Relatório - Trabalho 2
Otimização

Douglas Affonso Clementino

2021

INTRODUÇÃO

Toda manhã, Sr. Gump sai para caminhas por sua cidade. Ele não gosta de passar mais de uma vez por cada lugar. Mas ele gosta de demorar em seu passeio. Dado um conjunto de lugares (vértices), as ligações entre estes lugares (arestas) e os custos (tempo) de percorrer cada uma destas ligações, devemos encontrar um trajeto para o Sr. Gump que seja o mais longo possível, sem repetir lugares.

Este problema pode ser categorizado como, dado um grafo ponderado não dirigido $G(V, E)$, tal que $E = \{1, ..., n\}$, com n sendo o total vértices, achar o *Ciclo Simples* (FEOFILOFF, 2017) C que inicia no vértice 1 e cuja soma das arestas percorridas seja maximal.

Para resolver este problema, utilizaremos da técnica de *Branch and Bound* (KREHER; STINSON, 1999) em busca de propor uma forma otimizada encontrar tal ciclo *Ciclo Simples Maximal* C.

1 MODELAGEM

Dado o contexto do problema, e definido o grafo $G(V, E)$ efetuaresmo proposta incremental do algoritmo para encontrar o *Ciclo Simples Maximal* iniciado em vértice 1.

Será de definida uma Matriz de Adjacência (MATRIZ..., 2021) $A = [a_{ij}]$, com $i, j \in \{1..n\}$, tal que, se aresta $i, j \in E$ então $a_{ij} = peso(\{i, j\})$, caso contrário $a_{ij} = 0$.

Primeiramente será proposta uma solução de busca exaustiva utilizando do algoritmo de de busca em profundidade que já implementa *Bound por Inviabilidade* nas arestas. Após isso, será proposta uma segunda implementação com *Bound por Estimativa em Subárvore*, estimando valores de ramos a partir relaxamento na soma das arestas de vértices candidatos remanescentes, executando a poda caso estimativa contenha valor menor solução parcial já encontrada.

1.1 BOUND POR INVIABILIDADE

Nesta primeira proposta, a ideia consiste em explorar o grafo $G(E, V)$ que descreve os possíveis caminhos a seres percorridos utilizando método de busca em profundidade, utilizando a matriz de adjacência A e a lista ordenada de vértices não visitados L. Deste modo, a busca é feita da seguinte forma:

Inicializa-se lista $maiorCicloSimples = []$ $tamnhomaiorCicloSimples = 0$; define-se loop para percorrer todos os $w \in L$, caso exista aresta entre vértice 1 e w, remove-se esta aresta de A e inicia-se o algoritmo recursivo de busca em profundidade uma busca em profundidade (**Algoritmo 2**), indicando Matriz A modificada, vértice destino w, peso de 1, w e caminho atual, contendo apenas vértice 1. Este procedimento é indicado em **Algoritmo 1**.

Algoritmo 1: encontraMaiorCaminhada()

Result: Ciclo Maximal em $maiorCicloSimples$ e seu tamanho em $tamanhoCicloMax$

```
1  v = 1;
2  maiorCicloSimples = [ ];
3  tamanhoCicloMax = 0;
4  L = {1..n};
5  foreach w em L do
6  |   peso = A[a_vw];
7  |   if peso > 0 then
8  |   |   removeAresta(A, v, w);
9  |   |   buscaProdundidade(A, w, peso, [1]);
10 |   |   adicionaAresta(A, v, w);
11 |   end
12 end
13 return maiorCicloSimples, tamanhoCicloMax;
```

Quanto a busca em profundidade (**Algoritmo 2**), primeiramente é adicionando vértice v de entrada a caminho, caso $v = 1$, um novo Ciclo Simples (Solução Viável) foi encontrado, caso este novo ciclo seja maior que o encontrado anteriormente armazená-lo em $maiorCicloSimples$ e o seu tamanho em $tamanhoCicloMax$.

Caso ciclo não tenha sido encontrado, remover v de lista de vértices válidos L, indicando que este não deve ser adicionado em subárvores desta chamada recursiva. Iniciar uma nova busca por candidatos em L que sejam vizinhos de v, iniciando uma nova chamada

recursiva com *caminho* e *tamanho* atualizados.

Algoritmo 2: buscaProdundidade(A, v, *total*, *caminho*)

```
1  caminho.push(v);
2  if v > 1 then
3  |  if total > tamanhoCicloMax then
4  |  |  maiorCicloSimples = ciclo;
5  |  |  tamanhoCicloMax = tamaho;
6  |  end
7  |  caminho.pop();
8  |  return;
9  end
10 L = L \{v};
11 foreach w em L do
12 |  peso := A[a_vw];
13 |  if peso > 0 then
14 |  |  buscaProdundidade(A, w, total + peso, caminho);
15 |  end
16 end
17 L = insereOrdenado(L ,v);
```

Ao final, de todas as chamadas, $maiorCicloSimples$ e $tamanhoCicloMax$ conterão, respectivamente, o caminho do maior Maior Ciclo Simples e o tamanho deste Ciclo.

1.1.1 Por que Bound por Inviabilidade?

Tendo como exemplo o grafo $G(V, E)$ em **Figura 1**:

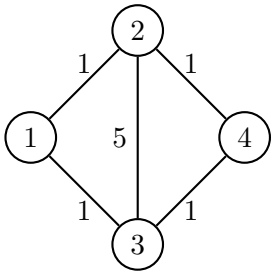
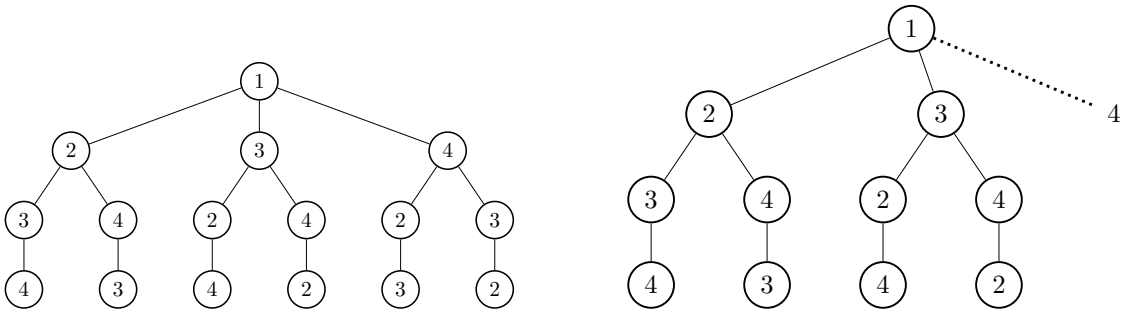


Figura 1 – Exemplo de entrada para problema.

Seria possível obter o maior caminho através de algoritmo de *backtrack* que retorna-se todas as permutações P de tamanhos 3 à $|V|$, verificando se $P \frown \{1\}$ é uma solução viável e escolhendo a maior. Para exemplo acima, tal método geraria uma árvore de busca como em **Figura 2a**.

Enquanto o algoritmo proposto em 1.1 considera não apenas os vértices disponíveis mas também as arestas (linhas 6 e 7 de **Algoritmo 1** e 12 e 13 de **Algoritmo 2**) no processo de busca, efetuando podas devido a inviabilidade da transição, reduzindo a árvore de busca, como pode ser observado em **Figura 2b**, onde é executada a poda na transição (aresta) $\{1, 4\}$ pela sua inexistência.



(a) Árvore Busca *backtrack* (b) Árvore Busca Bound por Inviabilidade

Figura 2 – Comparação entre árvores de busca. Vérices de árvore estão contidos em círculos, linha pontilhada indica poda.

1.2 BOUND POR ESTIMATIVA EM SUBÁRVORE

Nesta segunda proposta, busca-se definir uma limitação (*bound*) sobre o valor das arestas, estipulando se é possível ou não atingir uma melhor resposta para cada um das subárvores percorridas, efetuando poda caso resposta seja negativa. Desta forma, tendo X' como uma parcial obtida até determinado v o vértice atual de ramo e w o vizinho candidato a expanção, tendo $L = (E \backslash X') \cup \{1\}$ e $peso = custo(X')$, estima-se que, no melhor dos casos seria possível definir um ciclo composto por arestas de X' mais à aresta $\{v, w\}$ e às arestas de vértices remanescente em $L \backslash \{1\}$ e como destino vértices em L (que inclui o vértice de origem 1) de forma que este último conjunto de arestas seja máxima, assim, sendo:

$$Estimativa = custo(X') + custo(\{v, w\}) + \sum_{p \in L \backslash 1} max\{\{p, q\} \ \in E | q \in L\}$$

Por possibilitar a repetição de vértices de destino este método pode não atender aos requisitos da formação de um ciclo e menos ainda a de um ciclo simples, porém certamente servida como *upperbound* do custo dos possíveis ciclos simples obtidos a partir da parcial X'. Desta forma, caso esta estimativa já não ultrapasse o valor máximo atual ($tamanhoCicloMax$), não é necessário explorar a sua subárvore.

Abaixo, em **Algoritmo 3** esta exemplificado o pseudocódigo da implementação da função de bound ($boundSomaArestas()$).

Algoritmo 3: boundSomaArestas(A, L, $peso$)

```
Result: Soma de arestas
soma = peso;
listaOrigem = L\{1}
foreach v em listaOrigem do
    maximo = 0;
    foreach w em L do
        peso = A[a_vw];
        if peso > maximo then
            maximo = peso;
        end
    end
    soma = soma + maximo;
end
return soma;
```

Assim, para fazer uso deste bound, modifica-se os algoritmos propostos em 1.1 para inclusão deste *bound* como exemplificado em **Algoritmo 4** e **Algoritmo 5**.

Algoritmo 4: encontraMaiorCaminhadaBB()

Result: Ciclo Maximal em *maiorCicloSimples* e seu tamanho em *tamanhoCicloMax*

```
1  v = 1;
2  maiorCicloSimples = [ ];
3  tamanhoCicloMax = 0;
4  L = {1..n};
5  bound = boundSomaArestas(A, L, 0);
6  foreach w em L do
7  |  peso_aresta = A[a_vw];
8  |  if peso_aresta > 0 then
9  |  |  if (bound + peso_aresta) > tamanhoCicloMax then
10 |  |  |  removeAresta(A, v, w);
11 |  |  |  buscaProdundidadeBB(A, w, peso, [1]);
12 |  |  |  adicionaAresta(A, v, w);
13 |  |  end
14 |  end
15 end
16 return maiorCicloSimples, tamanhoCicloMax;
```

Algoritmo 5: buscaProdundidadeBB(*A*, *v*, *total*, *caminho*)

```
1  caminho.push(v);
2  if v == 1 then
3  |  if total > tamanhoCicloMax then
4  |  |  maiorCicloSimples = ciclo;
5  |  |  tamanhoCicloMax = tamaho;
6  |  end
7  |  caminho.pop();
8  |  return;
9  end
10 L = L \{v};
11 bound = boundSomaArestas(A, L, total);
12 foreach w em L do
13 |  peso_aresta := A[a_vw];
14 |  if peso_aresta > 0 then
15 |  |  if (bound + peso_aresta) > tamanhoCicloMax then
16 |  |  |  buscaProdundidadeBB(A, w, aux, caminho);
17 |  |  end
18 |  end
19 end
20 L = insereOrdenado(L ,v);
```

1.2.1 Exemplo de Poda

Continuando com grafo exemplo **Figura1**, somente com a aplicação de bound por inviabilidade foi possível obter a árvore **Figura3**.

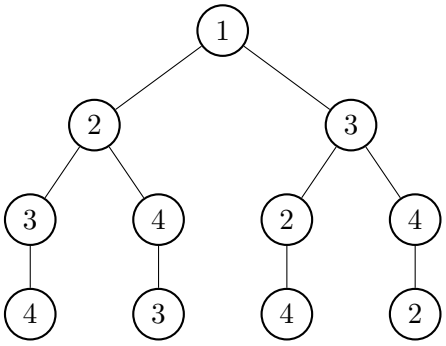


Figura 3 – Árvore resultado de algoritmo $encontraMaiorCaminhada()$ 1.

Aplicando o novo bound durante o processo de criação da árvore, o cenário de exploração em exemplo **Figura4**, onde sobre a esquerda das arestas é indicado o valor de maior ciclo encontrado, ao lado direito e sublinhado, a estimativa calculada para aquele ramo, linhas tracejadas indicam arestas de retorno (identificação de ciclo) e linhas pontilhadas indicam cortes de ramos.

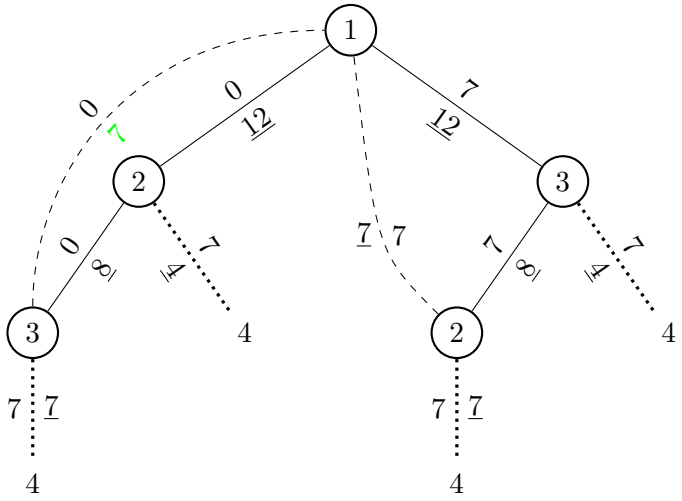


Figura 4 – Exploração efetuada por algoritmo $encontraMaiorCaminhadaBB()$ 4.

Incialmente, calcula-se o valor de bound para vertice $\{1\}$:

$$bound = \sum_{v \in [2,3,4]} max\{\{v, w\} \ \in E | w \in [1,2,3,4]\} = 11$$

Após isso, calcula-se a estimativa de expanção para ramo de aresta $\{1, 2\}$, calculado como:

$$Estimativa = custo\{1\} + custo\{1,2\} + bound$$

Utilizando como base o custo das arestas de **Figura1**, temos:

$$Estimativa = 0 + 1 + 11 = 12$$

Como custo de maior ciclo até momento é 0, explora-se o ramo. Remove-se aresta {1, 2} de A e inicia-se a esploração de vértice 2. Primeiro passo calcula-se o seu bound.

$$bound = \sum_{v \in [3,4]} max\{\{v, w\} \ \in E | w \in [1, 3, 4]\} = 2$$

E estima-se valores para próximo vértice a ser explorado, que é o 3. Dessa forma, calcula-se estimativa para aresta {2, 3}:

$$Estimativa = custo\{1, 2\} + custo\{2, 3\} + bound$$

$$Estimativa = 1 + 5 + 2 = 8$$

Novamente, como custo máximo atual é 0, explora-se vértice 3, primeiramente calculando seu valore de *bound*:

$$bound = \sum_{v \in [4]} max\{\{v, w\} \ \in E | w \in [1, 4]\}$$

Ao explorar vértice 3, identifica-se a aresta de retorno {3, 1}, e calcula-se a sua estimativa:

$$Estimativa = custo\{1, 2, 3\} + custo\{3, 4\} + bound$$
$$Estimativa = 6 + 1 + 0 = 7$$

Como estimativa é maior que ciclo atual, explora-se aresta {3, 1}, identificando assim um novo ciclo, definindo-se assim o ciclo simples {1, 2, 3, 1} de custo 7 (que é maior que cutos de ciclo atual, 0), armazenando estas informações em $maiorCicloSimples$ e $tamanhoCicloMax$ respectivamente.

Após isso, retorna-se a vertice 3, continuando sua exploração. Próximo passo é explorar a aresta {3, 4}, definindo a $Estimativa$ utilzando o *bound* calculado anteriormente ao entrar em vértice 3:

$$Estimativa = custo\{1, 2, 3\} + custo\{3, 4\} + bound$$
$$Estimativa = 6 + 1 + 0 = 7$$

Nesta rodada, como já encontramos um ciclo de tamanho 7, não é viável explorar tal vértice 4, desta forma poda-se a árvore. Como não existem mais vértices a serem explorados, retorna-se a vértice 2 e verifica-se ramo {2, 4}, com *bound* já calculado em ao entrar em vértice 2:

$$Estimativa = custo\{1, 2\} + custo\{2, 4\} + bound$$

$$Estimativa = 1 + 1 + 2 = 4$$

Como $Estimativa < 7$, não é possível obter melhor resposta de ramo, desta forma ele é descartado. Exploradas todas as possibilidade de vertice 2, retorna-se para vertice 1 onde para ramos a deireita os mesmos processos são repetidos. Importante observar que, apesar de outra extremidade contes o mesmo ciclo em ordem oposta, o ultimo vértice não é explorado, uma vez que a sua estimativa é 7, que não é maior que o comprimento do ciclo atual.

Ao final, obtem-se a resposta de $MaiorCicloSimples = \{1, 2, 3, 1\}$ e 7 sendo o seu custo. Árvore gerada para obter esta resposta pode ser vista em **Figura5**.

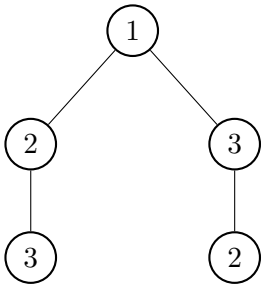


Figura 5 – Árvore resultado de $encontraMaiorCaminhadaBB()$ 4.

2 IMPLEMENTAÇÃO

A implementação do projeto foi efetuada na linguagem *Python 2*, para a execução dos códigos é utilizada a implementação de *Python 2*, *pypy*.

Os códigos descritos em 1.2 refletem a implementação, diferindo na representação das estruturas L, A, $maiorCicloSimples$, $tamanhoCicloMax$ e as funções $removeAresta()$ e $adicionaAresta()$ são gerenciadas através da classe $Problema$ (atributos $verticesValidos$, $matAdj$, $maiorCicloSimples$, $tamanhoCicloMax$ e métodos $removeAresta(v, w)$ e adicionaAresta(v, w, peso_aresta), respectivamente)

2.1 EXEMPLOS DE ENTRADA

Entrada para exemplo utilizado em exemplificação de sessões anteriores 1 pode ser encontrado em *exemplos/entrada/exemplo1_trabalho.txt*, assim como demais entradas utilizadas para desenvolvimento do trabalho.

3 RESULTADOS

A seguir, constam gráficos para os resultados obtidos por versão de Busca em Profundidade Clássica (disponível em *bb_caminhada/classico.py*) e algoritmo com técnica de *Branch Bound* (contido em *bb_caminhada/main.py*).

Nos gráficos de Figuras 6 e 7, são comparadas saídas de quantidade de nós para entradas grafos simples (6) e grafos complexos (7).

O mesmo ocorre para gráficos de figuras 8 e 9, onde os mesmos experimentos são observados, porem explorando-se o tempo de execução dos algoritmos.

Importante pontuar que não foi possível concluir o experimento *teste_14completoRandomico* em versão *Classico*. De forma que, para este experimento, existe apenas resultados para implementação de *Branch Bound*.

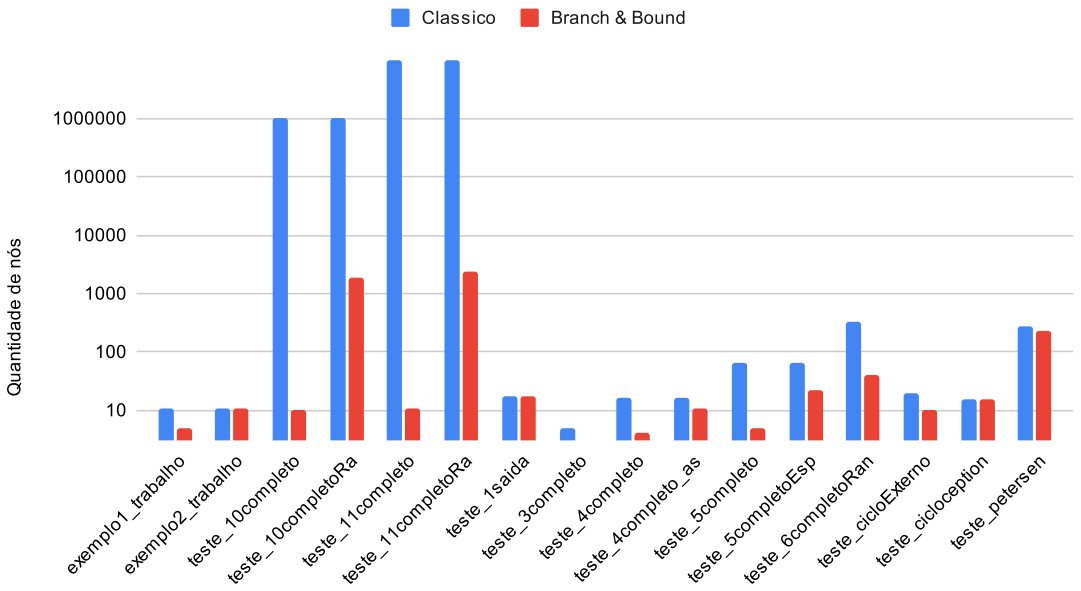


Figura 6 – Quantidade de nós em arvode de exploração para grafos simples (Diretório *./exemplos/entrada/*).

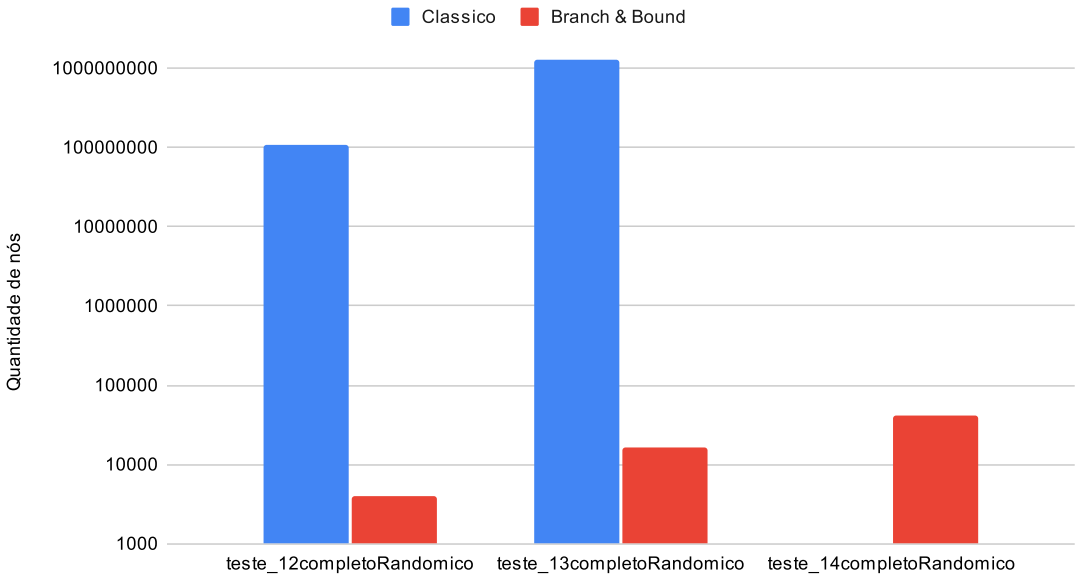


Figura 7 – Quantidade de nós em arvode de exploração para grafos grande (Diretório *./exemplos/entrada/grande/*).

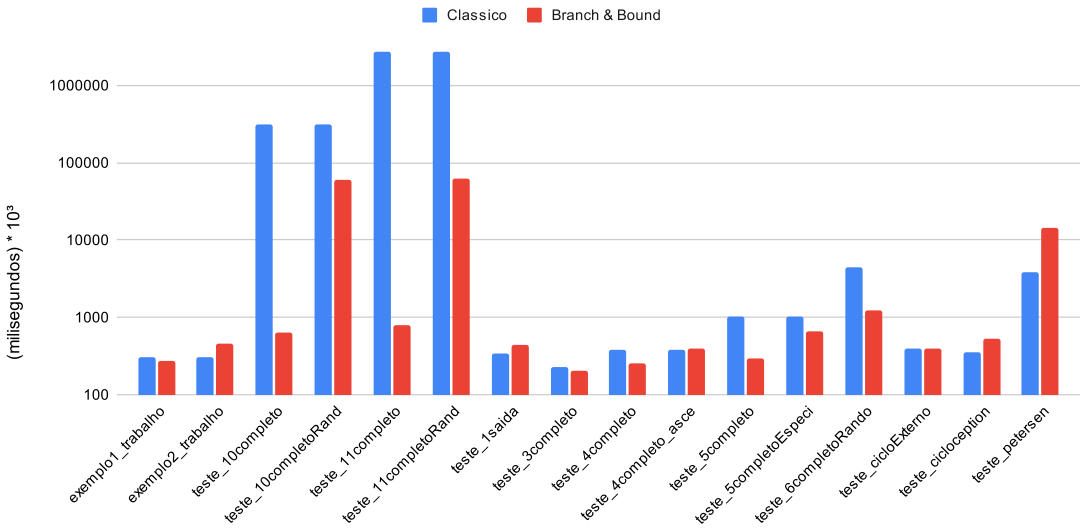


Figura 8 – Tempo (em $milisegundos * 10^3$) dispendido para exploração de grafos simples (Diretório *./exemplos/entrada/*).

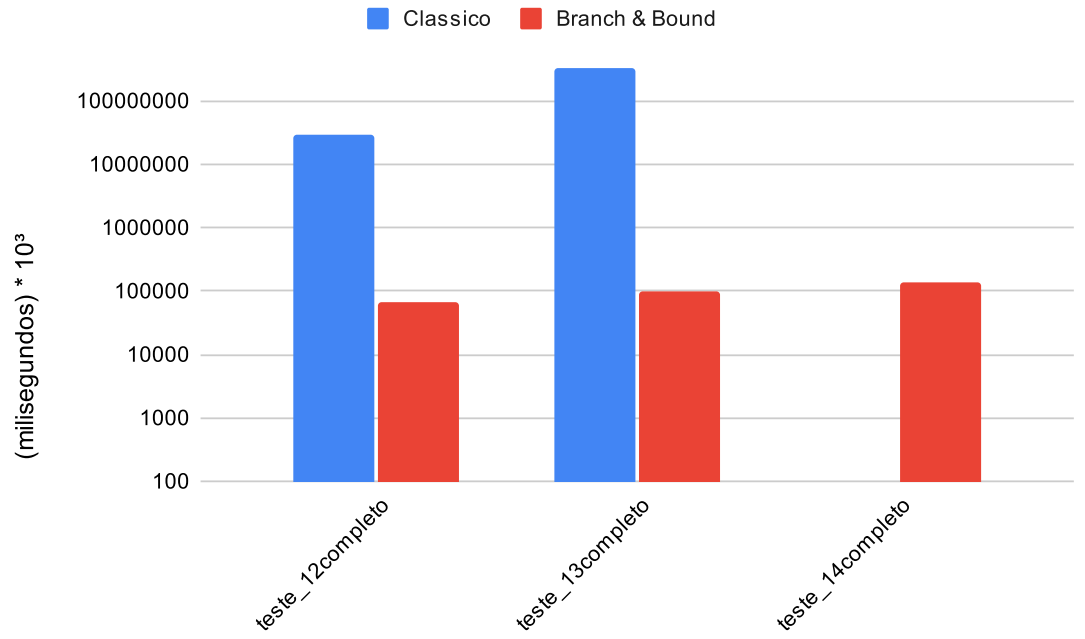


Figura 9 – Tempo (em $milisegundos * 10^3$) dispendido para exploração de grafos grande (Diretório *./exemplos/entrada/grande/*).

4 CONCLUSÃO

A partir dos resultados é notável que a aplicação da técnica de *Branch Bound* proposta é efetiva na otimização da busca por *Ciclos Simples Maximal iniciado em vértice 1*, não apenas obtendo ganhos na aplicação do processo mesmo que com a adição de bound de complexidade $O(L^2)$, mas tambem possibilitando a exploração de problemas mais complexos, como *teste_14completoRandomico* o que não seria possível no modelo Clássico.

Porem o modo como está proposto é diretamente afetado pela ordem em que os vértices estão distribuídos uma vez que a única ordenação de percurso dos vértices proposta é a léxica.

Outro ponto é a definição da função de *bound*, que por considerar que todos os vértices remanescentes possam contribuir para formação de um novo ciclo torna-se muito abrangente não efetuando podas nos casos em que existem arestas "pesadas"fora de um Ciclo Simples que envolva o vértice 1 ou em casos que o grafo em que não existam tais ciclos iniciados no vértice 1. Nestes casos, o desempenho do algoritmo contendo *bound* será pior que o original, devidas as operações extras feitas para calcular *bound*. Isso que pode ser observado para os experimentos *exemplo2_trabalho*, *teste_1saida* e *teste_cicloception*, onde o a quantidade de nós explorados é a mesma porem o tempo de algoritmo *Branch Bound* é maior que o a Busca Clássico.

Além disso, o algoritmo base descrito em 1.1 efetua a exploração de um mesmo ciclo mais de uma vez, diferindo, por exemplo de uma busca em profundidade clássica (BUSCA..., 2020) que visita cada um dos vértices por apenas uma vez porem que só possibilita a identificação da presença de ciclos em tempo de busca, mas não a obtenção destes ciclos (caminho, comprimento, etc.).

Dessa forma, é possível concluir que a solução proposta provém ganho de desempenho para obtenção da resposta ótima de problema estudado, mas ainda possui margem para aprimoramento, tanto na escolha de vértices a serem percorridos quanto na função de *bound*.

Referências

BUSCA em profundidade. 2020. [Acessado em 10 Julho 2021]. Disponível em: <https://pt.wikipedia.org/wiki/Busca_em_profundidade>. Citado na página 12.

FEOFILOFF, P. *Caminhos e ciclos em grafos*. 2017. [Acessado em 10 Julho 2021]. Disponível em: <https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/paths-and-cycles.html>. Citado na página 1.

KREHER, D. L.; STINSON, D. R. *Combinatorial algorithms: generation, enumeration, and search*. [S.l.]: CRC press, 1999. Citado na página 1.

MATRIZ de adjacência. 2021. [Acessado em 10 Julho 2021]. Disponível em: <https://pt.wikipedia.org/wiki/Matriz_de_adjac%C3%AAncia>. Citado na página 1.