Emparelhamento em grafos: Comparação de desempenho entre o algoritmo força-bruta e resolvedor minisat

Gisele Goulart Tavares da Silva[1], **Guilherme Almeida Félix da Silva**[1]

[1]Departamento de Ciência da Computação
Universidade Federal de Juiz de Fora

giselegoulart@ice.ufjf.br, guilherme.felix@engenharia.ufjf.br

Resumo. *Trabalho prático apresentado como parte da avaliação da disciplina Análise e Projeto de Algoritmos - DCC001 da Universidade Federal de Juiz de Fora - UFJF. O problema tratado é o de emparelhamento maximal mínimo em grafos. Apresenta-se um algoritmo que produz a solução exata, sua complexidade é avaliada e, por tratar-se de um problema NP-Completo, é esperado que a partir de determinado tamanho de instância de teste o algoritmo proposto não retorne a solução exata em tempo factível. Ainda, apresenta-se um modelo deste problema na forma de um problema SAT e, utilizando um resolvedor de problemas SAT, uma solução é obtida. É feito um estudo comparativo com instâncias de tamanhos pequeno (até 10 vértices), médio (30 vértices) e grandes (50 vértices).*

1. Introdução

Um grafo é um conjunto finito e não vazio de vértices e arestas, indicado por $G(V, E)$, de maneira que cada aresta está associada a dois vértices, chamados de **extremidades**. O conjunto de vértices de G é representado por $V(G)$ e o conjunto de arestas de G é indicado por $E(G)$.

O número de arestas de um grafo é denotado por $|E|$ e o número de vértices é denotado por $|V|$. A **cardinalidade** de um conjunto de quaisquer elementos de um grafo representa a número de elementos desse conjunto.

Sejam *u* e *v* dois vértices pertencentes a um grafo G, indica-se a aresta entre esses vértices como (u, v). Neste contexto não há distinção entre arestas caso a ordem dos vértices seja trocada, ou seja, a aresta (u, v) é a mesma que a aresta (v, u) (o grafo é dito ser *não direcionado*).

Duas arestas são consideradas *adjacentes* se possuem um vértice em comum. Um **grafo completo** é um grafo tal que todos os vértices são vizinhos entre si, ou seja, todo vértice possui aresta que o liga a todos os demais.

Um **subgrafo** H de um grafo G é tal que todos os seus vértices e arestas estão em G e as extremidades de cada aresta de H são as mesmas em G.

Um **emparelhamento** *M* em um grafo *G (V,E)* é um subconjunto de arestas tais que não são adjacentes duas a duas. Um emparelhamento *M* é **maximal** se não é possível adicionar mais arestas a *M*, e será **máximo** se for um emparelhamento maximal que contém o maior número de arestas possível.

O problema de determinar um emparelhamento máximo em um grafo é considerado tradicional na área de estudo de algoritmos e possui complexidade polinomial. O algoritmo de Edmonds permite obter o emparelhamento máximo em um grafo qualquer em tempo polinomial. O objetivo deste trabalho é escrever (e avaliar a complexidade e desempenho) de um algoritmo que encontre o **emparelhamento maximal mínimo**, ou seja, um emparelhamento maximal de **menor cardinalidade** possível. Este problema foi demonstrado por Yannakakis e Gavril ser NP-Completo [Centeno 2007]. Desta forma, sabe-se que o algoritmo apresentado aqui será capaz de encontrar a solução exata apenas para instâncias consideradas pequenas. Para instâncias a partir de um determinado tamanho, o tempo necessário para a obtenção da solução exata torna-se impraticável.

2. Algoritmo

O algoritmo apresentado a seguir foi construído de acordo com o paradigma *força-bruta*, que consiste em gerar todas as possibilidades e testar uma a uma afim de se encontrar a solução.

1. Gera todas as permutações de arestas do grafo G de entrada
2. Para cada permutação, seleciona a primeira aresta, verifica se as demais são vizinhas dela e inclui as não-vizinhas numa solução temporária.
3. Se a solução temporária tiver tamanho menor que a solução global, atualiza a solução global
4. Repete o procedimento iniciando de cada aresta da permutação
5. Retorna a solução global: Conjunto de arestas e seu tamanho.

A ideia do algoritmo é selecionar arestas e adicionar no conjunto solução, uma a uma, testando se são vizinhas, de todas as formas possíveis. Dessa maneira, primeiramente é gerada uma lista com uma ordenação de todas as arestas do grafo. A primeira da lista é colocada na solução candidata. Testa-se se todas demais arestas e as não vizinhas são adicionadas na solução candidata. Guarda-se essa solução. Em seguida, o processo é feito na mesma lista, porém iniciando o processo adicionando a aresta na segunda posição da lista. E isso é feito para todas as permutações com o conjunto de arestas do grafo.

A complexidade de gerar todas as permutações de arestas domina assintoticamente todas as demais operações, de forma que a complexidade desse algoritmo é $|E|!$. O algoritmo de Johnson Trotter foi usado nessa etapa. Assim, supondo o tempo necessário para gerar um subconjunto igual a $10^{-9}s$, para uma instância com 5 arestas o tempo gasto para o processamento seria de $1,2 \times 10^{-7}$ s. Para uma instância de 10 arestas, o tempo sobe para $1,6288 \times 10^{-3}$ s. Para 20 arestas o tempo seria $2,432 \times 10^{9}$ s, que já se mostra uma instância intratável.

O grafo está representado como uma lista de adjacências e diversas listas ligadas foram utilizadas em todo o procedimento, como estruturas auxiliares. Também é utilizada uma estrutura `listaAresta`, onde são incluídas todas as arestas do grafo e onde o algoritmo força bruta realiza sua busca. A estrutura do grafo em si não é utilizada durante a execução do algoritmo força bruta e da transformação da entrada para SAT. A montagem do grafo foi realizada em um primeiro momento para uma tentativa de solução do problema utilizando abordagem gulosa, de modo a retornar uma solução aproximada. Funções de inclusão, exclusão e busca foram implementadas na estrutura `lista` (que representa um grafo). A implementação do algoritmo força bruta, apesar de mais custosa, retorna a solução exata do problema. Desse modo, partindo da estrutura do grafo já pronta, implementamos estruturas auxiliares para tratar apenas das arestas do grafo, onde as permutações são realizadas.

2.1. Resultados - Algoritmo Força-Bruta

Os testes foram executados em uma máquina com a seguinte especificação: 8GB de memória RAM, 1TB de HD, processador i5 quinta geração (2 cores físicos, total de 4 com hyperthreading).

Instancia	$\|V\|$	$\|E\|$	Solução	Tempo de Execução (ms)	Observações
teste1_wiki.txt	6	6	1 aresta	7.348	Processo concluído
teste3_wiki.txt	5	6	2 arestas	12.543	Processo concluído
teste2_wiki.txt	6	7	2 arestas	89.513	Processo concluído
teste4.txt	8	8	3 arestas	1239.88	Processo concluído
teste_10.txt	6	10	2 arestas	189420	Processo concluído
teste_30.txt	9	30	4 arestas	2077118 ($\approx$ 35 min)	Processo encerrado. Solução parcial

Tabela 1. Resultados algoritmo força bruta

O consumo de memória foi da ordem de 90 % As figuras a seguir indicam algumas instâncias de teste e a solução está indicada em vermelho.

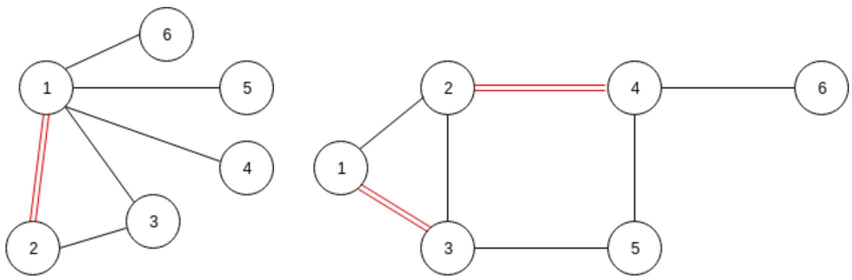

Figura 1. Instancias *teste1_wiki.txt* e *teste2_wiki.txt*

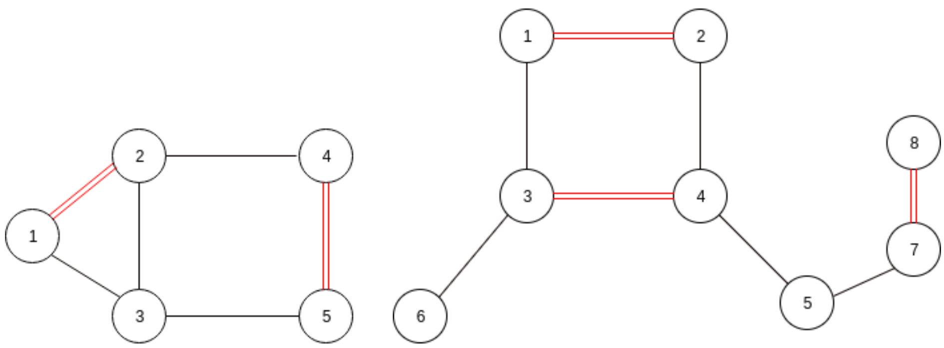

Figura 2. Instancias *teste3_wiki.txt* e *teste4_wiki.txt*

3. Formulação SAT

O problema da **satisfatibilidade** consiste em determinar se uma fórmula proposicional na *forma normal conjuntiva* (FNC) é satisfatível. Uma fórmula proposicional consiste de variáveis booleanas, conectivos $\wedge$ (conjunção), $\vee$ (disjunção) e $\neg$ (negação). A fórmula é dita ser *satisfatível* se, e somente se, existir uma atribuição de valores *verdadeiro* e *falso* para as variáveis de forma que toda a expressão assuma o valor lógico *verdadeiro*. A fórmula está na FNC se for uma conjunção de *cláusulas*. Uma *cláusula* é uma disjunção de *literais*. Um *literal* é uma variável (booleana) ou sua negação. [Stamm-Wilbrandt 1993].

Esse problema, também conhecido por SAT, foi o primeiro a ser mostrado NP-completo, ou seja, é um problema que não se conhece algoritmo que o resolva em tempo polinomial. Sua importância é permitir identificar outros problemas que também pertençam a essa classe (NP-completo). Para isso é necessário que se faça uma *redução* do problema SAT ao problema que está sendo estudado. Em outras palavras, deve-se reescrever o problema SAT de forma que ele corresponda a uma instância do problema a ser trabalhado. Ao se obter um algoritmo que faça essa "tradução"da entrada de um problema SAT na entrada do problema a ser tratado em tempo polinomial e garantindo que a solução do problema a ser tratado indique a solução do problema SAT, concluímos a *redução*[Levitin 2012]. Em resumo, é possível resolver um problema por meio da solução de uma instância do problema SAT.

No caso deste trabalho, o problema consiste em converter a entrada do problema de emparelhamento (um grafo e um inteiro *k*) na entrada de um problema SAT (uma fórmula booleana na FNC).

3.1. Modelo SAT

O enunciado do problema original pode ser reescrito na forma de um problema de decisão como: Existe um subconjunto *M* de arestas em um grafo *G*, de tamanho não maior do que um dado *K* tal que as arestas de *M* não sejam vizinhas entre si duas a duas?

A formulação em uma expressão booleana na forma normal conjuntiva é apresentada a seguir:

$$F = \bigwedge_{1 \le i \le K} noMaximoUm\{[e,i] | e \in E\} \wedge \bigwedge_{e,f \in E}^{e \cap f \neq \emptyset} noMaximoUm\{[e,i],[f,i] | 1 \le i \le K\}$$
$$\wedge \bigwedge_{e \in E} nenhum[e,i] | 1 \le i \le K \Rightarrow peloMenosUm\{[f,i] | 1 \le i \le K, f \in E, e \cap f \neq \emptyset\}$$

Nessa formulação $K \le |E|$ e as funções $noMaximoUm$ e $peloMenosUm$ são escritas como:

$$noMaximoUm\{l_1, l_2, ..., l_x\} := \bigwedge_{1 \le i,j \le x} (\bar{l_i} \vee \bar{l_j})$$

$$peloMenosUm\{l_1, l_2, ..., l_x\} := (l_1 \vee l_2 \vee \cdots \vee l_x)$$

A primeira delas indica que a expressão inteira será verdadeira se no máximo um literal assumir o valor verdadeiro. A segunda delas indica que a expressão inteira será verdadeira se ao menos um literal for verdadeiro.

A relação $a \Rightarrow b$ é equivalente a $\neg a \vee b$ e também foi utilizada para fins de implementação. Todas essas expressões e o modelo foram retirados de [Stamm-Wilbrandt 1993]

3.2. MiniSat - SAT Solver

O resolvedor de problemas SAT usado foi o miniSat[1], que recebe como entrada um arquivo de texto contendo uma cláusula por linha e cada linha contem um literal separado por espaço. O fim da cláusula é representado por um caracter "0". Nessa representação, a negação de um literal é representado pelo símbolo "-"e cada literal é indicado por um número inteiro, no nosso caso, o identificador de cada aresta.

Já estando na FNC, cada linha armazena as disjunções de cada literal e de uma linha para a outra está representada a conjunção de cada cláusula. Para produzir este arquivo de entrada, foram escritas as funções $noMaximoUm$ e $peloMenosUm$ e uma função maior que representa a fórmula completa.

A função $noMaximoUm$ consiste em colocar cada literal negado dois a dois em uma cláusula e, em seguida, fazer a conjunção de cada uma delas. A função $peloMenosUm$ consiste em fazer uma disjunção de cada literal.

Para encontrar a solução do problema, é feito um laço, com a variável K iniciando em 1 e indo até o total de arestas do grafo, e para cada valor de K é gerada uma fórmula completa, que é passada para o miniSat. Quando a resposta gerada for *SATISFIABLE*, o laço é encerrado. O K é guardado (e então temos a cardinalidade do conjunto obtido) e o retorno do *solver* indica o valor lógico de cada literal da entrada, que é interpretado diretamente: variável negada, aresta fora da solução, variável verdadeira, aresta na solução.

A função que gera a fórmula completa consiste na conjunção de três expressões. A primeira delas itera sobre as arestas do grafo, identificando-as unicamente, para cada K. Assim, para K = 1, todas as arestas do grafo serão identificadas com um novo rótulo ([1, 1], [2, 1], [3, 1], etc.), colocadas em uma lista auxiliar (na qual cada identificador corresponde a um literal) e passada para a função $noMaximoUm$, que se encarregará de escrever no arquivo de entrada para o miniSat essa parte da fórmula completa.

A segunda expressão itera sobre as arestas, porém só identifica com novos rótulos e inclui na lista auxiliar a ser passada para a função $noMaximoUm$ as arestas que são vizinhas. Portanto, antes de se incluir na lista auxiliar é feita iteração na lista de arestas para identificar quais são vizinhas ($e \cap f \neq \emptyset$).

A terceira (e última) expressão foi reescrita como

$$\bigwedge_{e \in E} (peloMenosUm[e,i] | 1 \leq i \leq K) \quad \vee \quad (peloMenosUm\{[f,i] | 1 \leq i \leq K, f \in E, e \cap f \neq \emptyset\})$$

utilizando a relação $a \Rightarrow b$ é equivalente a $\neg a \vee b$ e o fato que a negação de "nenhum"é "pelo menos um". Esse trecho também identifica, para cada aresta do grafo quais são

[1] disponível em http://minisat.se/Main.html

suas vizinhas, atribui novos rótulos e as inclui em uma lista auxiliar, a ser passada para a função $peloMenosUm$.

4. Resultados miniSat

Instancia	$\|V\|$	$\|E\|$	Saída do miniSat	Valor de K	Tempo de Execução (s)
teste1_wiki.txt	6	6	Satisfiable	1 aresta	0
teste3_wiki.txt	5	6	Satisfiable	2 arestas	0
teste2_wiki.txt	6	7	Satisfiable	2 arestas	0
teste4.txt	8	8	Satisfiable	2 arestas	0
teste_10.txt*	6	10	Satisfiable	2 arestas	0
teste_30.txt**	9	30	Satisfiable	2 arestas	0

Tabela 2. Resultados do miniSat

As duas últimas instâncias (* e **) foram executadas com a função do força bruta desabilitado, por conta do consumo de memória. Para as quatro primeiras instâncias os resultado foi o esperado, ao menos na cardinalidade da solução. Porém, inspecionando a solução mostrada pelo miniSat observamos claramente inconsistência na saída. Para o arquivo `teste2_wiki.txt`, por exemplo, a saída foi [-1 -2 -3 4 5 -6 -7 -8 -9 -10 11 -12 13 -14 0], que indica a seleção de quatro arestas para a solução. O arquivo `teste4.txt` que também apresentou solução com K = 2, teve como saída [-1 -2 -3 4 -5 -6 7 -8 9 -10 -11 -12 13 -14 15 -16 0], também indicando mais de duas arestas para a solução. Além desses erros, a solução do miniSat para todas as instâncias (a menos da primeira na tabela) foi exatamente a mesma, o que é, no mínimo, suspeito.

Considerando que o modelo teórico utilizado já é consolidado na comunidade científica e que o solver também tem reconhecimento de "estado da arte"atestado, os indícios mais fortes dão conta de que a fonte do problema está relacionada à transcrição das cláusulas no arquivo de entrada para o tratamento do miniSat.

De qualquer forma, é notório o péssimo desempenho do algoritmo força bruta, o que o torna proibitivo em problemas cuja complexidade é conhecidamente alta (neste caso NP-Completo), muito embora retorne a solução exata. Sua aplicabilidade está restrita a instâncias de tamanho muito reduzido, o que não reflete a maior parte dos problemas a serem tratados no mundo real, tanto pelo tamanho da instância quanto pela restrição de tempo de solução. Assim, para a referida classe de problemas, outras estratégias trazem melhor resultado, como heurística gulosa, por exemplo.

Referências

Centeno, C. C. (2007). Sobre emparelhamento maximal mínimo em certas classes de grafos. Master's thesis, Universidade Federal de Goiás.

Levitin, A. (2012). *Introduction to the design & analysis of algorithms*. Boston: Pearson,.

Stamm-Wilbrandt, H. (1993). Programming in propositional logic or reductions: Back to the roots (satisability). Technical report, Institut für Informatik III, Universität Bonn.