Algoritmo A* para resolução da Árvore Geradora de Rótulos Mínimos

Marco Cezar Moreira de Mattos[1], **Rômulo Manciola Meloca**[1]

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

{marco.cmm,rmeloca}@gmail.com

Resumo. *Este relatório apresenta o emprego do algoritmo A* para a resolução da Árvore Geradora de Rótulos Mínimos, tornando evidente o problema em torno deste, bem como suas origens e dificuldades e implicações. Apresenta-se os trabalhos relacionados em especial o que dá origem à implementação proposta para o problema, discutindo os resultados dos testes aplicados na solução teórica e na solução proposta.*

1. Introdução

Em teoria dos grafos há um problema muito comum: Dado um grafo G não-dirigido e conexo cujas arestas E sejam ponderadas e não negativas, encontrar uma árvore T sobre a qual seja possível induzir um subgrafo G' de G, tal que G' seja conexo e que a soma do peso de todas as arestas $w(E')$ seja a menor possível. A este problema chama-se Problema da Árvore Geradora de Pesos Mínimos (PAGPM), do inglês *Minimum Weight Spanning Tree*.

Definindo: Um grafo $G = \{V, E\}$ é um conjunto de vértices $v \in V$ conectados por meio de arestas $E = \{\{u, v\} \mid u, v \in V \wedge u \neq v\}$; Cada aresta e possui dois vértices (u, v) (abuso de notação adotado), podendo esta ser omnidirecional ou bidirecional caracterizando um grafo dirigido ou não-dirigido, respectivamente. As arestas podem portar informações relativas a distâncias caracterizando um grafo cujas arestas sejam ponderadas em sua presença ou não-ponderadas na ausência destas informações; Um vértice é uma unidade mínima que porta informações que deseja-se modelar.

Outras definições: Uma árvore T não é senão um grafo bipartido, planar, acíclico e conexo, de modo que uma árvore é uma particularidade de um grafo (assim como quadrado está para retângulo); Um grafo bipartido é aquele cujos vértices podem ser divididos em dois conjuntos disjuntos, isto é, cuja intersecção entre os conjuntos seja nula; Um grafo planar é aquele que pode ser representado em um plano (bidimensional) no qual nenhuma aresta seja concorrente a qualquer outra; Um grafo acíclico é aquele que não apresenta arestas que formem um caminho que contenha um ou mais vértices repetidos; Um grafo conexo, no âmbito de grafos não-dirigidos, é aquele que conecta todos os vértices do grafo, de modo que é possível acessar qualquer vértice a partir de qualquer outro vértice.

Mais definições: Um subgrafo $G' = \{V', E'\}$ de um grafo $G = \{V, E\}$ é um grafo em que $V' \subseteq V \wedge E' \subseteq E$; Um grafo G' induzido por V' é um subgrafo de G em que o conjunto de arestas E' é formado a partir do conjunto de vértices V'

tal que para todo vértice v adiciona-se em E' todas as arestas relacionadas a este originalmente contidas em E. Matematicamente define-se um grafo induzido como sendo um grafo $G' = \{V', E'\} \mid E' = \{(u, v) \in E : u, v \in V'\} \wedge V' \subseteq V \wedge E' \subseteq E$. Nota-se que também é possível induzir um grafo a partir de outros conjuntos; Uma árvore geradora T é um subgrafo G' de G tal que este seja conexo, não-dirigido e que possua o mesmo conjunto inicial de vértices $V' = V$; Define-se o peso total das arestas $w(E')$ como sendo $w(E') = \sum_{(u,v \in E')} [w(u, v)]$.

Para formar uma árvore geradora são necessários exatamente $|V| - 1$ arestas, tal que $|V|$ seja o número de vértices pertencentes ao grafo; Esta regra advém da definição das árvores geradoras, uma vez que esta não admite ciclos e conecta todos os vértices do grafo, disto deriva-se que para conectar todos os vértices são necessárias tantas arestas quantos vértices houverem, com exceção de uma, a fim de admitir no mínimo dois vértices que sejam finais.

Um grafo $G = \{V, E\}$, no entanto, pode admitir várias árvores geradoras. É possível discretizar o número máximo de árvores geradoras que um grafo admite fazendo a combinação do número de arestas $|E|$ em grupos de $|V| - 1$, uma vez que cada árvore geradora possui $|V| - 1$ arestas, isto é, $G' = \{E', V' \mid |E'| = |V| - 1 \wedge V' = V\}$, conforme discutido. Fazendo a combinação matemática $\binom{|E|}{|E'|}$, obtém-se o número de conjuntos distintos que E' pode assumir, gerados a partir E. Evidentemente, nem todas combinações possíveis são árvores geradoras à medida em que elementos do conjunto de possíveis valores para E' podem admitir ciclos, descaracterizando-se como uma árvore, tampouco geradora.

Sejam $e = |E|$ e $e' = |E'| = |V| - 1$,

$$C_e^{e'} = \frac{e!}{e' \cdot [e - e']!}$$

Nota-se que quanto mais denso for o grafo, isto é quanto maior for o número de arestas que o grafo contiver, mais árvores geradoras o grafo admite.

Para solucionar o PAGPM é preciso obter a melhor árvore dentre todo o conjunto de árvores geradas $C = \{E'_1, E'_2, \ldots, E'_n \mid n = C_{|E|,|V|-1}\}$ a qual possua o menor peso total. Obviamente o conjunto de árvores geradas para o PAGPM é grande o suficiente para que não seja viável computar todo o conjunto e buscar nele uma solução exata (em tese), ou seja, muitas soluções são até mesmo NP-Completas! O que demanda uma solução de busca inteligente. Deste modo, é possível utilizarmos busca cega, heurística ou local.

Utilizar um algoritmo de busca implica em obter um ótimo local ao invés da solução exata, entretanto, alguns problemas de busca operam sobre um conjunto cujos elementos são linearmente independentes, de modo que o conjunto é um matroide.

Análogo ao PAGPM, há ainda o Problema da Árvore Geradora de Rótulos Mínimos (PAGRM), do inglês *Minimum Labelling Spanning Tree*, o que lhe difere do PAGPM é que neste, cada aresta possui um rótulo e o objetivo é encontrar um árvore geradora utilizando o menor número de rótulos possíveis. Neste caso, a combinação

matemática é feita a partir do número de rótulos e não a partir do número de arestas, de modo que a computação da solução exata (não fosse a propriedade matemática explorada), levaria muito tempo para ser feita, uma vez que não sabe-se de antemão quantos rótulos distintos serão necessários para conectar todo o grafo

Na seção que segue é demonstrado o escopo do PAGRM.

2. O problema

Seja $G = \{V, E, L\}$ um grafo rotulado, não-dirigido e conexo, tal que $V = \{v_1, v_2 \dots v_{|V|}\}$ seja o conjunto de vértices, $E = \{(u, v) \mid u, v \in V \wedge u \neq v\}$ seja o conjunto de arestas e $L = \{l_1, l_2 \dots l_{|L|}\}$ seja o conjunto de rótulos tal que cada aresta e contenha um rótulo l, deseja-se obter um subgrafo $G' = \{V', E', L' \mid V' = V \wedge E' \subseteq E \wedge |E'| = |V| - 1 \wedge L' \subseteq L \wedge \min |L'|\}$. Nestas condições, pode-se chamar G' de T, isto é, uma árvore geradora de rótulos mínimos.

A solução exata do PAGRM demandaria computar todas as possibilidades de combinação de rótulos com tamanhos variados, que pode ser calculado através da combinação de todos rótulos $\binom{|L|}{i}$ em conjuntos de $i = 1 \dots |L|$.

Sejam L o conjunto de rótulos do grafo G, C o conjunto de combinações possíveis para L' e $l = |L|$,

$$|C| = \sum_{i=1}^{l} \frac{l!}{i! \cdot [l - i]!}$$

Deste modo, define-se C por $C = \{L'_1, L'_2 \dots L'_n \mid n = \sum_{i=1}^{|L|} C^i_{|L|}\}$, e o conjunto objetivo L' por um dos elementos do conjunto C tal que $|L'| = \min_{\forall l' \in C} |l'|$.

Nota-se que ao combinar determinados rótulos o subgrafo obtido pode admitir ciclos, e portanto não configura-se como árvore, todavia, diferente do PAGPM esta pode ser a solução desde que conecte todo o grafo e utilize o menor número possível de rótulos. Neste caso, para tornar o subgrafo uma árvore geradora, basta remover uma aresta qualquer de cada ciclo, uma vez que o objetivo é atingir o menor número de rótulos; Remover uma aresta em nada impacta o número de rótulos utilizados, pois se o fizesse então o subconjunto L' de rótulos em questão não seria a solução uma vez que ao remover uma aresta que forme ciclo obteria-se um número menor de rótulos utilizados e, portanto, esta seria a solução.

Evidentemente, não seria necessário percorrer todos os elementos do conjunto C uma vez que, sendo este gerado de maneira ordenada progressivamente pelo número de rótulos utilizados, o algoritmo chegaria em seu objetivo na primeira ocorrência de uma árvore geradora que satisfaça os critérios de saída do algoritmo. Nestas circunstâncias, para obter-se a solução haveria-se necessário buscar linearmente em no máximo todas as combinações de todos os rótulos possíveis com o número de rótulos variando de $1 \dots |V| - 1$, uma vez que o número máximo de arestas necessárias para conectar um grafo é $|V| - 1$ arestas e que cada uma pode assumir um rótulo diferente. Caso o número de rótulos seja menor que $|V| - 1$, então este é o máximo a ser buscado.

$$\sum_{i=1}^{\min(|L|,|V|-1)} \binom{|L|}{i}$$

Contudo, lançando mão da propriedade matemática de matroides e independência linear, pode-se utilizar um algoritmo de busca (cega, heurística ou local), que não expanda todos os nós e caminhe de acordo o objetivo de maneira a reduzir o número de nós expandidos e, por consequência, o tempo de computação.

Na seção a seguir são tratados os trabalhos relacionados ao tema.

3. Trabalhos Relacionados

Revisando a literatura, verificam-se várias tentativas de otimização para o problema, combinando variadas formas de algoritmos de busca que venham a fornecer melhores resultados.

Vários testes são executados em [**?**]

4. Solução Proposta

4.1. Busca A*

Uma vez que o conjunto onde buscar a solução para o PAGRM é muito grande e este seja um matroide, isto é, o conjunto de florestas, é possível buscar a solução exata para o problema com um algoritmo de busca gulosa, como é o A*, que sempre escolhe para expandir o nó que estiver mais próximo do objetivo, segundo uma função heurística que estima a distância a ser percorrida.

Para adequar o algoritmo A* ao PAGRM, de acordo com a literatura, definiu-se o que são os nós e como é dada a função heurística que mede a distância já percorrida e a distância a ser percorrida. Para o problema definiu-se que os nós são os subgrafos induzidos pelo conjunto L', tal que o conjunto $L' = \{l'_1, l'_2 \dots l'_n \mid n = |L'|\} \wedge L' \subseteq L$ são os rótulos utilizados no subgrafo. A função heurística $f(node) = g(node) + h(node)$ têm seus valores calculados tal que $g(node)$ meça os rótulos utilizados no nó e $h(node)$ é estime os nós a serem utilizados, computado por meio do menor número de rótulos que cobrem o número de arestas que ainda faltam ser cobertas.

Deste modo, para um subgrafo $G' = \{V', E', L'\}$ de um grafo $G = \{V, E, L\}$, seja $E = \{e_1, e_2 \dots e_n \mid e_1 \geq e_2 \geq \dots \geq e_n \wedge n = |L'| \wedge e = |E(l)|\}$. Cada elemento e corresponde ao número de arestas que o rótulo cobre no grafo.

$g(node) = |L'|$

$h(node) = \min j \sum_{i=1}^{j} e_i \geq (|V| - 1) - |E'| \mid E' \in T'$

Na prática a lista de rótulos não utilizados é ordenada pelo número de arestas que cobre. Para os cálculos com o número de arestas que um rótulo cobre, são removidos os ciclos. Abaixo ilustra-se a busca A* entre todas as combinações possíveis de rótulos, como já discutido. Seu funcionamento orientado ao objetivo pode ser observado.

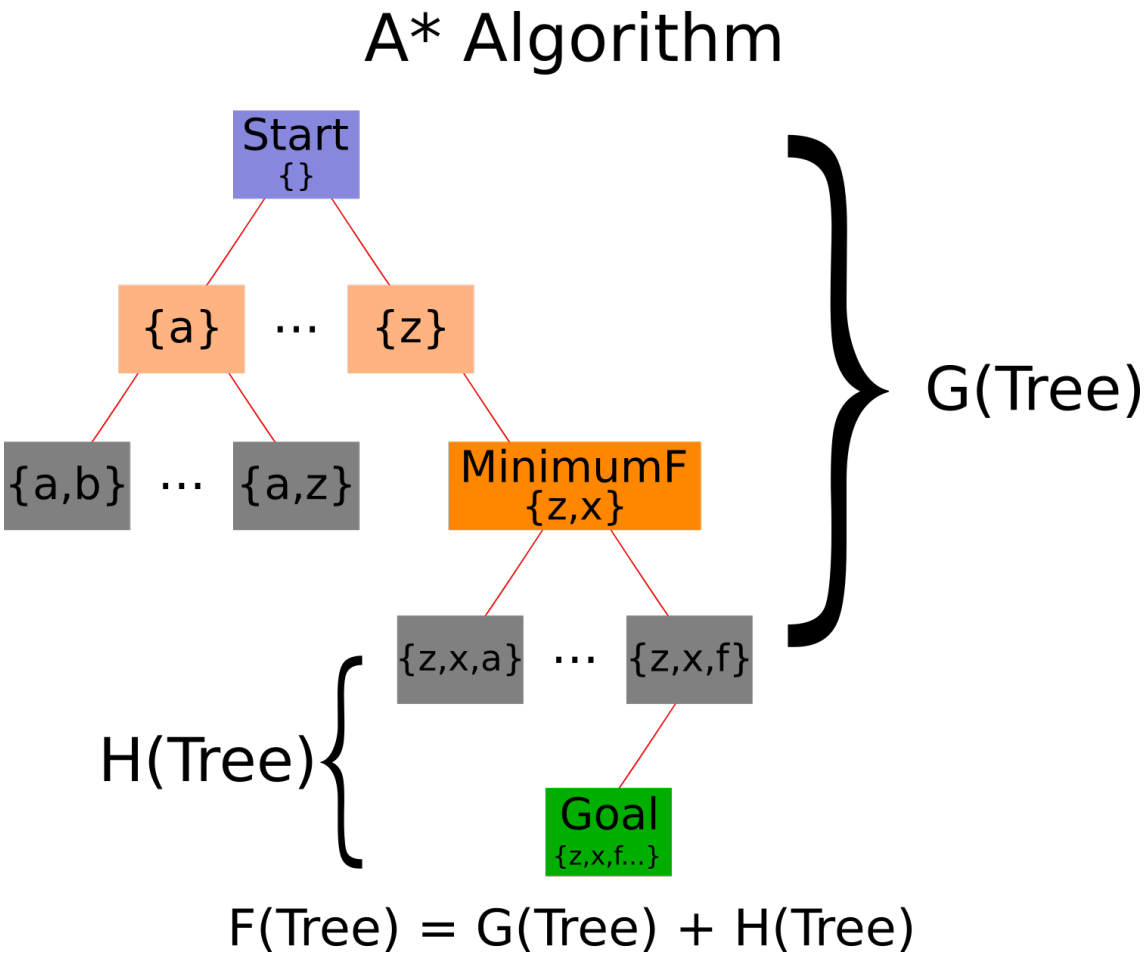

Figura 1. Algoritmo A*

Cumpre dizer que durante a execução do algoritmo os nós já expandidos, porém não selecionados (por não possuírem menor função heurística), são armazenados em uma lista para que sempre seja obtido o nó que, de fato, possuir menor função heurística.

4.2. Implementação

Evidentemente é possível fazer uma espessa camada de abstrações que permite modelar um grafo por meio de uma matriz, uma lista, vetores, objetos recursivamente encadeados ou qualquer outro tipo abstrato de dados (TAD), armazenando mais informações ou menos informações conforme a necessidade e devida observação ao *trade-off* consumo de RAM vs. consumo de CPU.

Para este trabalho, optou-se por consumir mais RAM que CPU, de modo que para cada nó (conjunto de rótulos sobre os quais induz-se uma floresta, isto é, um subgrafo do grafo original) armazena-se também a lista de arestas cobertas pelos rótulos. Armazena-se ainda, para cada rótulo, as arestas que cada qual cobre. Além das arestas cobertas e rótulos utilizados, armazena-se a lista de rótulos não selecionados.

Implementou-se a solução na linguagem de programação Java, lançando mão do arcabouço de métodos, interfaces e classes previamente implementadas pelo tutor da disciplina.

Respaldado pelo algoritmo proposto por [**?**], implementou-se este cuja ideia central em altíssimo nível é:

Data: Grafo rotulado e conexo.
Result: Árvore geradora de rótulos mínimos.
Ordene a lista de rótulos pelo número de arestas que cada rótulo cobre;
Gere o primeiro nó;
while *Não for obtida uma árvore geradora de rótulos mínimos* **do**
    Obtenha o nó mais próximo do objetivo;
    forall *filhos do nó objetivo* **do**
        Calcule o custo de cada filho do nó mais próximo do objetivo;
    end
end
Retorne a árvore geradora de rótulos mínimos;

Algorithm 1: Busca A* para resolver 8-Puzzle

Cabe comentar que apropriou-se da ideia do algoritmo de Prim para resolução da árvore geradora de pesos mínimos. Neste caso, utilizou-se da ideia de englobar sempre ao conjunto de vértices pertencentes ao corte S aquele vértice pertencente à $V - S$ ao qual houver uma aresta segura incindindo nele. No escopo deste problema não há a necessidade de comparar os pesos das arestas, uma vez que são apenas rótulos. Ao final da execução do algoritmo de Prim, obtém-se uma árvore, isto é, os ciclos são removidos. Conforme discutido a respeito da remoção de uma aresta qualquer que forme um ciclo, aplicou-se uma adaptação do algoritmo de Prim que escolha quaisquer arestas seguras para T'.

4.3. Diagramação

Para a solução projetada, utilizou-se a seguinte organização da solução, evidenciada no diagrama de classes.

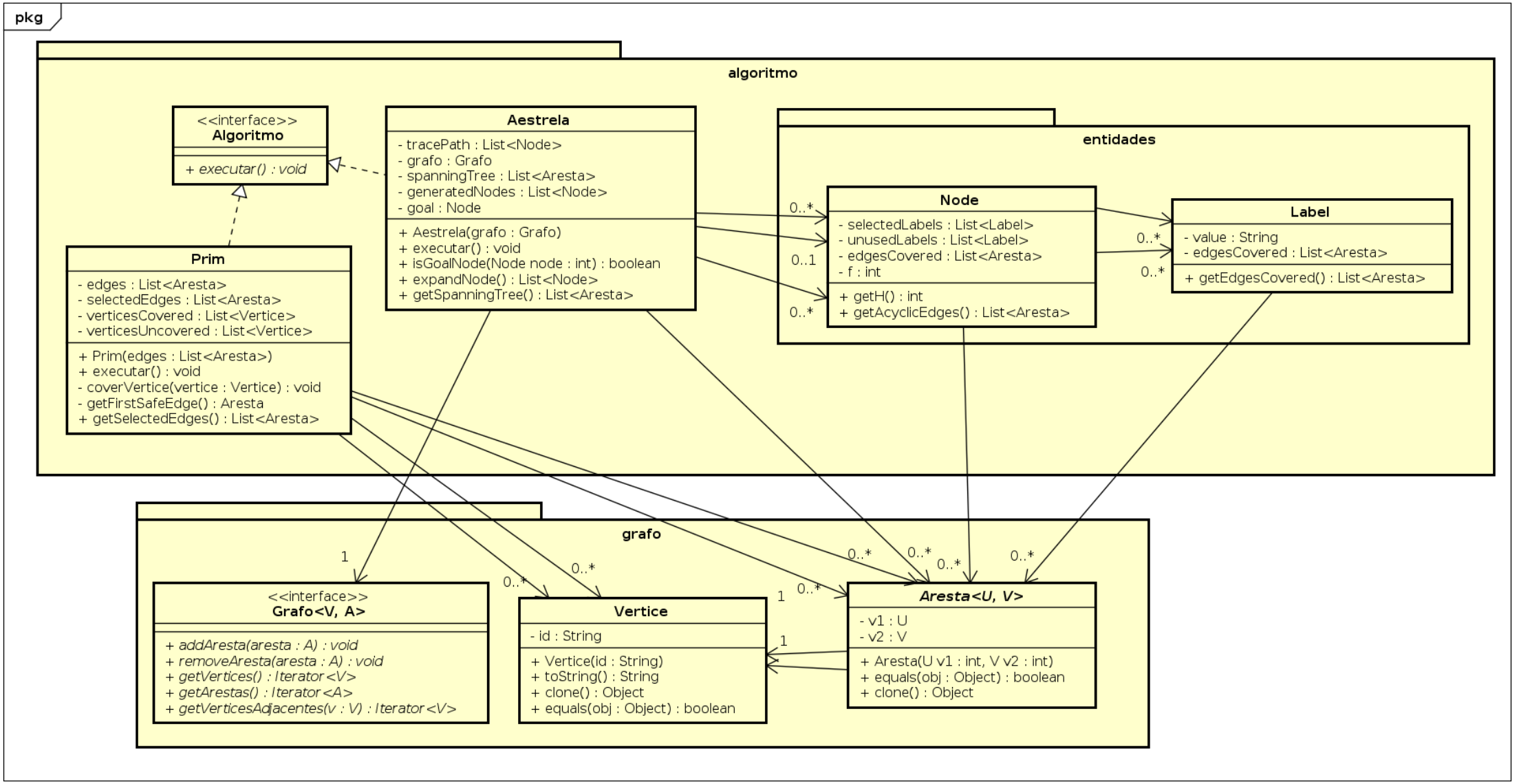

Figura 2. Diagrama de Classes

5. Testes

Nesta seção comparou-se o tempo computacional e qualidade da solução do algoritmo A* proposto pelo [**?**]. E executou-se a implementação deste algoritmo para uma base de dados quem contém 10(dez) grafos cada arquivo, e os arquivos variavam a quantidade de labels, vértices e a densidade.

A base de dados proposta pelo [**?**] foi executada em um computador Intel(R) Core(TM) i7-5500U CPU @ 2.40GHz com 16GB de RAM. E obteu-se os seguintes resultados da seguinte seção.

5.1. Resultados

Utilizou-se 16 grupos, variando a quantidade de vértices de 20 até 500, a quantidade de labels de 20 até 625. Para cada grupo, existem 3(três) densidades diferentes, sendo elas 0.2, 0.5 e 0.8. As tabelas a baixo, mostram, de par em par, a qualidade da solução e o tempo computacional.

n	l	d	Exact	A*
			$\lvert\bar{L'}\rvert$	$\lvert\bar{L'}\rvert$
20	20	0.8	2.4	2.8
		0.5	3.1	3.6
		0.2	6.7	7.8
30	30	0.8	2.8	2.9
		0.5	3.7	4.1
		0.2	7.4	8.7
40	40	0.8	2.9	2.7
		0.5	3.7	4.1
		0.2	7.4	8.9
50	50	0.8	3	3.0
		0.5	4	4.5
		0.2	8.6	10.5

Tabela 1. Qualidade da solução

n	l	d	Exact	A*
			$Tempo(ms)$	$Tempo(ms)$
20	20	0.8	0	11
		0.5	0	2
		0.2	11	4
30	30	0.8	0	88
		0.5	0	103
		0.2	138	719
40	40	0.8	2	490
		0.5	3.2	323
		0.2	100200	144
50	50	0.8	3.1	2155
		0.5	21.9	19866
		0.2	66300	18839

Tabela 2. Tempo computacional

n	l	d	Exact	A*
			$\|\tilde{L'}\|$	$\|\tilde{L'}\|$
100	25	0.8	1.8	1.8
		0.5	2	2
		0.2	4.5	NF
100	50	0.8	2	2
		0.5	3	NF
		0.5	6.7	NF
100	100	0.8	3	NF
		0.5	4.7	NF
		0.2	NF	NF
100	125	0.8	4	NF
		0.5	5.2	NF
		0.2	NF	NF

Tabela 3. Qualidade da solução

n	l	d	Exact	A*
			$\|\tilde{L'}\|$	$\|\tilde{L'}\|$
200	50	0.8	2	2
		0.5	2.2	NF
		0.2	5.2	NF
200	100	0.8	2.6	NF
		0.5	3.4	NF
		0.5	NF	NF
200	200	0.8	4	NF
		0.5	NF	NF
		0.2	NF	NF
200	250	0.8	4	NF
		0.5	NF	NF
		0.2	NF	NF

Tabela 4. Qualidade da solução

n	l	d	Exact	A*
			$Tempo(ms)$	$Tempo(ms)$
100	25	0.8	0	2101
		0.5	0	2303
		0.2	11	NF
100	50	0.8	0	19384
		0.5	0	NF
		0.2	138	NF
100	100	0.8	2	NF
		0.5	3.2	NF
		0.2	100200	NF
100	125	0.8	3.1	NF
		0.5	21.9	NF
		0.2	66300	NF

Tabela 5. Tempo computacional

n	l	d	Exact	A*
			$Tempo(ms)$	$Tempo(ms)$
200	50	0.8	29	57509
		0.5	32	NF
		0.2	5400	NF
200	100	0.8	138	NF
		0.5	807	NF
		0.5	NF	NF
200	200	0.8	22500	NF
		0.5	NF	NF
		0.2	NF	NF
200	250	0.8	206000	NF
		0.5	NF	NF
		0.2	NF	NF

Tabela 6. Tempo computacional

5.2. Análise

Observou-se uma excelente qualidade da solução, sendo todos os casos a solução ideal. Porém o custo computacional foi muito elevado, impossibilitando a execução do algoritmo para instâncias com muitos vértices e/ou muitas labels.

6. Considerações Finais

Considera-se acerca deste trabalho três aspectos, a saber: Sua concepção, seu desenvolvimento e sua execução.

Concebeu-se este trabalho como um problema cujo espaço de soluções é grande o suficiente para que seja necessário o maior número possível de otimizações, deste modo,

Desenvolveu-se todo o trabalho, desde a elaboração da solução até a fase de implementação pensando em todas as melhorias possíveis para que o algoritmo pudesse ser executado em menor tempo possível. Conforme discutido, optou-se por consumir mais RAM que processamento, contudo as

Testou-se.

Deste modo, pôde-se,.

Referências

[1] Chang, R., Leu, S. *The minimum labeling spanning trees.* Information Processing Letters, 63(5), 277282, 1997.

[2] Consoli, S., Darby-Dowman, K., Mladenovic, N., Moreno-P'erez, 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.