Corte Máximo em Grafos

Notas de aula de MAC-5727

(Material Extra do Capítulo 2)

Prof.ª Yoshiko Wakabayashi

– Versão pós-aula feita por Bruno Takahashi C. de Oliveira em 09/03/09 –

15 de agosto de 2016

1 Definições básicas e notação

Neste texto vamos considerar apenas grafos sem laços.

Seja $G = (V, E)$ um grafo. Seja $X \subset V$ (i.e. um subconjunto dos vértices do grafo) e seja $\bar{X}$ seu complemento. Denotamos por $\delta(X)$ o conjunto de todas as arestas de G que têm uma ponta em X e a outra em $\bar{X}$. Ou seja,

$$\delta(X) := \left\{ e = \{x, y\} \in E \mid x \in X,\ y \in \bar{X} \right\}.$$

Definimos um **corte** (*cut*) de um grafo $G = (V, E)$ como um subconjunto de arestas da forma $\delta(X)$ para algum $X \subset V$. Dizemos que um corte C de um grafo é **máximo** (resp. **mínimo**) se o número de arestas em C é máximo (resp. mínimo), ou seja, se não existe no grafo um corte com mais (resp. menos) arestas do que C.

Para simplificar a notação, se v é um vértice de um grafo, escrevemos $\delta(v)$ em vez de $\delta(\{v\})$.

Denotamos por $g(v)$ o **grau** de um vértice v, e observamos que $g(v) = |\delta(v)|$.

2 O problema

O problema do Corte Máximo é o seguinte:

Problema MaxCut (G): *Dado um grafo $G = (V, E)$, encontrar em G um corte máximo.*

Este problema é NP-difícil [1]. Aproveitamos para lembrar que o problema do *corte mínimo* pode ser resolvido de maneira exata em tempo polinomial através de algoritmos que encontram fluxo máximo.

Veremos a seguir dois algoritmos bem simples para o MaxCut. Um algoritmo mais sofisticado para este mesmo problema será visto no Capítulo 7 do livro-texto.

3 Algoritmo de busca local

A ideia por trás de algoritmos baseados em *busca local* é a seguinte: começa-se com uma solução viável qualquer e então tenta-se encontrar uma outra solução melhor, fazendo algum passo de mudança local. Repete-se o processo até que não seja mais possível encontrar uma solução melhor; finaliza devolvendo a solução obtida até então.

No caso do problema em foco, explicando de maneira intuitiva, temos o seguinte algoritmo: iniciamos com um conjunto arbitrário X de vértices. A cada iteração, verificamos se "vale a pena" transferir um vértice de X para $\bar{X}$ ou transferir um vértice de $\bar{X}$ para X, e, se detectamos que essa mudança aumenta o tamanho do corte, fazemos a mudança. Repetimos esse processo até o momento em que não é possível obter nenhuma melhora.

Mais formalmente, temos o seguinte algoritmo.

Algoritmo MaxCut-BuscaLocal (G)

1 Tome arbitrariamente $X_1 \subset V$, $i \leftarrow 0$.

2 $i \leftarrow i + 1$.

3 Se $\exists v \in \bar{X}_i$ tal que $\delta(X_i \cup \{v\}) > \delta(X_i)$,
  então $X_{i+1} \leftarrow X_i \cup \{v\}$ e volte ao passo 2.

4 Se $\exists v \in X_i$ tal que $\delta(X_i \setminus \{v\}) > \delta(X_i)$,
  então $X_{i+1} \leftarrow X_i \setminus \{v\}$ e volte ao passo 2.

5 Devolva $\delta(X_i)$.

Note que, após o algoritmo terminar, se $\delta(X)$ é o corte devolvido pelo algoritmo, então nenhum vértice de X (resp. $\bar{X}$) pode ter mais arestas "internas", i.e., arestas ligando-o a vértices de X (resp. $\bar{X}$) do que arestas "de corte", i.e., arestas ligando-o a vértices de $\bar{X}$ (resp. X). De fato, senão teria sido vantajoso mover um tal vértice para $\bar{X}$ (resp. X), e portanto o algoritmo tê-lo-ia feito. Logo, para qualquer vértice v em V, e lembrando que $g(v)$ denota o grau do vértice, temos

$$|\delta(v) \cap \delta(X)| \geq \frac{g(v)}{2}.$$

Ou seja, cada vértice v contribui para o corte com pelo menos $g(v)/2$ arestas. Este fato é crucial na análise do desempenho do algoritmo que faremos a seguir.

Teorema 1: *O algoritmo* MaxCut-BuscaLocal (G) *é uma 1/2-aproximação polinomial para o problema* MaxCut(G).

Demonstração: Primeiramente observamos que

$$opt(G) \leq |E|. \tag{1}$$

Para mostrar que o algoritmo é polinomial, observe que o número de arestas no corte inicial é $|\delta(X_1)|$, e este valor vai sendo incrementado de pelo menos uma unidade. Como $opt(G) \leq |E|$, então esse número inicial será incrementado no máximo $|E|$ vezes. Além disso, observe que para fazer cada incremento, é necessário visitar no máximo todos os vértices do grafo. Logo, o algoritmo é polinomial.

Seja $\delta(X)$ o corte devolvido pelo algoritmo. Como vimos anteriormente, temos que

$$|\delta(v) \cap \delta(X)| \geq \frac{g(v)}{2} \text{ para todo vértice } v. \tag{2}$$

Note também que

$$\delta(X) = \frac{1}{2} \sum_{v \in V} |\delta(v) \cap \delta(X)|. \tag{3}$$

De fato, para contar as arestas do corte $\delta(X)$, é suficiente somar apenas as "contribuições" que cada vértice de X (resp. $\bar{X}$) dá ao corte. Como na somatória do lado direito de (3) estamos incluindo as "contribuições" dos vértices de X e também dos vértices de $\bar{X}$, então cada aresta do corte é contada duas vezes.

Combinando as desigualdades (2) e (3) obtemos:

$$\delta(X) \geq \frac{1}{2} \sum_{v \in V} \frac{g(v)}{2}.$$

Mas, em qualquer grafo, a soma dos graus de seus vértices é igual a duas vezes o número de suas arestas. Utilizando esse fato na desigualdade acima, e combinando com a delimitação dada em (1) concluímos que

$$\delta(X) \geq \frac{|E|}{2} \geq \frac{1}{2}\, opt(G).$$

□

4 Algoritmo guloso

Apresentamos agora uma outra ideia bastante usada para projetar algoritmos. Neste segundo algoritmo, num passo genérico temos dois conjuntos de vértices X e Y, que definem um corte formado pelas arestas que vão de X a Y, visitamos o próximo vértice v (seguindo uma ordem pré-estabelecida) e decidimos em qual dos conjuntos X ou Y devemos colocar esse vértice de forma a maximizar o corte que se obtém, no grafo definido pelos vértices $X \cup Y \cup \{v\}$. No final, devolvemos $\delta(X)$.

Antes de iniciar, faremos as seguinte definição. Se $G = (V, E)$ é um grafo, então dados dois conjuntos $X, Y \subset V$, tais que $X \cap Y = \emptyset$, definimos $\beta(X, Y)$ como o número de arestas em G que têm uma ponta num dos vértices de X e outra ponta num dos vértices de Y. Ou, mais formalmente,

$$\beta(X, Y) = |\{e = \{v, u\} \in E \mid v \in X,\ u \in Y\}|.$$

Utilizaremos também a seguinte notação simplificada: quando qualquer um dos parâmetros da função β for um conjunto unitário, eliminamos o uso das chaves. Ou seja, se v é um vértice então escrevemos $\beta(v, X)$ em vez de $\beta(\{v\}, X)$.

Observe também a seguinte propriedade. Se $v \notin X$ e $v \notin Y$, então

$$\beta(X \cup \{v\}, Y) = \beta(X, Y) + \beta(v, Y). \tag{4}$$

Estamos agora prontos para descrever mais formalmente o algoritmo.

Algoritmo MaxCut-Guloso (G):
0 Considere uma ordenação qualquer dos vértices de G: $v_1, v_2, \ldots v_n$.
1 $X_1 \leftarrow v_1, Y_1 \leftarrow \emptyset$.
2 Para cada $i = 2, 3, \ldots, n$,
3 se $\beta(v_i, Y_{i-1}) \geq \beta(v_i, X_{i-1})$,
4 então $X_i \leftarrow X_{i-1} \cup v_i$, $Y_i \leftarrow Y_{i-1}$;
5 senão, $X_i \leftarrow X_{i-1}$, $Y_i \leftarrow Y_{i-1} \cup v_i$.
6 Devolva $\delta(X_n)$.

Análise do Algoritmo Guloso

Para os propósitos da análise, faremos a seguinte definição. Para cada aresta, definimos o **vértice responsável** por essa aresta como sendo a sua ponta de maior índice. Ou seja, se $e = \{v_i, v_j\}$, e $i < j$, então v_j é o vértice responsável pela aresta e. Portanto, cada aresta tem exatamente um vértice responsável.

Para cada vértice v_i, definimos r_i como o **número de arestas pelas quais v_i é responsável**. Então, é imediato que

$$\sum_{i=1}^{n} r_i = |E|. \tag{5}$$

Note que, em qualquer iteração i do algoritmo, todos os vértices de X_{i-1} e de Y_{i-1} têm índices menores do que v_i, pois foram processadas antes. Logo, todas as arestas que ligam v_i aos vértices de X_{i-1} e Y_{i-1} são de *responsabilidade* de v_i (conforme a definição de "responsabilidade" dada anteriormente). Isto nos leva à seguinte observação:

$$r_i = \beta(v_i, X_{i-1}) + \beta(v_i, Y_{i-1}).$$

Logo, como em qualquer soma, se uma das parcelas é maior ou igual à outra, certamente esta parcela é maior ou igual à metade da soma. Ou seja:

$$\text{se } \beta(v_i, Y_{i-1}) \geq \beta(v_i, X_{i-1}), \text{ então } \beta(v_i, Y_{i-1}) \geq r_i/2. \tag{6}$$

O análogo vale para $\beta(v_i, X_{i-1})$.

O foco da nossa análise será mostrar que, a cada iteração, o número de arestas no corte aumenta de pelo menos $r_i/2$. Então, no total, o corte terá no mínimo $\sum r_i/2 = |E|/2$ arestas, que é a metade do limitante do ótimo (já que $opt(G) \leq |E|$).

Separamos a análise em dois casos. Utilizamos neste raciocínio os resultados (4) e (6) apresentados anteriormente.

Caso 1: $\beta(v_i, Y_{i-1}) \geq \beta(v_i, X_{i-1})$. Neste caso temos:

$$\begin{aligned} \beta(v_i, Y_{i-1}) &\geq r_i/2 \\ \beta(v_i, Y_{i-1}) + \beta(X_{i-1}, Y_{i-1}) &\geq r_i/2 + \beta(X_{i-1}, Y_{i-1}) \\ \beta(X_{i-1} \cup v_i, Y_{i-1}) &\geq r_i/2 + \beta(X_{i-1}, Y_{i-1}) \\ \beta(X_i, Y_i) &\geq r_i/2 + \beta(X_{i-1}, Y_{i-1}). \end{aligned}$$

Caso 2: $\beta(v_i, Y_{i-1}) < \beta(v_i, X_{i-1})$. Neste caso temos:

$$\begin{aligned} \beta(v_i, X_{i-1}) &\geq r_i/2 \\ \beta(v_i, X_{i-1}) + \beta(X_{i-1}, Y_{i-1}) &\geq r_i/2 + \beta(X_{i-1}, Y_{i-1}) \\ \beta(X_{i-1}, Y_{i-1} \cup v_i) &\geq r_i/2 + \beta(X_{i-1}, Y_{i-1}) \\ \beta(X_i, Y_i) &\geq r_i/2 + \beta(X_{i-1}, Y_{i-1}). \end{aligned}$$

Em ambos os casos, chegamos à mesma conclusão, ou seja, a de que:

$$\beta(X_i, Y_i) \geq r_i/2 + \beta(X_{i-1}, Y_{i-1}).$$

Vimos então que em cada iteração do algoritmo o tamanho do corte gerado é pelo menos $r_i/2$ unidades maior que o anterior. Então temos que

$$\beta(X_n, Y_n) \geq \sum_{i=1}^{n} r_i/2.$$

Usando (5) na desigualdade acima, e o fato de que $opt(G) \leq |E|$, temos que

$$\beta(X_n, Y_n) \geq \frac{|E|}{2} \geq \frac{1}{2}\, opt(G).$$

E, naturalmente, como no final Y_n é o complemento de X_n, então $\delta(X_n)$ é um corte do grafo G com $\beta(X_n, Y_n)$ arestas. Logo o corte $\delta(X_n)$ devolvido pelo algoritmo tem ao menos $1/2\, opt(G)$ arestas. Com isso, demonstramos que o algoritmo MaxCut-Guloso é uma 1/2-aproximação para o problema MaxCut.

Referências

[1] R.M. Karp. Reducibility among combinatorial problems. In R.E. Miller and J.M. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum, 1972.