Universidade Federal de Juiz de Fora

Departamento de Ciência da Computação

Teoria dos Grafos

Algoritmos Construtivos para o Problema do Subconjunto Dominante Mínimo

Grupo 19

João Victor Lopes Borges – 201665528AB

Mathews Edwirds Gomes Almeida – 201765503AB

Matheus dos Reis Casarim – 201765512AB

Professor: Stênio Sã Rosário F. Soares

Relatório do trabalho final da disciplina DCC059 - Teoria dos Grafos, parte integrante da avaliação da mesma.

Juiz de Fora

Novembro de 2020

1. Introdução

O Relatório Técnico presente tem como objetivo central descrever o uso dos algoritmos gulosos construtivos para o Problema do Subconjunto Dominante Mínimo. Considerou-se neste trabalho as características do problema modelado sobre um grafo simples, usando lista de adjacências como principal estrutura de dados. Foram desenvolvidos dois algoritmos, guloso e randomizado. Os mesmos foram avaliados sobre um conjunto de instâncias e os resultados foram comparados com os apresentados na secção 4 deste relatório.

O restante do trabalho está estruturado da seguinte forma: na Seção 2 o problema é descrito formalmente através de um modelo em grafos; a Seção 3 descreve as abordagens propostas para o problema, enquanto a Seção 4 apresenta os experimentos computacionais, onde se descreve o design dos experimentos, o conjunto de instâncias (*benchmarks*), bem como se apresenta a análise comparativa dos resultados dos algoritmos desenvolvidos e a literatura; por fim, a Seção 5 traz as conclusões do trabalho e propostas de trabalhos futuros.

2. Descrição do problema

O Problema do Subconjunto Dominante, refere-se à necessidade de se encontrar o conjunto de vértices em um grafo de forma a dominar os demais nós do mesmo, sem que sejam inseridos vértices já dominados no agrupamento, evitando assim, duplicidades no subconjunto solução. De modo que cada elemento ou será selecionado ou será vizinho de um vértice já selecionado.

No que se refere ao Problema do Subconjunto Dominante Mínimo, estamos preocupados em encontrar o subconjunto de vértices de menor tamanho do grafo analisado.

As aplicações práticas desse problema, encontram-se na criação e elaboração de redes sem fio, na detecção precoce e vacinação dirigida em surtos de doenças infecciosas[1] e em aplicações de redes sociais.

3. Abordagens gulosas para o problema

Algoritmos Gulosos são soluções simples e de fácil implementação para problemas em que se espera que ao escolher a opção mais promissora dada a iteração atual, a mesma acarreta em menores custos e evite problemas futuros em nível global, sendo que para cada tomada de decisão há uma tratativa diferente de acordo com o tipo de algoritmo guloso escolhido. Para a resolução do problema do Subconjunto Dominante Mínimo, o critério utilizado para tomadas de decisões locais trata-se da análise referente aos vértices com maior número de vizinhos ao seu redor, também foram estudados e implementados 2 algoritmos gulosos: simples e randomizado, os quais serão melhor explicitados a seguir.

3.1 Algoritmo guloso

O Algoritmo Guloso é utilizado como solução comum para problemas onde buscamos a solução global otimizada, dado que as escolhas são definitivas e o algoritmo não sofre arrependimentos durante a sua execução. A concepção da solução por meio deste algoritmo começa pela elaboração de uma lista de candidatos a qual é alterada a cada etapa, sendo que o melhor candidato é o que apresenta o maior número de vizinhos que ainda não estão presentes no conjunto solução. A cada etapa é adicionado o melhor nó local, ou seja, aquele que possui mais vizinhos que não estão presentes na solução, após essa adição é recalculado o possível melhor candidato.

Algorithm 1: Algoritmo Guloso simples genérico

```
1:  funcao ALGORITMOGULOSO();
2:  s ← ∅ ;
3:  candidatos ← |V| ;
4:  nosDominados ← false;
5:  while numInteracoes < |V| do
6:  |   initialize candidatos;
7:  end
8:  while !nosDominados do
9:  |   s ← candidatos[melhorCandidato];
10: |   nosDominados[melhorCandidato] ← true;
11: |   while aresta ≠ NULL do
12: |   |   nosDominados[aresta → id]← true;
13: |   end
14: |   remove candidatos[melhorCandidato];
15: |   while |candidatos| do
16: |   |   candidatos → numVizinhos ←∑ vizinhos == false
17: |   end
18: end
```

3.2 Algoritmo guloso randomizado

O algoritmo randomizado irá seguir o processo de tomada de decisões do guloso inicialmente proposto, porém, com a diferença de que ao invés de se coletar sempre o candidato supostamente mais promissor (maior número de vizinhos), trocando-o por uma janela de possíveis melhores candidatos onde a escolha dos elementos do conjunto solução tem como base um valor aleatorizado com auxílio de um parâmetro alfa fornecido na função.

Algorithm 1: Algoritmo Guloso Randomizado

```
funcao ALGORITMOGULOSORANDOMIZADO(alpha, seed);
Input: alpha, seed
Output: melhorSolucao
1: s ← ∅ ;
2: melhorSolucao ← ∅ ;
3: candidatos ← |V| ;
4: nosDominados ← false;
5: srand(seed);
6: while do
7:     while numInteracoes < |V| do
8:         initialize candidatos;
9:     end
10:    while !nosDominados do
11:        s ← candidatos[melhorCandidato];
12:        nosDominados[melhorCandidato] ← true;
13:        while aresta ≠ NULL do
14:            nosDominados[aresta → id]← true;
15:        end
16:        remove candidatos[melhorCandidato];
17:        while l < |candidatos| do
18:            if candidatos[l].numVizinhos ≥ melhorCandidato.numVizinhos −
                melhorCandidato.numVizinhos ∗ alpha then
19:                candidatos → numVizinhos ←∑ vizinhos == false;
20:                cont++;
21:            end
22:        end
23:        random ← rand() % cont;
24:        solucao ← candidatos[candidatos.size() - random - 1].verticeId;
25:        dominados[solucao] ← true;
26:        aresta ← solucao → aresta;
27:        candidatos.erase(candidatos.end() - random - 1);
28:        if melhorSolucao = NULL then
29:            melhorSolucao = solucao;
30:        else
31:            if melhorSolucao.size > solucao.size then
32:                melhorSolucao = solucao;
33:            else
34:                delete solucao;
35:            end
36:        end
37:    end
38: end
```

4. Experimentos computacionais

Nesta seção estão descritos os experimentos computacionais realizados, especificando as instâncias e as configurações da máquina utilizada para realizar o experimento.

4.1. Descrição das instâncias

As instâncias utilizadas e suas informações foram disponibilizadas pelo Professor Stênio Sã na plataforma classroom.

Instância	Nº de Vértices	Nº de arestas
Problem.dat_50_50_0	50	50
Problem.dat_50_100_0	50	100
Problem.dat_50_500_0	50	500
Problem.dat_100_250_0	100	250
Problem.dat_100_2000_0	100	2000
Problem.dat_300_300_0	300	300
Problem.dat_300_1000_0	300	1000
Problem.dat_500_1000_0	500	1000
Problem.dat_800_0_0	800	799
Problem.dat_800_5000_0	800	5000

Tabela 1: Instâncias utilizadas no experimento

4.2. Ambiente computacional do experimento e conjunto de parâmetros

Todos os testes foram feitos em ambiente Linux (Ubuntu) utilizando o terminal do Ubuntu via linha de comando (g++ -c *.cpp;g++ -o main -O3 *.o;./main <arquivoEntrada> <arquivoSaida>). O código foi implementado em C++ e as configurações da máquina utilizada para testes foram:

Máquina	Notebook Acer Aspire 5
Sistema Operacional	Ubuntu 9.3.0-17 / Windows 10
Processador	Intel Core i5 - 8265U 1.6GHz
Memória	DDR4 2400MHz SDRAM 8GB
Versão do compilador GCC	(Ubuntu 9.3.0-17 ubuntu1~20.04) 9.2.0

Tabela 2: Configurações da máquina

4.3. Resultados quanto à qualidade e tempo

Após o desenvolvimento do problema, são demonstrados na tabela 3 abaixo, os resultados com relação a qualidade de cada instância, referindo-se ao conjunto dominante mínimo obtido.

Onde, na tabela 3, tem-se a segunda coluna representando a melhor solução obtida para cada instância, quando comparados os 6 tipos de processos utilizados. As demais colunas, apresentam o subconjunto mínimo obtido por cada algoritmo, com seu respectivo critério.

Seguindo, adiante encontra-se a tabela 4, onde são comparados os métodos, e sua eficácia em relação aos demais. Novamente, a segunda coluna representa o melhor subconjunto solução do problema, para cada instância, em seguida, a terceira

coluna apresenta o RDI calculado para cada algoritmo levando-se em conta seus critérios individuais, ressaltando o fato de que tal valor é obtido através do seguinte cálculo de desvio percentual médio:

- *RDI (a) = 100 x (Media (a) - b) / (c - b)*

Sendo "*a*" o algoritmo utilizado, portanto "*Media (a)*" representa o valor médio das soluções obtidas pelo mesmo, seguindo, "*b*" trata-se da melhor média obtida dentre todos os algoritmos e por fim "*c*" assume o pior dos casos calculados.

Adiante, já na quarta coluna são coletadas as informações referentes ao tempo de processamento de cada execução analisada. Em seguida, as demais intercalam-se entre RDI calculado para cada evento e novamente o tempo do mesmo.

Assim, as heurísticas de construção foram executadas para cada instância em duas diferentes formas: gulosas e gulosas randomizadas. Os valores em negrito, destacados, indicam as melhores soluções na execução de cada instância.

Com o intuito de realizar os testes comparativos, foi utilizado um conjunto de 10 instâncias, onde as 4 primeiras são consideradas pequenas (poucos vértices e poucas arestas), seguidas das 3 subsequentes intermediárias, e finalmente as demais 3 grandes instâncias.

Observando mais atentamente os dados coletados, é possível destacar algumas constatações iniciais:

- O algoritmo guloso, possui em 100% dos casos o melhor tempo de execução, fato este que se dá através de sua abordagem direta sobre cada uma das instâncias analisadas, uma vez que o mesmo toma decisões locais de acordo com o princípio de vértice com maior número de vizinhanças, optando sempre pelo candidato a princípio mais promissor. Sem considerar outras opções que inicialmente, não se apresentam como as melhores, porém, poderiam representar melhoras significativas, em uma solução mais adequada ao problema.
- A declaração anterior, se confirma ao realizar a comparação do RDI de cada instância, onde em poucos casos, o algoritmo guloso se apresentou como a melhor solução (RDI = 0), nos demais, mesmo nas ocasiões onde o algoritmo

guloso randomizado, possui um alfa mais elevado, ainda sim, as médias apresentadas eram superiores ao guloso.

- Os melhores resultados referentes ao sub conjunto dominante mínimo, encontraram-se, no guloso randomizado (com alfa = 0.1), em todos os casos testados. A explicação para este fenômeno, se dá pelo fato de um valor mais baixo de alfa, concentrar cada vez mais a tomada de decisões do algoritmo, nos candidatos que apresentam-se como promissores localmente, devido a heurística utilizada que visa o número de vértices adjacentes ao candidato. Comprovando que, a vizinhança realmente é um fator preponderante na resolução do problema.
- Tais resultados, confirmam a hipótese de que a randomização das escolhas locais, aumentam a probabilidade de melhores soluções. E ainda, que a heurística escolhida aparenta ser a mais eficaz, uma vez que, demais métodos provaram-se inferiores aos resultados do algoritmo guloso, métodos estes que foram testados internamente pelo grupo.
- Por fim, as instâncias estudadas, também representam um fator importante na conclusão do trabalho, pois, como observado abaixo, quanto maior o grafo analisado, tanto em número de vértices quanto de arestas, maiores as possibilidades disponíveis para escolha de candidatos a solução, nas execuções randomizadas, o que justifica a maior discrepância de RDI's nas últimas instâncias analisadas.

Valores Médios dos Algoritmos							
Instância	**Melhor Média**	**Guloso**	**Randomizado 0.1**	**Randomizado 0.2**	**Randomizado 0.3**	**Randomizado 0.5**	**Randomizado 0.7**
Problem.dat_50_50_0	**18**	25	**18**	19	19	21	22
Problem.dat_50_100_0	**13**	15	**13**	14	**13**	15	16
Problem.dat_50_500_0	**4**	**4**	**4**	**4**	**4**	**4**	5
Problem.dat_100_250_0	**21**	26	**21**	24	24	27	30
Problem.dat_100_2000_0	**5**	**5**	**5**	**5**	**5**	6	6
Problem.dat_300_300_0	**119**	135	**119**	131	132	140	153
Problem.dat_300_1000_0	**55**	61	**55**	62	64	73	84
Problem.dat_500_1000_0	**133**	137	**133**	147	149	161	175
Problem.dat_800_0_0	**329**	340	**329**	362	366	420	426
Problem.dat_800_5000_0	**94**	97	**94**	105	109	127	151

Tabela 3: Valores Médios dos algoritmos

Desvio Percentual Médio												
Instância	**Melhor Média**	**Guloso**	**Tempo**	**Randomizado 0.1**	**Tempo**	**Randomizado 0.2**	**Tempo**	**Randomizado 0.3**	**Tempo**	**Randomizado 0.5**	**Tempo**	**Randomizado 0.7**
Problem.dat_50_50_0	**18**	100	**33ms**	**0**	4s	14,28	5.2s	14,28	5.72s	42,85	5.472s	57,14
Problem.dat_50_100_0	**13**	66,67	**40ms**	**0**	3.3s	33,34	7.23s	**0**	11.82s	66,67	10.88s	100
Problem.dat_50_500_0	**4**	**0**	**18ms**	**0**	4.2s	**0**	6.22s	**0**	6.52s	**0**	7.05s	100
Problem.dat_100_250_0	**21**	55,56	**0ms**	**0**	13.10s	33,33	14.31s	33,33	14.33s	66,67	14.23s	100
Problem.dat_100_2000_0	**5**	**0**	**0ms**	**0**	6s	**0**	7s	**0**	7.7s	100	9.77s	100
Problem.dat_300_300_0	**119**	47,06	**0ms**	**0**	135.1s	35,29	153.81s	38,24	147.78s	61,76	162.28s	100
Problem.dat_300_1000_0	**55**	20,68	**0.93s**	**0**	133.3s	24,13	146.6s	31,03	14.66s	62,07	146.6s	100
Problem.dat_500_1000_0	**133**	9,52	**2.53s**	**0**	466.6s	33,33	513.3s	38,09	518s	66,66	545.1s	100
Problem.dat_800_0_0	**329**	11,34	**6.33s**	**0**	1380s	34,02	1538s	38,14	1500s	93,81	1516s	100
Problem.dat_800_5000_0	**94**	5,26	**3.9s**	**0**	1309s	19,29	1470s	26,31	1447s	57,89	1644s	100

Tabela 4: Desvio Percentual Médio

5. Conclusões e trabalhos futuros

O problema proposto, consistiu em calcular de modo eficiente um subconjunto solução para o conceito descrito nos primeiros tópicos listados anteriormente, de acordo com a teoria abordada sobre a disciplina de Grafos (DCC-059). Entre as dificuldades da equipe, as principais envolveram o desenvolvimento de uma heurística eficiente para o problema, uma vez que cada tema possui uma abordagem diferente, e ainda a modelagem de diferentes instâncias utilizando-se diferentes tipos de arquivos de entrada, visto que, este projeto funciona tanto para as instâncias da fase 1, quanto da segunda etapa do problema.

Para sempre obter o subconjunto dominante mínimo de um grafo, seria necessário executar um algoritmo de força bruta, algo que se mostraria inviável pelo longo tempo de execução exigido.

Buscando novos horizontes de aprimoramento da pesquisa aqui apresentada, existe a possibilidade de aprofundar estudos em diferentes heurísticas não tratadas no texto, e também não estudadas pela equipe, no intuito de melhorar os resultados coletados, caso existam. E ainda novos métodos de abordar o problema excluindo-se os algoritmos gulosos (e suas variações), bem como também a força bruta. Fica em aberto, a possibilidade de demais abordagens que fujam deste conceito.

Referências Bibliográficas

[1] Eubank, S., Guclu, H., Anil Kumar, V. *et al.* **Modelling disease outbreaks in realistic urban social networks.** *Nature* **429,** 180–184 (2004). Disponível em: https://www.nature.com/articles/nature02541?message=remove&lang=pt. Acesso em: 14 nov. 2020.

[2] ALBUQUERQUE, Mayra Carvalho. **Matheuristics for Variants of the Dominating Set Problem**. 2018. 88 f. Tese (Doutorado) - Curso de Programa de Pós–Graduação em Informática, Departamento de Informática, Puc-Rio, Rio de Janeiro, 2018.

[3] P. O. BOAVENTURA NETTO. **Grafos: Teoria, Modelos e Algoritmos.** Editora Edgard Blucher Ltda, 1996.

[4] ROCHA, Anderson; DORINI, Leyza E. Baldo. **Algoritmos gulosos: definições e aplicações.** 2004. Instituto de Computação, Unicamp, Campinas.