Universidade Federal de Juiz de Fora
Departamento de Ciência da Computação
Teoria dos Grafos

Título do Relatório - Por exemplo: Algoritmos Construtivos para o Problema da Árvore de Steiner em Grafos

Grupo 14

João Vitor Gomes da Silva - MAT 202035035

Kaio Vinycius Braga dos Santos - MAT 202035019

Charles Lelis Braga - MAT 202035015

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

Janeiro de 2023

1 Introdução

O presente Relatório Técnico tem como objetivo central descrever o uso de algoritmos construtivos gulosos para o Problema do Subconjunto Dominante Ponderado. Considerou-se neste trabalho as características do problema modelado sobre um grafo não direcionado G = (V, E). Foram desenvolvidos três algoritmos gulosos e randomizados. Os mesmos foram avaliados sobre um conjunto de instâncias e os resultados foram comparados com os apresentados em [**?**].

O restante do trabalho está assim estruturado: 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

Dado um grafo não direcionado G = (V, E) e uma função f:$V \rightarrow A$ que associa a cada vértice de V um peso, apresente um subconjunto $D \subseteq V$, tal que, para todo vértice $i \in V$, $i \in D$ ou existe uma aresta $(i, j) \in E$ com $j \in D$ e a soma dos pesos dos vértices de D seja mínima. A minimização é feita com o objetivo de diminuir o valor das somas dos pesos dos vertices

3 Abordagens gulosas para o problema

Nesta seção são descritos os algoritmos desenvolvidos. Aqui, deve-se justificar a escolha da função critério utilizada e a estratégia de atualização da lista de candidatos.

3.1 Algoritmo guloso

O algoritmo guloso é um algoritmo de busca que, a cada passo, escolhe a opção que parece ser a "mais promissora"naquele momento. Essa escolha é baseada em uma função de escolha (também conhecida como função heuristica), que avalia cada opção e escolhe a melhor. O algoritmo adotado é $10/peso + grau(V)$

Algorithm 1: Algoritmo Guloso

Input: S, e

Output: *Solução completa*

```
1 Inicialize a solução S como conjunto vazio;
2 while Sfinal ≠ completa do
3 |   e = heuristica(V, E);
4 |   if e ∪ s ∈ F then
5 |   |   s = s ∪ e
6 end
7 return S;
8
```

3.2 Algoritmo guloso randomizado

O algoritmo guloso randomizado é o mesmo que o de um algoritmo guloso comum. Ele começa com uma solução vazia e, em seguida, itera até que a solução esteja completa. Em cada iteração, o algoritmo escolhe a próxima opção de acordo com a função de escolha e adiciona a opção escolhida à solução. Depois disso, ele atualiza as informações da solução e verifica se a solução está completa. Se sim, o algoritmo termina e retorna a solução. Se não, ele continua iterando.

Algorithm 2: Algoritmo Guloso Randomizado

Input: $I, alfa, numInter$

Output: *Solução completa*

```
1  solBest = {}, s i = 1, k
2  while i < numIter do
3  |   LC = ordenaCandidatos(I);
4  |   i++;
5  |   s = { };
6  |   while S não finalizada do
7  |   |   k = randomRange(0,alfa*LC.count()-1));
8  |   |   if sol U LC[k] é viável then
9  |   |   |   sol = sol U LC[k];
10 |   |   atulaizaListaCandidatos(LC, k);
11 |   end
12 end
13 return solBest;
```

3.3 Algoritmo guloso randomizado reativo

O algoritmo é semelhante ao algoritmo guloso randomizado, começando com uma solução vazia e iterando até que a solução esteja completa. Em cada iteração, o algoritmo escolhe a próxima opção de acordo com a função de escolha e adiciona a opção escolhida à solução. Depois disso, ele atualiza as informações da solução e verifica se a solução está completa. Se sim, o algoritmo termina e retorna a solução. Se não, ele continua iterando.

Algorithm 3: Algoritmo Guloso Randomizado Reativo

Input: $I, alfa, numInter, bloco$
Output: *Solução completa*

```
1  solBest = {}, s i = 1, k
2  inicializaVetores(P, A, m);
3  while i < numIter do
4      if i % bloco == 0 then
5          atualizaProbabilidades(P,A,solBest);
6      LC = ordenaCandidatos(I);
7      i++;
8      s = { };
9      α = escolheAlfa(P);
10     while S não finalizada do
11         k = randomRange(0,*LC.count()-1));
12         if sol U LC[k] é viável then
13             sol = sol U LC[k];
14         atulaizaListaCandidatos(LC, k);
15     end
16     atualizarMedias(A, s, α);
17     if s.val < solBest.val then
18         sol = sol U LC[k];
19 end
20 return solBest;
```

4 Experimentos computacionais

Para cada instancia, foi executado uma unica vez uma configuração de algoritmo. Uma vez o algoritmo guloso tres vezes o algoritmo randomizado, uma para cada alpha (0.15,0.30,0.50) com 500 interações e foram executadas uma vez para cada instancia o algoritmo reativo com 500 interações

4.1 Descrição das instâncias

São grafos não direcionados não nulos com vertices ponderadas

1.

Instance Name	**#edge**	**#vertex**	$\#K$	$\#L$
Problem.dat_50_50_3	50	50	3	552
Problem.dat_50_250_3	50	250	3	192
Problem.dat_100_250_3	100	250	3	672
Problem.dat_100_500_3	100	500	3	432
Problem.dat_150_150_3	150	150	3	1608
Problem.dat_250_750_3	250	750	3	1640
Problem.dat_300_500_3	300	500	3	2723

Tabela 1: Tabela com descrição das instâncias 3-regular

4.2 Ambiente computacional do experimento e conjunto de parâmetros

O ambiente computacional utilizado foi a linguagem c, sendo compilado no compilador Gcc, com o processador Intel Xeon(R) CPU ES-1650 0 3.20GHz x 12,com o gerador de numeros aleatorios mersenne_twister_engine.

Para cada instancia, foi executado uma unica vez uma configuração de algoritmo. Uma vez o algoritmo guloso tres vezes o algoritmo randomizado, uma para cada alpha (0.15,0.30,0.50) com 500 interações e foram executadas uma vez para cada instancia o algoritmo reativo com 500 interações

4.3 Resultados quanto à qualidade e tempo

Usamos este cálculo para diferença percentual:

$$gap(a) = \frac{Media(a) - b}{b} \tag{1}$$

Em geral, a versão randomizada do algoritmo guloso tende a ter um custo menor do que o algoritmo guloso original, mas leva muito mais tempo para rodar. A diferença de custo entre os dois algoritmos geralmente é maior quando as instâncias são maiores (por exemplo, quando o número de itens e/ou o número de mochilas é maior).

Instância	**Melhor / Lit.**	**Guloso**	**Randomizado**			**Reativo**	
			0,15	**0,30**	**0,50**		
Problem.dat_50_50_3	552,00	1,39	1,25	1,25	1,25	0,96	
Problem.dat_50_250_3	192,00	3,02	2,13	2,13	2,13	1,38	
Problem.dat_100_250_3	672,00	2,27	1,90	1,90	1,90	1,12	
Problem.dat_100_500_3	432,00	2,60	1,66	1,66	1,66	1,41	
Problem.dat_150_150_3	1608,00	1,82	1,73	1,73	1,73	1,51	
Problem.dat_250_750_3	1640,00	2,75	2,44	2,44	2,44	1,62	
Problem.dat_300_500_3	2723,00	2,24	2,12	2,12	2,12	1,69	
Problem.dat_300_2000_3	1533,00	2,01	1,26	1,26	1,26	1,05	
Problem.dat_500_500_0	1825,00	7,82	7,67	7,67	7,67	7,22	
Problem.dat_500_500_3	1824,00	7,50	7,31	7,31	7,31	6,64	

Tabela 2: Tabela de comparação dos resultados da literatura com os resultados dos algoritmos de construção

Também é importante notar que o custo do algoritmo guloso é afetado pelo valor do terceiro parâmetro no nome do arquivo de instâncias, que é 3 na maioria dos casos. Não é especificado o que este parâmetro representa na definição do problema.

É importante notar também que a versão randomizada do algoritmo guloso com alfa = 0,3 parece ter custos menores do que a versão com alfa = 0,15, mas o tempo de execução é semelhante.

O algoritmo guloso tem uma Taxa de Comparação em torno de 2 para a maioria das instâncias, indicando que a solução obtida é aproximadamente o dobro do custo da solução ótima.

As versões randomizadas do algoritmo guloso geralmente possuem taxas de comparação mais baixas do que o algoritmo guloso, indicando que suas soluções estão mais próximas da solução ótima. A taxa de comparação diminui à medida que o valor de alfa aumenta. Por exemplo, a taxa de comparação é 1,19 quando alfa = 0,5.

Parece que a versão reativa do algoritmo randomizado tem desempenho semelhante em termos de custo e tempo de execução, mas a amostra fornecida é pequena para uma conclusão definitiva.

Em geral, parece que a versão aleatória do algoritmo guloso é um trade-off entre custo e tempo, pode alcançar soluções melhores do que o algoritmo guloso, mas leva muito mais tempo para ser executado.

Instância	**Guloso**	**Randomizado**			**Reativo**
		0,15	**0,30**	**0,50**	
Problem.dat_50_50_3	0,00238	1,11651	1,11651	1,11651	1,49959
Problem.dat_50_250_3	0,00109	0,60135	0,60135	0,60135	0,76239
Problem.dat_100_250_3	0,00950	4,42924	4,42924	4,42924	5,61792
Problem.dat_100_500_3	0,00685	2,48772	2,48772	2,48772	3,28504
Problem.dat_150_150_3	0,04275	18,20760	18,20760	18,20760	24,52960
Problem.dat_250_750_3	0,15807	56,87820	56,87820	56,87820	62,31200
Problem.dat_300_500_3	0,31646	158,95600	158,95600	158,95600	136,00400
Problem.dat_300_2000_3	0,11067	46,41200	46,41200	46,41200	48,18680
Problem.dat_500_500_0	1,49902	737,49600	737,49600	737,49600	572,17300
Problem.dat_500_500_3	1,42826	743,38800	743,38800	743,38800	580,81100

Tabela 3: Tabela de comparação dos tempos de execução dos resultados dos algoritmos de construção

5 Conclusões e trabalhos futuros

Ao final do experimento notamos que o melhor algoritmo neste caso foi Algoritmo Randomizado.

Referências

[1] A. El-Sayed and A. Eltawil, "Matheuristics for Variants of the Dominating Set Problem,"Springer, 2019.