Uma abordagem heurística para o problema da Árvore Geradora Mínima de Grau Restrito

RUBEM KALEBE SANTOS

Natal, Brasil
Novembro de 2015

Sumário

1 Introdução

O termo "heurístico" vem do grego *heuriskein*, que é sinônimo para descobrir ou encontrar algo, do mesmo radical que deu origem a palavra "heureca", interjeição imortalizada pelo matemático e filósofo grego Arquimedes (287-212 a.C.). Uma heurística é um conjunto de regras e métodos que conduzem à descoberta, à invenção e à resolução de problemas.

No contexto dos algoritmos, uma heurística pode ser vista como um procedimento exploratório que visa resolver um problema e que geralmente não envolve a implementação computacional de um conhecimento especializado, isto é, um procedimento heurístico para resolver uma equação de segundo grau não usaria, necessariamente, a *Fórmula de Bhaskara*, mas buscaria, por outros métodos, uma solução que atendesse à equação. Para tanto, as heurísticas normalmente tendem a apresentar um certo grau de conhecimento acerca do comportamento do problema a ser resolvido, gerando um número muito menor de soluções possíveis.

Uma solução ótima de um problema nem sempre é o alvo dos métodos heurísticos, uma vez que, tendo como ponto de partida uma solução viável, baseiam-se em sucessivas aproximações direcionadas a um ponto ótimo. Logo, estes métodos costumam encontrar as melhores soluções possíveis para problemas, e não soluções exatas, perfeitas, definitivas.

Como visto no trabalho anterior, a versão de decisão do problema da Árvore Geradora Mínima com restrição de grau (*Degree-constrained minimum spanning tree* – DCMST) pertence a classe NP-completo [GJ79] e os métodos exatos são computacionalmente custosos e demandam muito tempo (séculos, para algumas instâncias) para retornar uma solução para uma instância próxima da realidade, isto é, com, no mínimo, centenas de vértices e arestas.

Os processos heurísticos exigem muitas vezes menos tempo que os procedimentos exatos, aproximam-se mais da forma como o ser humano raciocina e chega às resoluções dos problemas, e garantem soluções eficientes. Para quem precisa de uma resposta (pelo menos próxima da ótima) e tem limitação de recursos (tempo, dinheiro) os algoritmos heurísticos são interessantes pois produzem uma boa resposta (melhor do que não ter nenhuma resposta), em tempo polinomial, e evitam um possível desperdício de dinheiro, como também podem possibilitar o lucro (se for encontrado um algoritmo heurístico "melhor" que outro, por

exemplo).

Neste trabalho serão apresentados dois algoritmos heurísticos para resolver o problema em questão. O primeiro deles faz uso de uma meta-heurística bastante utilizada nesse contexto e que produz bons resultados. O outro algoritmo é mais específico, sendo uma variação de soluções exatas do problema, e que apresenta bons resultados em pouquíssimo tempo.

O trabalho está sendo mantido em um repositório *online* e pode ser acessado a partir deste *link*: https://github.com/rubemkalebe/DCMST.

2 Revisão bibliográfica

Antes de discutir as soluções heurísticas propostas é necessário descrever os métodos utilizados e do que se tratam.

As soluções heurísticas podem ser classificadas em alguns grupos, como: *construtivas*, *de melhoramento*, *de relaxação* e *meta-heurísticas*. Segundo a definição original, as meta-heurísticas são métodos de solução que coordenam procedimentos de busca locais com estratégias de mais alto nível, de modo a criar um processo capaz de escapar de ótimos locais e realizar uma busca robusta no espaço de soluções de um problema [GK03]. Basicamente, elas combinam diferentes estratégias para percorrer o espaço de busca tentando escapar de ótimos locais. Essas diferentes estratégias podem incluir inclusive outras heurísticas, elementos de memória, diversificação e/ou intensificação, e múltiplas vizinhanças.

As meta-heurísticas são aplicadas para encontrar respostas a problemas sobre os quais há poucas informações: não se sabe como é a aparência de uma solução ótima, há pouca informação heurística disponível e algoritmos exatos são inviáveis devido ao espaço de soluções ser muito grande. Porém, dada uma solução candidata ao problema, esta pode ser testada e sua otimalidade, averiguada.

Elas diferenciam-se, entre si, pelo critério de escolha da solução inicial, da definição da vizinhança, da forma de seleção do "vizinho" e do critério de parada; pode-se dizer que uma de suas principais características é ser de uso geral, no sentido que podem ser adaptadas para resolver problemas diversos, diferentemente das heurísticas simples que são específicas para um dado problema.

Algumas das meta-heurísticas mais conhecidas são:

- Busca Tabu (*Tabu Search*);
- GRASP (*Greedy Randomized Adaptive Search Procedure*);
- *Simulated Annealing*;
- Enxame de Partículas (*Particle Swarm Optimization*);
- Colônia de Formigas (*Ant Colony Optimization*);
- Algoritmos Genéticos (*Genetic Algorithm*).

Nesse trabalho abordaremos a meta-heurística *Colônia de Formigas*, que será descrita nas próximas subseções. Além disso, também será

discutido a aplicação dela em outros trabalhos que abordam o mesmo problema.

2.1 Otimização por Colônia de Formigas

A Otimização por Colônia de Formigas (*Ant Colony Optimization* – ACO) é uma meta-heurística inspirada na natureza, mais especificamente no comportamento utilizado pelas colônias de formigas para traçar rotas entre o formigueiro e as fontes de alimentação. O principal aspecto desse comportamento é uma substância chamada **feromônio**, que é secretada pelas formigas durante seus percursos, de forma a indicar caminhos prometedores a outras formigas. A ideia básica do método é utilizar formigas artificiais, representadas por processos concorrentes, que traçam caminhos em um grafo cujos vértices, ou arestas, podem representar componentes da solução.

Segundo [dFV09], trata-se de uma técnica enquadrada em um ramo específico da inteligência coletiva (*swarm intelligence*), inspirada no comportamento social que as formigas apresentam ao buscarem por fontes de alimento para seus ninhos. Segundo o mesmo autor, quanto mais feromônio depositado na busca pelo alimento, menor é o caminho e, consequentemente, mais atraídas para essa trajetória serão as formigas vindouras.

No processo de busca pelo alimento, as formigas são, também, capazes de se adaptar a alterações do meio ambiente. Por exemplo, voltam a encontrar o caminho mais curto entre a fonte de comida e o ninho após o aparecimento de um obstáculo que impede a circulação pelo percurso inicial.

Segundo [PZ07], a analogia do comportamento das formigas com a otimização se realiza do seguinte modo:

a) A busca de alimento é equivalente à exploração das soluções factíveis em um problema de otimização combinatória.

b) A quantidade de alimento é similar ao valor da função objetivo.

c) O rastro de feromônio é a memória adaptativa do método.

Os parâmetros deste método são, de acordo com [Per07]:

a) Ponderação do feromônio.

b) Percentual de evaporação do feromônio.

c) Quantidade de formigas.

d) Número máximo de iterações.

Tais parâmetros devem ser calibrados para cada problema específico e seu funcionamento ocorre da seguinte forma: quanto mais vezes um determinado caminho for percorrido, maior será a quantidade de feromônio deixada nesse rastro, indicando a direção que a pesquisa tenderá a percorrer. O procedimento de busca é repetido até que um número máximo de iterações seja atingido ou não se verifique melhoria na qualidade das soluções obtidas após um determinado número de iterações [Per07].

2.1.1 Construção da solução

Em algoritmos baseados nesta técnica as soluções são construídas iterativamente pelas formigas. Cada formiga constrói uma solução e é capaz de avaliar o valor dessa solução (completa ou parcial). Enquanto a constrói, a formiga toma decisões baseadas na informação disponível para as possíveis escolhas. Essas informações vêm da experiência de outras formigas, baseado na quantidade de feromônio depositada em um determinado caminho, e de sua própria experiência, através de uma heurística (geralmente uma *lista tabu* – que não tem muito a ver com a *Busca Tabu*).

A probabilidade de uma formiga escolher a transição do estado i para o estado j, $p(e_{ij})$, é dada pela equação 1, onde $\mathcal{N}_i$ representa o conjunto de estados vizinhos de i, τ_{ij} e η_{ij} representam, respectivamente, a quantidade de feromônio e a informação heurística associada a transição entre os estados i e j, e α e β são parâmetros que ponderam a importância da informação que vem do feromônio e da heurística, respectivamente. Quanto maior o α, maior será a preferência pelo feromônio nas trilhas, e quanto maior o β, mais "guloso" será o algoritmo. Cada vez que uma solução é construída, um depósito de feromônio pode ser efetuado, que evapora com o tempo.

$$p(e_{ij}) = \begin{cases} \dfrac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{h \in \mathcal{N}_i} \tau_{ih}^{\alpha} \cdot \eta_{ih}^{\beta}}, & \text{se } j \in \mathcal{N}_i. \\ 0, & \text{caso contrário.} \end{cases} \tag{1}$$

A informação heurística é um valor associado a uma aresta (i, j) que representa a atratividade da formiga por visitar o vértice i depois de

visitar j e é dado pela equação 2, onde o valor η_{ij} é inversamente proporcional a distância, ou custo, d_{ij} entre os vértices i e j.

$$\eta_{ij} = \frac{1}{d_{ij}} \tag{2}$$

A quantidade de feromônio na transição e_{ij} em uma iteração $t+1$ é calculada conforme a equação 3, onde ρ é a taxa de evaporação. O depósito de feromônio $\Delta\tau_{ij}$ é a soma dos depósitos de feromônio de todas as formigas em uma dada iteração t. Cada um desses depósitos no *Ant System* é calculado usando a equação 4, onde Q é um parâmetro do algoritmo e $L^a(t)$ é o comprimento do caminho π_a construído por uma formiga a em uma iteração t:

$$\tau_{ij}(t+1) = \rho \cdot \tau_{ij}(t) + \Delta\tau_{ij}(t, t+1) \tag{3}$$

$$\Delta\tau_{ij}^{a}(t, t+1) = \begin{cases} \dfrac{Q}{L^a(t)}, & \text{se } e_{ij} \in \pi_a. \\ 0, & \text{caso contrário.} \end{cases} \tag{4}$$

Vários melhoramentos foram, e são, sugeridos para o *Ant System*. Uma das primeiras e mais conhecidas estratégias foi apresentada em [Dor92]. A ideia é similar a estratégia de elitismo dos algoritmos genéticos e, basicamente, fornece um forte reforço adicional nas arestas que pertencem ao melhor percurso achado desde o início do algoritmo.

2.1.2 Aplicação ao problema

Soluções utilizando esta técnica podem ser encontradas em [BHE05], [BZ06] e [BDZ12]. A principal diferença entre as propostas de [BZ06] e [BDZ12] para a proposta em [BHE05] é que nas duas primeiras o autor trabalha com uma variante dessa meta-heurística, na qual uma formiga encontra apenas uma parte da solução, em contraste com os algoritmos de Colônia de Formigas comuns, nos quais uma formiga constrói uma solução completa. Segundo o autor, essa diferença permite que as formigas considerem seções menores e mais localizadas do problema, o que torna os algoritmos bem adaptados a programação concorrente [BZ06]. A solução em [BDZ12] apresenta um fase de otimização local que permite uma melhora significativa nos resultados dos experimentos em relação ao trabalho anterior.

Em [BHE05] são apresentadas duas soluções utilizando Otimização por Colônia de Formigas: uma usa os vértices do grafo como componentes da solução e a outra usa as arestas. Na primeira as formigas constroem uma solução seguindo o *algoritmo de Prim* [Pri57] e na outra elas seguem o *algoritmo de Kruskal* [Kru56].

3 Descrição da solução

Este problema já vem sendo bastante explorado na literatura e várias soluções foram propostas. A primeira solução proposta foi apresentada em [NH80], nos anos 80, e era baseada em uma técnica de *branch-and-bound*. No mesmo trabalho também foram apresentados dois algoritmos baseados em técnicas de Programação Linear que conseguem encontrar soluções para instâncias de até 100 vértices.

Variadas soluções são encontradas na literatura. Em [dSM08] é apresentada uma solução utilizando a meta-heurística *Variable Neighborhood Search* (VNS), em [ALM06] o autor traz uma solução heurística baseada em relaxação Lagrangiana, em [Ern10] é encontrada uma meta-heurística híbrida que mescla relaxação Lagrangiana e Enxame de Partículas e em [Tor13] é mostrada uma abordagem um pouco diferente, baseada em técnicas de aprendizado por reforço (*learning automata*). Além dessas, algoritmos genéticos também são extensivamente usados, como em [ZG97], [RJ00] e [KC00].

Algoritmos baseados no comportamento das formigas também são usados e foram apresentados na seção anterior. Nesta seção serão apresentadas duas soluções. A primeira delas faz uso da Otimização por Colônia de Formigas e a outra é um algoritmo heurístico orientado por um processo de *backtracking*.

3.1 Algoritmo baseado em Colônia de Formigas

O algoritmo desse estilo que foi implementado segue bem o padrão dos algoritmos de Otimização por Colônia de Formigas. Basicamente, ele implementa tudo o que foi visto na seção 2, incluindo as fórmulas.

De maneira geral, um conjunto de arestas $E = \{e_{ij} | i \in V, j \in V, i \neq j\}$, conjunto de todas as arestas possíveis no grafo, contém os componentes das soluções que as formigas construirão de forma incremental. O primeiro passo do algoritmo é ordenar esse conjunto de arestas de forma crescente pelo peso de suas arestas. A partir daí inicia-se o *Ant System* que executa dentro de um laço com no máximo $maxIterations$. Dentro desse laço temos métodos comuns a meta-heurística:

- `setupAnts()`: aqui um número $numAnts$ (definido por $numVertices \times numAntFactor$) são ativadas no sistema, dentro do qual cada uma é colocada em uma aresta aleatória do grafo.

- `moveAnts()`: neste método cada formiga irá construir uma solução completa. Cada escolha [de aresta] pode se dar de maneira aleatória (se o número aleatório gerado for menor que uma constante pr) ou probabilisticamente (de acordo com a equação 1 na seção 2). A escolha aleatória é útil para causar diversificação no processo (todavia, ela deve ser bem ajustada para não acarretar em soluções ainda mais longes da ótima).

- `updateTrails()`: neste método serão feitas, respectivamente, a evaporação do feromônio (de acordo com a taxa predefinida) e a atualização do feromônio nas arestas de acordo com a contribuição de cada formiga.

- `updateBest()`: este método é responsável por intermediar o armazenamento da melhor solução até um momento.

Resumidamente, o algoritmo pode ser expresso no pseudocódigo 1.

Algoritmo 1 Pseudocódigo que ilustra o *Ant System* proposto

```
1:  function antSystem
2:      Ordene, de forma crescente pelo peso, as arestas
3:      for t = 1 to maxIterations do
4:          Coloque cada formiga em uma aresta aleatória
5:          for k = 1 to |V| − 1 do
6:              while formiga k não construir a solução S_k do
7:                  Selecione a próxima aresta pela regra p_ij^k ou aleatoriamente
8:              Calcule o custo C_k da solução S_k
9:              if C_k < C* then
10:                 S* = S_k
11:                 C* = C_k
12:         Atualize os feromônios
        return S*
```

Note que o pseudocódigo apresentado está bem abstrato. O passo da linha 7 inclui também verificação da restrição de grau e da aresta (no caso se ela já tiver sido inserida na solução outra aresta deverá ser selecionada). Na implementação, se a aresta não puder ser inserida na solução (caso quebre a restrição ou se já tiver sido incluída) é iniciado um laço que tenta inserir a próxima aresta válida, a partir do índice da aresta anterior.

Para evitar que uma formiga inclua uma mesma aresta mais de uma vez na solução é usada uma lista tabu, que é associada a cada formiga e é basicamente um vetor de booleanos que indica se a aresta da posição i foi ou não visitada.

Afim de minimizar o impacto do tempo gasto na operação de exponenciação – usada no momento de escolhas probabilísticas de acordo com a equação 1 – foi utilizada uma função otimizada que é apresentada em [Ank]. O uso dessa função gerou, como esperado, impacto positivo no tempo gasto pela solução, em comparação com a função da biblioteca padrão do C++.

3.2 Algoritmo heurístico

Afim de obter valores limite para o custo das soluções foi implementado também um algoritmo heurístico que utiliza a técnica de *backtracking* com alguns ajustes. O primeiro ajuste foi o reordenamento do espaço de busca (conjunto de arestas) de forma crescente pelos seus respectivos pesos. O outro ajuste foi a interrupção do *backtracking* em um certo momento. Essa interrupção é dada em função do número de iterações sem mudanças, isto é, número de iterações sem gerar uma nova solução ótima. Se tal número alcançar o valor máximo estipulado, o algoritmo é encerrado e a melhor solução armazenada é retornada. O valor máximo estipulado *escape* é dado em função do número de arestas da solução, tal que $escape = (|E|^2) \times 5$. Essa equação é baseado no número de arestas por esse ser um dos principais fatores que influenciam na geração da solução e testes mostraram que tal fórmula apresenta bons resultados, permitindo o algoritmo encontrar uma solução melhor em várias instâncias. De certa forma este algoritmo lembra um algoritmo guloso, chegando a se comportar como um algumas vezes, mas a possibilidade de eventualmente melhorar a solução ao continuar procurando, durante um certo tempo, por outra solução é que diferencia ele. De fato, a ideia do algoritmo é semelhante ao raciocínio humano, de forma que "se até dado momento não for achada uma solução melhor, esta deve ser a melhor (ou no mínimo, é muito boa)".

Como todo algoritmo heurístico, não há como provar que esse algoritmo sempre irá retornar a melhor resposta. Todavia, tendo em vista que, provavelmente, a melhor solução envolve arestas com baixo custo, é natural se pensar que a melhor solução não demore muito tempo para ser encontrada. Em dado momento, na procura por outras soluções, pode ser que estejamos "perdendo tempo", por isso a interrupção se faz necessária. Mesmo assim, no mínimo, temos uma solução muito boa.

Destacam-se duas vantagens nesse algoritmo: bom tempo de resposta

e possível melhora da solução, já que diferentemente de um algoritmo guloso, ele pode melhorar a solução, se até dado momento for encontrada uma melhor. O bom tempo de resposta advém de sua complexidade, que é em torno de $O(n^2)$, já que limitamos o número de iterações. Perceba também que o *backtracking* serve apenas para guiar o processo de busca por uma melhor solução.

Como não há elementos não determinísticos, o algoritmo sempre retorna a mesma resposta para uma mesma instância. Alguma "pitada" de aleatoriedade poderia ser inserida no algoritmo, mas o mesmo mostrou-se suficiente para retornar boas respostas.

O procedimento heurístico é apresentado no código 1. Saiba que nas soluções geradas o grafo é armazenado em uma lista de arestas e o grau de cada vértice é mantido em um vetor de inteiros. Além disso, uma estrutura de conjuntos disjuntos (*Union-Find*) é usada para checar se o grafo é conexo e/ou cíclico.

Código 1: Método responsável por procurar a solução

```
1  void Heuristic::combinations(Edge link[], int length, int size, int startPosition, Tree
       &tree) {
2          if(size == 0) {
3                  if((solutions == 0) || (tree.totalCost() < bestTree->totalCost())) {
4                          bestTree->update(tree);
5                          iterationsWithoutChange = 0;
6                  }
7                  solutions++;
8                  return;
9          }
10         for(int i = startPosition; i <= length - size; i++) {
11                 if(++iterationsWithoutChange > escape) {
12                         return; // Aqui ocorre a interrupcao do algoritmo
13                 } else if((solutions == 0 || tree.totalCost() + link[i].getCost() <
                       bestTree->totalCost()) && tree.addEdge(link[i])) {
14                         combinations(link, length, size-1, i+1, tree);
15                         tree.removeEdge();
16                 } else {
17                         combinations(link, length, size, i+1, tree);
18                         return;
19                 }
20         }
21 }
```

Perceba também que o algoritmo de *backtracking* só busca árvores geradoras de grau restrito que podem ser mínimas. Isso é garantido através da "poda" feita na linha 13, onde ele só insere uma aresta se, além de obedecer as restrições do problema, a inserção dela não produzir uma árvore de custo maior do que a melhor solução atual.

Nesse trabalho iremos nos referir a esse algoritmo simplesmente como "algoritmo heurístico".

4 Experimentos Computacionais

Afim de avaliar a eficiência dos algoritmos propostos, nesta seção serão apresentados os resultados da execução dos programas implementados. Além dos propostos na seção 3 será considerado também um algoritmo exato que foi implementado no trabalho anterior.

4.1 Instâncias

No trabalho anterior foi implementado um gerador de instâncias euclidianas e neste trabalho ele foi melhorado para que pudesse gerar também instâncias não euclidianas. Sendo assim, aqui considera-se instâncias euclidianas e não euclidianas. As instâncias variam de 5 vértices até 1000 vértices, com o grau máximo variando entre 2 e 6.

4.2 Metodologia de Comparação

Os testes foram realizados em um computador com processador Intel i3 2ª geração, 4 núcleos e frequência de 3.1 GHz, e 4 GB de memória RAM. Para cada instância, 30 execuções independentes de cada um desses algoritmos foram realizadas.

Todos os algoritmos foram implementados em linguagem C++ e compilados com o compilador g++ versão 4.8.4. Também foi utilizada a *flag* de otimização `-O3` disponibilizada pelo compilador, que possibilitou um bom ganho de desempenho.

4.3 Configuração dos Parâmetros

No *Ant System* primeiramente foram considerados os valores sugeridos em [Dor92] e alguns pequenos ajustes foram feitos em seguida baseando-se em vários testes, afim de melhorar os resultados obtidos. Percebeu-se, por exemplo, que quando se aumentava demais a probabilidade de escolha aleatória o algoritmo ficava mais distante da solução ótima.

Os valores finais para os parâmetros desse algoritmo foram:

- *Alpha*: 1.0;
- *Beta*: 5.0;
- Quantidade inicial de feromônio: 5.0;
- Máximo de iterações: 300;

- Taxa de evaporação do feromônio: 0.5;
- Fator de número de formigas usadas ($numAnts = fator \times |V|$): 0.8;
- Taxa de evaporação do feromônio: 0.5;
- Coeficiente de depósito de feromônio = 500.0;
- Probabilidade de seleção puramente randômica: 0.1;

No outro algoritmo heurístico o único parâmetro usado é a quantidade máxima de iterações, *escape*, sem gerar uma nova solução. A saber, $escape = (|E|^2) \times 5$. Os testes feitos justificaram essa escolha. Essa equação permite o algoritmo achar boas soluções para instâncias muito grandes.

4.4 Resultados

Nas tabelas que seguem, o algoritmo 1 corresponde ao *backtracking* implementado no trabalho anterior, o algoritmo 2 ao algoritmo heurístico e o algoritmo 3 ao *Ant System*. As abreviações "res", "tem" e "dif" correspondem, respectivamente, a melhor resposta, tempo médio gasto nas execuções e diferença, percentual, em relação à melhor solução. A melhor resposta considerada foi sempre a de menor custo resultante das execuções, o tempo médio calculado é resultante de uma média aritmética e para o cálculo da diferença percentual foi usada a fórmula 5. Na identificação das instâncias, o número que aparece imediatamente após o 'n' corresponde ao número de vértices e o que aparece após o 'd' corresponde ao grau máximo.

$$diff = \frac{melhorConhecida - melhorDaAtual}{melhorConhecida} \times 100\%. \qquad (5)$$

As tabelas estão separadas por tipo de instância. A tabela 1 apresenta os resultados para instâncias euclidianas e a tabela 2 para instâncias não euclidianas. Nas tabelas 3 e 4 são apresentado os resultados obtidos pelo algoritmo heurístico para instâncias euclidianas e não euclidianas, respectivamente.

Instância	Alg1-Res	Alg1-Tem	Alg2-Res	Alg2-Tem	Alg2-Dif	Alg3-Res	Alg3-Tem	Alg3-Dif
n5d2	139	0.000020	139	0.000026	0.0%	139	0.004709	0.0%
n5d3	115	0.000070	115	0.000030	0.0%	115	0.004978	0.0%
n5d4	163	0.000060	163	0.000035	0.0%	163	0.004888	0.0%

n7d2	207	0.000714	207	0.000203	0.0%	207	0.011772	0.0%
n7d3	164	0.001103	164	0.000169	0.0%	164	0.011881	0.0%
n7d4	165	0.001734	165	0.000166	0.0%	165	0.012312	0.0%
n7d5	122	0.001680	122	0.000185	0.0%	122	0.012131	0.0%
n7d6	171	0.000677	171	0.000177	0.0%	171	0.012287	0.0%
n9d2	220	0.034301	220	0.000414	0.0%	220	0.031865	0.0%
n9d3	178	0.037462	178	0.000397	0.0%	178	0.031673	0.0%
n9d4	196	0.019090	196	0.000295	0.0%	196	0.031431	0.0%
n9d5	169	0.024031	169	0.000331	0.0%	169	0.032211	0.0%
n9d6	173	0.010895	173	0.000290	0.0%	173	0.031050	0.0%
n11d2	235	0.306797	235	0.000777	0.0%	235	0.064171	0.0%
n11d3	227	1.207769	227	0.000628	0.0%	233	0.064185	-4.02%
n11d4	265	1.155119	265	0.000569	0.0%	265	0.065557	0.0%
n11d5	187	0.755387	187	0.000597	0.0%	187	0.071179	0.0%
n11d6	195	0.451963	195	0.000525	0.0%	198	0.071082	-1.54%
n13d2	276	27.850982	287	0.001344	-3.98%	320	0.135790	-15.94%
n13d3	216	37.244088	216	0.000879	0.0%	224	0.131974	-3.70%
n13d4	246	22.948303	246	0.000853	0.0%	246	0.140689	0.0%
n13d5	231	13.997855	231	0.000820	0.0%	238	0.136903	-3.03%
n13d6	240	38.044140	240	0.000864	0.0%	240	0.148766	0.0%
n15d2	-	-	318	0.002070	0.0%	331	0.255481	-4.09%
n15d3	-	-	255	0.001311	0.0%	280	0.253883	-9.80%
n15d4	-	-	265	0.001451	0.0%	308	0.245949	-16.22%
n15d5	-	-	274	0.001345	0.0%	305	0.251463	-11.31%
n15d6	-	-	266	0.001316	0.0%	292	0.253594	-9.78%
n17d2	-	-	260	0.002419	0.0%	260	0.398820	0.0%
n17d3	-	-	304	0.001944	0.0%	304	0.420232	0.0%
n17d4	-	-	208	0.001941	0.0%	210	0.415808	0.96%
n17d5	-	-	298	0.001950	0.0%	328	0.402224	-10.07%
n17d6	-	-	276	0.002030	0.0%	276	0.414382	0.0%
n21d2	-	-	367	0.006268	0.0%	420	0.956256	-14.44%
n21d3	-	-	286	0.004186	0.0%	286	0.971535	0.0%
n21d4	-	-	289	0.004208	0.0%	289	0.992212	0.0%
n21d5	-	-	302	0.003988	0.0%	331	0.948059	-9.60%
n21d6	-	-	308	0.004586	0.0%	310	0.992067	-0.65%
n23d2	-	-	364	0.008850	0.0%	382	1.420184	-4.94%
n23d3	-	-	314	0.006510	0.0%	321	1.456688	-2.23%
n23d4	-	-	284	0.006337	0.0%	285	1.482029	-0.35%
n23d5	-	-	308	0.005497	0.0%	325	1.438004	-5.52%
n23d6	-	-	290	0.006263	0.0%	304	1.479924	-4.83%
n24d2	-	-	386	0.006817	-4.61%	369	1.722451	0.0%
n24d3	-	-	384	0.006456	-7.56%	357	1.811610	0.0%
n24d4	-	-	300	0.005660	0.0%	300	1.754042	0.0%
n24d5	-	-	317	0.006428	0.0%	325	1.752380	-2.52%
n24d6	-	-	307	0.006603	0.0%	307	1.778197	0.0%
n27d2	-	-	363	0.011368	0.0%	363	2.758018	0.0%
n27d3	-	-	331	0.010222	0.0%	346	2.769980	-4.53%
n27d4	-	-	347	0.010403	0.0%	347	2.795224	0.0%
n27d5	-	-	308	0.009686	0.0%	308	2.801596	0.0%
n27d6	-	-	327	0.009712	0.0%	341	2.835082	-4.28%
n30d2	-	-	446	0.014172	-1.83%	438	4.453857	0.0%
n30d3	-	-	404	0.011452	0.0%	419	4.430795	-3.71%
n30d4	-	-	332	0.011350	0.0%	351	4.434365	-5.72%
n30d5	-	-	371	0.011643	0.0%	396	4.493155	-6.74%
n30d6	-	-	376	0.011815	0.0%	388	4.390252	-3.19%
n32d2	-	-	426	0.013236	0.0%	427	5.719953	-0.23%

Tabela 1: Tabela contendo o tempo gasto na execução dos programas para instâncias euclidianas. O traço "-" indica que o algoritmo não foi executado (quando não houve execução foi devido a prevista longa demora). O tempo relatado está em segundos.

Instância	Alg1-Res	Alg1-Tem	Alg2-Res	Alg2-Tem	Alg2-Dif	Alg3-Res	Alg3-Tem	Alg3-Dif
n5d2	55	0.000023	55	0.000015	0.0%	55	0.009295	0.0%
n5d3	101	0.000060	101	0.000024	0.0%	101	0.009252	0.0%
n5d4	64	0.000046	64	0.000027	0.0%	64	0.008769	0.0%
n7d2	69	0.000212	69	0.000085	0.0%	109	0.010774	-57.91%
n7d3	169	0.000918	169	0.000258	0.0%	169	0.011281	0.0%
n7d4	132	0.000358	132	0.000135	0.0%	134	0.011874	-1.51%
n7d5	150	0.000756	150	0.000233	0.0%	150	0.011829	0.0%
n7d6	92	0.000106	92	0.000160	0.0%	94	0.013828	-2.17%
n9d2	133	0.008657	133	0.000778	0.0%	144	0.029689	-8.27%
n9d3	93	0.004294	93	0.000234	0.0%	93	0.032489	0.0%
n9d4	143	0.007243	146	0.000400	-2.10%	146	0.034610	-2.10%
n9d5	124	0.009499	124	0.000432	0.0%	160	0.028592	-29.03%
n9d6	40	0.001762	40	0.000209	0.0%	48	0.029134	-20.00%
n11d2	198	0.192667	206	0.001306	-4.04%	236	0.061569	-19.20%
n11d3	121	0.075377	121	0.000431	0.0%	121	0.061608	0.0%
n11d4	139	0.075914	139	0.000547	0.0%	139	0.064358	0.0%
n11d5	97	0.022261	97	0.000420	0.0%	97	0.058753	0.0%
n11d6	110	0.044850	110	0.000546	0.0%	110	0.068228	0.0%
n13d2	125	0.602458	128	0.001195	-2.40%	129	0.124811	-3.20%
n13d3	123	0.330412	123	0.000874	0.0%	123	0.133068	0.0%
n13d4	124	1.423995	124	0.000854	0.0%	124	0.137242	0.0%
n13d5	93	0.177294	93	0.000595	0.0%	93	0.128644	0.0%
n13d6	153	1.393111	153	0.000466	0.0%	153	0.132992	0.0%
n15d2	131	58.178153	165	0.002223	-25.95%	204	0.249119	-55.72%
n15d3	152	73.637764	152	0.001228	0.0%	152	0.249027	0.0%
n15d4	-	-	195	0.001247	0.0%	195	0.253527	0.0%
n15d5	-	-	127	0.001119	0.0%	135	0.242873	-6.30%
n15d6	-	-	94	0.001138	0.0%	127	0.258329	-35.10%
n17d2	-	-	167	0.003326	0.0%	181	0.406037	8.38%
n17d3	-	-	95	0.001855	0.0%	95	0.404365	0.0%
n17d4	-	-	176	0.001886	0.0%	176	0.394759	0.0%
n17d5	-	-	132	0.001741	0.0%	147	0.390634	-11.36%
n17d6	-	-	65	0.001714	0.0%	65	0.399128	0.0%
n21d2	-	-	138	0.004181	0.0%	160	0.948918	-15.94%
n21d3	-	-	154	0.004086	0.0%	154	0.956272	0.0%%
n21d4	-	-	155	0.004057	0.0%	156	0.977288	-0.64%
n21d5	-	-	169	0.003955	0.0%	176	0.957149	-4.14%
n21d6	-	-	91	0.003766	0.0%	106	0.930561	-16.48%
n23d2	-	-	192	0.011580	0.0%	214	1.460626	-11.45%
n23d3	-	-	144	0.005300	0.0%	148	1.454038	-2.78%
n23d4	-	-	134	0.005511	0.0%	146	1.427827	-8.95%
n23d5	-	-	132	0.004747	0.0%	132	1.440289	0.0%
n23d6	-	-	225	0.004723	0.0%	244	1.446813	-7.45%
n24d2	-	-	288	0.008934	0.0%	288	1.776445	0.0%
n24d3	-	-	164	0.006670	0.0%	179	1.751796	-9.14%
n24d4	-	-	122	0.006041	0.0%	122	1.759900	0.0%
n24d5	-	-	99	0.006432	0.0%	99	1.750699	0.0%
n24d6	-	-	207	0.005662	0.0%	207	1.742212	0.0%

n27d2	-	-	255	0.011501	-18.05%	216	2.816323	0.0%
n27d3	-	-	155	0.009531	0.0%	167	2.788440	-7.74%
n27d4	-	-	130	0.009057	0.0%	130	2.779777	0.0%
n27d5	-	-	117	0.005975	0.0%	132	2.801596	-12.82%
n27d6	-	-	86	0.006383	0.0%	86	2.764790	0.0%
n30d2	-	-	175	0.019332	0.0%	215	4.464686	-22.86%
n30d3	-	-	149	0.012092	0.0%	149	4.412225	0.0%
n30d4	-	-	121	0.011339	0.0%	121	4.399531	0.0%
n30d5	-	-	166	0.009193	0.0%	175	4.428729	-5.42%
n30d6	-	-	117	0.011819	0.0%	117	4.398214	0.0%
n32d2	-	-	216	0.017449	0.0%	219	5.667936	-1.39%

Tabela 2: Tabela contendo o tempo gasto na execução dos programas para instâncias não euclidianas. O traço "-" indica que o algoritmo não foi executado (quando não houve execução foi devido a prevista longa demora). O tempo relatado está em segundos.

Com base nas tabelas 1 e 2 percebe-se que o algoritmo heurístico produz bons resultados e em pouco tempo, sendo ideal para comparar a eficácia de outros algoritmos heurísticos. Para instâncias euclidianas os algoritmos parecem mais estáveis, diferente do que acontece com instâncias não euclidianas, onde, principalmente, o algoritmo bioinspirado apresenta aparente instabilidade, mesmo para instâncias pequenas (para instâncias euclidianas pequenas tal algoritmo tende quase sempre a achar a melhor solução). O algoritmo heurístico tende a encontrar soluções melhores na maioria dos casos, mas, eventualmente, o *Ant System* encontra soluções melhores, mostrando seu poder. Percebe-se também que a restrição de grau não afeta tanto o desempenho da solução, sendo o número de vértices (que afeta o número de arestas possíveis) o principal fator de impacto no tempo gasto pela solução.

Mais uma vez percebe-se que o algoritmo exato demanda muito tempo, não sendo atrativo, principalmente pelo fato dos outros dois algoritmos apresentarem boas (às vezes ótimas) respostas em bem menos tempo. O limite de processamento do algoritmo exato também é algo negativo para o mesmo, já que ele só consegue lidar com instâncias pequenas.

De acordo com o que foi encontrado na literatura esse parece ser, de fato, o "limite" do algoritmo de Colônia de Formigas. Em [BZ06], para alguns tipos de instâncias, com 30 vértices, o programa leva em torno de 2-4 segundos (dependendo do tipo da instância e do grau máximo), enquanto que o algoritmo apresentado aqui demora em torno de 4 segundos. Em [BHE05], considerando o algoritmo *k-ACO* (baseado em arestas usando o *algoritmo de Kruskal*), é apresentado o melhor tempo gasto para uma instância com 50 vértices e grau máximo igual a 5.

Considerando uma instância gerada para o presente trabalho e com o mesmo tamanho, o nosso algoritmo demorou cerca de 5 vezes a mais para encontrar uma solução. Ainda em [BHE05], instâncias consideradas "difíceis" mostraram-se desafiadoras para o algoritmo proposto por eles, de maneira que o programa levou muito tempo para encontrar uma resposta para uma instância com 30 vértices e grau máximo igual a 5.

Instância	Resultado	Tempo gasto
n50d2	475	0.072427
n50d3	409	0.057690
n50d4	487	0.058321
n50d5	451	0.057219
n50d6	419	0.058356
n70d2	643	0.344489
n70d3	593	0.204170
n70d4	538	0.216844
n70d5	571	0.200412
n70d6	537	0.202022
n100d2	794	0.948395
n100d3	653	0.805271
n100d4	646	0.806377
n100d5	625	0.813301
n100d6	578	0.817732
n120d2	864	1.869989
n120d3	721	1.657576
n120d4	620	1.660118
n120d5	580	1.659136
n120d6	702	1.676140
n200d2	1206	14.334644
n200d3	824	12.591498
n200d4	803	12.637128
n200d5	809	12.559576
n200d6	802	12.559729
n300d2	1562	10.328687
n300d3	1051	9.226556
n300d4	948	9.270004
n300d5	1042	9.216142
n300d6	905	9.242652
n400d2	1749	12.298225
n400d3	1149	11.288737
n400d4	1096	11.311935
n400d5	1093	11.260020
n400d6	1102	11.234823
n500d2	1996	3.870720
n500d3	1260	3.223677
n500d4	1196	3.241394
n500d5	1204	3.230087
n500d6	1203	3.220576
n1000d2	2680	13.376668
n1000d3	1793	12.816912
n1000d4	1746	12.705796
n1000d5	1756	12.677630
n1000d6	1703	12.878792

Tabela 3: Tabela contendo o tempo gasto na execução do algoritmo 2 para instâncias euclidianas de tamanho elevado. O tempo relatado está em segundos.

Instância	Resultado	Tempo gasto
n50d2	265	0.060669
n50d3	165	0.052780
n50d4	141	0.052506
n50d5	144	0.052759
n50d6	143	0.053297
n70d2	275	0.219067
n70d3	177	0.204852
n70d4	181	0.197634
n70d5	151	0.199766
n70d6	130	0.196161
n100d2	372	0.890618
n100d3	184	0.809712
n100d4	178	0.820967
n100d5	169	0.804329
n100d6	160	0.807012
n120d2	353	2.245975
n120d3	193	1.663869
n120d4	202	1.663745
n120d5	179	1.682029
n120d6	193	1.674867
n200d2	439	13.217291
n200d3	256	12.678352
n200d4	252	12.733202
n200d5	246	12.805465
n200d6	224	12.753360
n300d2	531	13.491904
n300d3	335	9.387831
n300d4	326	9.455228
n300d5	328	9.439317
n300d6	311	9.269899
n400d2	657	11.570871
n400d3	412	11.213285
n400d4	408	11.184295
n400d5	411	11.218412
n400d6	407	11.127786
n500d2	740	3.342783
n500d3	506	3.167749
n500d4	506	3.150350
n500d5	501	3.152460
n500d6	502	3.178546
n1000d2	1268	16.581386
n1000d3	999	12.734792
n1000d4	999	12.743657
n1000d5	999	12.679602
n1000d6	999	12.670419

Tabela 4: Tabela contendo o tempo gasto na execução do algoritmo 2 para instâncias não euclidianas de tamanho elevado. O tempo relatado está em segundos.

O algoritmo heurístico se comportou bem com os dois tipos de instância, porém não temos nenhum outro resultado para comparar, devido ao tamanho das instâncias. O fato dele apresentar uma boa resposta para instâncias razoavelmente grandes e em pouco tempo o torna interessante para uso. Se a resposta dada por ele não for a melhor, deve ser uma bem próxima da melhor. Algo interessante foi o desempenho dele para instâncias com 500 vértices, que foi mais rápido do que para instâncias com 200 e 300 vértices. Não se soube identificar a causa do desempenho obtido para essas instâncias. Talvez o parâmetro *escape* seja responsável por isso.

5 Considerações finais

Este trabalho apresentou uma abordagem heurística para solucionar o problema da Árvore Geradora Mínima de Grau Restrito. Para tanto, foram apresentados dois algoritmos: um faz uso da Otimização por Colônia de Formigas e o outro é um algoritmo heurístico orientado por um processo de *backtracking*. O *Ant System* se mostrou bastante adequado ao problema, apresentando boas respostas e robustez. Todavia, ele se mostrou um pouco lento, de forma que ainda não é a melhor solução para o problema. Para comparar os resultados produzidos pelo *Ant System* implementou-se um algoritmo heurístico que interrompe um processo de *backtracking* em um momento específico. Esse algoritmo é melhor do que o algoritmo guloso porque consegue procurar, até certo momento, se existe uma solução melhor. Nos experimentos percebeu-se que esse algoritmo possibilitou uma boa métrica, assim como deu respostas (que são boas) para instâncias grandes. Todavia, em algumas poucas vezes o *Ant System* apresentou melhores resultados.

Percebe-se que uma abordagem heurística é necessária, e aceitável, para este problema.

Na tentativa de encontrar uma solução melhor pode-se tentar o seguinte: melhorar o algoritmo heurístico usando *branch-and-bound* ou procurar outras formas de resolução de problema, que não sejam, necessariamente, meta-heurísticas.

Ao longo dos dois trabalhos também foram apresentados dois programas que podem servir em trabalhos futuros sobre esse problema ou até de outros do gênero: o validador e o gerador de instâncias. Por estarem implementados em uma linguagem popular e disponíveis no repositório *online* apresentado na seção 1, eles podem ser úteis em outros trabalhos, podendo ser parcialmente ou completamente utilizados (desde que a fonte seja citada). Além disso, os casos de teste criados podem, e são úteis, em abordagens para este problema. O projeto também está disponível no mesmo repositório, o que facilita eventuais consultas e pesquisas ou até dar ideias para implementações de soluções para esse ou outros problemas.

6 Referências

[ALM06] Rafael Andrade, Abilio Lucena, and Nelson Maculan. Using lagrangian dual information to generate degree constrained spanning trees. *Discrete Applied Mathematics*, 154(5):703–717, 2006.

[Ank] Martin Ankerl. Optimized approximative pow() in c / c++. `http://martin.ankerl.com/2012/01/25/optimized-approximative-pow-in-c-and-cpp/`. Acessado em 22-11-2015.

[BDZ12] Thang N Bui, Xianghua Deng, and Catherine M Zrncic. An improved ant-based algorithm for the degree-constrained minimum spanning tree problem. *Evolutionary Computation, IEEE Transactions on*, 16(2):266–278, 2012.

[BHE05] Yoon-Teck Bau, Chin Kuan Ho, and Hong Tat Ewe. An ant colony optimization approach to the degree-constrained minimum spanning tree problem. In *Computational Intelligence and Security*, pages 657–662. Springer, 2005.

[BZ06] Thang N Bui and Catherine M Zrncic. An ant-based algorithm for finding degree-constrained minimum spanning tree. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 11–18. ACM, 2006.

[dFV09] Felipe Fonseca Tavares de Freitas and Guilherme Ernani Vieira. Tendências de aplicações da otimização por colônia de formigas na progamação de job-shops. *Revista Produção Online*, 10(1), 2009.

[Dor92] Marco Dorigo. Optimization, learning and natural algorithms. *Ph. D. Thesis, Politecnico di Milano, Italy*, 1992.

[dSM08] Mauricio C de Souza and Pedro Martins. Skewed vns enclosing second order algorithm for the degree constrained minimum spanning tree problem. *European Journal of Operational Research*, 191(3):677–690, 2008.

[Ern10] Andreas T Ernst. A hybrid lagrangian particle swarm optimization algorithm for the degree-constrained minimum span-

ning tree problem. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8. IEEE, 2010.

[GJ79] Michael R Garey and David S Johnson. Computers and intractability: a guide to the theory of np-completeness. 1979. *San Francisco, LA: Freeman*, 1979.

[GK03] Fred W. Glover and Gary A. Kochenberger. *Handbook of Metaheuristics*. Springer US, 2003.

[KC00] Joshua Knowles and David Corne. A new evolutionary approach to the degree-constrained minimum spanning tree problem. *Evolutionary Computation, IEEE Transactions on*, 4(2):125–134, 2000.

[Kru56] Joseph B Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956.

[NH80] Subhash C Narula and Cesar A Ho. Degree-constrained minimum spanning tree. *Computers & Operations Research*, 7(4):239–249, 1980.

[Per07] João Paulo Gonçalves Pereira. Heurísticas computacionais aplicadas à otimização estrutural de treliças bidimensionais. 2007.

[Pri57] Robert Clay Prim. Shortest connection networks and some generalizations. *Bell system technical journal*, 36(6):1389–1401, 1957.

[PZ07] Ignacio Payá-Zaforteza. Optimización heurística de pórticos de edificación de hormigón armado. 2007.

[RJ00] Günther R Raidl and Bryant A Julstrom. A weighted coding in a genetic algorithm for the degree-constrained minimum spanning tree problem. In *Proceedings of the 2000 ACM symposium on Applied computing-Volume 1*, pages 440–445. ACM, 2000.

[Tor13] Javad Akbari Torkestani. Degree constrained minimum spanning tree problem: a learning automata approach. *The Journal of Supercomputing*, 64(1):226–249, 2013.

[ZG97] Gengui Zhou and Mitsuo Gen. A note on genetic algorithms for degree-constrained spanning tree problems. *Networks*, 30(2):91–95, 1997.