Algoritmo Exato para o problema de Conjunto de Vértices de Retroalimentação

Irene Ginani Costa Pinheiro[1]

[1]Departamento de Informática e Matemática Aplicada
Universidade Federal do Rio Grande do Nore (UFRN)

ireneginani@gmail.com

Abstract. *The work in question aims to explain, show and analyze the problem of the set of feedback vertices, constructing an exact algorithm for this problem showing its pseudocode, using the GRASP metaheuristic.*

Resumo. *O trabalho em questão visa explicar, mostrar e analisar o problema do conjunto de vértices de retroalimentação, construindo um algoritmo aproximativo para esse problema mostrando seu pseudocódigo e utilizando a metaheurística GRASP.*

1. Introdução

Dado um grafo $G = (V, E)$, direcionado ou não, um conjunto de vértices de retroalimentação ou feedback vertex set é um conjunto de vértices de forma que se ele for retirado do grafo, este se torna acíclico. Assim dizer se um grafo possui um conjunto de vértices de retroalimentação de tamanho k é considerado um problema NP-Completo e assim tentar achar o menor conjunto possível de vértices de forma a tornar um grafo acíclico é considerado um problema NP-Difícil, podemos observar a prova dessa afirmação em [Libeskind-Hada 2007].

Outra vertente do problema é atribuindo pesos aos vértices e assim o objetivo não é mais achar a menor quantidade, mas sim a menor combinação de pesos que retirando os vértices correspondentes no grafo ele se torne acíclico. Porém nesse trabalho iremos tratar apenas o primeiro caso.

Dessa forma os algoritmos aproximativos para esse problema são uma alternativa para sua resolução, visto que os algoritmos exatos possuem uma complexidade exponencial, assim o objetivo do trabalho em questão é construir um algoritmo aproximativo para o problema em questão, utilizando os conceitos aprendidos em [PANOS M. PARDALOS 1999], para assim buscar uma complexidade aceitável dado o estado atual do problema.

2. Problematização e Modelagem

O problema estudado em questão, possui diversas aplicações, sendo citado em problemas como design de chips e casos de sequências de genomas [Jiong Guo 2006], porém sua principal aplicação é em resolução de deadlocks. Na área de sistemas operacionais um deadlock refere-se a uma situação de impasse onde dois ou mais processos ficam bloqueados, aguardando a resposta uns dos outros. Assim é de extrema importância que o sistema operacional não entre em colapso, portanto um algoritmo eficiente para o tratamento dos

deadlocks é considerado importante. Assim temos como vértices os processos que estão em execução no sistema, enquanto as arestas são tratadas como um fluxo de informação, mostrando que um processo precisa do outro para terminar sua execução e sempre que houver um ciclo iremos ter um deadlock.

Dessa forma utilizamos o conceito de conjunto de vértices de retroalimentação para acabar com o mínimo de processos possível de forma a não prejudicar o funcionamento do computador. Assim os vértices do conjunto são os processos que devem ser terminados.

3. Algoritmos para Conjunto de Vértices de Retroalimentação

O problema de conjunto de vértices de retroalimentação, ou seja, achar o menor conjunto de vértices que torne um grafo acíclico é considerado NP-Difícil, dessa forma os algoritmos exatos que resolvem o problema possuem complexidade exponencial, mas é possível resolver o problema por meio de uma aproximação que pode ou não coincidir com a resposta ótima.

3.1. Algoritmos Exatos

O algoritmo exato, em grafos não-direcionados. com a melhor complexidade encontra o conjunto de vértices de retroalimentação em $O(1.7347^n)$ partindo da ideia de dado um conjunto independente F de vértices do grafo, construindo uma floresta induzida, verificando cada caso necessário para que essa floresta seja máxima e sem ciclos. Dessa forma o complementar a essa floresta será o conjunto e vértices de retroalimentação. Podemos ver melhor o algoritmo e as provas necessárias para as afirmações feitas em [Fedor V. Fomin 2007]. Enquanto para grafos direcionados temos essa complexidade em $O(1.9977^n)$, porém no trabalho em questão iremos avaliar apenas grafos não-direcionados.

3.2. Algoritmos Aproximativos

Os algoritmos aproximativos surgem como uma solução mais viável para os problemas NP-Completos de forma que enquanto o algoritmo exato demoraria anos para que uma solução viável surgisse, o algoritmo aproximativo resultará em uma solução mais rápida e ainda assim aceitável. Dessa forma para os algoritmos aproximativos podemos utilizar várias técnicas dentre elas as metaheurísticas. Para o trabalho em questão escolhi utlizar a metahurística GRASP, visto que em alguns artigos ela é bem utilizada para o problema como em [PANOS M. PARDALOS 1999]

3.3. Algoritmo Proposto

Como dito anteriormente o algoritmo proposto segue a metaheurística GRASP, utilizando então, sendo assim é necessário definir 4 algoritmos que fazem parte do método: um algoritmo guloso, um aleatório, um adaptativo e por fim a busca local que irá tentar melhorar a solução. Dessa forma iniciemos pelo algorito guloso. A função que mede o "potencial"de cada vértice para verificar se ele é o melhor no momento foi definida como sendo a quantidade de vizinhos de um vértice v, ou seja o grau desse vértice, visto que quanto maior seu grau será mais provável que este faça parte de um ciclo, e assim precise ser retirado. Assim o algoritmo guloso irá montando uma lista de elite dos vértices presentes no grafo, de forma que o vértice v entrará nessa lista se:

$$Graus(v) >= Menor\_grau + \alpha * (Maior\_grau - Menor\_grau)$$

Onde o alfa é definido de forma que se ele é $\alpha = 0$ o algoritmo é totalmente guloso e se $\alpha = 1$, o algoritmo é totalmente aleatório.

Após definir a primeira função iremos definir a função aleatória que dada a lista de elite construída anteriormente o algoritmo deve escolher um vértice dessa lista e colocálo na solução corrente ate o momento.

Por fim, a função adaptativa irá percorrer todo o grafo e verificar os vizinhos do vértice e questão e para eles irá diminuir seu grau em 1 unidade.Dessa forma, o algoritmo repitirá esses três passos enquanto não houver uma solução viável ou seja, enquanto o grafo ainda possuir ciclos.

Após encontrar uma solução viável, o algoritmo irá armazená-la em uma lista de soluções, e então efetuará a busca local dentro dessa lista de soluções para encontrar a melhor solução que lá está armazenada. Após esse procedimento, a melhor solução é retornada ao usuário como sendo o conjunto de vértices de retroalimentação.

3.3.1. Pseudocódigo

Segue abaixo, de forma mais detalhada, a implementação do algoritmo:

Algorithm 1 Algoritmo Proposto

```
Contador
Grafo G
function ALGORITMO PROPOSTO(G)
    Lista_Elite
    Lista_Solução
    Solucoes
    while Contador ¡ 250 do
        while Existe_Ciclo(G) do
            Função_Gulosa(GrafoG, Lista_Elite)
            Função_Aleatória(Lista_Elite, Verticev)
            Função_Adaptativa(GrafoG, Verticev    ▷ Onde v é o vértice escolhido na função aleatória
            Lista_Solucao = Lista_Solucao ∪ v
            Retira de G os vértices em Lista_Solução
        end while
      Soluções = Lista_Solucao ∪ Soluções
        Busca_Local(Solucoes, Melhor_Solução)
        if Melhor_Solução <= Solução then
            Solucao = Melhor_Solução
        end if
        Contador = Contador + 1
    end while
    Imprima Solução
```

Onde as partes de cada algoritmo chamados aqui já foram discutidas anterior-

mente.

3.3.2. Complexidade e Corretude

como temos um algoritmo aproximativo sua corretude não poderá ser provada, a solução não ser exata. Porém é asssegurado que a solução que o algoritmo que imprimirá uma solução viável visto que o algoritmo é exacutado enquanto houver um ciclo no grafo, quando este não existe mais, teremos a solução. Em relação a complexidade do algoritmo temos que cada parte do seu funcionamento teve que percorrer no mínimo o grafo inteiro resultado em $O(n^2)$, individualmente. Porém cada funcionalidade é executada enquanto houver ciclos e por fim, o algoritmo utiliza a métrica curta executando 100 vezes, de modo que sua complexiade final irá depender de quantas vezes o algoritmo de detecção de ciclo é executado. De forma que ao decorrer do algoritmo ele irá retirando do grafo os vértices da solução e verificando se ainda há ciclo.

3.3.3. Testes

Para efetuar os testes foram utilizados dois tipos de gerador instâncias, um onde podemos colocar o grafo que desejamos manuealmente em um arqquivo e outro onde o grafo é gerado aleatoriamente. As instâncias encontradas de benchmarck estão disponíveis em [ckomus], porém somente algumas foram utilizadas de modo que em vértices a partir de 150 vértices o algoritmo executava mais de 2 horas, onde foi interrompido. Dessa forma os testes foram rodados em um computador Dell Vostro 5470, com processador i7 com quatro núcleos e memória RAM de 8GB.

Dessa forma, para exibir o funcionamento do algoritmo, utilizamos o grafo da figura 1.

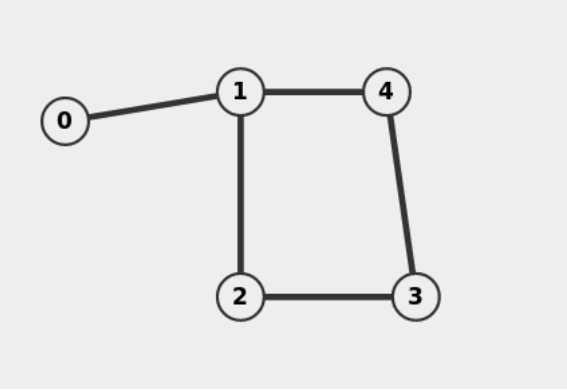

Figura 1. Exemplo 1 - grafo com 5 vértices

Dessa forma temos a seguinte saída para a entrada acima:

```
irene@irene-Vostro-5470:~/Documentos/2017.2/PAA/Trabalho 2/Testes$ ./teste teste
2.txt
8323722262 ns
tamanho do conjunto: 1
Vertices na solução:
1
```

Figura 2. Exemplo 1 - grafo com 5 vértices

Assim, o conjunto de vértices de retroalimentação tem apenas um vértice, e sua retirada resulta na figura 3

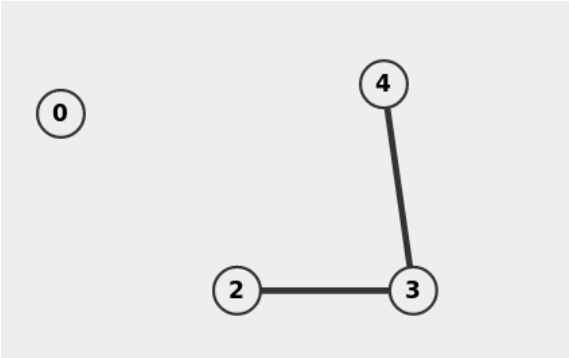

Figura 3. Exemplo 1 - grafo com 5 vértices

Conforme aumentamos as instâncias dos grafos, começamos gerá-los aleatoriamente de forma que não seria mais possível acompanhar os passos graficamente, como feito anteriormente, assim agora, serão exibidos os resultados para alguns grafos gerados de forma aleatória, além de utilizar as instâncias encontradas em [ckomus], porém também não foi possível acompanhar seu resultado graficamente devido ao tamanho das instâncias, dessa forma abaixo temos algumas capturas de tela com resultados de algumas execuções para evidenciar o funcionamento do algoritmo.

```
irene@irene-Vostro-5470:~/Documentos/2017.2/PAA/Trabalho 2/Testes$ ./teste t
2.txt
72313061530 ns
tamanho do conjunto: 2
Vertices na solução:
1
0
irene@irene-Vostro-5470:~/Documentos/2017.2/PAA/Trabalho 2/Testes$
```

Figura 4. Exemplo 2 - grafo com 10 vértices

Figura 5. Exemplo 3 - grafo com 20 vértices

Tabela 1. Execuções do Algoritmo proposto

Instância	Alfa	Quantidade de Vértices no FVS	Média das execuções
graph1_10	0.5	2	2007833002 ns
graph1_30	0.5	3	29986247246 ns
graph1_50	0.5	3	158440728885 ns
graph1_100	0.5	10	173700373015

4. Resultados

Com os testes sendo executados podemos observar que o algoritmo proposto é mais rápido do que o exato, porém ainda é lento para algumas instâncias demorando mais do que o tempo máximo estimado que seriam 3 horas. Dessa forma podemos realizar a comparação do algoritmo proposto e suas instâncias além do algoritmo aproximado com o exato. Incialmente observemos o comportamento do algoritmo proposto.

Na tabela 1 temos que a primeira coluna simboliza a instância usada com a quantidade de vértices, a segunda o valor do alfa utilizado, a terceira a quantidade no conjunto de vértices de retroalimentação e por fim a média de 30 execuções em relação ao tempo que levou cada execução.

Assim, observemos a comparação do algoritmo proposto com o algoritmo exato implementado anteriormente.

Tabela 2. Comparações entre algoritmos

	Vértices	Tempo(ns)
Algoritmo Proposto	5	8316383766
Algoritmo B&B	5	139842823299171
Algoritmo Proposto	10	2007833002
Algoritmo B&B	10	1140475865992439
Algoritmo Proposto	20	446118703915
Algoritmo B&B	20	140010826368935
Algoritmo Proposto	40	158440728885
Algoritmo B&B	40	140311569576172

Na tabela 2 temos qual algoritmo causou aquele teste na primeira coluna, a quantidade de vértices na segunda coluna e os tempos do respectivo algoritmo descrito anteriormente e o entregue na unidade passada. Por fim, na terceira coluna em nanossegundos temos o tempo de execução de cada um, e assim podemos constatar que o algoritmo proposto é mais rápido, porém é aproximado, podendo não imprimir a resposta ótima.

5. Considerações Finais

Assim, observamos que o algoritmo proposto possui conceitos que aproximam da solução ótima, mas utilizando algoritmos aproximativos nem sempre essa resposta é exata, de

forma que o algoritmo é mais rápido, porém nem sempre é correto. Assim é uma característica que precisa ser analisada em situações reais, se a aproximação é suficiente para a solução do problema.

6. Compilação

Para compilar o algoritmo é necessário um compilador de c++, o g++ 5.4 e digitar o comando g++ gerar_grafo_arquivo.cpp -std=c++11 -o teste, depois pode-se rodar o programa digitando o comando: ./teste nome.txt onde nome.txt é o arquivo de teste o qual deseja-se utilizar. Para utilizar a geração aleatória é necessário digitar: g++ gerar_grafo_aleatorio.cpp -std=c++11 -o teste e depois ./teste. Lembrando que é necessário está no diretório onde esses programas estão com o código fonte.

Referências

ckomus. Pace-fvs.

Fedor V. Fomin, Serge Gaspers, A. V. P. I. R. (2007). On the minimum feedback vertex set problem: Exact and enumeration algorithms.

Jiong Guo, Jens Gramm, F. H. R. N. S. W. (2006). Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization.

Libeskind-Hada, R. (2007). An example of a short np-completeness proof.

PANOS M. PARDALOS, TIANBING QIAN, M. G. R. (1999). A greedy randomized adaptive search procedure for the feedback vertex set problem.