Estudo do Problema de Árvores Geradoras de Rótulos Mínimos*

[Implementação e análise de instâncias usando JPSO] †

Paulo Batista da Costa
Universidade Tecnológica Federal do Paraná
2015 Campo Mourão
Campo Mourão, Brasil
pauloc@alunos.utfpr.edu.br

Felipe Veiga Ramos
Universidade Tecnológica Federal do Paraná
2015 Campo Mourão
Peabiru, Brasil
fveigaramos@gmail.com

ABSTRACT

Um dos assuntos fundamentais da área da Teoria dos grafos são as árvores geradoras de rótulos mínimos. Elas tem por objetivo encontrar o menor número possível de rótulos de um grafo não dirigido conexo que cubram todos os vértices possíveis. Os problemas de árvores geradoras de rótulos mínimos ou PAGRM são de grande importância para áreas que projetam redes de comunicação, elétrica, transporte dentre outras. Desse modo, para estudar e ampliar os conhecimentos na área, foi proposto este trabalho. Ele consiste na implementação de um algoritmo – no caso o JPSO – e na sua comparação com demais algoritmos propostos pela literatura de acordo com uma base comum. Assim, este trabalho é uma oportunidade de compreender e estudar na prática conteúdos que envolvem teoria dos grafos.

Keywords

PAGRM; Rótulos mínimos; Teoria dos Grafos; JPSO;

1. INTRODUÇÃO

Um dos assuntos fundamentais da área de Teoria dos grafos são as árvores geradoras de rótulos mínimos. Esta pode ser entendida como uma árvore geradora obtida através de um grafo conectado não direcionado . Os Problemas das árvores geradoras de rótulos mínimos (PAGRM) podem ser formalmente definidos como: Dado um grafo rotulado não dirigido $G = (V, E, L)$ no qual V é o conjunto n de vértices, E é o conjunto de m arestas e L é o conjunto de l rótulos, encontrar uma árvore geradora T de G tal que $|L_T|$ é minimizado. Define-se $|L_T|$ como o conjunto de diferentes rótulos das arestas em uma árvore geradora T. [7]

Tal é utilizado para representar situações presentes no mundo real como redes de comunicação, elétrica, transporte dentre outros. Sua compreensão e estudo é fundamental para a elaboração de soluções que visam reduzir custos e ampliar eficiência. Em outras palavras, o objetivo do estudo e aplicações de algoritmos para resolver o PAGRM é gerar - a partir de um grafo não dirigido - uma árvore que contenha o menor número possível de rótulos e ao mesmo tempo cobrindo todos os seus vértices.

Este trabalho tem por finalidade analisar o conjunto de instâncias – o mesmo conjunto utilizado em [3] – através da implementação (realizada na linguagem de programação JAVA) e execução do algoritmo JPSO (*Jumping Particle Swarm Optimization*) [4], e dessa forma estabelecer uma comparação entre os resultados obtidos desta aplicação e aqueles provindos das heurísticas e pseudo-código apresentados em [3]. Para mais informações sobre implementações e pseudocódigos, o material está disponível neste link .

Tal comparação levou em consideração fatores como a complexidade dos grafos (número de vértices e rótulos) , valor médio da função objetivo e o tempo computacional. Dessa forma, os resultados são comparados da mesma forma como reportado em [3].

O restante deste trabalho está organizado em seções da seguinte forma: Na seção 2 é realizada uma revisão literária, na 3 é apresentada a proposta de algoritmo exato para solução do PAGRM (no caso o JPSO), em 3.2 os resultados da implementação são mostrados, em 4 é explicitada uma comparação entre os resultados da literatura e os obtidos neste trabalho e por último são apresentadas as conclusões em 5.

2. REVISÃO LITERÁRIA

Algoritmos propostos para cobrir o máximo número de vértices, do inglês *Maximum vertez covering algorithm – MVCA*, tem sua origem da necessidade de criar sistemas em rede com menor custo e maior eficiência. Em 1997, Chang and Leu (1997) [2] introduziram tal assunto e hoje este é conhecido como problema de árvore geradora de rótulos mínimos (PAGRM). Eles introduziram discussões sobre PAGRM expõem sua primeira solução e provam ser um problema NP completo [6], ou seja, que é um problema cuja resolução não pode ser determinada em um tempo polinomial. Eles propõem inicialmente a seguinte heurística: Começa-se com um

*(Produces the permission block, and copyright information). For use SIG-ALTERNATE.CLS. Supported by ACM.

†A full version of this paper is available as *Author's Guide to Preparing ACM SIG Proceedings Using LaTeX2$_\epsilon$ and BibTeX* at `www.acm.org/eaddress.htm`

DOI: 10.1145/1235

grafo vazio e adicione sempre arestas que cubram o maior número de vértices não visitados. Tal solução gera uma árvore geradora de rótulos mínimos. Entretanto, a solução inicial apresentou falhas – gerando grafos não conexos – e posteriormente sofreu modificações. No ano seguinte, Krumke e Wirth [5] propuseram a correção que gerava apenas grafos conexos que minizava o nímero de componentes. Essa correção alcançou complexidade $(1 + 2\log n)$. Entretanto,o trabalho mais recente [8] melhorou tal complexidade para $(1 + \log(n + 1))$. Em [9], os autores utilizaram outra aproximação para resolução: Algoritmos Genéticos que utilizam conceitos provindos da biologia para resolver PAGRM. Em seguida, Cerulli et al em [1] aplicaram o *Pilot Method* e tal foi testado em conjuntos de instâncias obtendo a melhor resposta na maioria dos casos. Por sua vez, em 2006 Xiong et al em [10] implementou versões simplificadas do *Pilot Method* com modificações de algoritmos genéticos (MGA) e obteve uma performance melhor quanto a obtenção de respostas (capacidade de diversificação).

Em trabalho datado mais recente [4] – em 2009 – há a apresentação de Consoli et Al na qual a solução dada ao PGRM é abordada através de um algoritmo baseado em comportamento de seres presentes em ambiente natural. Especificamente esse trabalho trata de derivações do PSO (*Particle Swarm Optimization*) para resolver problemas de natureza discreta, como é o caso do problema em questão. A ideia de sua proposta é que as partículas (soluções) iterem de forma a combinarem entre si a cada iteração e aproximarem continuamente da resposta tida como a melhor possível. Assim, partículas tendem a convergir a melhor resposta. Tal heurística pode ser analogamente dita como uma estratégia de inteligência coletiva e deriva conceitos presentes na área de inteligência artificial. Esta solução foi a eleita para caso de estudo, pelo fato de ser fácil quanto a sua implementação.

3. PROPOSTA DE ALGORITMO – O JPSO

3.1 Heurística - A ideia do JPSO

O PSO (Do inglês *Particle Swarm Optmization*) é um algoritmo que simula o comportamento de revoada de indivíduos na natureza. Cada solução do problema é tido como um indivíduo que pode voar dentro do espaço de soluções. A melhoria das posições, ou seja, as melhores respostas são obtidas através da continua movimentação das partículas que constituem tal revoada. É um algoritmo proposto por [4] que utiliza o conceito de inteligência coletiva, no qual as partículas tendem a convergir rumo a uma solução que esteja mais próxima do ótimo desejado e é capaz de auxiliar na busca por respostas em problemas de PAGRM. Entretanto, esse algoritmo é formulado para resolver problemas contínuos, o que contradiz o uso para tal. Isto pois, um PAGRM constitui um cenário de resolução discreta.

Portanto, para a utilização da ideia provinda do PSO é necessário que o algoritmo foque problemas discretos. Assim, uma adaptação é utilizada: O JPSO (*Jumping Particle Swarm Optimization*).O JPSO é um DPSO (Do inglês,*Discrete Particle Swarm Optimization*) [] , ou seja, possui a mesma essência da heurística aplicada pelo PSO com a diferença de ser voltado para problemas discretos. A ideia é que a cada iteração as possíveis resposta estejam mais próximas do que seria considerado como ótimo.

3.2 Explicações da implementação do JPSO

A implementação para este trabalho está disposta na linguagem de programação JAVA e está de acordo com o pseudo-código descrito em [4], ou seja, o código gerado neste trabalho é representado pelo que está presente naquele trabalho. Para mais detalhes o artigo, bem como, toda a estrutura (artigos, referências para implementação , pseudo-código são encontradas na neste – link– referente ao controle de versionamento utilizado. Assim, a entrada para o JPSO é um grafo não dirigido e conexo $G = (V, E, L)$, com n vértices, m arestas, l rótulos e $Q \subseteq V$ nós. Além disto, tem como saída uma árvore geradora a qual objetiva ser mínima, ou seja, conter o menor número de rótulos possível e ao mesmo tempo cobrir todos os vértices do grafo.

Inicialmente foram carregadas as instâncias presentes nos arquivos disponibilizados em ambiente virtual referente aos grafos sobre os quais a análise foi aplicada e que são as mesmas utilizadas por [3]. Estas foram carregadas em estruturas as quais retornam um grafo não dirigido e conexo provindo da base de dados.

Em seguida, a partir de cada grafo abstraído da base de dados foram geradas 100 partículas. Estas partículas são a representação das possíveis soluções e são inicialmente formadas de forma randômica: Para cada partícula é induzido um grafo aleatoriamente a partir dos rótulos do grafo original. A partir desse momento iniciam-se as iterações que buscam a melhor resposta, elas são caracterizadas pelos seguintes passos: 1. atualiza-se a melhor resposta (componente conexa com menor número de rótulos) para cada partícula , 2. atualiza-se a melhor resposta global já obtida, 3. Para cada partícula há a tentativa de reduzir o número de rótulos mantendo as propriedades necessárias, 4. tentativa de combinar duas soluções gerando um grafo com menor número de rótulos.

As tentativas de modificações de cada partículas são randômicas. Além disso é importante ressaltar que a cada iteração as melhores respostas possíveis nunca apresentarão maior número de rótulos do que na iteração anterior, pois toda vez que ocorrer aumento no número de rótulos há o descarte da solução mantendo a anterior. Outro ponto importante é o critério de parada para o algoritmo:É finalizado a partir do momento em que se completam 100 iterações sem modificação no melhor resultado global.

Assim, o JPSO deve retornar a melhor resposta obtida durante sua execução. Apesar de garantir o retorno da melhor solução, não garante que esta seja a solução ótima. Em outras palavras, não retorna necessariamente a arvore com menor rótulos possível de ser formada. Isto devido ao fato de basear-se na aleatoriedade inicial das soluções, o que também implica que execuções distintas sobre a mesma base de dados retorne uma soluções diferentes.

Resultado da execução - O Algoritmo e sua Ação sobre as Instâncias

A aplicação do JPSO para PAGRM (que é um problema NP completo [6]) sobre a base disponibilizada gerou dados de saída que foram organizados em um arquivo texto em formato *txt*. Eles podem ser verificados de forma organizada nas tabelas 1 2 3 4 5 6 7 e 8 e posteriormente são comparados aos resultados dos algoritmos exibidos em [3], estes sobre a mesma base de dados.

N	L	D	F. O. JPSO	F. O. VNS
20	20	0.8	2.4	2.4
		0.5	3.1	3.1
		0.2	6.8	6.7
30	30	0.8	2.8	2.8
		0.5	3.7	3.7
		0.2	7.5	7.4
40	40	0.8	2.9	2.9
		0.5	3.7	3.7
		0.2	7.7	7.4
50	50	0.8	3.0	3
		0.5	4.5	4
		0.2	8.8	8.6
		Total	56.9	55.7

Table 1: N é o número de nós, L é o número de rótulos e D é a densidade. A tabela mostra a função objetido do PSO (F. O. JPSO) e a do VNS (F. O. VNS)

N	L	D	Tempo JPSO	Tempo VNS
20	20	0.8	1002	0
		0.5	1.543	0
		0.2	2.156	1,6
30	30	0.8	1.890	0
		0.5	4.291	0
		0.2	6.191	5.2
40	40	0.8	3.021	0
		0.5	8.018	3.100
		0.2	14.583	9.600
50	50	0.8	6.729	0
		0.5	15.001	4.100
		0.2	3.113	11.900
		Total	67.538	35.500

Table 2: N é o número de nós, L é o número de rótulos e D é a densidade. A tabela mostra o tempo de execução de cada – JPSO e VNS – em milissegundos

N	L	D	F. O. JPSO	F.O. VNS
100	25	0.8	1.8	1.8
		0.5	2.0	2.0
		0.2	4.6	4.5
	50	0.8	2.0	2
		0.5	3.0	3
		0.2	6.9	6.7
	100	0.8	3.8	3
		0.5	5.3	4.7
		0.2	11.3	9.7
	125	0.8	4.1	4
		0.5	6.2	5.2
		0.2	13.1	11
	Total –		64.1	57.6

Table 3: N é o número de nós, L é o número de rótulos e D é a densidade. A tabela mostra a função objetido do PSO (F. O. JPSO) e a do VNS (F. O. VNS)

N	L	D	Tempo JPSO	T. VNS
100	25	0.8	11440	0
		0.5	16.664	0
		0.2	46.836	3.100
	50	0.8	31.146	7.700
		0.5	34.667	4.240
		0.2	107.502	4.970
	100	0.8	37.138	215.000
		0.5	96.218	114.700
		0.2	192.166	414.800
	125	0.8	60.090	10.100
		0.5	113.139	551.100
		0.2	239.019	420.400
	Total –		986.025	1800.000

Table 4: N é o número de nós, L é o número de rótulos e D é a densidade. A tabela mostra o tempo de execução de cada – JPSO e VNS – em milissegundos

N	L	D	F. O. JPSO	F. O. VNS
200	50	0.8	2.0	2
		0.5	2.3	2.2
		0.2	5.7	5.2
	100	0.8	2.9	2.6
		0.5	4.1	3.4
		0.2	9.4	7.9
	200	0.8	4.6	4
		0.5	6.8	5.4
		0.2	15.3	12
	250	0.8	5.1	4
		0.5	8.0	6.3
		0.2	17.9	13.9
		Total –	84.1	68.9

Table 5: N é o número de nós, L é o número de rótulos e D é a densidade. A tabela mostra a função objetido do PSO (F. O. JPSO) e a do VNS (F. O. VNS)

N	L	D	Tempo JPSO	Tempo VNS
200	50	0.8	53185	0
		0.5	130.555	17.200
		0.2	306.403	241.300
	100	0.8	99.256	123.200
		0.5	157.123	151.100
		0.2	574.387	1700.000
	200	0.8	254.051	32.000
		0.5	655.515	971.900
		0.2	1184.675	12800.000
	250	0.8	566.244	1100.000
		0.5	933.404	3400.000
		0.2	1441.719	3200.000
		Total –	6856.514	23700.000

Table 6: N é o número de nós, L é o número de rótulos e D é a densidade. A tabela mostra o tempo de execução de cada – JPSO e VNS – em milissegundos

N	L	D	F. O. JPSO	F. O. VNS
500	125	0.8	2.0	2.0
		0.5	3.1	2.6
		0.2	7.3	6.2
	250	0.8	3.1	3
		0.5	4.9	4.1
		0.2	12.9	9.9
	500	0.8	5.8	4.7
		0.5	8.7	6.5
		0.2	21.3	15.8
	625	0.8	6.7	5.1
		0.5	10.5	7.9
		0.2	25.4	18.3
		Total –	111.7	86.1

Table 7: N é o número de nós, L é o número de rótulos e D é a densidade. A tabela mostra a função objetido do PSO (F. O. JPSO) e a do VNS (F. O. VNS)

N	L	D	Tempo JPSO	Tempo VNS
500	125	0.8	580854	17.100
		0.5	1535.064	1100.000
		0.2	4508.425	3900.000
	250	0.8	2351.650	14.200
		0.5	6289.072	84000.000
		0.2	9052.493	5100.000
	500	0.8	6751.334	22300.000
		0.5	16560.601	32300.000
		0.2	16661.087	139700.000
	625	0.8	18957.465	16100.000
		0.5	17878.234	44700.000
		0.2	18310.678	155500.000
		Total –	119436.957	504900.000

Table 8: N é o número de nós, L é o número de rótulos e D é a densidade. A tabela mostra o tempo de execução de cada – JPSO e VNS – em milissegundos

4. COMPARAÇÃO DE RESULTADOS

Dentre as tarefas necessárias para a realização deste trabalho está a realização da comparação entre os algoritmos presentes em [3] e o escolhido para a realização de implementação. Em [3] os algoritmos discutidos e executados sobre a mesma base de dados utilizada neste trabalho são: PILOT, MGA, GRASP e VNS. Assim, como os algoritmos são aplicados em uma base comum a deste trabalho, é possível estabelecer uma comparação plausível.

Tal comparação está divida em 3 grupos, baseados nas características de suas instâncias: grupo 1 é referente a instâncias pequenas que possuem número de vértices iguais ao número de rótulos, o grupo 2 considera instâncias maiores com o número de rótulos definidos por $0,25 * n$, onde n é o número de vértices e os rótulos varia de 25 a 125. Por último, o grupo 3 são as maiores instâncias presentes na base, pois é composta por parâmetros como número de vértices igual a 200 e número de rótulos que variam de 50 a 250 como visto na tabela 3 e número de vértices igual a 500 com número de rótulos variando de 125 a 625 como visto na tabela 4.

4.1 Comparações no grupo 1

No grupo 1 (120 instâncias), dentre os algoritmos apresentados em [3] os algoritmos MGA e PILOT são antagônicos: O segundo é mais rápido que o primeiro, entretanto provendo respostas piores. Enquanto isso, o segundo é mais demorado quanto a execução, mas gera soluções melhores. Para concluir se a resposta de uma heurística é melhor do que outra é necessário comparar suas funções objetivas, ou seja, a taxa referente a média de rótulos por instância. Por sua vez, GRASP e VNS são os mais rápidos e promovem a solução exata. Estes são excelente para essas instâncias. Comparando o JPSO implementado é perceptível que este é mais rápido que MGA (em função do tempo de execução) e mais lento que os demais. Além disso, o JPSO não promove a certeza de que a solução obtida é a exata (ótima).

O MGA é o mais lento por ser o que mais diversifica as possíveis soluções (no objetivo de encontrar a melhor), porém retornando uma resposta melhor que o PILOT que diversifica pouco, mas apresenta uma resposta pior. Assim, o JPSO é mais executa mais rápido do que o algoritmo mais lento apresentado em [3] , levando em considerações as instâncias do grupo 1 (cujas características podem ser vistas na tabela 2). Ao comparar o algoritmo implementado com o melhor caso considerado para comparação, vê-se que além do JPSO ter maior tempo de execução ainda tem uma função objetiva maior do que o VNS. Isto implica que em seu resultado há um número maior por instâncias. Em outras palavras, a qualidade das soluções geradas pelo JPSO possuem uma qualidade inferior ao VNS. Além disso, é importante constatar que a implementação aqui realizada gerou neste grupo a maior função objetivo. Para maiores referências ver tabelas 1 e 2

4.2 Comparações no grupo 2

No grupo 2, instâncias com número de vértices igual a 100 (360 instâncias), a melhor performance é tida pelo algoritmo VNS. Isto, pois apresenta como solução a melhor resposta (exata) e no menor tempo dentre os algoritmos apresentados. GRASP por sua vez também pode ser enquadrado como um algoritmo bom e obtém a mesma solução que o VNS (diferença mínima ao ser verificado o valor de suas funções objetivo). Aqui PILOT apresenta o segundo maior tempo de execução, tendo respostas piores que os demais. MGA continua sendo aquele de maior tempo. Aqui é possível perceber que ao aumentar o número de instâncias, o PILOT passa a perder aquilo que ele tinha de vantagem em relação aos demais que é a rapidez de execução. O JPSO aqui implementado possui um tempo de maior e ainda não produz uma resposta com a garantia que seja a solução ótima, além disso pode ser verificado que o JPSO gerou uma média maior quanto ao número de rótulos por instância (o que pesa negativamente para o valor de sua solução gerada) 3. Portanto a melhor opção para resolver um PAGRM é o uso do VNS. Para referência deste grupo ver tabelas 3 e 4

4.3 Comparações no grupo 3

Para instâncias maiores, como as que compõem o grupo 3 (referente à tabela 5, 6, 7 e 8) as heurísticas referentes ao VNS e ao GRASP se mantiveram superiores nos quesitos qualidade da resposta e tempo de execução. O GRASP passa a ter menor tempo de execução em relação ao VNS quão maior for o PGRM em questão. Em relação ao JPSO, é perceptível através da tabela 6 que em instâncias com o número de vértices igual a 200 possui um tempo de execução melhor apenas que o MGA. Segundo a tabela 5 é possível também notar que JPSO continua gerando mais rótulos por instância.Já em instâncias com o número de vértices igual a 500 (as maiores instâncias) é notável que ele apresenta o maior tempo de execução sem garantir o resultado ótimo.

5. CONCLUSÃO

Em nosso trabalho implementamos o algoritmo JPSO para fim de execução da base de dados disponibilizada. Esta é a mesma aplicada em [3] e este fato permitiu que fossem comparados os resultados obtidos pela implementação aqui discutida e aqueles apresentados por demais soluções. O objetivos deste trabalho foi ampliar os conhecimentos a respeito do PAGRM que é o foco dos algoritmos apresentados.

Nota-se que para instâncias maiores os algoritmos VNS e GRASP são os mais compensativos, levando em consideração aspectos como tempo de execução e garantia de solução ótima. O JPSO implementado possui um baixo rendimento para resolver um JPSO quando comparado às duas melhores heurísticas, pois não garante a obtenção da solução exata e ainda tem um tempo de execução maior, sendo compensativo em alguns casos em relação ao MGA e ao PILOT. Acredita-se que o motivo do execução do JPSO implementado está atrelado ao critério de parada do laço de repetição contido no algoritmo: A execução continuará até que a melhor solução global não se altere por até 100 iterações, assim a redução desse critério influencia no tempo de execução. Outro fator que é negativo para o JPSO implementado é o fato de gerar mais número de rótulos por instância do que a média geral apresentada por demais heurísticas. Além disso, foi possível notar que dependendo do objetivo e das condições apresentadas em mundo real, uma solução rápida de ser executada pode gerar uma resposta que não compensa – caso do PILOT em um conjunto de instâncias pequenas. Este trabalho foi fundamental para ampliar a visão a respeito do PAGRM e a forma como especialistas da área de grafos estudam heurísticas candidatas a serem boas alternativas de resolução.

.

6. REFERENCES

[1] R. Cerulli, A. Fink, M. Gentili, and S. Voß. Metaheuristics comparison for the minimum labelling spanning tree problem. In *The Next Wave in Computing, Optimization, and Decision Technologies*, pages 93–106. Springer, 2005.

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

[3] S. Consoli, K. Darby-Dowman, N. Mladenović, and J. M. Pérez. 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.

[4] S. Consoli, J. M. Pérez, K. Darby-Dowman, and N. Mladenović. Discrete particle swarm optimization for the minimum labelling steiner tree problem. In *Nature inspired cooperative strategies for optimization (NICSO 2007)*, pages 313–322. Springer, 2008.

[5] S. O. Krumke and H.-C. Wirth. On the minimum label spanning tree problem. *Information Processing Letters*, 66(2):81–85, 1998.

[6] R. E. Ladner. On the structure of polynomial time reducibility. *J. ACM*, 22(1):155–171, Jan. 1975.

[7] S. Pettie and V. Ramachandran. An optimal minimum spanning tree algorithm. *J. ACM*, 49(1):16–34, Jan. 2002.

[8] Y. Wan, G. Chen, and Y. Xu. A note on the minimum label spanning tree. *Information Processing Letters*,

84(2):99–101, 2002.

[9] Y. Xiong, B. Golden, and E. Wasil. A one-parameter genetic algorithm for the minimum labeling spanning tree problem. *Evolutionary Computation, IEEE Transactions on*, 9(1):55–60, 2005.

[10] Y. Xiong, B. Golden, and E. Wasil. Improved heuristics for the minimum label spanning tree problem. *Evolutionary Computation, IEEE Transactions on*, 10(6):700–703, 2006.