Relatório de Implementação do Algoritmo de Busca por Interseção de Conjuntos para o Problema de Árvore Geradora de Rótulos Mínimos

Giovani Henrique Bertuzzo[1], Sávio de Oliveira Camacam[1]

[1]Departamento de Computação – Universidade Tecnológica Federal do Paraná (UTFPR)
Caixa Postal 271 – 87.301-899 – Campo Mourão – PR – Brazil

giovani.hbertuzzo@gmail.com, saviocamacam@alunos.utfpr.edu.br

Abstract. ***This paper describes the process of implementation of the Intersection Search of Sets for the problem of Trees Generating Minimum Labels. The problem of Trees Generating Minimum Labels consists in finding in a graph a tree that connects all the vertices of the graph - when it contains all the vertices connected through edges, each edge contains a label - using the least amount of labels many different. The work still presents, comparisons with other different algorithms of the literature proposed to solve the same problem.***

Resumo. ***Este trabalho descreve o processo de implementação da Busca por Interseção de Conjuntos para o problema de Árvores Geradoras de Rótulos Mínimos. O problema de Árvores Geradoras de Rótulos Mínimos consiste, em encontrar em um grafo, uma árvore que conecta todos os vértices do grafo - quando o mesmo contém todos os vértices ligados através de arestas, cada aresta contém um rótulo - utilizando a menor quantidade de rótulos diferentes. O trabalho ainda apresenta, comparações com outros diferentes algoritmos da literatura propostos para resolver o mesmo problema.***

1. Introdução

Um dos ramos da computação onde é estudado uma estrutura de dados que representa problemas através de grafos é a teoria dos grafos. Tal estrutura pode ser utilizada para representar diferentes problemas, modelando-o com vértices e arestas. Um dos problemas mais conhecidos é o da Árvore Geradora de Rótulos Mínimos, este trabalho apresenta o algoritmo de Busca por Interseção de Conjuntos, que soluciona tal problema.

a. ***Árvores Geradoras de Rótulos Mínimos***

O problema de Árvores Geradoras de Rótulos Mínimos consiste, em encontrar em um grafo não dirigido, uma árvore de busca que conecta todos os vértices do grafo, utilizando a menor quantidade de rótulos diferentes. Os vértices do grafo, são conectados através de arestas, onde cada aresta possui um rótulo representando o

custo de transição de um vértice para o outro.

b. Seções

A seção dois deste trabalho, apresenta diferentes trabalhos da literatura onde são demonstrados, diferentes algoritmos para solução do problema de Árvore Geradora de Rótulos Mínimos, na seção três é apresentada o algoritmo de Busca por Interseção de Conjuntos. A seção quatro descreve os principais métodos do algoritmo proposto, e a seção cinco relata os resultados, e faz comparações com os resultados dos demais algoritmos apresentados. Por fim na seção seis, é inferido uma conclusão obtida através de todo o estudo feito sobre o problema.

2. Trabalhos Relacionados

Ao realizar uma rápida pesquisa, é possível encontrar diferentes algoritmos com heurísticas utilizadas para solucionar o problema da Árvore Geradora de Rótulos Mínimos. Iremos fazer uma breve apresentação dos algoritmos mais conhecidas da literatura, seus resultados serão comparados com o algoritmo proposto, baseando-se em atributos para comparação como, número de vértices do grafo, tempo de execução, e a média dos valores da função objetivo (média do número de rótulos utilizados).

2.1. Modified Genetic Algorithm (Algoritmo Genético Modificado)

Os algoritmos genéticos baseiam-se no princípio de evolução, utilizando operações como mutação e cruzamento, aplicadas ao conceito de fitness (função objetivo) e então são gerados novos indivíduos. Para o problema de Árvore Geradora de Rótulos Mínimos, utiliza-se como fitness o número de rótulos utilizados na solução do problema.

A população inicial é gerada pela adição de rótulos aleatoriamente a conjuntos vazios, até surgirem soluções viáveis. As operações de cruzamento e mutação são então aplicadas para construir uma geração a partir da anterior. A operação de crossover cria um grupo de indivíduos a partir de seus "pais", que são soluções viáveis.

A partir dos pais P1 e P2, é feito a união e começa a se formar o subgrafo mais abrangente P. Em seguida, adiciona rótulos do subgrafo P ao conjunto solução inicialmente vazio, até obter uma solução viável. Por outro lado, a operação de mutação consiste em adicionar um novo rótulo aleatoriamente e, em seguida, tentar remover os rótulos (ou seja, as arestas associadas), do rótulo com menor freqüência ao mais freqüente, mantendo a viabilidade.

2.2. Algoritmo de Máxima Cobertura de Vértices (MVCA)

Dado um Grafo G = (V, E, L), com n vértices (V), m arestas (E) e l labels (L), L é o conjunto de possíveis rótulos para todas as arestas. Definimos H como um subgrafo de G que não tem aresta e tem todos os vértices de G. Enquanto o subgrafo H não for uma componente conexa, o algoritmo encontra o rótulo l, que as suas arestas (que contêm o rótulo l) irá cobrir o maior número de vértices que ainda não foram descobertos. Então adiciona-se todas as arestas com rótulo l ao subgrafo H. Este

procedimento é repetido até que se tenha um subgrafo H que contém todos os vértices de G e seja conexo.

2.3. Greedy Randomized Adaptive Search Procedure (Procedimento de busca adaptativa aleatória gulosa)

Dado um Grafo G = (V, E, L), com n vértices (V), m arestas (E) e l rótulos (L), onde L é o conjunto de possíveis rótulos para todas as arestas. C é o conjunto dos rótulos usados H é o subgrafo que contém todos os vértices de G induzido pelos rótulos de C, Comp(C) é o número de componentes conexas de H. C' é o conjunto global dos rótulos usados. H' é o subgrafo que contém todos os vértices de G induzidos pelos rótulos de C'. RCL é a lista de rótulos candidatos. Alpha é o número de rótulos em L.

Primeiramente o conjunto de rótulos C é vazio, então ele cria um grafo induzido por C, sem arestas, depois disso, o algoritmo é executado e coloca na RCL apenas os rótulos que tem o número de componentes conexas menor do que o valor alpha pré definido, se o número de iterações for maior que 2 então RCL vai ser o conjunto de rótulos e alpha agora vai ser o número de rótulos em L, após isso é escolhido um rótulo aleatória em RCL. Essa rótulo é adicionada no conjunto de rótulos usados C, o subgrafo H agora é o subgrafo induzido pelo novo conjunto de rótulos usados C e o comp(C) é atualizado. Enquanto o subgrafo tiver mais de uma componente, ou seja, Comp(C) > 1, RCL vai receber todos os rótulos que diminuem o número de componentes do grafo, agora novamente é escolhido uma rótulo aleatória em RCL esse rótulo é adicionada ao conjunto de rótulos usados C, o subgrafo H é atualizado junto com o Comp(C).

Após o algoritmo percorrer todas os rótulos do conjunto C, com cada uma delas ele deleta o rótulo de C atualiza o subgrafo H com base no novo conjunto C, atualiza o Comp(C) e verifica se o subgrafo continua com apenas uma componente conexa, se não for mais conexo ele desfaz a remoção e termina. Após os dois procedimentos ele verifica se |C'| > |C| então C' agora vai ser igual a C e o H' será induzido pelo novo C'. Isso se repete até que se obtenha a solução desejada.

3. Método proposto: Busca por Interseção de Conjuntos

A partir da visão da matriz de incidência de cada grafo como sendo uma composição de camadas que formava uma grande componente conexa, para solução do problema de Árvore Geradora de Rótulos Mínimos foi elaborada a proposta de decomposição dessas camadas. Cada camada aqui é entendida como a matriz de adjacência de um único rótulo.

A implementação desse algoritmo segue os seguintes pressupostos:

- Um grafo conexo pode ser compreendido como um conjunto que possui todos os vértices do grafo;
- A partir da extração da matriz de incidências de cada rótulo é possível reconhecer suas componentes internas, esse procedimento será explicado no tópico **Método: *getAllSubGraphs()***;
- É chamado apenas subgrafo de ***l***, um novo grafo que contém apenas as componentes de vértices que são alcançados por ***l***.
- Se um grafo possui ***L*** rótulos são necessários ***L*** subgrafos de ***l*** em ***L*** com

suas componentes internas ligadas apenas por seu próprio rótulo.

- Compreendendo as componentes internas de cada subgrafo ***l*** como conjuntos de vértices cobertos por ***l***, a solução para o problema nesse algoritmo segue os procedimentos:
    - Escolher um conjunto de heurísticas que elega os subgrafos mais importantes para gerar o resultado;
    - Usar os subgrafos eleitos numa lista de prioridade do subgrafo mais importante para o menos importante;
    - Escolher uma política de iteração por essa lista que otimize a interseção do subgrafo mais importante daquele que é o segundo mais importante e assim por diante, considerando que a relevância de um subgrafo pode mudar a cada iteração na lista de prioridades;
    - Com sucessivas iterações pela lista de prioridades de subgrafos considere a união das componentes internas de cada subgrafo de forma que progrida as iterações de forma que a cada iteração seja maximizada a adição de novos vértices e minimizada a criação de novas componentes tendendo sempre a 1 que vai ser ao final da execução a componente que contém todos os vértices de grafos com o menor número de rótulos compondo essa solução.

4. Algoritmo

A solução apresentada é composta por vários métodos importantes que trazem uma solução menor para o problema principal.

O procedimento abaixo denominado **bic-MLST-procedure** compõe a execução principal desta solução, sendo que dentro desse procedimento existem outros sub-procedimentos que serão explicados mais adiante.

```
bic-MLST-procedure(G)
        subgraphs = G.getAllSubGraphs()
        subgraphs.order()
        mainSubgraph = subgraphs.first()
        labelSet.add( mainSubGraph.label )

        while mainSubGraph.maxComponent.size != G.totalVertex do
                nextMainSubgraph = subgraphs.nextMostImportant()
                List listOfIndexUnion
                List<Components> auxListComponents
                Bool unionFlag
                for each component c1 in mainSubgraph.components do
                        for each component c2 in nextMainSubgraph.components
```

```
                index = nextMainSubgraph.components.index(c2)
                Set intersection = c2 union c1
                if [intersection.size != 0 and
                intersection.size < c2.size and
                intersection.size < c1.size or
                        c1.size < c2.size and
                        intersection.size == c1.size
                ] then
                        listOfIndexUnion.add(index)
                else [if intersection = mainSubgraph.vertex union
                nextMainSubgraph.vertex != void
                ] then
                        unionFlag = true
                end else if
        end for
        Component newComponent = c1
        if listOfIndexUnion.size > 0 then
                unionFlag = true
                for each index in listOfIndexUnion do
                        newComponent.addAll(nextMainSubgraph.
                        components(index))
                end for
                labelSet.add(nextMainSubgraph.label)
        end if
        auxListComponents.add(newComponent)
        listOfIndexUnion.reset
end for
Set indexRemove
List<Components> finalListComponents
for each Component c1 in auxListComponents do
        for each Component c2 in auxListComponents do
                if c1.size > c2.size and c1 != c2 then
                        Component c = c1 intersection c2
                        if c.size != 0 then
                                c1.addAll(c2)
                                indexRemove.add(auxListComponents.index(c2))
                        end if
                end if
        end for
end for
```

```
            for i=0 to auxListComponents.size do
                    if indexRemove.notContais(i) then
                            Component c = auxListComponents.get(i)
                            finalListComponents.add(c)
                    end if
            end for
            mainSubgraph.setComponents(finalListComponents)
            if unionFlag then
                    mainSubgraph.vertex.addAll(nextMainSubgraph.vertex)
                    unionFlag = false
            end if
        end while
        return labelSet.size
end procedure
```

5. Processo de Implementação

O programa foi escrito na linguagem Java e consistiu no uso de estruturas de dados básicas, como estruturas de conjuntos, listas e essencialmente matrizes. O procedimento de busca dos resultados é feito pasta por pasta do diretório de amostras e de cada arquivo nesses diretórios são extraídos os dez grafos que serão processados com o procedimento **bic-MLST-procedure** para trazer a média de valor objetivo do problema.

5.1. Principal método auxiliar: *getAllSubGraphs()*

Este método é responsável pela extração de cada subgrafo que compõe as componentes cobertas por um único rótulo. O procedimento é executado ***L*** vezes, sendo ***L*** a quantidade de rótulos do grafo chamando o método ***getSubgraphOf()***.

Considerando a matriz de adjacência aquela que compreende valores abaixo da diagonal principal de tamanho ***V x V*** o processo de extração é executado ***L x (V²/2)*** vezes.

```
get-Subgraph-Of-Procedure(L)
        matrix-subgraph = adjacence-matrix.clone()
        List<Component> component-list
        component-list.add(new-component-void)
        Set vertex-set
        jDimension = 0
        for i=0 to totalvertices do
                for j=0 to jDimension do
                        if [matrix-subgraph[i][j] != label and
                        matrix-subgraph[i][j] != total-labels] then
```

```
                        matrix-subgraph[i][j] = L
                end if
                if matrix-subgraph[i][j] == L
                        vertex-set.add(i)
                        vertex-set.add(j)
                        boolean added-I = false
                        boolean added-J = false
                        index-J = 0, index-I = 0
                        for Component c in component-list do
                                if c.is-empty() or c.contains(i) then
                                        c.add(j)
                                        added-I = true
                                        index-I = component-list.indexOf(c)
                                end if
                        end for
                        for Component c in component-list do
                                if c.is-empty() or c.contains(j) then
                                        c.add(i)
                                        added-J = true
                                        index-J = component-list.indexOf(c)
                                        if index-J != index-I then break
                                end if
                        end for
                        if(index-I !- index-J and added-I and added-J then
                                Component c = component-list.get(index-I)
                                component-set.get(index-J).addAll(c)
                                component-set.remove(c)
                        end if
                        if !added-I and !added-J then
                                Component c
                                c.add(i)
                                c.add(j)
                                component-set.add(c)
                        end if
                end if
        end for
        if jDimension <= total-vertex then jDimension++
end for
SubGraph s(L, matrix-subgraph)
s.setVertexSet(vertex-set)
```

```
        s.setComponentSet(component-list)
        return s
end procedure
```

O que esse método realmente faz é, dada a matriz de adjacências do grafo original, é clonada essa matriz e V²/2 vezes verifica se é o rótulo analisado em questão faz ligação entre um par de vértices, se não é o dado rótulo, preenche com o número padrão (quantidade de rótulos), e enquanto isso, faz verificações simples de conjuntos, é a estrutura básica usada em component-list, uma lista de conjuntos, que é iniciado com um primeiro conjunto, que representa uma componente, iniciada vazia para primeira interação.

A cada verificação entre um par de vértices, são buscados os possíveis conjuntos (componentes) onde esses vértices podem estar, se num dado momento, um deles está em um conjunto já existente, o outro apenas é adicionado, se cada um está em um conjunto diferente, então um conjunto consome o outro e se não estiver em nenhum conjunto, então um novo conjunto é criado com eles dois.

5.2. Outros Métodos

5.2.1. Método: *getMaxComponent()*

Esse método é usado na ordenação dos subgrafos na eleição do subgrafo mais importante no critério de escolha do subgrafo mais importante ser aquele que possui a maior componente conexa dentre dos os subgrafos.

Em outro momento, nas iterações do ***loop*** principal de busca das rótulos, ele é consultado para retornar o tamanho da máxima componente do subgrafo principal, como condição de parada desse ter a maior componente com tamanho igual ao total de vértices do grafo. O algoritmo não foi representado por julgar ser redundante.

5.2.2. Método: *orderSubGraphs()*

Já esse outro método é onde fica a dinamicidade da solução proposta, já que ele pode ser ajustado de acordo novas heurísticas de classificação dos subgrafos, considerando que a lista ordenada pode ser modificada a cada iteração do ***loop*** principal de escolha dos rótulos.

6. Resultados

Infelizmente a solução só encontra resultado próximo de resultados ótimos analisados da literatura, que inclusive são apresentados nas tabelas seguintes, para valores de ***n*** baixos ou com alta densidade, o que indica que o projeto ainda não tem um resultado razoável para se apresentar como possível solução.

Os resultados alcançados para as amostras que houveram resultados calculados que foram atestadas sua corretude, a quantidade de labels sempre se aproxima de duas vezes a quantidade de labels que outros algoritmos conseguiram, os valores podem ser verificados nas tabelas. Entretanto, a ser levado em conta os requisitos do computador e a linguagem de programação empregados, em detrimento daqueles usados pelos estudos

comparados da literatura, o tempo de execução da busca se mostrou satisfatório e de ordem quase constante, não sendo sensível a densidade do grafo, apenas em relação ao tamanho de ***n.***

Os valores apresentados foram retirados da última fase da implementação do projeto, que representa uma solução ***dummy*** por enquanto, já que não foi possível elaborar uma heurística mais completa combinada com uma política de recálculo das prioridades e relevâncias dos subgrafos a serem agregados a solução.

Essencialmente na busca feita da solução apresenta abaixo, o procedimento BIC executa uma quantidade de iterações fixa entre os subgrafos trazendo o conjunto de labels eleito pela solução ***dummy***, que é uma solução básica, ao invés de fazer n iterações e recalcular as prioridades e talvez até retirando labels que já haviam sido eleitos com o uso de pesos de probabilidade entre e a combinação dos labels na solução presente. Uma solução mais elaborada pode ser tópico de estudo em outros projetos na área de Árvores Geradoras Mínimas para acadêmicos que se interessarem pelo método proposto aqui.

Tabela 1: Resultados computacionais para grupo 1 (Tempo máximo = 1000 milisegundos)

Média de valores da função objetivo								
n	l	d	Exact	Pilot	MGA	GRASP	VNS	BIC
20	20	0.8	2.4	2.4	2.4	2.4	2.4	3.3
		0.5	3.1	3.2	3.1	3.1	3.1	4.4
		0.2	6.7	6.7	6.7	6.7	6.7	7.9
30	30	0.8	2.8	2.8	2.8	2.8	2.8	3.7
		0.5	3.7	3.7	3.7	3.7	3.7	5.6
		0.2	7.4	7.4	7.4	7.4	7.4	9.8
40	40	0.8	2.9	2.9	2.9	2.9	2.9	4.3
		0.5	3.7	3.7	3.7	3.7	3.7	6.3
		0.2	7.4	7.6	7.4	7.4	7.4	11.4
50	50	0.8	3	3	3	3	3	4.7
		0.5	4	4	4.1	4	4	6.5
		0.2	8.6	8.6	8.6	8.6	8.6	13.3
Total			55.7	56	55.8	55.7	55.7	81.2
Tempo computacional (milisegundos)								

n	l	d	Exact	Pilot	MGA	GRASP	VNS	BIC
20	20	0.8	0	0	15.6	1.6	0	15
		0.5	0	1.6	22	0	0	0.3
		0.2	11	3.1	23.4	0	1.6	0.6
30	30	0.8	0	3	9.4	1.6	0	2.6
		0.5	0	3.1	26.5	0	0	1.7
		0.2	138	4.7	45.4	1.5	5.2	1.2
40	40	0.8	2	6.3	12.5	1.5	0	5.3
		0.5	3.2	7.9	28.2	1.5	3.1	2.5
		0.2	$100.2x10^3$	10.8	120.3	15.6	9.6	2.6
50	50	0.8	3.1	17.1	21.8	3	0	5.4
		0.5	21.9	20.2	531.3	9.4	4.1	5.4
		0.2	$66.3x10^3$	17.2	93.6	3.2	11.9	4.1
Total			$166.7x10^3$	95	950	38.9	35.5	46.7

Tabela 2: Resultados computacionais para grupo 2 com n=100 (Tempo máximo = $20x10^3$ milisegundos)

Média de valores da função objetivo								
n	l	d	Exact	Pilot	MGA	GRASP	VNS	BIC
100	25	0.8	1.8	1.8	1.8	1.8	1.8	1.8
		0.5	2	2	2	2	2	2.7
		0.2	4.5	4.5	4.5	4.5	4.5	7.3
	50	0.8	2	2	2	2	2	3
		0.5	3	3.1	3	3	3	4.9
		0.2	6.7	6.9	6.7	6.7	6.7	11.9
	100	0.8	3	3	3	3	3	6.1
		0.5	4.7	4.7	4.7	4.7	4.7	8.8
		0.2	NF	10.1	9.9	9.8	9.7	18.7
	125	0.8	4	4	4	4	4	6.7

		0.5	5.2	5.4	5.2	5.2	5.2	9.6
		0.2	NF	11.2	11.1	11	11	20.5
Total			-	58.7	57.9	57.7	57.6	102
Tempo computacional (milisegundos)								
n	l	d	Exact	Pilot	MGA	GRASP	VNS	BIC
100	25	0.8	9.4	4.7	26.5	0	0	10
		0.5	14	12.6	29.7	4.6	0	4.9
		0.2	34.3	23.2	45.3	9.3	3.1	5.7
	50	0.8	17.8	67.3	23.5	6.4	7.7	16.4
		0.5	23.5	90.7	106.2	51.6	42.4	7.1
		0.2	$10.2x10^3$	103.2	148.3	57.8	49.7	13.2
	100	0.8	142.8	378.1	254.7	61	215	24.1
		0.5	$2.4x10^3$	376.2	300	28.2	114.7	27.4
		0.2	NF	399.9	$9.4x10^3$	$1.2x10^3$	414.8	18.5
	125	0.8	496.9	565.7	68.7	9.4	10.1	25.9
		0.5	$179.6x10^3$	576.3	759.4	595.4	551.1	19.8
		0.2	NF	634.5	$2x10^3$	562.9	420.4	21.4
Total			-	$3.2x10^3$	$13.2x10^3$	$2.6x10^3$	$1.8x10^3$	194.4

Tabela 3: Resultados computacionais para grupo 2 com n=200 (Tempo máximo = $60x10^3$ milisegundos)

Média de valores da função objetivo								
n	l	d	Exact	Pilot	MGA	GRASP	VNS	BIC
200	50	0.8	2	2	2	2	2	2
		0.5	2.2	2.2	2.2	2.2	2.2	3.1
		0.2	5.2	5.2	5.2	5.2	5.2	8.6
	100	0.8	2.6	2.6	2.6	2.6	2.6	3.9
		0.5	3.4	3.4	3.4	3.4	3.4	6.2
		0.2	NF	8.3	8.3	8.1	7.9	14.9

	200	0.8	4	4	4	4	4	6.5
		0.5	NF	5.5	5.4	5.4	5.4	10.5
		0.2	NF	12.4	12.4	12.2	12	25.2
	250	0.8	4	4	4	4.1	4	8.3
		0.5	NF	6.3	6.3	6.3	6.3	12.2
		0.2	NF	13.9	14	13.9	13.9	28.6
Total			-	69.8	69.8	69.4	68.9	130
Tempo computacional (milisegundos)								
n	l	d	Exact	Pilot	MGA	GRASP	VNS	BIC
200	50	0.8	29.7	90.7	26.5	20.5	0	49
		0.5	32.7	164.1	68.8	14.2	17.2	43.2
		0.2	$5.4x10^3$	320.4	326.6	37.5	241.3	24.6
	100	0.8	138.6	876.5	139.3	45.3	123.2	107.9
		0.5	807.8	$1.2x10^3$	$1.6x10^3$	176.6	151.1	41.9
		0.2	NF	$1.3x10^3$	$2.2x10^3$	667.2	$1.7x10^3$	44.8
	200	0.8	$22.5x10^3$	$5.9x10^3$	204.6	43.6	32	133.9
		0.5	NF	$5.6x10^3$	$16.1x10^3$	885.6	971.9	97.5
		0.2	NF	$5x10^3$	$12.7x10^3$	$9.4x10^3$	$12.8x10^3$	80.1
	250	0.8	$20.6x10^3$	$9.1x10^3$	$2.2x10^3$	$4.9x10^3$	$1.1x10^3$	175.6
		0.5	NF	$8.4x10^3$	$17.6x10^3$	506	$3.4x10^3$	112.9
		0.2	NF	$8x10^3$	$26.4x10^3$	$1.4x10^3$	$3.2x10^3$	72.3
Total			-	$45.9x10^3$	$79.6x10^3$	$18.1x10^3$	$23.7x10^3$	983.7

Tabela 4: Resultados computacionais para grupo 2 com n=500 (Tempo máximo = $300x10^3$ milisegundos)

Média de valores da função objetivo								
n	l	d	Exact	Pilot	MGA	GRASP	VNS	BIC
500	125	0.8	2	2	2	2	2	2.3
		0.5	2.6	2.6	2.6	2.6	2.6	3.8

		0.2	NF	6.3	6.2	6.2	6.2	11.2
	250	0.8	3	3	3	3	3	4.5
		0.5	NF	4.2	4.3	4.2	4.1	7.1
		0.2	NF	9.9	10.1	9.9	9.9	19.7
	500	0.8	NF	4.8	4.7	4.7	4.7	7.9
		0.5	NF	6.7	7.1	6.5	6.5	13.4
		0.2	NF	15.9	16.6	15.9	15.8	33.7
	625	0.8	NF	5.1	5.4	5.1	5.1	10.3
		0.5	NF	8.1	8.3	7.9	7.9	16.4
		0.2	NF	18.5	19.1	18.4	18.3	38.4
Total			-	87.1	89.4	86.4	86.1	168.7
Tempo computacional (milisegundos)								
n	l	d	Exact	Pilot	MGA	GRASP	VNS	BIC
500	125	0.8	370	$3.4x10^3$	18	152	17.1	474.5
		0.5	597	$6.6x10^3$	$2.6x10^3$	455	$1.1x10.^3$	449.4
		0.2	NF	$11.9x10^3$	$57.1x10^3$	$4x10^3$	$3.9x10^3$	396.4
	250	0.8	$5.3x10^3$	$35.49x10^3$	516	248	142.3	1034.8
		0.5	NF	$65.3x10^3$	$28x10^3$	583	$84x10^3$	895.2
		0.2	NF	$156.4x10^3$	$181.2x10^3$	$3.3x10^3$	$5.1x10^3$	698.5
	500	0.8	NF	$200.5x10^3$	$117.5x10^3$	$28.1x10^3$	$22.3x10^3$	1693.3
		0.5	NF	$190.1x10^3$	$170.9x10^3$	$90.9x10^3$	$32.3x10^3$	1657.9
		0.2	NF	$300.6x10^3$	$241.8x10^3$	$20.2x10^3$	$139.7x10^3$	1212.2
	625	0.8	NF	$184.3x10^3$	$51.9x10^3$	$4.9x10^3$	$16.1x10^3$	1944.7
		0.5	NF	$200.9x10^3$	$22.2x10^3$	$35.7x10^3$	$44.7x10^3$	1932.3
		0.2	NF	$289.9x10^3$	$297.8x10^3$	$53.1x10^3$	$155.5x10^3$	1482.6
Total			-	$1645.3x10^3$	$1371.5x10^3$	$213.8x10^3$	$504.9x10^3$	13871.8

7. Conclusão

A implementação da solução na versão até então apresentada ficou com algumas ameaças a validade dos resultados alcançados, especialmente pelo tamanho da base, que apesar de não ser tão grande, ainda é extensa para ser avaliada passo-a-passo quanto a corretude dos seus resultados, no que diz respeito, à filtragem dos grafos com vértices que não são alcançados por nenhum label, e que logo são componentes isoladas, o caso de labels que também não ligam nenhum par de vértices, ou seja, alguns casos pontuais que nossa solução pode apresentar alguma inconsistência, considerando que grafos que não tem as características acima citada, haverá um resultado não-ótimo, mas factível.

Ao iniciar a implementação da solução para o problema de Árvores Geradoras de Rótulos Mínimos, nos deparamos com inúmeras questões, mas a essencial, é a escolha das heurísticas de classificação e busca dos labels e o algoritmo em si, que será implementado, dentre o uso de algoritmos consolidados da literatura, ou o risco de implementar uma nova solução, que torna o processo mais intenso, já que todo novo problema ou impasse é inédito e toda primeira solução, é por assim dizer, uma solução óbvia demais.

Ao analisar o problema, podemos enxergar um grafo em sua essência como uma estrutura a ser representada computacionalmente, de fato como um grafo, com seus nós e arestas e todo o mapeamento e orientação a objetos necessária para colocar o problema físico do mundo real no computador, ou podemos levar o problema para um outro tipo de abstração, que foi o que fizemos levando a uma interpretação mais abstrata com conjuntos e padrões matriciais, que julgamos ser mais eficiente, sendo essa uma afirmação passível de mais testes e comparações dos resultados para atestar que o ***bic-procedure*** representa de fato, uma solução plausível para esse problema, gerando assim apontamentos para futuros estudos.

Referências

S. Consoli a,*, K. Darby-Dowman a , N. Mladenovic´ a , J.A. Moreno Pérez b. (2009) “Greedy Randomized Adaptive Search and Variable Neighbourhood Search forthe minimum labelling spanning tree problem”, In: European Journal of Operational Research disponível em https://www.journals.elsevier.com/european-journal-of-operational-research/