# 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, complexity and correctness, using the branch and bound technique.*

***Resumo.*** *O trabalho em questão visa explicar, mostrar e analisar o problema do conjunto de vértices de retroalimentação, construindo um algoritmo exato para esse problema mostrando seu pseudocódigo, complexidade e corretude, utilizando a técnica de branch and bound.*

## 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 exatos para esse problema possuem de complexidade exponencial, assim o objetivo do trabalho em questão é construir um algoritmo exato para o problema em questão, utilizando os conceitos aprendidos em [Fedor V. Fomin 2007], 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. Algoritmo Proposto

O algoritmo proposto pelo trabalho em questão utiliza a mesma ideia encontrada em [Fedor V. Fomin 2007], tentando induzir uma floresta máxima no grafo de forma que seu complemento sera o conjunto de vértices de retroalimentação. A diferença implementada é a técnica de desenvolvimento de algoritmos avançados chamada branch and bound. Além disso, utilizaremos o conceito de vizinho genérico, onde dada um conjunto independente de vértices e um vértice aleatório do grafo,uma contração é quando contraímos todas as arestas do conjunto no vértice, assim se v é vizinho de u após a contração ele é considerado um vizinho genérico. Assim o limite inferior do algoritmo será o conjunto F, de vértices independentes, e o limite superior será a quantidade de vértices que o grafo contém. Dessa forma, inicialmente iremos calcular o conjunto independente utilizando um algoritmo aproximativo. A partir do conjunto encontrado iremos tentar induzir uma floresta máxima observando os seguintes casos:

- Se o Conjunto Independente encontrado é igual a quantidade de vértices no grafo;
- Se todos os vértices do grafo possuem grau igual ou menor que 1;

Após essas verificações o algoritmmo segue sua execução para todos os vértices presentes no conjunto independente de forma a avaliar os seguintes casos:

- Se para um vértice t em F todos seus vizinhos estão em então construa um novo conjunto independente I a partir do grafo construído na proposição 3 em [Fedor V. Fomin 2007] e a árvore será a soma dos vértices de F + I, e o conjunto de vértices de retroalimentação será o restante dos vértices;
- Se o vértice v em N(t) onde t está em F tem 1 vizinho genérico, então adicione ele em F e chame novamente o algoritmo;
- Se o vértice v em N(t) onde t está em F tem 4 ou mais vizihos genéricos efetue duas chamadas para o algoritmo, uma para o F com o vértice v e a outra removendo o vértice v do grafo;
- Se o vértice v em N(t) onde t está tem exatamente dois vizinhos genéricos então execute duas novas chamadas recursivas do algoritmo, uma adicionando o vértice v em F e outra removendo v do grafo e adicionando os dois vizinhos de v em F;
- Se todos os vértices vizinhos de t possuem exatamente 3 vizinhos genéricos então escolha um vértice v em N(t) e efetue 3 chamadas recursivas de modo que a primeira será adicionando v em F; a segunda será removendo v do grafo G e adicionando um vizinho de v $w_1$ e F e por fim removendo v e $w_1$ de G e acrescentando os outros dois vizinhos de v, $w_2$ e $w_3$ em F.
  Se alguma adição de vértices em F causa um ciclo, basta ignorar esse passo de adição.
  Dessa forma iremos sempre comparar as chamadas recursivas com o limite inferior e o superior definido para poder eliminar execuções que não estejam dentro do esperado. Além disso iremos procurar o resultado máximo das chamadas recursivas e por fim efetuar uma operação de complementar com o grafo original para assim achar o conjunto de vértices de retroalimentação.

### 3.2.1. Pseudocódigo

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

**Algorithm 1** Algoritmo Proposto

CI = Conjunto Independente máximo do grafo G
lb = tamanho de CI
up = tamanho de vértices do grafo
F = CI
**function** ALGORITMO PROPOSTO($M_{nxn}, lb, up, FVS, CI, F$)
adicione F a uma lista de conjuntos
**if** Máx F em lista de conjuntos $>$ up **then return** 0;
**end if**
**if** Min F em lista de conjuntos $<$ lb **then return** 0;
**end if**
**if** Máx F em lista de conjuntos ¡ up **then**
up = tamanho máximo de lista de conjuntos
**end if**
**if** Min F em lista de conjuntos ¿ lb **then**
lb = tamanho mínimo de lista de conjuntos
**end if**
**if** $CI_{tamanho}$ = $M_{tamanho}$ **then return** FVS é vazio
**end if**
**if** Maior grau de M é 1 **then return** FVS é vazio
**end if**
**for** $i = 0$ to $CI_{tamanho}$ **do**
**if** $V - CI$ = $N(i)$ **then**
Construa um grafo H como já citado
faça seu conjunto independente I **return** $CI + I$
**end if**
**for** todos os v vizinhos de i **do**
Calcule os vizinhos genéricos de v
**if** V tem um vizinho genérico **then** adicione v a F
AlgoritmoProposto($M_{nxn}, lb, up, FVS, CI, F$)
**end if**
**if** v tem quatro ou mais vizinhos genéricos **then**
AlgoritmoProposto($M_{nxn} - v, lb, up, FVS, CI, F$)
AlgoritmoProposto($M_{nxn}, lb, up, FVS, CI, F + v$)
**end if**
**if** v tem exatamente dois vizinhos genéricos **then**
AlgoritmoProposto($M_{nxn}, lb, up, FVS, CI, F + v$)
**if** F+$w_1, w_2$ não possui ciclo **then**
AlgoritmoProposto($M_{nxn}, lb, up, FVS, CI, F + w_1, w_2$) ▷ Onde $w_1$ e $w_2$ são os vizinhos de v
**end if**
**end if**
**if** todos os vizinhos de v tem 3 vizinhos genéricos **then**
AlgoritmoProposto($M_{nxn}, lb, up, FVS, CI, F + v$)
AlgoritmoProposto($M_{nxn} - v, lb, up, FVS, CI, F + w_1$)
**if** F+$w_1, w_2$ não possui ciclo **then**
AlgoritmoProposto($M_{nxn} - v, w_1, lb, up, FVS, CI, F + w_3, w_2$) ▷ Onde $w_1$, $w_2$ e $w_3$ são os vizinhos de v
**end if**
**end if**

**end for**

### 3.2.2. Complexidade

Em cada caso iremos abrir novas chamadas recursivas da passando a mesma entrada, porém aumentando o conjunto que gera a árvore máxima. Assim como cada caso exclui os demais, temos que no pior dos casos o algoritmo efetua três novas chamadas recursivas para todos os vértices disponíveis no conjunto que gera a árvore. Temos então que no pior dos casos essa árvore terá n-1 vértices, no caso do conjunto de vértices de retroalimentação mínimo ser igual a um, já que o caso de quando não há vertices é identificado ainda antes da recurssão. Assim a complexidade do algoritmo desenvolvido será $O(3^n)$.

### 3.2.3. Corretude

A prova da corretude de cada caso, utilizando os conceitos e operações acima podemos encontrar em [Fedor V. Fomin 2007], a única diferença são as chamadas recursivas, de modo que no algoritmo proposto haverão mais pelas combinações. Assim ao invés de comparar as chamadas em cada subcaso iremos apenas invocá-las para assim gerar todas as combinações possiveis e poder efetuar a técninca de branch and bound. Portanto como o algoritmo gera todas essas combinações e as compara extraíndo a menor, podemos afirmar que ele é correto.

### 3.2.4. Testes

Para efetuar os testes foram utilizados dois tipos de gerador instâncias, um onde podemos colocar o grafo que desejamos manuealmente e outro onde o grafo é gerado aleatoriamente. As instâncias encontradas de benchmarck estão disponíveis em [ckomus ], porém não foram usadas devido ao tamanho, já que o algoritmo exato implementado suporta até 50 vértices. 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.

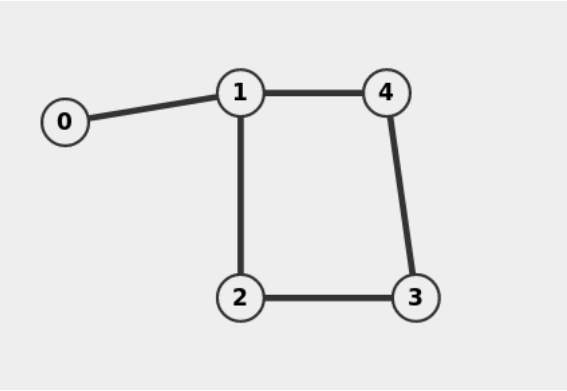


**Figura 1. Exemplo 1 - grafo com 5 vértices**

Dessa forma temos a seguinte saída para a entrada acima:

```
1
563707 msirene@irene-Vostro-5470:~/Documentos/2017.2/PAA/Trabalho 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

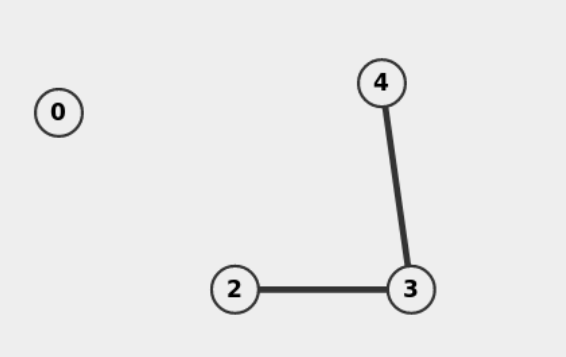


**Figura 3. Exemplo 1 - grafo com 5 vértices**

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 10, 20 e 40 vértices, que foi até onde foi possível obter uma saída.

```
35
140503878968913 msirene@irene-Vostro-5470:~/Documentos/2017.2/PAA/Trabalho
```

**Figura 4. Exemplo 2 - grafo com 10 vértices**

Por fim temos um exemplo de um grafo com 200 vértices, o qual não chegou ao fim, mas no momento da interrupção havia explorado alguns ramos da árvore de branch and bound.

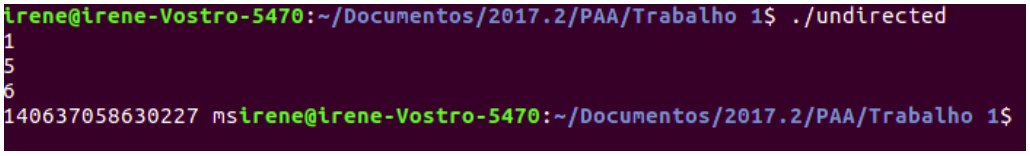


**Figura 5. Exemplo 3 - grafo com 20 vértices**


**Figura 6. Exemplo 4 - grafo com 40 vértices**

## 4. Resultados

Com os testes sendo executados podemos observar que o algoritmo proposto é mais lento do que o que foi inspirado, porém utiliza a técnica de branch and bound, o que explora mais possibilidades além do que o mencionado. Além disso, para fins comparativos foi implementado outro algoritmo para achar o conjunto de vértices de retroalimentação, de forma que o algoritmo parte de uma heurística qualquer, também utilizando branch and bound, chegando no resultado. Dessa forma segue, na tabela 1, a comparação dos dois algoritmos.

**Tabela 1. Comparações entre algoritmos**

| | Vértices | Tempo(ms) |
|---|---|---|
| Algoritmo Proposto | 5 | 563707 |
| Algoritmo B&B | 5 | 139842823299171 |
| Algoritmo Proposto | 10 | 140503878968913 |
| Algoritmo B&B | 10 | 1140475865992439 |
| Algoritmo Proposto | 20 | 140637058630227 |
| Algoritmo B&B | 20 | 140010826368935 |
| Algoritmo Proposto | 40 | 140069400646747 |
| Algoritmo B&B | 40 | 140311569576172 |

Na tabela 1 temos qual algoritmo causou aquele teste na primeira coluna, a quan-


**Figura 7. Exemplo 5 - grafo com 200 vértices**

tidade de vértices na segunda coluna e os tempos dos respectivos algoritmos descritos anteriormente na terceira coluna em microsegundos, e assim podemos constatar que o algoritmo proposto é mais rápido, porém o tempo vai depender da heurística utilizada.

## 5. Considerações Finais

Assim, observamos que o algoritmo proposto possui conceitos que aproximam da solução ótima, mas utilizando a técnica de branch and bound se torna um pouco mais lento em termos de sua complexidade. Assim para trabalhos futuros será feita uma implementação, utilizando meta-heurísticas, para o problema de conjunto de vértices de retroalimentação.

## 6. Compilação

Para compilar o algoritmo é necessário um compilador de c++, o g++ 5.4 e digitar o comando g++ undirected.cpp -std=c++11 -o undirected, depois pode-se rodar o programa digitando o comando: ./undirected

Para testar cada caso citado é necessário abrir o algoritmo em algum editor de texto de comentar o restante dos testes e descomentar o desejado, fazendo uso também pelo método de geração do algoritmo.

## 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.