Árvore Geradora de Rótulos Mínimos

Guilherme Castro Diniz[1], **Mariane Affonso Medeiros**[2]

[1]Universidade Tecnológica Federal do Paraná (UTFPR)
Campo Mourão, PR, 87301-899

[2]Departamento Acadêmico de Computação (DACOM)

{guizao.seri,marianeaffonsomedeiros}@gmail.com

Resumo. *Neste artigo estudamos o problema da árvore geradora de rótulos mínimos (MLSP - Minimum labelling spanning tree). A ideia deste problema é construir uma árvore geradora utilizando o mínimo possível de labels (pesos das arestas) diferentes. Neste artigo propomos o algoritmo de vizinhanças variáveis (variable neighbourhood search) para resolver o problema MLSP. Os resultados obtidos pelo algoritmo proposto neste trabalho serão comparados com os rresultados do trabalho [Consoli et al. 2009]. .*

1. Introducão

O problema da árvore geradora de peso mínimo é um dos problemas mais clássicos da teoria dos grafos. Sua ideia consiste em encontrar uma árvore que passe por todos os vértices do grafo, utilizando as arestas mais adequadas, de maneira que ao fim tenha o menor peso total de arestas possível. Este problema, gerou inúmeras ramificações de si, como a árvore geradora de diâmetro mínimo, árvore geradora de minimo e máximo, árvore geradora de rótulos mínimos, entre outros. São inúmeras as aplicações deste problema na prática, como por exemplo, em um projeto de circuito eletrônico, muitas vezes é necessário que os pinos de vários componentes se tornem eletricamente equivalentes, o que é alcançado ligando-os uns aos outros. Para interconectar um conjunto de *n* pinos, podemos usar um arranjo de *n - 1* fios, cada qual conectando dois pinos. De todos os arranjos possíveis, o que utiliza a mínima quantidade de fio é normalmente a mais desejável [Cormen et al. 2001].

Neste artigo estudamos e definimos uma das ramificações do problema da árvore geradora de peso mínimo, a árvore geradora de rótulos mínimos (MLSP). A MLSP consiste em encontrar uma árvore geradora que percorra todos os vértices passando por o minimo possivel de labels (pesos das arestas) diferentes. A principal diferença entre a árvore geradora de pesos mínimos e a árvore geradora de rótulos mínimos é que, o foco da primeira é encontrar o menor caminho possivel (menor soma possivel dos pesos das arestas) que passe por todos os vértices. Enquando o MLSP busca encontrar um caminho que passe pela menor quantidade de labels (pesos de arestas) diferentes contemplando todos os vértices do grafo. No segundo problema, é preferivel percorrer um caminho maior, mas que passe pelo minimo de labels diferentes do que percorrer um caminho menor mas que passe por várias labels diferentes. Lembamos que durante este trabalho quando nos referirmos a labels, estamos nos referindo ao peso que a aresta possui.

MLSP é um problema classificado como NP-dificil [Chang and Shing-Jiuan 1997], em que dado um grafo ponderado e não direcionado, busca-se uma árvore geradora com o menor número de labels diferentes possíveis.

Este modelo representa muitos problemas reais de telecomunicação. Vamos considerar uma rede com muitos tipos de midias de comunicação diferentes, como fibra optica, cabo coaxial, microonda e linha de telefone. Um nó de comunicação pode se comunicar com diferentes nós, escolhendo tipos diferentes de midia de comunicação. Dado um conjunto de nós de uma rede de comunicação, o problema consiste em encontrar um árvore geradora(uma rede de comunicação conectada) que utilize a menor quantidade de diferentes tipos de comunicação possível. Esta árvore geradora irá reduzir o custo de construção e a complexidade da rede. Este problema pode ser formulado como um problema de grafos. Dado um grafo G = (V, E) e uma função label L(e) para todas as arestas $e \in E$ onde os vértices são representados pelos nós de comunicação. As arestas e suas labels são representam as linhas de comunicação e seus tipos de midia de comunicação. O objetivo é encontrar uma árvore geradora que utilize a menor quantidade de dierentes tipos de labels [Chang and Shing-Jiuan 1997].

O problema da árvore geradora de rótulos mínimos é definida formalmente da seguinte forma:

> *MLSP :* Dado um grafo ponderado e não direcionado $G = (V, E, L)$, onde V é o conjunto de n vértices, E é o conjunto de m arestas, e L é o conjunto de l labels, encontre uma árvore geradora T de G de forma que $|L_T|$ seja minimo, onde L_T é o conjunto de labels usadas em T.

O restante do artigo é organizado em revisão bibliográfica, onde fazemos uma revisão da literatura do problema. ...

2. Revisão Bibliográfica

Inspirados pelos problemas de design de telecomunicação de redes, Chang e Leu proporam em 1997 o problema da árvore geradora de rótulos mínimos. Eles provaram que o MLSP é um problema NP-dificil e propuseram uma heurística de tempo polinomial para resolver o problema, a heuristica de cobertura máxima de vértices (*maximum vertex covering algorithm* MVCA). Este algoritmo tenta construir uma árvore geradora gradualmente, a cada tempo selecionando uma label de forma que as arestas que possuem esta label selecionada cubram a maior quantidade possível de vértices que ainda não foram descobertos [Chang and Shing-Jiuan 1997]. No entando, com este algoritmo há a possibilidade de, embora todos os vértices do grafo sejam visitados, ele não seja um grafo conectado e então o algoritmo falha [Consoli et al. 2009], pois busca-se uma árvore, e uma árvore precisa de todos os nós conectados .

Por este motivo, em 1998 Krumke e Wirth revisaram o algoritmo proposto por Chang e Leu, o MVCA, e fizeram as correção necessárias para que este problema não ocorrece. A versão do MVCA corrigida funciona da seguinte maneira, inicia com um grafo vazio, e adiciona aleatoriamente uma label a um conjunto de labels de forma que minimize o número de componentes conectados. O procedimento continua até que haja apenas uma componente conectada, isto é, quando apenas um subgrafo conectado é obtido.

Em 2003 Brüggemann-etal abordou o problema sob uma perspectiva diferente,

utilizando uma busca local para obter uma árvore geradora, e provou que considerando um caso especial do problema, quando uma determinada label apareça no máximo $r = 2$ vezes no grafo em questão, o problema pode ser resolvido em tempo polinomial. Em 2005 Xiong-etal propôs um algoritmo genético para resolver o problema da MLST, que superou o tempo de execução do MVCA em vários casos.

Ao decorrer dos anos, vários outros algoritmos foram propostos para resolver o problema da árvore geradora de rótulos mínimos. Algoritmos como por exemplo, Método Piloto (*Pilot Method*) prosposto em 1999 por Duin-Voß. Este método utiliza uma heuristica básica como processo de candidatura, e então realiza diversas iterações do processo de candidatura em relação a uma já fornecida solução *master*. As iterações da heurística básica são realizadas até que todas as possíveis escolhas locais com respeito a solução *master* sejam avaliadas. Ao fim de todas as iterações, uma nova solução *master* é obtida estendendo a solução atual com as alterações correspondentes para obter o melhor resultado [Consoli et al. 2009]. Ou seja, a cada nova iteração é escolhido uma nova solução *master* melhor que a anterior.

Outro algoritmo proposto, é o da Busca Gulosa Adaptativa e Randômica (*Greendy Randomized Adaptative Search* - GRASP) apresentado em 1995 por Feo-Resende. O GRASP considera uma solução inicial e efetua uma busca local para melhorar a qualidade da solução inicialmente proposta. Esta solução inicial é gerada por um procedimento guloso, aleatório e randomizado.

Por fim, temos o algoritmo de Busca por Vizinhanças Variáveis (*Variable Neighbourhood Search* - VNS). Esta meta-heurística é baseada na mudança dinâmica dos vizinhos durante o processo de busca, isto é, a cada iteração uma solução aperfeiçoada *x'* na vizinhança *N(x)* da solução atual *x* é obtida, até que mais nehuma solução melhor seja encontrada [Mladenović and Hansen 1997]. Foi prosposta em 1997 por Mladenović-Hansen.

3. Busca por Vizinhanças Variáveis

A VNS foi proposta em 1997 por Mladenović-Hansen, sua prosposta é de uma metaheuristica simples e efetiva que obtém através de um processo sistemático de mudança de vizinhos dentro de uma busca local os resultados almejados. A VNS não segue uma trajetória, mas sim, explora cada vez mais a vizinhança da solução atual em busca de uma solução melhor e muda sua melhor solução apenas quando a solução do vizinho é melhor que a solução atual.

As heurisiticas *Variable Neighbourhood Search* e *Greendy Randomized Adaptative Search* obtiveram os melhores resultados baseados nos teste executados em [Consoli et al. 2009]. Como neste artigo iremos utilizar a mesma base da dados para testes usado por Consoli-etal, optamos por implementar a metaheurisca VNS. A implementação para o problema MLST utilisando VNS esta descrito no Algoritmo 1.

Nesse algoritmo, parte-se de uma solução inicial qualquer e a cada iteração seleciona-se um vizinho C' aleatório, dentro da vizinhança $N_k(s)$ da solução C corrente. Esse vizinho C' então, é submetido a uma busca local. Se a solução ótima local, C'', for melhor que a solução C atual, a busca continua de C'' recomeçando da primeira estrutura de vizinhança $N_1(s)$. Caso contrário, continua-se a busca a partir da próxima estrutura

de vizinhança $N_{(k+1)}(s)$. Este procedimento é encerrado quando uma condição de parada for atingida, tal como o tempo máximo permitido de CPU, o número máximo de iterações ou número máximo de iterações consecutivas entre dois melhoramentos.

Algoritmo 1 VNS

```
1:  function VNS(G)
2:      C = Gerar-Solucao-Inicial-Aleatoriamente()
3:      while (não atingir as condições) do
4:          k = 1 , k_max = (|C| + |C|/3)
5:          while k < k_max do
6:              C' = ShakingPhase(k)
7:              BuscaLocal(C')
8:              if |C'| < |C| then
9:                  C = C'
10:                 k = 1 , k_max = (|C| + |C|/3)
11:             else
12:                 k++
13:     Atualiza o grafo H de acordo com as labels que estão no conjunto C
14:     return T
```

O VNS funciona da seguinte forma, dado um grafo ponderado e não direcionado, $G(V, E, L)$, com n vértices, m aresta e l labels, busca-se uma árvore geradora com mínimo possível de labels diferentes. O algoritmo começa com algumas inicializações que são, C conjunto de labels utilizadas na solução atual, $H(V, E(C))$, constitui-se apenas pelas labels contidas em C, C' conjunto de labels usadas para comparar com C, $H'(V, E(C'))$, grafo formado a partir das labels em C', e $CommpC$ quantidade de componentes conexas de C'. O conjunto C é composto da seguinte forma, em cada índice correspondente a label usada no conjunto, coloca-se 1 na posição da label. Por exemplo, se a label 17, faz parte do conjunto C, no índice 17 de C haverá o número 1 indicando que a label está no conjunto.

Após as inicializações, o algoritmo inicia com uma solução inicial escolhida aleatoriamente. Para obter esta solução aleatória, sorteamos um vértice aleatório x dentro do intervalo n, e pegamos os vértices adjacentes de x para formar o grafo inicial, o conjunto C então é atualizado, junto com o grafo $H(V, E(C))$. Em seguida é calculado o valor de k e k_{max}, onde para calcular k_{max} considera-se a quantidade de labels presentes em C.

O valor de k_{max} é um importante parâmetro para obter um bom balanço entre recurso de capacitação e diversificação. Escolhendo um valor pequeno para k_{max}, produz um alto recurso de intensificação e um baixo recurso de diversificação, resultando em um algoritmo rápido, porém com uma alta probabilidade de cair em mínimos locais. Já, escolhendo um alto valor para k_{max}, produz um baixo recurso de intensificação e um alto recurso de diversificação, resultando em um algoritmo lento, porém com baixa probabilidade de cair em mínimos locais [Consoli et al. 2009]. Baseado nisto, utilizamos a equação $k_{max} = (|C| + |C|/3)$, proposta por [Consoli et al. 2009] para obter um bom valor de k_{max}.

A próximo passo no algoritmo é o método *ShakingPhase*, que muda a estrutura

dos vizinhos quando a busca local fica presa em uma minimo local [Consoli et al. 2009]. Este método é implementado da seguinte forma, coloca em C' o conteúdo de C. Percorre os k vizinhos de C', sorteando um número rnd entre 0 e 10. Se rnd for menor ou igual a 5, deleta uma label aleatoriamente de C', caso contrário, verifica quais as labels ainda não contidas na solução atual C, e adiciona em C' uma label ainda não presente em C. Atualiza $H'(V, E(C))$ e $CompC$.

Em seguida, inicia-se a busca local em C'. Enquanto a quantidade de componentes conexas de C' ($CompC$) for maior que 1, considera-se um conjunto S de labels não utilizadas na solução que minimize a quantidade de componentes conexas de C', seleciona e adiciona em C' uma label aleatória x de S. Então atualiza H' e $CompC$. Quando se obter C' com uma componente apenas, inicia-se um for sobre as labels usadas em C', excluindo cada label i de C', e verificando se a $CompC$ aumenta ou diminui, caso aumente, adiciona novamente a label escolhida a C'.

Com C' já definido pela busca local, verifica se a solução proposta por C', é melhor ou pior que a solução atual C, se for melhor troca C por C', caso contrario continua buscando uma solução mais plausível na vizinhança de C, aumentando o número de vizinhos (k). Neste ponto do algoritmo (linha 8 do Algoritmo 1), fizemos uma modificação quanto ao algoritmo de [Consoli et al. 2009]. Em vez de verificarmos se a quantidade de labels de C' é menor que C, verificamos se a quantidade de componentes conexas geradas pelo conjunto C' é menor que a quantidade de componentes conexas geradas pelo conjunto C. E acreditamos que devido a esta modificação, nosso algoritmo apresenta resultados diferentes do algoritmo de [Consoli et al. 2009].

4. Resultados Computacionais

Nesta seção comparamos a qualidade da solução e o tempo computacional do algoritmo de VNS implementado neste trabalho, com o algoritmo VNS implementado por [Consoli et al. 2009]. A base de dados usada para testes usa várias instâncias do problema MLSP, para que se possa avaliar como o algoritmo é influenciado pelos parâmetros de arestas, número de vértices e labels.

A base de testes utilizada neste artigo foi proposta por [Cer 2005]. Os experimentos foram feitos em um computador Intel(R) Core(TM) i3-2328M CPU @ 2.20GHz com 4GB de RAM. A base de testes, considera diferentes conjuntos de dados, cada um contendo 10 instancias do problema MLSP.

Média da função Objetivo - Grupo 1 | Tempo computacional (segundos)

Parâmetros		Consoli-etal:2009	Diniz-Medeiros:2015	Consoli-etal:2009	Diniz-Medeiros:2015
n	d	VNS	VNS	VNS	VNS
20	0.8	2.4	3.4	0.0	0.8
	0.5	3.1	4.4	0.0	0.1
	0.2	6.7	12.3	1.6	0.4
30	0.8	2.8	3.8	0.0	0.2
	0.5	3.7	5.3	0.0	0.5
	0.2	7.4	9.5	5.2	2.9
40	0.8	2.9	4.0	0.0	0.9
	0.5	3.7	5.9	3.1	1.3
	0.2	7.4	10.7	9.6	7.7
50	0.8	3.0	4.3	0.0	1.3
	0.5	4.0	6.2	4.1	3.2
	0.2	8.6	15.7	11.9	26.4
Total		**55.7**	**85.5**	**35.5**	**45.7**

Média da função Objetivo - Grupo 2, n = 100 | Tempo computacional (segundos)

Parâmetros			Consoli-etal:2009	Diniz-Medeiros:2015	Consoli-etal:2009	Diniz-Medeiros:2015
n	l	d	VNS	VNS	VNS	VNS
100	25	0.8	1.8	2.0	0.0	0.2
		0.5	2.0	2.5	0.0	0.5
		0.2	4.5	6.6	3.1	5.3
	50	0.8	2.0	2.9	7.7	1.7
		0.5	3.0	4.2	42.4	4.7
		0.2	6.7	9.5	49.7	49.1
	100	0.8	3.0	4.6	215.0	11.9
		0.5	4.7	7.4	114.7	36.6
		0.2	9.7	NF	414.8	NF
	125	0.8	4.0	6.2	10.1	22.6
		0.5	5.2	7.9	551.1	26.3
		0.2	11.0	NF	420.4	NF
	Total		**57.6**	**53.8**	**1.8 x** 10^3	**158.9**

Média da função Objetivo - Grupo 2, n = 200 | Tempo computacional (segundos)

Parâmetros			Consoli-etal:2009	Diniz-Medeiros:2015	Consoli-etal:2009	Diniz-Medeiros:2015
n	l	d	VNS	VNS	VNS	VNS
200	50	0.8	2.0	2.1	0.0	1.4
		0.5	2.2	3.1	17.2	6.6
		0.2	5.2	7.3	241.3	10.5
	100	0.8	2.6	3.4	123.2	14.9
		0.5	3.4	5.1	151.1	0.4
		0.2	7.9	NF	1.7 x 10^3	NF
	200	0.8	4.0	5.4	32.0	226.6
		0.5	5.4	NF	971.9	NF
		0.2	12.0	NF	12.8 x 10^3	NF
	250	0.8	4.0	6.2	1.1 x 10^3	384.5
		0.5	6.3	NF	3.4 x 10^3	NF
		0.2	13.9	NF	3.2 x 10^3	NF
	Total		**68.9**	**32.6**	**23.7 x** 10^3	**644.9**

Para testes consideramos dois grupos de instancias diferentes, que incluem números de vértices e quantidade de labels de 20 até 500. O número de arestas é determinada pela densidade do grafo, trabalhamos com três densidades: 0.8 grafo denso, 0.5 grafo meio denso, 0.2 grafo pouco denso. Consideramos que um algoritmo é pior, ou

menos eficiente se um possui tempo computacional ou a média da função objetivo muito mais alta que o outro.

O grupo 1, possui instâncias pequenas do problema, com o número de vértices igual ao número de labels. Já o grupo 2 possui número de vértices de labels desiguais.

Analisando as tabelas, podemos verificar que o algoritmo implementado para este trabalho, não obteve resultados muito próximos, do algoritmo implementado por [Consoli et al. 2009]. No algoritmo deste trabalho, existem alguns gargalos de programação, que podem ser otimizados, levando a se obter melhores resultados.

5. Conclusão

Neste artigo nós estudamos a meta-heurística de Vizinhanças Variáveis para o problema da Arvore Geradora de Rótulos Mínimos. Nosso objetivo foi implementar o VNS e comparar os resultados da nossa implementação com os resultados da implementação de [Consoli et al. 2009]. Para esta comparação realizamos experimentos computacionais com a mesma base de testes usada por [Consoli et al. 2009]. As comparações foram baseadas no tempo de execução do algoritmo e no valor da função objetivo, que constitui da média de labels por solução da árvore geradora.

Os resultados apontaram que a solução proposta neste artigo não foi tão eficiente quando a solução de Consoli-etal:2009.

Referências

(2005). Metaheuristics comparison for the minimum labelling spanning tree problem. In Golden, B., Raghavan, S., and Wasil, E., editors, *The Next Wave in Computing, Optimization, and Decision Technologies*, volume 29. Springer US.

Chang, R.-S. and Shing-Jiuan, L. (1997). The minimum labeling spanning trees. *Information Processing Letters*, 63(5):277 – 282.

Consoli, S., Darby-Dowman, K., Mladenović, N., and Pérez, J. M. (2009). Greedy randomized adaptive search and variable neighbourhood search for the minimum labelling spanning tree problem. *European Journal of Operational Research*, 196(2):440 – 449.

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

Mladenović, N. and Hansen, P. (1997). Variable neighborhood search. *Computers Operations Research*, 24(11):1097 – 1100.