Algoritmo Genético para resolução da Árvore Geradora de Rótulos Mínimos

Emanuel Felipe Giroldo Mazzer[1], João Vitor Foralosso Gris [1], Leonardo Pontes Baiser [1]

[1]DACOM – Universidade Tecnológica Federal do Paraná (UTFPR)
Caixa Postal 271 – 87301-899 – Campo Mourão – PR – Brazil

{emanuelgiroldo,joaogrisxv, lpbaiser}@gmail.com

Resumo. *Este relatório apresenta a construção de uma algoritmo genético para a resolução do Problema de Árvore Geradora de Rótulos Mínimos(PAGRM). Neste artigo propomos apresentar o problema relacionado, os trabalhos correlacionados a este assunto, a implementação do algoritmo para solução do problema e a demonstração dos resultados obtidos nos testes em comparação com os resultados reportados em [5].*

1. Introdução

Em geral, a maioria dos problemas as serem resolvidos em grafos consistem em encontrar arvores geradoras de peso mínimos e otimizar alguma medida. O Problema da Árvore Geradora de Peso Mínimos é um dos problemas mais clássicos da teoria dos grafos. Para muitos desses problemas, existem algoritmos em tempo polinomial que utilizam a seguinte ideia: Dado um grafo G não direcionado, com arestas E ponderadas, deve-se encontrar uma árvore T onde todos os vértices V do subgrafo induzido em T sejam conexos e que a soma do peso de todas as arestas seja o menor possível. Existe inúmeras ramificações para este problemas, são algumas delas: Árvore geradora de mínimo e máximo, árvore geradora de diâmetro mínimo, árvore geradora de rótulos mínimos, entre outros.

De fato existem muitas aplicações da solução deste problema, como exemplo, dado um projeto de circuito eletrônico, deseja-se que os pinos de vários componentes se tornem eletricamente equivalentes, isto somente é alcançado ligando-se os pinos uns aos outros. Para construir a conexão de um conjunto de n pinos, podemos gerar um conjunto de n - *1* fios, onde cada fio conecta dois pinos. Dados todos os conjuntos de fios possíveis, o qual utiliza a mínima quantidade de fio é o escolhido para o circuito [Cormen et al. 2001].

Neste relatório apresentamos um dos ramos do problema da árvore geradora de peso mínimo, a árvore geradora de rótulos mínimos - PAGRM (do inglês *minimum labelling spanning tree*). O objetivo do PAGRM, dado que cada aresta possui um rótulo, encontrar uma árvore geradora que percorra todos os vértices utilizando a menor quantidade de rótulos possíveis. Diferente da árvore geradora de pesos mínimos que consiste em encontrar o menor caminho possível ou seja a menor soma de pesos das aretas, a árvore geradora de rótulos mínimos utiliza de uma combinação matemática sobre os rótulos para encontrar um caminho que viste todos os vértices do grafo e utilize a menor quantidade de rótulos possíveis.

2. O problema

Em um PAGRM, é dado um grafo com cada uma de suas arestas marcadas com uma *label* e procura-se um grafo que seja induzido pelo menor número de *labels* possível.

O problema pode ser visto no mundo real em redes de comunicação, redes elétricas, dentre outros. Por exemplo em linhas de telefone, cada nó de comunicação pode se comunicar com diferentes nós escolhendo diferentes rotas. O grafo é representado pelo conjunto de nós de comunicação e as linhas que os ligam, cada uma com seu meio, o problema é encontrar uma arvore geradora que utiliza o numero mínimo de meios diferentes. Essa árvore geradora pode reduzir o custo e a complexidade para construção da rede.

Definição: Dado um grafo rotulado não-dirigido $G = \{V, E, L\}$, no qual $V = \{v_1, v_2 \dots v_{|V|}\}$, ou seja o conjunto de n vértices, $E = \{(u, v) \mid u, v \in V \wedge u \neq v\}$, o conjunto de m arestas e $L = \{l_1, l_2 \dots l_{|L|}\}$, o conjunto de l rótulos, encontrar uma árvore geradora T apartir de G tal que $|LT|$ é mínimo. Define-se LT como o conjunto de diferentes rótulos das arestas em uma árvore geradora T. Uma solução factível é definida como um conjunto de rótulos C contidos em L, tal que todas as arestas com rótulos em C representam um subgrafo conexo de G o qual contém todos os vértices de G. Se C é uma solução realizável, então qualquer árvore geradora possui no máximo $|C|$ rótulos. Se a solução é ótima, então qualquer árvore geradora C é uma árvore geradora de rótulos mínimos. Assim, para resolver o problema, devemos encontrar uma solução possível com o mínimo número de rótulos possíveis.

3. Trabalhos Relacionados

Nos tópicos que seguem serão demonstrados alguns trabalhos correlacionados com o problema da árvore geradora de rótulos mínimos.

Algoritmo de substituição de arestas (Edge Replacement Algorithm)

Dado um Grafo G = (V, E, L), com n vértices (V), m arestas (E) e l *labels* (L), onde L é o conjunto de possíveis *labels* para todas as arestas.

Algorithm 1: Algoritmo de substituição de arestas (Edge Replacement Algorithm)

Encontra uma spanning tree ST arbitrária de G;
for *Para todas as arestas i que não estão na arvore* **do**
  Label(i) vai ser igual ao *label* da aresta i. Se Label(i) ainda não apareceu na ST vai para o passo 7;
  Encontre o ciclo C, que é criado adicionando a aresta i;
  Calcula a heurística para cada possível movimento;
  A variável current_label_count vai ser o número de vezes que label(i) apareceu em C, a variável minimun_*label* será o *label* que menos apareceu em C e a variável minimun_count será o número de vezes que a *label* minimun_*label* apareceu;
  Se o current_label_count ¿ minimun_count e o label(i) != do minimun_*label* então adicione a aresta i na *ST* e remova da árvore as arestas da arvore cuja o *label* é igual ao minimun_label;
end
Encontre uma ST arbitrária em H

Primeiro encontra-se uma árvore de escoamento (ST) arbitrária, então, procurando reduzir o número total de *labels* da ST, testa para cada aresta que não esta na arvore, se após adcioná-la na ST e remover uma aresta já contida, há a possibilidade de se obter uma ST com menor número de *labels*.

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), onde L é o conjunto de possíveis *labels* para todas as arestas. Saída ST

Algorithm 2: Algoritmo de Máxima Cobertura de Vértices (MVCA)

H é um subgrafo de G que não tem nenhuma aresta e tem todos os vértices de G, onde H = (V,)
while *Enquanto H não é conexo* **do**
  Encontre o *label* l que as arestas com este *label* vai cobrir o maior número de vértices que ainda não foram cobertos Adiciona todas as arestas com *label* l no subgrafo H
end

Este algoritmo é baseado em construir uma *ST* gradativamente. Isso acontece selecionando o *label* cuja as arestas que contém esse *label* cubram a maior quantidade de vértices que ainda não foram cobertos, este procedimento é repetido até que se tenha um subgrafo H que contém todos os vértices de G e é conexo.

Algoritmo Exato (*ExactAlgorithm*)

Ele é baseado no algoritmo A*, e funciona de forma que o nó a ser expandido é sempre o de menor custo. Para calcular o custo f de cada nó são necessárias duas partes g(x) e h(x) onde g(x) é igual ao custo do caminho de busca atual, da raiz r até esse nó x, onde g(r) = 0 e h(x) é um custo estimado do melhor caminho de x até um nó objetivo, onde o h(nó objetivo) = 0.
Dado um Grafo G = (V, E, L), com n vértices (V), m arestas (E) e l *labels* (L),

onde L é o conjunto de possíveis *labels* para todas as arestas.
Saída ST com o número mínimo de *labels*.

Algorithm 3: Algoritmo Exato (*ExactAlgorithm*)

Coloque o nó raiz r em OPEN - OPEN é o lugar que armazena todos os nós gerados e ainda não expandidos
Se OPEN estiver vazio, retorna erro - se existe uma solução este passo não será executado
Remove-se de OPEN e coloca-se em CLOSED um nó n cuja o seu f é mínimo - CLOSED é o lugar que armazena todos os nós expandidos
Remove-se de OPEN e coloca-se em CLOSED um nó n cuja o seu f é mínimo - CLOSED é o lugar que armazena todos os nós expandidos
Se n é um nó objetivo vai para o passo 8
Senão expanda n. Se existem k *labels* não selecionados, então n tem k filhos (um pra cada *label* não selecionada)
for *Para cada filho n' de n* **do**
    Se n' não esta em OPEN ou em CLOSED, calcula-se h(n') e f(n') = g(n') + h(n') onde g(n') = g(n) + 1 e g(r) = 0 Coloque n' em OPEN
end
Encontre a ST do subgrafo

Primeiramente coloca-se o nó raiz no conjunto dos nós gerados OPEN. Agora começa a iteração que vai remover de OPEN o nó n com menor custo f e adicioná-lo ao conjunto dos expandidos CLOSED, após isso, verificamos se n é um nó objetivo, ou seja, se obteve-se um subgrafo de espalhamento, se n for um nó objetivo, sai da iteração e tenta encontrar a ST do subgrafo. Se n não for um nó objetivo, ele irá expandir para verificar para todos os filhos n′ de n se n′ já foi criado ou expandido, se ele não estiver nem em OPEN nem em CLOSED será calculado então o custo de n′ e ele sera adicionado ao conjunto OPEN, após isso retorna-se ao passo de remover de OPEN e assim sucessivamente até encontrar um nó objetivo, e depois a ST do Subgrafo.

Busca Gulosa Randômica Adaptativa (*GreedyRandomizedAdaptativeSearchProcedure*)

Dado um Grafo G = (V, E, L), com n vértices (V), m arestas (E) e l *labels* (L), onde L é o conjunto de possíveis *labels* para todas as arestas. Saída ST.

Algorithm 4: Busca Gulosa Randômica Adaptativa (*GreedyRandomizedAdaptativeSearchProcedure*)

begin
C é o conjunto das *labels* usadas H é o subgrafo que contém todos os vértices de G induzido pelos *labels* de C Comp(C) é o numero de componentes conexas de H C' é o conjunto global das *labels* usadas H' é o subgrafo que contém todos os vértices de G induzido pelos *labels* de C'
end
repeat
C ←0
H vai ser o subgrafo induzido de G com as *labels* que agora estão em C Construction-Phase(C)
Local-Search(C)
if *C' > C* **then**
C' = C H' vai ser o subgrafo induzido de G com as *labels* que agora estão em C'
end
until *Até que se obtenha uma ST de H';*
;

Algorithm 5: Procedure Construction-Phase(C)

begin
RCL ←0, vai ser a lista de candidatos de tamanho α
end
if *Number of Iterations > 2* **then**
RCL ←L α←l Pega uma *label* c randomica em RCL Adiciona a *label* c no conjunto de *labels* usadas C H vai ser o subgrafo induzido de G com as *labels* que agora estão em C Comp(C) é atualizado
end
while *Comp(C) > 1* **do**
RCL vai ser o conjunto de todas as *labels* em L que minimizem Comp(C) Pega uma *label* c randomica em RCL Adiciona a *label* c no conjunto de *labels* usadas C H vai ser o subgrafo induzido de G com as *labels* que agora estão em C Comp(C) é atualizado
end

Algorithm 6: Procedure Local Search(C)

for *i = 1, enquanto i for menor que o numero de labels em C* **do**
Remove a *label* i do conjunto C H vai ser o subgrafo induzido de G com as *labels* que agora estão em C Comp(C) é atualizado
end
if *Comp(C) > 1* **then**
Adiciona o *label* i ao conjunto C H vai ser o subgrafo induzido de G com as *labels* que agora estão em C Comp(C) é atualizado
end

Primeiramente o conjunto de *labels* C é vazio, então ele cria um grafo induzido por C, ou seja, sem arestas, depois disso, o procedimento Construction-Phase é executado, o Construction-Phase coloca na RCL apenas as *labels* que tem o numero de componentes conexas menor do que o valor alpha pré definido, se o numero de iterações for maior que 2 então RCL vai ser o conjunto de *labels* e alpha agora vai ser o numero de *labels* em L, após isso é escolhido uma *label* aleatória em RCL essa *label* é adicionada no conjunto de *labels* usadas C, o subgrafo H agora é o subgrafo induzido pelo novo conjunto de *labels* usadas C e o comp(C) é atualizado, agora, enquanto o subgrafo tiver mais de uma componente, ou seja, Comp(C) > 1, RCL vai receber todas as *labels* que diminuem o numero de componentes do grafo, agora novamente é escolhido uma *label* aleatória em RCL essa *label* é adicionada ao conjunto de *labels* usadas C, o subgrafo H é atualizado junto com o Comp(C). Após chamar o Construction-Phase o algoritmo chama o Local-Search, o algoritmo Local-Search percorre todas as *labels* do conjunto C, com cada uma delas ele deleta a *label* 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' sera induzido pelo novo C'. Isso se repete até que se obtenha a árvore de escoamento.

Algoritmo VNS

Dado um Grafo $G = (V, E, L)$, com n vértices (V), m arestas (E) e l *labels*(L), onde L é o conjunto de possíveis *labels* para todas as arestas.

Algorithm 7: VNS

```
begin
    C é o conjunto global das labels usadas;
    H é o subgrafo que contém todos os vértices de G induzido pelos
     labels de C;
    Comp(C) é o numero de componentes conexas de H;
    C' é o conjunto das labels;
    H' é o subgrafo que contém todos os vértices de G induzido pelos
     labels de C';
    Comp(C') é o numero de componentes conexas de H';
    C ←Solução Inicial Randômica
end
repeat
    K ←1;
    Kmax ←—C— + —C—/3;
    while K ¡ Kmax do
        C' ←Shaking-Phase(Nk(C));
        Local-Search(C');
        if |C'| < |C| then
            C ←C' ;
            k ←1;
            Kmax ←—C— + —C—/3;
        else
            k ←k+1;
        end
    end
until Até satisfazer a condição de ter uma ST de H;
```

Algorithm 8: Procedure Shaking-Phase(Nk(C))

```
begin
    C' ←C;
end
for i = 1 enquanto i ¡ k do
    rnd = Random(0, 1);
    if rnd ≤0,5 then
        Remove um label aleatório em C';
    else
        Adiciona um label que não esta sendo usado ainda em C';
        Atualiza o subgrafo H' com o novo valor de C';
    end
end
```

Algorithm 9: Procedure Local Search(C)

```
while Comp(C') ¿ 1 do
    S é o conjunto de labels não utilizados que minimizam o numero de
     componentes conexos;
    Seleciona um label randômico u que pertence a S;
    Adiciona u em C';
    Atualiza o subgrafo H' com o novo valor de C' e o Comp(C');
end
for i = 1 , enquanto i ¡ —C'— do
    Remove a label i do conjunto C';
    H' vai ser o subgrafo induzido de G com as labels que agora estão em
     C';
    Comp(C') é atualizado;
    if Comp(C') ¿ 1 then
        Adiciona o label i ao conjunto C';
        H' vai ser o subgrafo induzido de G com as labels que agora estão
         em C';
        Comp(C') é atualizado;
    end
end
```

Primeiramente ele inicia C com uma solução exequível e deixa o parâmetro K variando durante a execução, ele muda conforme o valor de $|C'|$ com relação ao $|C|$, o procedimento Shaking-Phase representa a ideia central do VNS, ele muda a estrutura da vizinhança quando o Local-Search esta preso em um minimo local. O Local-Search funciona adicionando *labels* em C' até que o grafo H' tenha apenas um componente conexo, depois disso ele vai removendo os *labels* de C' com intuito de diminuir o numero de *labels* de H' sem aumentar o número de componentes conexas. Depois disso o algoritmo verifica se o numero de *labels* de C' é menor do que o gerado aleatoriamente no começo do algoritmo, se for menor então C vai ser igual a C', K recebe 1 e Kmax $|C| + |C|/3$, senão for menor incrementa o tamanho da estrutura dos vizinhos ($k = k + 1$), faz isso até encontrar uma ST arbitrária.

4. Solução Proposta

A solução que a equipe escolheu para resolver o PAGRM foi o *Genetic Algorithm.* No PAGRM nos é dado um Grafo G = (V, E, L), com n vértices (V), m arestas (E) e l *labels* (L), onde L é o conjunto de possíveis *labels* para todas as arestas, uma solução factível para o PAGRM é um conjunto de *labels* C que contém algumas *labels* de L e o grafo induzido pelas *labels* em C forma uma grafo que contém todos os vértices de G e apenas uma componente conexa.

Inicialização: O algoritmo é inicializado recebendo um grafo, após isso o que deve ser feito é, a partir deste grafo, montar um conjunto de *labels* que induzam o grafo conexo. Esse conjunto de *labels* pode ser obtido de forma aleatória, ou seja adicionando uma *label* por vez nele e testando se o grafo induzido por essas *labels* tem apenas uma componente conexa, ou pode ser obtido usando um algoritmo como o MVCA, assim obtendo-se uma população incial.

Crossover: O crossover gera um descendente da união de dois indivíduos, para gerar o novo descendente o crossover ordena as *labels* do descendente gerado em ordem decrescente por sua frequência de aparição no grafo, o operador vai adicionando os *labels* no novo descendente (inicialmente vazio) em ordem decrescente, até que esse descendente se torne uma solução fazível.

Algorithm 10: Crossover(s[1], s[2])

S = s[1] U s[2];
T = {};
Ordena os *labels* de S em ordem decrescente por sua frequência em G;
Adiciona *labels* de S do primeiro ao último em T, até que T seja uma solução factível;
Retorna T;

Mutação: Dada uma solução viável, uma nova solução pode ser obtida utilizando a mutação. Primeiramente um rótulo que não esta na solução apresentada é adicionado a ela, após a inserção, o algoritmo remove um a um os rótulos em ordem crescente de frequência de aparição, desde que, depois da remoção, a solução continue sendo factível.

Algorithm 11: Mutation(S)

Seleciona aleatóriamente um *label* c que não esta em S;
T = S U c;
Ordena os *labels* em T em ordem decrescente por sua frequência em G;
O algoritmo remove um a um os *labels* em ordem crescente por sua frequência de aparição em G, desde que T se mantenha com uma solução factível;
Repete a remoção do passo quatro até que não se possa mais remover *labels* ;
Retorna T;

5. Resultados

Após terminada a implementação do algoritmo, executamos o experimento sobre a base proposta por [Cerulli et al. 2005] e comparamos os resultados computacionais obtidos do algoritmo genético proposto neste trabalho, com o algoritmo genético implementado por [Consoli et al.2009]. A base de testes utiliza 10 instâncias de cada grafo com um valor de densidade, sendo a densidade das aresta no grafo variando de alta densidade(0.8), média densidade(0.5), baixa densidade(0.2), com o intuito de avaliar o comportamento do algoritmo quando influenciado pelo tamanho de um grafo e a densidade de rótulos, foram executados testes sobre o grupo 1, grupo 2 com n=100, grupo 2 com n=200 e grupo 2 com n=500. Os testes foram executados em uma máquina Intel I7 CPU @ 3.3GHz com 8 GB RAM, salvo que no experimento realizado por [Cerulli et al. 2005] foi utilizado uma máquina Pentium Centrino microprocessador at 2.0 GHz com 512 MB RAM.

Tabela 1. Grupo 1

Média da Função Objetivo					Tempo computacional (ms)	
Parâmetros			Consoli et al. 2009	SPMGA:2016	Consoli et al. 2009	*SPMGA:2016
n	l	d	MGA	MGA	MGA	MGA
20		0.8	2.4	6.3	15.6	55
		0.5	3.1	7.6	22	36
		0.2	6.7	10	23.4	98
30		0.8	2.8	6.8	9.4	19
		0.5	3.7	9.3	26.5	62
		0.2	7.4	14.5	45.4	33.5
40		0.8	2.9	9.2	12.5	71
		0.5	3.7	9.6	28.2	45.6
		0.2	7.4	16	120.3	87.3
50		0.8	3	11.4	21.8	64.5
		0.5	4.1	13.8	531.3	99
		0.2	8.6	16.6	93.6	98
Total			55.8	131,1	950	768,9

Tabela 2. Grupo 2 n=100

Média da Função Objetivo					Tempo computacional (ms)	
Parâmetros			Consoli et al. 2009	SPMGA:2016	Consoli et al. 2009	SPMGA:2016
n	l	d	MGA	MGA	MGA	MGA
100	25	0.8	1.8	6.2	26.5	81.6
		0.5	2	3.4	29.7	67.1
		0.2	4.5	2.2	45.3	98.4
	50	0.8	2	8.2	23.5	108.9
		0.5	3	5.7	106.2	102.5
		0.2	6.7	2.8	148.3	95.9
	100	0.8	3	16.8	254.7	212.6
		0.5	4.7	23	300	301.6
		0.2	9.9	21	$9.4x10^3$	346,5
	125	0.8	4	19.2	68.7	231.8
		0.5	5.2	NF	759.4	NF
		0.2	11.1	NF	$2x10^3$	NF
Total			57.9	108.5	$13.2x10^3$	$1,6x10^3$

Tabela 3. Grupo 2 n=200

Média da Função Objetivo					Tempo computacional (ms)	
Parâmetros			Consoli et al. 2009	SPMGA:2016	Consoli et al. 2009	*SPMGA:2016
n	l	d	MGA	MGA	MGA	MGA
200	50	0.8	2	9.4	26.5	288.2
		0.5	2.2	5.6	68.8	225.6
		0.2	5.2	3.8	326.6	165.2
	100	0.8	2.6	13.2	193.3	360.1
		0.5	3.4	5	$1.6x10^3$	300
		0.2	8.3	3.7	$2.2x10^3$	217.1
	200	0.8	4	24.3	204.6	$1,1x10^3$
		0.5	5.4	24.7	$16.1x10^3$	$1,0x10^2$
		0.2	12.4	30	$12.7x10^3$	$1,7x10^3$
	250	0.8	4	28.8	$2.2x10^3$	$1,9x10^3$
		0.5	6.3	24	$17.6x1^3$	$1,1x10^3$
		0.2	14	32	$26.4x10^3$	$1,5x10^3$
Total			69.8	204.5	$79.6x10^3$	$10,085x10^3$

Tabela 4. Grupo 2 n=500

Média da Função Objetivo					Tempo computacional (ms)	
Parâmetros			Consoli et al. 2009	SPMGA:2016	Consoli et al. 2009	SPMGA:2016
n	l	d	MGA	MGA	MGA	MGA
500	125	0.8	2	13,4	18	$3,9x10^3$
		0.5	2.6	7,8	$2,6x10^3$	$3,2x10^3$
		0.2	6.2	5,8	$57,1x10^3$	$2,5x10^3$
	250	0.8	3	16,7	516	$12,8x10^3$
		0.5	4.3	9,2	$28x10^3$	$8,8x10^3$
		0.2	10.1	7,6	$181x10^3$	$5,4x10^3$
	500	0.8	4.7	37,2	$117,5x10^3$	$47,6x10^3$
		0.5	7.1	41,6	$170,9x10^3$	$38,4x10^3$
		0.2	16.6	39	$241,8x10^3$	$28,2x10^3$
	625	0.8	5.4	41,25	$51,9x10^3$	$81,9x10^3$
		0.5	8.3	51	$222,2x10^3$	$72,7x10^3$
		0.2	19.1	45	$297,8x10^3$	$84,3x10^3$
Total			69.8	315,55	$1371,5x10^3$	$390,3x10^3$

6. Análise

Analisando os resultados obtidos descritos nas tabelas acima, notamos que algoritmo implementado ficou um tanto quanto distante de obter os resultados semelhantes aos obtidos por [Consoli et al. 2009], tratando-se de um algoritmo genético que para cada grafo gera dois indivíduos passíveis de solução, porém ainda estão com um número de rótulos relativamente grandes, devido a geração de rótulos randomicamente, ou seja a cada no execução, este número de rótulos é diferente, alguns pontos desta geração de indivíduos podem serem otimizados com o intuito de minimizar ainda mais número de rótulos que constituem uma árvore geradora de rótulos mínimos.

7. Considerações Finais

Neste trabalho após compreendermos sobre algoritmos genéticos utilizados para solucionar o problema da Árvore Geradora de Rótulos Mínimos. Estabeleceu como objetivo a implementação de uma nova versão deste algoritmo afim de comparar os resultados obtidos como os resultados obtidos por [Consoli et al. 2009], os resultados obtidos foram avaliados pelo seu tempo de execução médio para um conjunto de grafos e pela média da função objetiva. Os resultados como visto nas tabelas (1,2,3,4) apresentou grande variação, em alguns casos mostrou-se melhor que a solução comparada, porém em sua maioria a solução apresentada não foi suficientemente eficiente quanto a solução [Consoli et al. 2009].

Deste modo, devemos admitir que para problemas envolvendo está complexidade, deve-se adotar resultados não tão quanto ótimos quando comparado com a literatura, sendo assim tal qual a solução alcançada por este trabalho seria de suficiente aprovação dado que para um grafo extenso, sua árvore geradora mínima resultou em uma quantidade consideravélmente reduzida de rótulos.

8. Referências

Consoli, S., Darby-Dowman, K., Mladenovic, N., Moreno-Pèrez, J.A. Greedy randomized adaptive search and variable neighbourhood search for the minimum labelling spanning tree problem. European Journal of Operational Research 196(2), 440-449, 2009.

Xiong, Y., Golden, B., Wasil, E. A One-Parameter Genetic Algorithm for the Minimum Labeling Spanning Tree Problem. IEEE Transactions on Evolutionary Computation, 9(1), 5560, 2005.

Cormen, T., Leiserson, C., Rivest, R., and Stein, C. (2001). Introduction To Algorithms. MIT Press.