	11	31.671	21	0.02
CETSP-10-16	11	29.763	47	0.06
CETSP-10-17	11	34.073	63	0.08
CETSP-10-18	11	37.721	33	0.04
CETSP-10-19	11	38.688	77	0.10
CETSP-10-20	11	32.356	79	0.11
CETSP-10-21	11	39.490	46	0.06
CETSP-10-22	11	36.241	63	0.08
CETSP-10-23	11	33.579	53	0.06
CETSP-10-24	11	38.026	141	0.18
CETSP-10-25	11	35.192	63	0.08
CETSP-10-26	11	36.788	161	0.22
CETSP-10-27	11	30.414	80	0.10
CETSP-10-28	11	36.087	64	0.08
CETSP-10-29	11	33.712	63	0.08
CETSP-12-0	13	37.472	120	0.17
CETSP-12-1	13	34.105	87	0.12
CETSP-12-2	13	45.216	96	0.13
CETSP-12-3	13	33.168	72	0.10
CETSP-12-4	13	33.247	63	0.08
CETSP-12-5	13	38.045	207	0.29
CETSP-12-6	13	36.152	72	0.10
CETSP-12-7	13	35.054	61	0.08
CETSP-12-8	13	38.392	135	0.19
CETSP-12-9	13	32.537	72	0.10
CETSP-12-10	13	40.701	109	0.15
CETSP-12-11	13	35.910	63	0.08
CETSP-12-12	13	33.445	51	0.06
CETSP-12-13	13	36.958	100	0.13
CETSP-12-14	13	33.757	80	0.10
CETSP-12-15	13	40.832	47	0.06
CETSP-12-16	13	38.250	101	0.14
CETSP-12-17	13	35.078	81	0.11
CETSP-12-18	13	38.755	80	0.10
CETSP-12-19	13	38.255	126	0.18
CETSP-12-20	13	32.621	135	0.20
CETSP-12-21	13	38.628	101	0.14
CETSP-12-22	13	36.827	147	0.20
CETSP-12-23	13	32.797	79	0.10
CETSP-12-24	13	36.264	95	0.13
CETSP-12-25	13	38.079	79	0.10
CETSP-12-26	13	33.076	79	0.11
CETSP-12-27	13	34.343	89	0.12
CETSP-12-28	13	35.105	77	0.10
CETSP-12-29	13	35.301	60	0.08

Tabela A.4: Resultados para as instâncias de Behdani $|V| = 15, 17, 19, 21$, $r = 0.50$

Instância	$\|V\|$	**Opt.**	**Arv.**	**CPU T(s)**
CETSP-14-0	15	38.144	145	0.20
CETSP-14-1	15	38.897	102	0.14
CETSP-14-2	15	36.873	224	0.36
CETSP-14-3	15	34.153	81	0.11
CETSP-14-4	15	37.938	266	0.35
CETSP-14-5	15	34.309	80	0.11
CETSP-14-6	15	34.163	94	0.13
CETSP-14-7	15	39.126	138	0.19
CETSP-14-8	15	34.576	155	0.22
CETSP-14-9	15	36.029	101	0.14
CETSP-14-10	15	34.984	124	0.18
CETSP-14-11	15	36.633	98	0.14
CETSP-14-12	15	35.211	109	0.16
CETSP-14-13	15	42.163	78	0.10
CETSP-14-14	15	39.660	172	0.27
CETSP-14-15	15	38.956	120	0.18
CETSP-14-16	15	43.036	129	0.18
CETSP-14-17	15	34.118	361	0.51
CETSP-14-18	15	39.138	193	0.28
CETSP-14-19	15	37.169	232	0.33
CETSP-14-20	15	33.705	129	0.19
CETSP-14-21	15	36.001	114	0.16
CETSP-14-22	15	40.335	293	0.41
CETSP-14-23	15	37.776	96	0.14
CETSP-14-24	15	38.901	100	0.14
CETSP-14-25	15	35.315	80	0.11
CETSP-14-26	15	41.050	142	0.20
CETSP-14-27	15	35.670	96	0.14
CETSP-14-28	15	37.160	178	0.23
CETSP-14-29	15	33.382	81	0.11
CETSP-16-0	17	38.275	201	0.33
CETSP-16-1	17	43.809	122	0.19
CETSP-16-2	17	35.349	172	0.26
CETSP-16-3	17	40.967	145	0.22
CETSP-16-4	17	36.532	185	0.30
CETSP-16-5	17	36.325	127	0.18
CETSP-16-6	17	40.611	276	0.41
CETSP-16-7	17	36.929	389	0.58
CETSP-16-8	17	36.339	169	0.26
CETSP-16-9	17	33.331	114	0.15
CETSP-16-10	17	34.640	135	0.21
CETSP-16-11	17	38.621	140	0.20
CETSP-16-12	17	48.726	149	0.22
CETSP-16-13	17	40.540	111	0.16
CETSP-16-14	17	43.490	174	0.22
CETSP-16-15	17	38.138	632	0.95
CETSP-16-16	17	38.114	317	0.49
CETSP-16-17	17	35.580	216	0.37
CETSP-16-18	17	41.865	426	0.61
CETSP-16-19	17	40.260	255	0.36
CETSP-16-20	17	43.267	182	0.26
CETSP-16-21	17	38.757	97	0.13
CETSP-16-22	17	36.616	80	0.12
CETSP-16-23	17	40.556	206	0.32
CETSP-16-24	17	36.057	724	1.08
CETSP-16-25	17	37.188	166	0.27
CETSP-16-26	17	38.817	186	0.26
CETSP-16-27	17	41.874	184	0.29
CETSP-16-28	17	41.669	177	0.26
CETSP-16-29	17	40.348	209	0.35

Instância	$\|V\|$	**Opt.**	**Arv.**	**CPU T(s)**
CETSP-18-0	19	38.647	203	0.34
CETSP-18-1	19	44.777	167	0.27
CETSP-18-2	19	38.075	274	0.42
CETSP-18-3	19	41.214	793	1.30
CETSP-18-4	19	41.760	197	0.33
CETSP-18-5	19	38.092	150	0.23
CETSP-18-6	19	37.199	661	0.97
CETSP-18-7	19	36.395	147	0.23
CETSP-18-8	19	33.670	109	0.16
CETSP-18-9	19	34.848	152	0.23
CETSP-18-10	19	41.544	145	0.22
CETSP-18-11	19	46.029	324	0.53
CETSP-18-12	19	40.859	181	0.27
CETSP-18-13	19	40.881	424	0.71
CETSP-18-14	19	42.097	1089	1.71
CETSP-18-15	19	39.184	342	0.53
CETSP-18-16	19	42.831	854	1.40
CETSP-18-17	19	41.837	408	0.64
CETSP-18-18	19	43.036	124	0.18
CETSP-18-19	19	40.200	191	0.30
CETSP-18-20	19	40.761	218	0.33
CETSP-18-21	19	35.102	665	1.18
CETSP-18-22	19	37.855	208	0.29
CETSP-18-23	19	38.742	152	0.22
CETSP-18-24	19	43.602	495	0.82
CETSP-18-25	19	42.263	146	0.22
CETSP-18-26	19	44.130	118	0.18
CETSP-18-27	19	39.536	195	0.32
CETSP-18-28	19	38.639	293	0.50
CETSP-18-29	19	43.128	515	0.73
CETSP-20-0	21	38.739	258	0.46
CETSP-20-1	21	45.003	327	0.52
CETSP-20-2	21	35.925	272	0.50
CETSP-20-3	21	47.701	1063	1.74
CETSP-20-4	21	38.803	334	0.58
CETSP-20-5	21	42.775	436	0.70
CETSP-20-6	21	43.403	452	0.70
CETSP-20-7	21	39.752	257	0.47
CETSP-20-8	21	38.485	287	0.44
CETSP-20-9	21	42.688	193	0.31
CETSP-20-10	21	42.292	424	0.72
CETSP-20-11	21	43.559	236	0.36
CETSP-20-12	21	39.616	628	1.10
CETSP-20-13	21	40.837	550	0.91
CETSP-20-14	21	39.776	393	0.58
CETSP-20-15	21	39.736	365	0.59
CETSP-20-16	21	43.453	483	0.75
CETSP-20-17	21	40.772	208	0.31
CETSP-20-18	21	42.364	232	0.40
CETSP-20-19	21	40.147	1180	1.91
CETSP-20-20	21	43.641	718	1.18
CETSP-20-21	21	40.437	223	0.33
CETSP-20-22	21	42.471	468	0.80
CETSP-20-23	21	44.577	709	1.23
CETSP-20-24	21	44.272	281	0.52
CETSP-20-25	21	42.111	722	1.14
CETSP-20-26	21	44.167	417	0.64
CETSP-20-27	21	41.926	324	0.55
CETSP-20-28	21	44.357	474	0.80
CETSP-20-29	21	39.296	200	0.32

Tabela A.5: Resultados para as instâncias de Behdani $|V| = 7, 9, 11, 13$, $r = 1.0$

Instância	$\|V\|$	**Opt.**	**Arv.**	**CPU T(s)**
CETSP-6-0	7	25.492	21	0.03
CETSP-6-1	7	30.070	7	0.01
CETSP-6-2	7	23.583	7	0.01
CETSP-6-3	7	21.242	7	0.01
CETSP-6-4	7	32.872	11	0.02
CETSP-6-5	7	18.328	33	0.04
CETSP-6-6	7	24.583	11	0.01
CETSP-6-7	7	31.874	7	0.01
CETSP-6-8	7	21.429	7	0.01
CETSP-6-9	7	24.651	3	0.01
CETSP-6-10	7	31.547	3	0.01
CETSP-6-11	7	27.599	7	0.01
CETSP-6-12	7	28.757	21	0.02
CETSP-6-13	7	22.859	11	0.02
CETSP-6-14	7	24.410	7	0.01
CETSP-6-15	7	30.915	21	0.02
CETSP-6-16	7	19.410	3	0.00
CETSP-6-17	7	19.719	3	0.01
CETSP-6-18	7	24.642	21	0.02
CETSP-6-19	7	30.289	1	0.01
CETSP-6-20	7	30.112	21	0.02
CETSP-6-21	7	27.666	33	0.04
CETSP-6-22	7	24.699	11	0.02
CETSP-6-23	7	26.323	21	0.02
CETSP-6-24	7	29.251	11	0.01
CETSP-6-25	7	27.683	3	0.01
CETSP-6-26	7	25.540	21	0.02
CETSP-6-27	7	25.691	32	0.04
CETSP-6-28	7	32.081	11	0.02
CETSP-6-29	7	36.183	21	0.02
CETSP-8-0	9	28.952	16	0.02
CETSP-8-1	9	31.040	40	0.05
CETSP-8-2	9	21.442	11	0.02
CETSP-8-3	9	32.872	11	0.02
CETSP-8-4	9	25.130	22	0.03
CETSP-8-5	9	29.458	7	0.01
CETSP-8-6	9	22.370	21	0.03
CETSP-8-7	9	24.883	16	0.02
CETSP-8-8	9	29.024	16	0.02
CETSP-8-9	9	34.773	21	0.03
CETSP-8-10	9	26.870	7	0.01
CETSP-8-11	9	25.265	47	0.06
CETSP-8-12	9	29.043	21	0.02
CETSP-8-13	9	25.410	33	0.04
CETSP-8-14	9	30.538	69	0.08
CETSP-8-15	9	29.667	33	0.04
CETSP-8-16	9	31.628	16	0.02
CETSP-8-17	9	27.788	11	0.02
CETSP-8-18	9	28.266	33	0.04
CETSP-8-19	9	30.907	21	0.03
CETSP-8-20	9	26.556	21	0.03
CETSP-8-21	9	29.067	33	0.04
CETSP-8-22	9	37.842	12	0.02
CETSP-8-23	9	28.988	27	0.03
CETSP-8-24	9	29.801	7	0.01
CETSP-8-25	9	34.837	46	0.06
CETSP-8-26	9	32.499	33	0.04
CETSP-8-27	9	31.435	26	0.03
CETSP-8-28	9	26.912	16	0.02
CETSP-8-29	9	31.118	7	0.01

Instância	$\|V\|$	**Opt.**	**Arv.**	**CPU T(s)**
CETSP-10-0	11	33.676	47	0.06
CETSP-10-1	11	33.975	22	0.03
CETSP-10-2	11	39.391	45	0.06
CETSP-10-3	11	25.712	21	0.03
CETSP-10-4	11	29.385	47	0.06
CETSP-10-5	11	30.549	33	0.04
CETSP-10-6	11	29.905	52	0.06
CETSP-10-7	11	31.081	16	0.02
CETSP-10-8	11	30.131	21	0.03
CETSP-10-9	11	26.385	72	0.10
CETSP-10-10	11	29.958	27	0.04
CETSP-10-11	11	25.754	46	0.06
CETSP-10-12	11	31.008	52	0.06
CETSP-10-13	11	31.673	33	0.04
CETSP-10-14	11	29.243	16	0.02
CETSP-10-15	11	29.146	21	0.02
CETSP-10-16	11	26.944	33	0.04
CETSP-10-17	11	30.583	47	0.06
CETSP-10-18	11	35.006	18	0.03
CETSP-10-19	11	35.341	33	0.04
CETSP-10-20	11	29.807	16	0.03
CETSP-10-21	11	36.794	21	0.03
CETSP-10-22	11	33.030	55	0.07
CETSP-10-23	11	30.539	47	0.06
CETSP-10-24	11	33.796	63	0.08
CETSP-10-25	11	31.857	33	0.04
CETSP-10-26	11	32.308	78	0.11
CETSP-10-27	11	27.130	58	0.08
CETSP-10-28	11	32.835	25	0.03
CETSP-10-29	11	31.368	33	0.04
CETSP-12-0	13	34.781	40	0.04
CETSP-12-1	13	31.290	15	0.02
CETSP-12-2	13	41.043	63	0.08
CETSP-12-3	13	29.392	47	0.06
CETSP-12-4	13	29.861	33	0.04
CETSP-12-5	13	32.559	120	0.18
CETSP-12-6	13	34.125	7	0.01
CETSP-12-7	13	31.965	47	0.06
CETSP-12-8	13	34.148	80	0.11
CETSP-12-9	13	28.670	63	0.08
CETSP-12-10	13	36.543	70	0.09
CETSP-12-11	13	32.700	39	0.05
CETSP-12-12	13	30.844	37	0.05
CETSP-12-13	13	33.012	67	0.09
CETSP-12-14	13	30.262	33	0.04
CETSP-12-15	13	37.461	33	0.04
CETSP-12-16	13	34.406	33	0.04
CETSP-12-17	13	32.065	33	0.04
CETSP-12-18	13	35.911	33	0.04
CETSP-12-19	13	34.633	63	0.08
CETSP-12-20	13	29.612	63	0.08
CETSP-12-21	13	34.539	46	0.06
CETSP-12-22	13	32.308	78	0.11
CETSP-12-23	13	29.230	47	0.06
CETSP-12-24	13	31.748	81	0.11
CETSP-12-25	13	34.127	62	0.08
CETSP-12-26	13	29.896	32	0.04
CETSP-12-27	13	30.937	60	0.08
CETSP-12-28	13	31.753	54	0.07
CETSP-12-29	13	32.269	47	0.06

Tabela A.6: Resultados para as instâncias de Behdani $|V| = 15, 17, 19, 21$, $r = 1.0$

Instância	$\|V\|$	**Opt.**	**Arv.**	**CPU T(s)**
CETSP-14-0	15	34.891	63	0.09
CETSP-14-1	15	34.924	33	0.04
CETSP-14-2	15	32.633	47	0.06
CETSP-14-3	15	31.126	47	0.06
CETSP-14-4	15	32.846	87	0.12
CETSP-14-5	15	31.186	47	0.06
CETSP-14-6	15	30.177	78	0.11
CETSP-14-7	15	34.487	79	0.11
CETSP-14-8	15	29.768	112	0.16
CETSP-14-9	15	33.177	21	0.03
CETSP-14-10	15	30.981	72	0.10
CETSP-14-11	15	32.312	81	0.12
CETSP-14-12	15	30.852	63	0.08
CETSP-14-13	15	38.265	46	0.06
CETSP-14-14	15	35.034	81	0.11
CETSP-14-15	15	35.122	81	0.12
CETSP-14-16	15	38.485	64	0.09
CETSP-14-17	15	29.067	158	0.24
CETSP-14-18	15	33.622	80	0.11
CETSP-14-19	15	32.327	98	0.14
CETSP-14-20	15	29.732	72	0.10
CETSP-14-21	15	32.089	80	0.12
CETSP-14-22	15	35.956	91	0.13
CETSP-14-23	15	33.998	59	0.08
CETSP-14-24	15	34.632	54	0.08
CETSP-14-25	15	32.228	47	0.06
CETSP-14-26	15	36.858	84	0.11
CETSP-14-27	15	31.327	47	0.07
CETSP-14-28	15	32.964	115	0.16
CETSP-14-29	15	30.086	47	0.07
CETSP-16-0	17	34.891	63	0.09
CETSP-16-1	17	39.564	63	0.09
CETSP-16-2	17	30.317	54	0.08
CETSP-16-3	17	36.763	109	0.16
CETSP-16-4	17	32.754	61	0.09
CETSP-16-5	17	32.136	101	0.14
CETSP-16-6	17	35.219	212	0.31
CETSP-16-7	17	30.988	208	0.36
CETSP-16-8	17	32.571	47	0.06
CETSP-16-9	17	30.038	33	0.05
CETSP-16-10	17	31.029	63	0.09
CETSP-16-11	17	34.141	79	0.12
CETSP-16-12	17	43.595	101	0.16
CETSP-16-13	17	37.177	63	0.09
CETSP-16-14	17	38.259	73	0.12
CETSP-16-15	17	32.576	169	0.27
CETSP-16-16	17	32.729	118	0.18
CETSP-16-17	17	30.857	101	0.17
CETSP-16-18	17	36.428	180	0.27
CETSP-16-19	17	35.113	95	0.14
CETSP-16-20	17	38.367	101	0.16
CETSP-16-21	17	34.724	55	0.08
CETSP-16-22	17	33.343	46	0.08
CETSP-16-23	17	36.513	101	0.15
CETSP-16-24	17	29.977	118	0.18
CETSP-16-25	17	32.967	63	0.09
CETSP-16-26	17	34.358	130	0.20
CETSP-16-27	17	36.953	112	0.17
CETSP-16-28	17	36.534	100	0.14
CETSP-16-29	17	36.786	21	0.03

Instância	$\|V\|$	**Opt.**	**Arv.**	**CPU T(s)**
CETSP-18-0	19	35.121	63	0.09
CETSP-18-1	19	39.579	91	0.14
CETSP-18-2	19	32.945	101	0.15
CETSP-18-3	19	35.849	170	0.24
CETSP-18-4	19	37.675	47	0.06
CETSP-18-5	19	34.073	86	0.12
CETSP-18-6	19	31.038	300	0.47
CETSP-18-7	19	32.758	47	0.07
CETSP-18-8	19	30.074	63	0.09
CETSP-18-9	19	31.082	54	0.07
CETSP-18-10	19	36.530	98	0.15
CETSP-18-11	19	40.579	146	0.22
CETSP-18-12	19	36.199	99	0.15
CETSP-18-13	19	35.674	191	0.28
CETSP-18-14	19	35.797	189	0.29
CETSP-18-15	19	34.262	108	0.16
CETSP-18-16	19	36.065	215	0.34
CETSP-18-17	19	35.929	131	0.21
CETSP-18-18	19	38.803	47	0.07
CETSP-18-19	19	36.045	87	0.13
CETSP-18-20	19	36.313	96	0.14
CETSP-18-21	19	29.907	81	0.12
CETSP-18-22	19	33.977	172	0.24
CETSP-18-23	19	33.849	101	0.15
CETSP-18-24	19	37.888	200	0.34
CETSP-18-25	19	37.034	91	0.14
CETSP-18-26	19	39.959	33	0.05
CETSP-18-27	19	35.011	63	0.09
CETSP-18-28	19	32.906	135	0.22
CETSP-18-29	19	36.725	200	0.30
CETSP-20-0	21	35.182	80	0.12
CETSP-20-1	21	39.549	81	0.12
CETSP-20-2	21	31.125	101	0.16
CETSP-20-3	21	41.023	225	0.36
CETSP-20-4	21	33.327	147	0.24
CETSP-20-5	21	36.919	184	0.32
CETSP-20-6	21	37.616	185	0.29
CETSP-20-7	21	35.581	91	0.13
CETSP-20-8	21	33.817	103	0.16
CETSP-20-9	21	37.473	79	0.11
CETSP-20-10	21	35.743	192	0.33
CETSP-20-11	21	38.366	137	0.22
CETSP-20-12	21	34.488	321	0.52
CETSP-20-13	21	34.494	157	0.26
CETSP-20-14	21	34.645	135	0.20
CETSP-20-15	21	34.790	163	0.22
CETSP-20-16	21	37.822	189	0.30
CETSP-20-17	21	36.094	109	0.16
CETSP-20-18	21	36.882	146	0.24
CETSP-20-19	21	32.868	197	0.31
CETSP-20-20	21	38.084	276	0.44
CETSP-20-21	21	35.649	166	0.28
CETSP-20-22	21	36.922	154	0.25
CETSP-20-23	21	38.927	254	0.45
CETSP-20-24	21	40.088	97	0.15
CETSP-20-25	21	35.367	250	0.42
CETSP-20-26	21	38.045	139	0.21
CETSP-20-27	21	36.404	356	0.62
CETSP-20-28	21	38.698	86	0.13
CETSP-20-29	21	34.393	86	0.12