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

Proposta de Solução para Árvore Geradora Mínima com Restrição de Grau

Grupo 15

Antônio Marcos Souza Pereira - 202065245A

Antony Leme Novais Ferreira - 202065009A

Igor Jose Costa Gonçalves - 202065138A

Vinícius de Oliveira Corbelli - 202065093A

Professores: Stênio Sã Rosário F. Soares
Luciana Brugiolo Gonçalves

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

Juiz de Fora

Setembro de 2021

1 Introdução

O presente relatório tem por objetivo descrever o desenvolvimento de três algoritmos - guloso, guloso randomizado e guloso randomizado reativo - para o problema de Árvore Geradora Mínima com Restrição de Grau Máximo, consiste na obtenção de uma árvore geradora mínima de um grafo onde cada vértice ou é folha da árvore ou satisfaz uma restrição de grau máximo. Os mesmos foram avaliados sobre um conjunto de instâncias, obtidos em [1], e os resultados foram comparados com os apresentados em [2].

2 Descrição do problema

O problema consiste em obter um uma solução que seja considerada ótima e para chegar até esta solução deve ser considerado um conjunto de soluções viáveis para o problema. Neste caso, o problema de obter soluções viáveis explora a necessidade de obter uma Árvore Geradora com Restrição de Grau, o que significa que os algoritmos não apenas precisam retornar uma árvore de custo mínimo, como também levar em consideração do grau máximo possível para um determinado vértice dessa solução, sendo esta medida como desclassificatória caso não atenda aos requisitos do problema.

Tendo obtido as soluções viáveis para o problema, o custo mínimo das soluções entra em consideração, ou seja, podem haver soluções viáveis(Árvore Geradora Com Restrição de Grau) sem necessariamente apresentar o custo mínimo possível dentre todas as alternativas. Portanto, um segundo desafio consiste em, dado as soluções viáveis, considerar aquela que fornece um custo mínimo dentre todas possíveis.

3 Abordagens gulosas para o problema

Para abordar o problema da Árvore Geradora com Restrição de Grau Máximo foi utilizado o Algoritmo de Kruskal, que visa construir uma árvore independentemente de grau em um primeiro momento, verificando apenas validade e qualidade das soluções, ou seja, onde o conjunto de arestas forma uma componente conexa e sem ciclos. Neste problema foi utilizado uma ordenação das arestas, que é um dos fatores que podem contribuir e facilitar para os algoritmos gulosos fazerem suas "escolhas" de uma maneira mais otimizada, levando os custos de cada adição de arestas em consideração. A ideia do algoritmo guloso, de tomar uma decisão sem se arrepender pressupõe a necessidade de parâmetro que lhe permita fazer escolhas consideradas ótimas dentro de um range de parâmetros válidos(soluções possíveis). Tendo gerado o conjunto de soluções viáveis a próxima etapa consiste em desqualificar aquelas que infringem a característica principal do problema que é o limite máximo "d" de um nó presente em cada solução. Para o problema dos algoritmos gulosos abaixo, foi utilizado um fator penalizador para orientar a seleção de arestas pelos algoritmos gulosos.

3.1 Algoritmo guloso

Neste algoritmo as arestas são inicialmente ordenadas através de seu score, pois para o problema do algoritmo guloso, o fator penalizador é constante, ou seja, dado que cada aresta tem seu peso o algoritmo se baseia em uma ordenação crescente que possibilita fazer escolhas otimizadas em relação ao custo total da solução. Dessa forma, o algoritmo de kruskal apresenta uma ordenação inicial sugerindo uma sequência de escolhas de arestas que forme uma árvore, sendo esta não necessariamente uma solução ótima. Para este algoritmo é definido como uma estratégia gulosa de escolha um conjunto ordenado de arestas pois assim o algoritmo tem a certeza de estar escolhendo a melhor opção dentre todas as possíveis.

Algorithm 1: Algoritmo Guloso

```
Input: Número máximo do grau do nó
Output: Melhor solução
foreach Aresta ∈Grafo do
    Seta a pontuação para cada Aresta do Grafo
    listEdgesAux ← Aresta
end
foreach ArestaListEdge ∈ listEdgesAux do
    contador ← 0
    foreach ArestaListFinal ∈ listEdgesFinal do
        if ArestaListEdge != ArestaListFinal then
            contador ← +1
    end
    if contador == tamanho de listEdgeFinal then
        arestaListEdgesFinal ← Aresta
    Ordena o array listEdgesFinal
    foreach no ∈ Grafo do
        AGMRG ← no
        noPai ← no
    end
    custoTotalArvore ← 0
    foreach Aresta ∈ listEdgesFinal do
        if grau da Aresta < número máximo do grau do nó then
            if não é ciclo then
                AGMRG ← Aresta
                custoTotalArvore ← + peso da Aresta
    end
    Imprime melhor solução
end
```

3.2 Algoritmo guloso randomizado

Considerando que o algoritmo tem a ordenação dos pesos das arestas como uma boa possibilidade para as escolhas gulosas do algoritmo, a ordenação sempre será em uma ordem fixa, logo, para gerar novas soluções é necessário gerar uma randomização das soluções viáveis. Essa randomização é definida no algoritmo como um fator(score) gerado em um range[1,5] multiplicado pelo peso da aresta mais um número aleatório multiplicado pelo peso do nó. Isso possibilita que uma nova sequência de arestas seja formada e que as escolhas do algoritmo variem.

Algorithm 2: Algoritmo Guloso Randomizado

```
Input: Número máximo do grau do nó
Output: Melhor solução
foreach Aresta ∈Grafo do
    Seta a pontuação para cada Aresta do Grafo
    listEdgesAux ← Aresta
end
for j ← 1 to 2500 do
    foreach ArestaListEdge ∈ listEdgesAux do
        contador ← 0
        foreach ArestaListFinal ∈ listEdgesFinal do
            if ArestaListEdge != ArestaListFinal then
                contador ← +1
        end
        if contador == tamanho de listEdgeFinal then
            arestaListEdgesFinal ← Aresta
        Ordena o array listEdgesFinal
        foreach no ∈ Grafo do
            AGMRG ← no
            noPai ← no
        end
        custoTotalArvore ← 0
        foreach Aresta ∈ listEdgesFinal do
            if grau da Aresta < número máximo do grau do nó then
                if não é ciclo then
                    AGMRG ← Aresta
                    custoTotalArvore ← + peso da Aresta
        end
        custo ← 0
        foreach Aresta ∈ AGRMG do
            custo ← custo + peso da Aresta
        end
        if custo < menorCusto then
            AGRMGBest ← AGMRG
            menorCusto ← custo
    end
    Imprime melhor solução
end
```

3.3 Algoritmo guloso randomizado reativo

Por definição, o algoritmo guloso randomizado reativo ordena as arestas em uma ordem fixa, logo, para gerar novas soluções há necessidade de gerar uma randomização das soluções viáveis. A randomização é gerada a partir de um fator(score) gerado em um range[1, 5], multiplicado pelo peso da aresta, adicionado um número gerado aleatoriamente, multiplicado pelo peso do nó. A cada iteração, o algoritmo pode incrementar ou decrementar nesse fator de penalidade, verificando se a iteração anterior foi uma solução melhor ou não; caso tenha sido, ele continua com a mesma lógica, caso não, ele faz o oposto.

Algorithm 3: Algoritmo Guloso Randomizado

```
Input: Número máximo do grau do nó
Output: Melhor solução
foreach Aresta ∈Grafo do
    Seta a pontuação para cada Aresta do Grafo
    listEdgesAux ← Aresta
end
for j ← 1 to 2500 do
    foreach ArestaListEdge ∈ listEdgesAux do
        contador ← 0
        foreach ArestaListFinal ∈ listEdgesFinal do
            if ArestaListEdge != ArestaListFinal then
                contador ← +1
        end
        if contador == tamanho de listEdgeFinal then
            arestaListEdgesFinal ← Aresta
        Ordena o array listEdgesFinal
        foreach no ∈ Grafo do
            AGMRG ← no
            noPai ← no
        end
        custoTotalArvore ← 0
        foreach Aresta ∈ listEdgesFinal do
            if grau da Aresta < número máximo do grau do nó then
                if não é ciclo then
                    AGMRG ← Aresta
                    custoTotalArvore ← + peso da Aresta
        end
        custo ← 0
        foreach Aresta ∈ AGRMG do
            custo ← custo + peso da Aresta
        end
        if custo < menorCusto then
            AGRMGBest ← AGMRG
            menorCusto ← custo
            if acrescentado then
                fatorPenalizador1 ← fatorPenzalidor1 + valor aleatorio
                fatorPenalizador2 ← fatorPenzalidor2 + valor aleatorio
                acrestado ← true
            else
                fatorPenalizador1 ← fatorPenzalidor1 - valor aleatorio
                fatorPenalizador2 ← fatorPenzalidor2 - valor aleatorio
                acrestado ← false
            end
            return
        if acrescentado then
            fatorPenalizador1 ← fatorPenzalidor1 - valor aleatorio
            fatorPenalizador2 ← fatorPenzalidor2 - valor aleatorio
            acrestado ← false
        else
            fatorPenalizador1 ← fatorPenzalidor1 + valor aleatorio
            fatorPenalizador2 ← fatorPenzalidor2 + valor aleatorio
            acrestado ← true
        end
    end
    Imprime melhor solução
end
```

4 Experimentos computacionais

4.1 Descrição das instâncias

Como instâncias para a execução destes algoritmos, o conjunto de instâncias exigiu algumas adaptações em algumas delas. Dado que em algumas o grafo já estava formado, o algoritmo já conseguiu executar sua tarefa com base nos princípios de Kruskal. A outra situação é onde os vértices eram considerados como pontos no espaço R2 e a distância euclidiana deveria ser considerada de cada ponto até todos os demais, sendo uma maneira de estabelecer um grafo e assim rodar o algoritmo.

Instance Name	**#edge**	**#vertex**
pr124	7626	124
pr136	9180	136
pr144	10296	144
d493	121278	493
d657	215496	657
u574	164451	574
rat575	165025	575
d1655	1368685	1655
d1291	832695	1291
d2103	2210253	2103

Tabela 1: Descrição das instâncias

4.2 Ambiente computacional do experimento e conjunto de parâmetros

Foi utilizado um ambiente computacional com as seguintes especificações: Sistema operacional: Windows 10, processador: AMD FX 8300, 4GB de memória ram. Foi utilizado como linguagem de programação C++ e compilado com g++ com um total de 2500 números de iterações e valores de alfa de 1.5 e 1.8.

4.3 Resultados quanto à qualidade e tempo

Qualidade: O algoritmo de kruskal já parte do pressuposto em que há um grafo previamente estabelecido e a partir daí a montagem de uma árvore de custo mínimo se baseia na seleção das arestas de menor custo, desde que, se juntas, não formam ciclos e ainda forme uma componente conexa. Considerando um único vértice como um ponto no espaço(conjunto R2) desprovido de qualquer relacionamento com qualquer outro vértice em específico, o maior desafio é estabelecer uma relação inicial entre todos os vértices presentes, formando um grafo completo. Tendo este se formado, o algoritmo de kruskal pode ser executado, tomando cada aresta deste grafo completo como uma possibilidade de estar na solução final deste algoritmo.

Instância	**Literatura**	**Melhor**	**Guloso**		**Randomizado**		**Reativo**	
			Custo	**Tempo**	**Custo**	**Tempo**	**Custo**	**Tempo**
pr124	50535	111471	**111471**	0,146	111471	0,149	111471	0,150
pr136	88964	18344	114818	0,182	114830	0,191	**18344**	0,183
pr144	49466	177690	**177690**	0,265	177690	0,283	177690	0,282
d493	29272	29272	**29272**	3,83520	29272	3,75238	29272	3,96340
d657	42490	30141	**30141**	9,03290	30141	9,21018	30141	10,21018
u574	36851	36851	**36851**	22,08811	36851	22,62329	36851	23,12113
rat575	6248	6248	**6248**	23,99137	6248	24,12011	6248	24,34062
d1655	56542	56542	**56542**	3198,87842	56542	3245,69825	56542	3344,72615
d1291	4693	46931	**46931**	4382,73835	46931	4464,18205	46931	4480,19014
d2103	76331	76331	**76331**	6822,83855	76331	6934,78513	76331	6974,65874

Tabela 2: Comparação de resultados dos algoritmos

5 Conclusões e trabalhos futuros

O presente trabalho apresenta contribuições para o Problema de Árvore Geradora Mínima com restrição de grau máximo, um problema de difícil solução. Para solucionar o problema em tempo hábil é necessária a utilização heurística, cuja utilidade é ainda mais evidente à medida que se utiliza instâncias maiores. Foi apresentado um algoritmo que se baseia no algoritmo de kruskal, com alterações nos critérios de seleção das arestas, possuindo uma pontuação (score) em cada uma delas que dependem do peso da aresta e do peso do nó que a aresta está incidindo. Além disso, um algoritmo randomizado que gera números aleatórios de heurísticas, a fim de possibilitar diferentes soluções e buscar a solução perfeita. O algoritmo randomizado reativo consegue realizar escolhas de valor de heurísticas a cada iteração com base nas escolhas anteriores, buscando encontrar a solução perfeita. Como trabalho futuro sugere-se melhorar ainda mais o desempenho do algoritmo de inclusão da lista de arestas possíveis para se tornar solução, através de restrição das arestas que ligam nós com pesos muito grandes. É possível que com tal alteração tenha uma melhora no tempo de execução total do algoritmo.

Referências

[1] Degree-constrained spanning tree benchmark instances. https://unsat.github.io/npbench/spanningtree.htmlsec-2. Acessado em 08/11/2021.

[2] Kavita Singh and Shyam Sundar. A heuristic for degree-constrained minimum spanning tree problem. page 357, 2018.