Heurísticas para o Problema do Ciclo Dominante com Coleta de Prêmios

Luis Henrique Pauleti Mendes
Instituto de Computação
Universidade Estadual de Campinas (UNICAMP) Campinas - SP - Brasil
luishpmendes@gmail.com

Fábio Luiz Usberti
Instituto de Computação
Universidade Estadual de Campinas (UNICAMP) Campinas - SP - Brasil
fusberti@ic.unicamp.br

RESUMO

Este trabalho propõem metodologias heurísticas para resolver o Problema do Ciclo Dominante com Coleta de Prêmios (PDCDP). Este problema consiste na composição de dois problemas NP-difíceis: o Problema do Conjunto Dominante e o Problema do Caixeiro Viajante. Brevemente, o objetivo do PDCDP consiste em encontrar um ciclo de custo mínimo em um grafo não-direcionado. O ciclo é trafegado por um viajante que necessita visitar um conjunto de clientes (vértices dominantes). A motivação do PCDCP consiste em sua aplicação prática para empresas de transporte, distribuição e coleta de produtos; em particular, no processo de definição de rotas para realização desses serviços.

PALAVRAS CHAVE. Conjunto Dominante em Grafos, Programação Inteira, Meta-heurística.

ABSTRACT

This work proposes heuristic methodologies to solve the Prize Collecting Dominating Cycle Problem (PCDCP). This problem is the composition of two NP-hard problems: the Dominating Set Problem and the Traveling Salesman Problem. Briefly, the aim of the PCDCP is to find a minimum cost cycle in an undirected graph. The cycle is traversed by a traveler who needs to visit a set of customers (dominant vertices). The motivation of the PCDCP consists in its practical application to companies that deal with transportation, distribution and collection of products; particularly, in the process of definition of the routes to carry out these services.

KEYWORDS. Dominating Set in Graphs, Integer Programming, Metaheuristic.

1. Introdução

Em teoria dos grafos, em particular nos ramos de problemas de dominação e problemas de roteamento, existem dois problemas representativos na literatura que já foram amplamente investigados: o Problema do Conjunto Dominante, do inglês *Dominating Set Problem* (DSP) e o Problema do Caixeiro Viajante, do inglês *Traveling Salesman Problem* (TSP). Dominação em teoria dos grafos é um modelo natural para vários problemas do mundo real. Haynes et al. [1998], ilustram vários problemas interessantes, incluindo roteamento de ônibus escolares, redes de computadores e estações de rádio. Também existem diversos tipos de aplicações no mundo real referente a problemas de roteamento, como coleta de lixo, serviços de correio postal, operações de frete, entrega de mercadorias, coleta de consumo de energia residencial, dentre outros.

As soluções para esses problemas são focadas em minimizar os custos logísticos, o que possibilita uma economia significativa para as entidades provedoras desses serviços. Porém, grande parte desses problemas são de difícil solução, o que motiva o interesse na pesquisa de algoritmos heurísticos que obtenham soluções de boa qualidade na prática.

Dado um grafo não-direcionado $G = (V, E)$, o DSP consiste em encontrar um subconjunto $D \subseteq V$ tal que cada vértice que não está em D é adjacente a algum vértice em D. Uma extensão do DSP, denominado k-DSP, generaliza a definição de conjunto dominante a partir do conceito de vizinhança entre vértices. No DSP um vértice u é vizinho de um vértice v se e somente se u é adjacente a v. Já no k-DSP, um vértice u é vizinho de um vértice v se e somente se $d(u, v) \leq k$, onde $d(u, v)$ é a distância (ou custo) de um caminho mínimo entre os vértices u e v, enquanto k é um parâmetro denominado raio de vizinhança.

O DSP já foi intensivamente estudado na literatura. Chang [1998] investigou diferentes abordagens de dominação em diversos tipos de grafos e Chen et al. [2004] estudaram o Problema do Conjunto Dominante com funções de medida e estenderam os resultados para grafos ponderados.

O TSP é um problema que consiste em encontrar um ciclo C em um grafo não-direcionado $G = (V, E)$ que passa por todos os vértices em V, tal que a distância (ou custo) total percorrida seja mínima. O TSP pode ser informalmente descrito como o problema de um vendedor ambulante que, partindo de uma cidade origem, precisa visitar um conjunto de cidades para comercializar seus produtos percorrendo a distância mínima. Existem estradas que ligam todas as cidades e cada cidade deve ser visitada somente uma vez. Após o término da viagem, o vendedor ambulante precisa retornar para a cidade de origem.

O TSP também já foi muito estudado na literatura. Chatterjee et al. [1996] e Laporte et al. [2000] mostram que o TSP pode ser aplicado em muitos problemas reais de diversas áreas do conhecimento, como logística, genética, manufatura e telecomunicações. Muitas metodologias exatas, aproximadas e heurísticas já foram propostas para o TSP, algumas das quais são exploradas por Applegate et al. [2011].

Neste trabalho foi estudado pela primeira vez o Problema do Ciclo Dominante com Coleta de Prêmios, do inglês *Prize Collecting Dominating Cycle Problem* (PCDCP). Trata-se de um problema NP-difícil que generaliza o DSP e o TSP. Problemas similares ao PCDCP já foram estudados na literatura. Bienstock et al. [1993] estudaram o Problema do Caixeiro Viajante com Coleta de Prêmios, do inglês *Prize Collecting Traveling Salesman Problem* (PCTSP), apresentando um algoritmo aproximado baseado no algoritmo de Christofides para o TSP. Snyder e Daskin [2005] estudaram o Problema do Caixeiro Viajante Generalizado, do inglês, *Generalized Traveling Salesman Problem* (GTSP), apresentando uma heurística eficaz que combina um algoritmo genético com uma busca local. Current e Schilling [1989] e Maziero et al. [2015] estudaram o Problema do Ciclo Dominante, do inglês *Dominating Cycle Problem* (DCP), apresentando formulações matemáticas e métodos heurísticos para resolvê-lo.

2. O Problema do Ciclo Dominante com Coleta de Prêmios

A definição do PCDCP é fornecida a seguir. Seja k um número não-negativo e $G = (V, E)$ um grafo onde V representa o conjunto de vértices, E o conjunto de arestas e $u \in V$ o vértice inicial. Associado a cada aresta $e \in E$ há um custo não-negativo c_e e a cada vértice $v \in V$ uma penalidade não-negativa π_v. Um subconjunto de vértices $D \subseteq V$ é um conjunto k-dominante de G se para cada vértice $v \in V \backslash D$ existe um vértice $w \in D$ tal que $d(w, v) \leq k$, onde $d(w, v)$ é a distância (ou custo) de um caminho mínimo entre os vértices w e v, enquanto k é o raio de vizinhança. O PCDCP tem por objetivo encontrar um ciclo de custo mínimo que também seja um conjunto k-dominante de G e que contenha o vértice inicial u. O custo do ciclo é composto pela soma dos custos de suas arestas e pela soma das penalidades dos nós não visitados pelo ciclo.

O PCDCP apresenta aplicações em problemas de logística e de distribuição de bens e serviços. Em particular, há uma forte relação entre o PCDCP e a logística de *unidades móveis de atendimento* para serviços de diversas natureza (médica, odontológica, veterinária, jurídica). Considere um cenário onde deseja-se determinar a rota de uma unidade móvel de atendimento para um conjunto de clientes dispersos geograficamente. Essa rota deve apresentar um conjunto de paradas que serão efetuadas ao longo do dia visando atender os clientes das proximidades. Além do objetivo de minimizar a distância total percorrida pela rota, é necessário que cada cliente esteja a uma distância adequada (menor que k) de pelo um dos pontos de parada. Supondo que clientes possam apresentar prioridades uns com relação aos outros, considera-se a aplicação de uma penalidade quando um cliente precisa se deslocar para uma unidade de atendimento.

Uma vez que o PCDCP é uma generalização do problema do ciclo dominante (DCP), problema do caixeiro viajante (TSP) e problema do conjunto dominante (DSP), consequentemente o PCDCP é NP-difícil. Desse modo, a solução exata do PCDCP requer tempo exponencial em função do tamanho da entrada, exceto se P = NP. No entanto, existem métodos de otimização denominados *heurísticas* que, apesar de não haver garantias teóricas de que possam obter uma solução ótima, possuem boa atuação na prática, obtendo soluções não triviais de boa qualidade.

3. Metodologias

Nesta seção serão apresentadas três metodologias heurísticas para solução do PCDCP. A primeira é denominada ”metodologia de duas fases”, uma fase para determinar um conjunto dominante e a outra fase para otimizar a rota sobre o conjunto dominante. A segunda metodologia consiste na implementação de uma meta-heurística GRASP. A terceira metodologia consiste em uma meta-heurística evolutiva, baseada em algoritmos genéticos

3.1. Metodologia de Duas Fases

A primeira metodologia heurística de solução proposta para o PCDCP é baseada na heurística para o DCP proposta por Current e Schilling [1989], que consiste em um método de duas fases: a primeira fase resolve o k-DSP, resultando em um subconjunto de vértices que dominam o grafo com penalidade mínima. A segunda fase busca resolver o TSP sobre os vértices pertencentes ao conjunto dominante obtido na primeira fase. Cabe observar que o ciclo obtido na segunda fase trata-se de uma solução viável (não necessariamente ótima) para o PCDCP, uma vez que seus vértices dominam o grafo.

Fase 1. Propõem-se o seguinte modelo PLI para a Fase 1:
Instância:

Um grafo $G = (V, E)$, um raio de vizinhança não-negativo k, um vértice inicial $u \in V$ e uma penalidade não-negativa π_v para cada $v \in V$.
Variáveis:

Para cada vértice $v \in V$, define-se uma variável binária y_v tal que $y_v = 1$ se e somente se o vértice v está na solução.

Função Objetivo:

$$min\{\sum_{vinV} \frac{-(1-y_v)}{\pi_v + 1}\} \tag{1}$$

Restrições:

$$y_u = 1 \tag{2}$$

$$\sum_{w \in N_k(v)} y_w \geq 1, \forall v \in V \tag{3}$$

A fase 1 tem por objetivo encontrar um conjunto k-dominante não trivial que seja interessante do ponto de vista das penalidades dos vértices dominados. Para isso, primeiramente formulamos a fase 1 utilizando as restrições 3 que garantem que a solução seja com conjunto k-dominante. A função objetivo 1 adota penalidades alteradas ϕ_v para os vértices dominados ($y_v = 0$). A penalidade alterada é da forma $\phi_v = \frac{-1}{\pi_v + 1}$, onde o termo "+1"é incluído para evitar divisão por zero na ocasião de penalidade original nula. Cabe notar que, como as penalidade alteradas são negativas e inversas às penalidade originais e a função objetivo é de minimização, o modelo é orientado a buscar um conjunto maximal de vértices dominados que minimizam a soma das penalidades originais.

Fase 2. Propõem-se o seguinte modelo PLI para a Fase 2:
Instância:

Um grafo $G = (V, E)$, um conjunto k-dominante D, um custo não-negativo c_e para cada aresta $e \in E$ e uma penalidade não-negativa π_v para cada $v \in V$.
Variáveis:

Para cada vértice $v \in V$, define-se uma variável binária y_v tal que $y_v = 1$ se e somente se o vértice v está na solução.

Para cada aresta $e \in E$, define-se uma variável binária x_e tal que $x_e = 1$ se e somente se a aresta e está na solução.
Função Objetivo:

$$min\{\sum_{e \in E} c_e x_e + \sum_{vinV} \pi_v(1 - y_v)\} \tag{4}$$

Restrições:

$$y_v = 1, \forall v \in D \tag{5}$$

$$\sum_{e \in \delta(\{v\})} x_e = 2y_v, \forall v \in V \tag{6}$$

$$\sum_{e \in \delta(S)} x_e \leq |S| - 1, \forall S \subset V \tag{7}$$

A fase 2 tem por objetivo encontrar um ciclo gerador que contenha os vértices do conjunto k-dominante D encontrado pela fase 1. Assim como na metodologia exata, a função objetivo 4 busca por uma solução de custo mínimo, onde o custo de uma solução é dado pelo somatório dos custos associados às arestas que pertencem a solução mais o somatório das penalidades pagas por vértices não visitados pela solução. As restrições 5 garantem que os vértices do conjunto k-dominante D façam parte da solução. As restrições 6 garantem que cada vértice v da solução será visitado apenas uma única vez. As restrições 7 garantem a não ocorrência de ciclos desconexos, ou seja, garantem que a solução consista de apenas um ciclo.

3.2. Metodologia GRASP

A segunda metodologia heurística proposta para o PCDCP é baseada na metodologia para o DCP proposta por Maziero et al. [2015]. Considerando que a solução exata para o PCDCP requer tempo exponencial, exceto se $P = NP$, propõe-se uma metodologia de solução para o PCDCP através da implementação da meta-heurística GRASP. Nesse sentido, para as instâncias onde a aplicação de uma metodologia exata se torna computacionalmente inviável, a meta-heurística torna-se uma alternativa para a obtenção de soluções de boa qualidade (sem garantias de otimalidade), em tempo computacional adequado.

Resende e Ribeiro [2002] apresentam a meta-heurística GRASP (*Greedy Randomized Adaptative Search Procedure*) como um processo iterativo, no qual cada iteração consiste de duas fases: construção e busca local. A fase de construção constrói uma solução factível, cuja vizinhança é investigada até que um mínimo local é encontrado durante a fase de busca local. Após certo número de iterações, a melhor solução encontrada é retornada como solução final da meta-heurística.

Construção Aleatória Gulosa. Na fase de construção da solução inicial, os elementos candidatos a participarem da solução inicial são escolhidos de modo aleatório e guloso até que a solução inicial seja uma solução completa. Este processo se realiza a partir da criação de uma lista de elementos candidatos a entrar na solução. Esta lista é então reduzida, criando-se uma lista restrita de candidatos (*restricted candidate list*, RCL) que contém os melhores candidatos a entrar na solução. A partir da lista RCL, os candidatos que efetivamente entram na solução são escolhidos aleatoriamente com distribuição uniforme.

No caso do PCDCP, a solução inicial S consiste em um ciclo e os elementos candidatos são os vértices em $V \setminus S$. A cada vértice $v \in V \setminus S$ é associado um custo incremental $c(S, v)$ que representa o custo mínimo de adicionar v em S, ou seja, $c(S, v) = \min_{(x,y)\in S} -\pi_v + D(x, v) + D(v, y)$ onde x e y são vértices adjacentes em S e $D(u, v)$ representa a distância do caminho mínimo entre u e v em G.

Neste trabalho, a RCL está associada com um parâmetro de limite $\alpha \in [0, 1]$. A lista de candidatos restrita é formada por todos os vértices $v \in V$ que podem ser inseridos na solução parcial S em construção sem destruir a viabilidade da solução e cuja qualidade é superior ao valor limite, isto é, $c(S, v) \in [c_{min}, c_{min} + \alpha(c_{max} - c_{min})]$.

Como o vértice u tem de fazer parte da solução necessariamente, S é inicializado como um ciclo degenerado contendo apenas o vértice u. A cada iteração, um vértice v de custo incremental $c(S, v)$ é adicionado à solução S, juntamente com os vértices dos caminhos mínimos que conectam v à S. O processo termina quando S constitui um conjunto k-dominante do grafo G.

Operadores de Busca Local. As soluções encontradas no primeiro estágio do GRASP não são necessariamente ótimos locais, logo a segunda fase passa a explorar a vizinhança dessas soluções. Uma vizinhança é definida como um conjunto de soluções que são alcançáveis a partir de uma solução inicial após a aplicação de um operador de busca local. Neste trabalho, foram implementados três operadores buscas locais: exclusão, troca e 2-OPT. Foi adotada a heurística de primeira melhora, do inglês *first improvement*, ou seja, move-se para a primeira solução que apresenta uma melhora no custo. O processo de busca local para quando nenhum operador de busca local consegue melhorar o custo da solução, neste caso dizemos que chegamos em um ótimo local.

Exclusão. Este operador de busca local tem por objetivo excluir da solução S vértices dominantes redundantes. Conta-se quantas vezes cada vértices ocorre na solução S e por quantos vértices dominantes cada vértice do grafo é dominado. A seguir, cada vértice v da solução S é analisado, verificando-se se é possível removê-lo e se sua remoção melhora o custo da solução S. O vértice v pode ser removido da solução se ele não for a única ocorrência do vértice inicial u e se todos os vértices dominados por v são dominados por pelo menos dois vértices, ou seja, a solução

continuará dominando todos os vértices do grafo mesmo que v seja removido. Para verificar se a remoção do vértice v da solução S melhora seu custo, analisa-se a variação $\Delta_{c_e}(S, v)$ no custo de S que a remoção de v proporciona, que depende de x e y, o predecessor e o sucessor de v em S, respectivamente. Se houver apenas uma ocorrência de v em S, temos que $\Delta_{c_e}(S, v) = \pi_v + c_{xy} - c_{xv} - c_{vy}$, caso contrário, se houver mais de uma ocorrência de v em S, temos que $\Delta_{c_e}(S, v) = c_{xy} - c_{xv} - c_{vy}$. Temos que a remoção do vértice v da solução S melhora o custo de S se $\Delta_{c_e}(S, v) < 0$, pois assim obteremos uma nova solução de custo menor.

Troca. Como o próprio nome indica, este operador de busca local tenta trocar os vértices da solução S por outros. Assim como no operador da seção **??**, conta-se quantas vezes cada vértice ocorre na S. A seguir, cada vértice v da solução S é analisado, verificando-se se é possível trocá-lo por outro vértice $w \in N_k(v)$ e se a troca melhora o custo da solução S. O vértice v pode ser trocado por w em S se ele não for a única ocorrência do vértice inicial u e se $N_k(v) \subseteq N_k(w)$, ou seja, se o vértice w domina todos os vértices dominados por v, de modo que a troca de v por w mantenha a propriedade de S ser um conjunto k-dominante do grafo. Para verificar se a troca do vértice v por w melhora o custo da solução S, analisa-se a variação $\Delta_{c_t}(S, v, w)$ no custo de S que tal troca proporciona, que depende de x e y, o predecessor e o sucessor de v em S, respectivamente. Temos que $\Delta_{c_t}(S, v, w) = c_{xw} + c_{wy} - c_{xv} - c_{vy}$; se houver apenas uma ocorrência de v em S, temos um acréscimo de π_v em $\Delta_{c_t}(S, v, w)$; analogamente, se não houver nenhuma ocorrência de w em S, temos um decréscimo de π_w em $\Delta_{c_t}(S, v, w)$. Temos que a troca do vértice v pelo vértice w na solução S melhora o custo de S se $\Delta_{c_t}(S, v, w) < 0$, pois assim obteremos uma nova solução de custo menor.

2-OPT. Este operador de busca local é um dos operadores clássicos para o TSP, sendo primeiramente proposto por Croes [1958]. Diferentemente dos dois operadores vistos anteriormente, o 2-opt tenta trocar as arestas da solução S. Analisa-se cada par de arestas $e = vw$ e $f = xy$ e verifica-se se é possível trocá-las pelas arestas $e' = vx$ e $f' = wy$ e se a troca melhora o custo da solução S. Para a troca ser possível, basta que as arestas e' e f' exitam no grafo.

3.3. Metodologia Genética

A terceira metodologia heurística proposta é baseada em um algoritmo genético proposto por Snyder e Daskin [2005] para o GTSP.

Um Algoritmo Genético, do inglês *Genetic Algorithm* (GA), é uma meta-heurística inspirada pelo processo de seleção natural na evolução biológica. Ao contrário das heurísticas que geram uma única solução e atuam exaustivamente para melhorá-la Em um GA, uma população de soluções candidatas (chamadas de indivíduos ou fenótipos) para um problema de otimização são evoluídas em direção a soluções melhores. Cada solução candidata possui um conjunto de propriedades (seus cromossomos ou genótipo) que pode ser alterado (mutado).

A evolução começa de uma população inicial de indivíduos gerados aleatoriamente, e é um processo iterativo, onde a população de cada iteração é chamada de uma geração. Em cada geração, a aptidão de cada indivíduo na população é avaliada; no caso do PCDCP, a aptidão é inversamente proporcional ao valor do custo da solução, ou seja, soluções de menor custo são mais aptas. Os indivíduos mais aptos são selecionados aleatoriamente da população atual e têm seus genomas modificados por operadores genéticos e de busca local para formar uma nova geração. A nova geração de soluções candidatas é então usada na próxima iteração do algoritmo.

Representação das Soluções. Neste trabalho, cada indivíduo, ou seja, cada solução S do PCDCP, é representada por meio de uma lista ligada circular L. Cada posição de L contém o identificador de um vértice de V e uma variável marcadora indicando se ele é um vértice representante do conjunto k-dominante. Vértices consecutivos em L são ligados por arestas em S.

A cada solução S está associada um cromossomo, ou seja, uma codificação. Neste trabalho, cada cromossomo é representado por uma permutação P dos vértices de V. A permutação P indica a ordem na qual os vértice representantes do conjunto k-dominante são visitados na solução

S. Os vértices representantes são aqueles localizados no começo da permutação P e que dominam todos os vértices de V. Os vértices restantes não são considerados, podendo aparecer em qualquer ordem na permutação P.

Operadores Genéticos. Um operador genético é um operador usado em um GA para guiar o algoritmo em direção a solução do problema. Existem três tipos de operadores (seleção, cruzamento e mutação), que devem trabalhar em conjunto para que o algoritmo seja bem-sucedido. Operadores genéticos são usados para criar e manter diversidade genética (mutação), combinar soluções existentes em novas soluções (cruzamento) e selecionar entre soluções (seleção).

Seleção. Seleção é o estágio de um algoritmo genético no qual os genomas dos indivíduos são escolhidos de uma população para serem reproduzidos. Indivíduos são selecionados através de um processo baseado em aptidão, onde soluções mais aptas são tipicamente mais prováveis de serem selecionadas.

Neste trabalho, foi utilizada a Seleção Proporcional à Aptidão, também conhecida como Seleção da Roleta. Na Seleção Proporcional à Aptidão, a aptidão de cada indivíduo é usada para associar uma probabilidade de seleção a cada cromossomo. Seja f_i a aptidão do indivíduo i na população, a sua probabilidade de ser selecionado é $p_i = \dfrac{f_i}{\sum_{j=1}^{N} f_j}$, onde N é o número de indivíduos na população. Como no PCDCP a aptidão é inversamente proporcional ao custo da solução, foi utilizado $f_i = \dfrac{1}{c_i + 1}$, onde c_i representa o custo da solução i e o termo "+1"é incluído para evitar divisão por zero na ocasião de custo nulo.

O processo de substituição de população adotado neste trabalho foi o de elitismo, no qual o indivíduo mais apto de uma geração é sempre carregado para a geração seguinte. Esta estratégia garante que a qualidade da solução obtida pelo GA não decairá de uma geração para a próxima.

Cruzamento. Cruzamento é um operador genético usado para variar a programação de cromossomos de uma geração para a próxima. Ele é análogo à reprodução e ao cruzamento biológico, nos quais GAs são baseados. Cruzamento é um processo de produzir uma solução-filha a partir de mais de uma solução-mãe selecionadas da população.

Neste trabalho, foi adotado um operador de cruzamento chamado C1, que foi apresentado em Reeves [1997]. Neste operador, um ponto de cruzamento X é escolhido aleatoriamente, e a seção esquerda do cromossomo da primeira solução-mãe é copiada para o cromossomo da solução-filha, com o cromossomo sendo completado tomando em ordem cada elemento não atribuído do cromossomo da segunda solução-mãe.

Mutação. Mutação é um operador genético usado para manter diversidade genética de uma geração para a próxima. Ela é análoga à mutação biológica. Mutação altera os valores de um ou mais genes de um cromossomo de uma solução, podendo mudar completamente a solução e, portanto, o GA pode chegar a uma solução melhor utilizando mutação. Neste trabalho, o operador de mutação consiste em escolher aleatoriamente dois genes do cromossomo e trocar os valores deles.

Operadores de Busca Local. A fim de acelerar o processo de convergência do GA, os operadores de busca local vistos na seção 3.2 foram aplicados em cada indivíduo da população.

4. Experimentos Computacionais

Os resultados consistem em experimentos computacionais sobre os modelos apresentados nas seções 3.1, 3.2 e 3.3. As instâncias utilizadas nos ensaios foram geradas distribuindo-se pontos (vértices) em um plano de dimensões 1×1 com coordenadas (x, y) geradas aleatoriamente no intervalo $[0, 1]$ com distribuição uniforme. Os custos das arestas são proporcionais ao comprimento do segmento de reta que une seus extremos, ou seja, à distância euclidiana entre seus extremos e as penalidades dos vértices são proporcionais à sua distância até um vértice de referência w. As

instâncias são categorizadas pela tupla (n, d, k, t), tal que n é o número de vértices, d é a densidade do grafo, k é o raio de vizinhança e t é um parâmetro tal que definimos w como sendo o vértice mais próximo do ponto $(0.5, 0.5)$ se $t = 0$ ou como o vértice mais distante do ponto $(0.5, 0.5)$ se $t = 1$. Foram geradas instâncias com valores $n \in \{50, 100, 200\}$, $d \in \{0.3, 0.5, 0.7\}$, $k \in \{0.0, 0.1, 0.2\}$ e $t \in \{0, 1\}$.

Foram testadas diversas configurações das metodologias GRASP e Genética. Para a metodologia GRASP, foram testados configurações com os valores $\alpha \in 0.3, 0.5, 0.7$ e, para o AG, configurações com valores Tamanho da População $\in 10, 50, 100$ e Taxa de Mutação $\in 0.1, 0.2, 0.3$. Como podemos ver nas tabelas 1 e 2, as configurações que obtiveram os melhores resultados foram $\alpha = 0.3$ para a Metodologia GRASP e Tamanho da População $= 10$ e Taxa de Mutação $= 0.3$ para a Metodologia Genética.

Tabela 1: Resultados obtidos pela metodologia GRASP para diferentes configurações

α	Custo Total
0.3	1404 ± 578
0.5	1424 ± 610
0.7	1426 ± 611

Tabela 2: Resultados obtidos pela metodologia Genética para diferentes configurações

Tamanho da População	Taxa de Mutação	Custo Total
10	0.1	1261.2 ± 555.7
10	0.2	1260.5 ± 554.9
10	0.3	1260.1 ± 557.1
50	0.1	1265.3 ± 546.7
50	0.2	1273.7 ± 560.6
50	0.3	1271.4 ± 553.9
100	0.1	1296.3 ± 576.5
100	0.2	1299.7 ± 578.1
100	0.3	1305.0 ± 582.1

A figura 1 apresenta um exemplo de uma instância do PCDCP, enquanto as figuras 2, 3 e 4 apresentam as soluções encontradas. Os vértices são representados por círculos pretos, de tamanho proporcional à penalidade enquanto a vizinhança de cada vértice é representada por uma circunferência vermelha.

O *solver* utilizado para a solução do modelo de duas fases foi o *Gurobi*. Todas as metodologias foram implementadas na linguagem de programação C++ e configuradas com tempo de execução máximo de 1 minuto. Os experimentos computacionais foram executados em um computador com processador Intel®Xeon®CPU X3430 2.4 GHz, com 8GB de RAM.

A metodologia de duas fases convergiu para a solução ótima dos subproblemas em 83,95% das instâncias. Já as metodologias GRASP e Algoritmo Genético encontraram soluções viáveis para todas as instâncias. A tabela 3 abaixo exibe informações sobre o desempenho das metodologias.

Tabela 3: Resultados obtidos por metodologia

Metodologia	Tempo (s)	Custo Total	Penalidades (%)	Custo da Rota (%)	$\lvert S\rvert/n$
Duas Fases	6 ± 17	1690 ± 1536	0.5 ± 0.3	0.5 ± 0.3	0.4 ± 0.3
GRASP	60.1 ± 0.2	1404 ± 579	0.3 ± 0.2	0.7 ± 0.3	0.8 ± 0.3
Genética	60.1 ± 0.1	1260 ± 559	0.3 ± 0.2	0.7 ± 0.2	0.7 ± 0.3

A coluna $\frac{|S|}{n}$ da tabela 3 exibe a proporção entre a quantidade de vértices na ciclo encontrado e a quantidade de vértices total do grafo, enquanto que as colunas Penalidades(%) e Custo da Rota (%) mostram como é a proporção do custo total dividido, respectivamente, entre as penalidades pagas por vértices não visitados e os custos pagos pelas arestas visitadas pela rota. Podemos ver que a Metodologia de Duas Fases obteve soluções com a menor quantidade de vértices, utilizando em média 40% dos vértices do grafo, fazendo com que uma maior parte dos custos das soluções fossem em decorrência das penalidades.

Podemos ver que a metodologia de Duas Fases obteve os menores tempos de processamento para as instâncias em que foi capaz de achar solução, embora tenha encontrado soluções de pior qualidade. Já as metodologias GRASP e AG obtiveram os maiores tempos de processamento, porém conseguiram encontrar soluções viáveis para todas as instâncias.

Na comparação entre as meta-heurísticas, cabe destacar o desempenho médio consideravelmente melhor do AG comparado com o GRASP, sendo que o primeiro conseguiu reduzir na ordem de 10%, em média, os custos das soluções. Uma possível explicação para a diferença de desempenho pode estar em um dos pontos fracos da meta-heurística GRASP, que corresponde à ausência de memória, ou seja, cada iteração é independente uma da outra. Em contrapartida, a memória está presente no AG, uma vez que são armazenados na população os genes que se mostraram mais competitivos ao longo de todas as gerações.

5. Conclusão

Os experimentos computacionais deste trabalho foram realizados utilizando três metodologias heurísticas: a primeira consiste em um método de solução de duas fases, quando na primeira fase é resolvido o modelo para o k-DSP, tal que a solução encontrada será usada na segunda fase, que consiste na solução do TSP; a segunda consiste de um método GRASP; e a última consiste de um algoritmo genético.

Os resultados obtidos pelos experimentos computacionais demonstraram o impacto da metodologia empregada na qualidade das soluções obtidas e no tempo de processamento. Na metodologia de duas fases, observamos um *trade-off* entre tempo de processamento e custo pago em penalidades, além de obter soluções que utilizam um menor número de vértices e cujo custo total é dividido mais igualitariamente entre penalidades e custo de arestas.

As meta-heurísticas GRASP e Algoritmo Genético obtiveram soluções não triviais para o problema em todas as instâncias consideradas, inclusive para as maiores instâncias de 200 vértices. O custo das soluções obtidas pelo Algoritmo Genético mostrou-se substancialmente inferior ao do GRASP, o que sugere que a característica do GRASP de não possuir memória pode ter prejudicado seu desempenho nos experimentos computacionais.

Como trabalhos futuros são sugeridos os estudos de modelos exatos e métodos de obtenção de limitantes duais. Além disso, cabe-se avaliar se este problema comporta algum fator de aproximação.

6. Agradecimentos

Este trabalho teve o suporte da Fapesp (processo número 2016/05036-1).

Referências

Applegate, D. L., Bixby, R. E., Chvatal, V., e Cook, W. J. (2011). *The traveling salesman problem: a computational study*. Princeton university press.

Bienstock, D., Goemans, M. X., Simchi-Levi, D., e Williamson, D. (1993). A note on the prize collecting traveling salesman problem. *Mathematical programming*, 59(1-3):413–420.

Chang, G. J. (1998). Algorithmic aspects of domination in graphs. In *Handbook of combinatorial optimization*, p. 1811–1877. Springer.

Chatterjee, S., Carrera, C., e Lynch, L. A. (1996). Genetic algorithms and traveling salesman problems. *European journal of operational research*, 93(3):490–510.

Chen, N., Meng, J., Rong, J., e Zhu, H. (2004). Approximation for dominating set problem with measure functions. *Computing and Informatics*, 23:37–49.

Croes, G. A. (1958). A method for solving traveling-salesman problems. *Operations research*, 6 (6):791–812.

Current, J. R. e Schilling, D. A. (1989). The covering salesman problem. *Transportation science*, 23(3):208–213.

Haynes, T. W., Hedetniemi, S., e Slater, P. (1998). *Fundamentals of domination in graphs*. CRC Press.

Laporte, G., Gendreau, M., Potvin, J.-Y., e Semet, F. (2000). Classical and modern heuristics for the vehicle routing problem. *International transactions in operational research*, 7(4-5):285–300.

Maziero, L. P., Usberti, F. L., e Cavellucci, C. (2015). O problema do ciclo dominante.

Reeves, C. R. (1997). Genetic algorithms for the operations researcher. *INFORMS journal on computing*, 9(3):231–250.

Resende, M. G. e Ribeiro, C. C. (2002). Greedy randomized adaptive search procedures. *Encyclopedia of optimization*, 2:373–382. [Online; acessado em 11 de dezembro de 2016].

Snyder, L. V. e Daskin, M. S. (2005). A random-key genetic algorithm for the generalized traveling salesman problem.

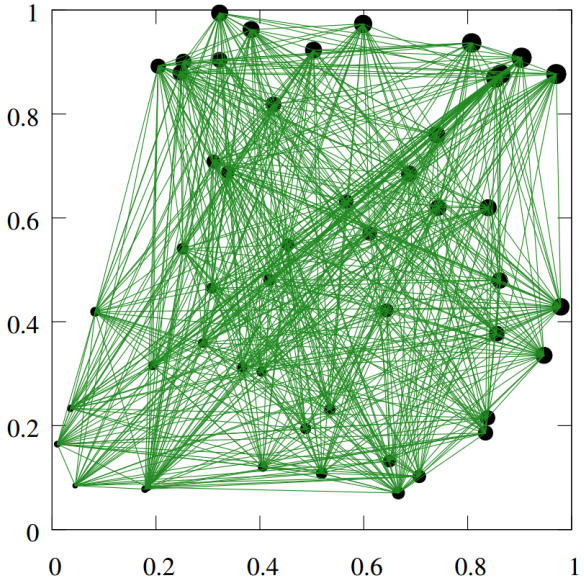

Figura 1: Instância do PCDCP com $n = 50$, $d = 0.5$, $k = 10$ e $t = 0$.

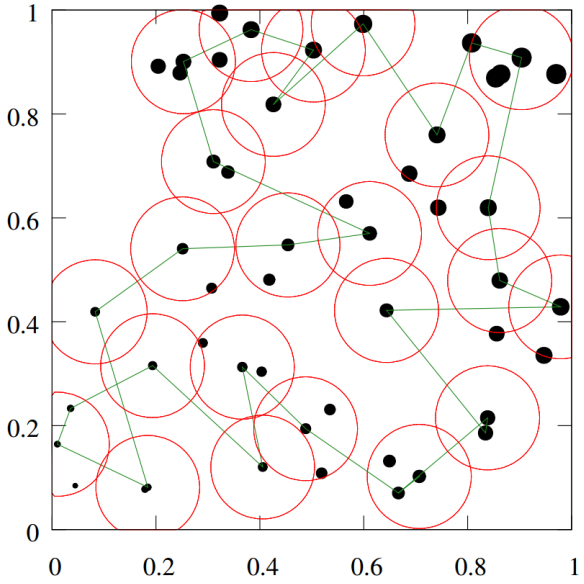

Figura 2: Solução da instância apresentada na Figura 1 com custo 913 obtida pela metodologia de duas fases.

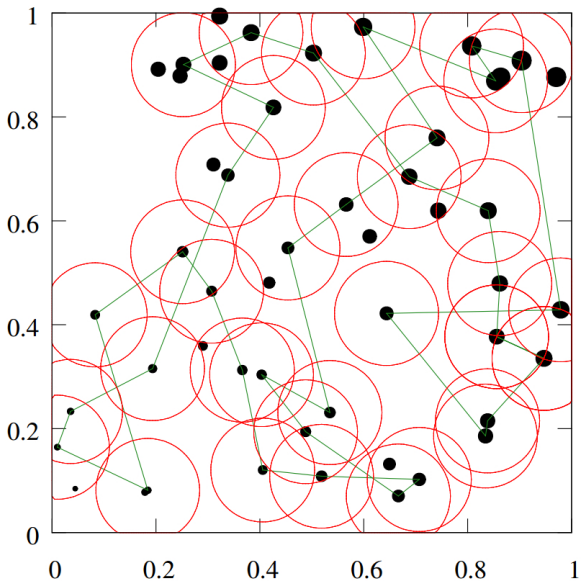

Figura 3: Solução da instância apresentada na Figura 1 com custo 902 obtida pela metodologia GRASP.

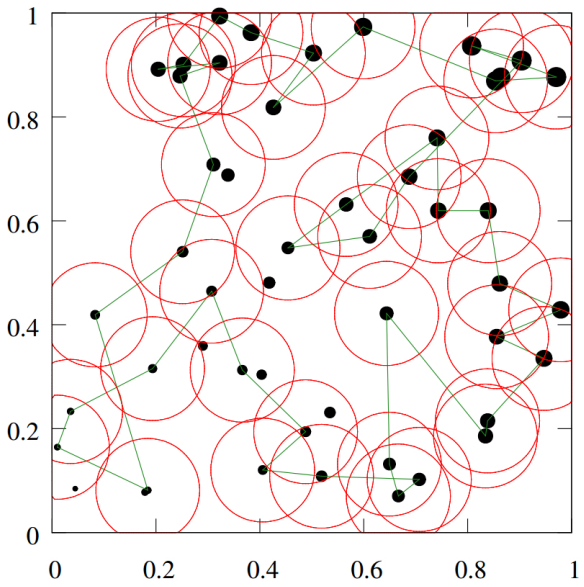

Figura 4: Solução da instância apresentada na Figura 1 com custo 724 obtida pela metodologia genética.