MO829 - Resumo de Artigo

Hugo Kooki Kasuya Rosado - 091539

29 de novembro de 2016

Resumo do Resumo

Este documento resume os resultados obtidos por J. Kneis, A. Langer e P. Rossmanith em "*A New Algorithm for Finding Trees with Many Leaves*"[9]. O artigo apresenta algoritmos parametrizados para problemas conhecidos como Maximum Leaf Spanning Tree cuja complexidade é $O(poly(|V|) + 4^k k^2)$ para grafos não-direcionados e $O(4^k|V||E|)$ para grafos direcionados.

1 Introdução

O artigo considera o problema teórico de se encontrar árvores e árvores geradoras em grafos tal que o número de folhas seja máximo. Para grafos direcionados, procuramos por *out-trees* ou *out-trees* geradoras. Definimos uma out-tree como uma árvore enraizada tal que existe um caminho direcionado da raíz até cada folha e dizemos que ela é geradora se existe caminho direcionado da raiz até cada vértices de G. O caso geral desse problema é classificado como APX-difícil [8] e atualmente existe, para o caso não-direcionado, uma 2-aproximação [12] e uma 3-aproximação com tempo quase linear [10]. Também há uma 3/2-aproximação para grafos cubos [3].

Na versão parametrizada do problema, procuramos decidir se existe uma árvore com pelo menos k folhas. Formalmente, o artigo ataca os três seguintes problemas:

Maximum Leaf Spanning Tree (MLST):
Entrada: Um grafo $G = (V, E)$ e $k \in \mathbb{N}$
Parâmetro: k
Problema: G contém uma árvore geradora com pelo menos k folhas?

Directed Maximum Leaf Out-Tree (DMLOT):
Entrada: Um digrafo $G = (V, E)$ e $k \in \mathbb{N}$
Parâmetro: k
Problema: G contém uma out-tree com pelo menos k folhas?

Directed Maximum Leaf Spanning Out-Tree (DMLST):
Entrada: Um grafo $G = (V, E)$ e $k \in \mathbb{N}$
Parâmetro: k
Problema: G contém uma out-tree geradora com pelo menos k folhas?

Note que, em um grafo não-direcionado, é fácil transformar uma árvore com pelo menos k folhas em uma árvore geradora com k folhas (se existe tal árvore geradora). Porém, em um grafo direcionado, o mesmo processo não garante uma árvore geradora; por isso a necessidade de se distinguir os problemas.

Sabemos, pelo teoremoa de Robertson e Seymor sobre minors, que MLST $\in FPT$ (exercício 6.15 da lista de exercícios 3). Porém, esse só garante a existência de um algoritmo $O(f(k)|V|^3)$. O primeiro algoritmo explícito para resolver o MLST foi criado por Boadlaender e utiliza técnicas com treewidth e possui complexidade de $O((17k^4)!|G|)$ [1]. Melhores resultados foram surgindo com o tempo, Downey e Fellows melhoraram a complexidade para $O((2k)^{4k}poly(G))$ [6]; Fellows, McCartin, Rosamond e Stege conseguiram $O(|G| + 14.23^k k)$ usando resultados de teoria de grafos muito sofisticados [11]; Bonsma, Brueggermann e Woeginger, $O(|V|^3 + 9.4815^k k^3)$ [2]; Estivill-Castro desenvolveu um kernel de tamanho $3.75k$ [7] - o que diretamente reduz a complexidade do algoritmo de Bonsma para $O(|V|^3 + 8.12^k)$; e Bonsma e Zickfield obtiveram $O(poly(|V|) + 6.75^k poly(k))$ [4].

Em relação ao DMLOT e ao DMLST, Bonsma e Dorn provaram que ambos eram FPT, utilizando técnicas de pathwidth, com complexidade de $O(2^{O(klog(k))}poly(|V|))$ [2]. No artigo que resumimos [9], Kneis, Langer e Rossmanith provam um algoritmo parametrizado $O(poly(|V|) + 4^k k^2)$ para o MLST e um $O(4^k|V||E|)$ para o DMLOT e DMLSOT. Mais recentemente, Daligault, Gutin, Kim e Yeo conseguiram um algoritmo $O(3.72^k|V|^{O(1)})$ com uma modificação do algoritmo de Kneis [5] para os mesmos problemas.

2 Preliminares

Seja $G = (V, E)$ um grafo, tomamos então que $n := |V|$ e $m := |E|$. Sem perda de generalidade, assumiremos que $k > 2$; c.c., o problema seria trivial. E, em um grafo direcionado, tomamos como vizinho de um vértice v todos os vértices u tal que $(v, u) \in E$. Seja T uma árvore (out-tree) de G, denotamos por $root(T)$ a raiz de T, $leaves(T)$ as folhas de T e por $inner(T) := V(T) - leaves(T)$ os vértices internos de T. Denotamos também por $N_{\overline{T}}(v) := N(v) - V(T)$ os vizinhos de v que não estão em T e por $T_v := (N[v], \cup_{u \in N(v)}\{(v, u)\})$ a estrela de centro v e seus vizinhos.

Se T é uma árvore com pelo menos k folhas, chamamos T de árvore de k-folhas. Se $N(inner(T)) \subseteq V(T)$, então dizemos que T é uma árvore inner-maximal, i.e., T é uma árvore que só pode crescer pelas suas folhas. Uma árvore rotulada é um 3-tupla (T, R, B) tal que T é uma árvore, e R e B particionam $leaves(T)$. Dizemos que uma árvore $T' \neq T$ estende T, denotando por $T' \succ T$ sss $root(T) = root(T')$ e T for um grafo induzido de T'. Uma árvore rotulada (T, R, B) é (folha-preservativa) estendida por T' se $T' \succ T$, $R \subseteq leaves(T')$ e denotamos por $T' \succ (T, R, B)$. Também dizemos que uma árvore rotulada (T', R', B') estende (T, R, B) sss $T' \succ (T, R, B)$. Em suma, (T, R, B) representa uma árvore T cuja folhas de R estão fixas, B são folhas potenciais - que podem ir para R ou virar vértices internos - e uma árvore T' estende (T, R, B) se $T' \succ T$ e preserva as folhas já fixadas em R.

3 Lemas

Apresentaremos nessa sessão diversos lemas que ajudarão a provar a complexidade do algoritmo apresentado no artigo. Como todos os lemas utilizados são bem intuitivos e simples, ou mesmo triviais, não apresentamos suas respectivas demonstrações.

Lema 3.1. *Seja (T, R, B) uma árvore rotulada inner-maximal e seja $T' \succ (T, R, B)$, então $B \neq \emptyset$.*

Lema 3.2. *Seja $G = (V, E)$ um grafo conexo não-direcionado, então G contém uma árvore de k-folhas sss G contém uma árvore de k-folhas geradora com k folhas.*

Lema 3.3. *Seja $G = (V, E)$ um grafo direcionado. Se G contém uma out-tree de k-folhas geradora enraizada em v, então qualquer out-tree de k-folhas enraizada em v pode ser expandida para uma out-tree de k-folhas geradora de G.*

Lema 3.4. *Seja $G = (V, E)$ um grafo, (T, R, B) uma árvore rotulada e $x \in B$. Então:*

1) *Se não há árvore de k-folhas T', tal que, $T' \succeq (T, R \cup \{x\}, B - \{x\})$, então toda árvore de k-folhas T' que estende (T, R, B) tem que $x \in inner(T')$.*

2) *Se existe árvore de k-folhas T', tal que $T' \succeq (T, R, B)$ e $x \in inner(T')$, então existe árvore de k-folhas $T'' \succeq (T + \{(x, y) : y \in N_{\overline{T}}(x)\}, R, N_{\overline{T}}(x) \cup B - \{x\})$.*

Lema 3.5. *Seja $G = (V, E)$ um grafo, (T, R, B) uma árvore de k-folhas e $x \in B$ tal que $N_{\overline{T}}(x) = \{y\}$. Se não existe árvore de k-folhas que estenda $(T, R \cup \{x\}, B - \{x\})$, então não existe árvore de k-folhas que estenda $(T + (x, y), R \cup \{y\}, B - \{x\})$.*

Corolário 3.6. *Seja $G = (V, E)$ um grafo, (T, R, B) uma árvore de k-folhas e $x \in B$. Se $N_{\overline{T}}(x) = \emptyset$ e se existe uma árvore de k-folhas que estenda (T, R, B), então existe árvore de k-folhas que estende $(T, R \cup \{x\}, B - \{x\})$.*

4 O Algoritmo

Algorithm 1 MaxLeaf(G, T, R, B, k)

Entrada: Um grafo G, uma árvore inner-maximal (T, R, B), $k \in \mathbb{N}$
Saída: Existe árvore de k-folhas $T' \succ (T, R, B)$?

1: **if** $|R| + |B| \geq k$ **then return** "SIM"
2: **if** $B = \emptyset$ **then return** "NAO"
3: Escolha um $u \in B$
//Tentar branch em que u é folha
4: **if** MaxLeaf$(G, T, R \cup \{u\}, B - \{u\}, k)$ retorna "SIM" **then return** "SIM"
//Se u não é folha, u deve ser interno em todas as soluções
5: $B \leftarrow B - \{u\}$
6: $N \leftarrow N_{\overline{T}}(u)$
7: $T \leftarrow T \cup \{(u, u') : u' \in N\}$
//Crescer caminho único: Lema 3.5
8: **while** $|N| = 1$ **do**
9: Seja $N = \{v\}$
10: $N \leftarrow N_{\overline{T}}(v)$
11: $T \leftarrow T \cup \{(v, v') : v' \in N\}$
//Não crie branch se não há mais vizinhos: Corolário 3.6
12: **if** $N = \emptyset$ **then return** "NAO"
13: **return** MaxLeaf$(G, T, R, B \cup N, k)$

O algoritmo MaxLeaf apresentado é recursivo e a sua complexidade é calculada limitando-se a árvore de busca. O algoritmo decide se há árvore de k-folhas em G enraizado em um dado vértice v. Assim, executamos o algoritmo para cada vértice de V para decidir a existência de tal árvore.

Lema 4.1. *Seja $G = (V, E)$ um grafo e $k > 2$. Se G não contém uma árvore de k-folhas,* MaxLeaf(G, T, R, B, k) *retorna "NAO"para cada $v \in V$. Se G contém tal árvore enraizada em r, então* MaxLeaf$(G, T_r, \emptyset, N(r), k)$ *retorna "SIM".*

Demonstração. Primeiramente mostramos que toda chamada de MaxLeaf é sempre feita com uma árvore inner-maximal rotulada (T, R, B). A estrela T_r é trivialmente inner-maximal, então $(T_r, \emptyset, N(r))$ é uma árvore inner-maximal rotulada. Seja então (T, R, B) inner-maximal. Se $u \in B$ é fixada como folha, então $(T, R \cup \{u\}, B - \{u\}) \succ (T, R, B)$ é inner-maximal (chamada da Linha 4). Caso contrário, u se torna interno de uma árvore T' obtida incluindo-se os vértices $N_{\overline{T}}(u)$ como folhas. Como $N(inner(T')) = N(inner(T)) \cup N(u) \subseteq V(T) \cup N(u) = V(T')$, temos que T' é inner-maximal, logo $(T', R, N_{\overline{T}}(u) \cup B - \{u\})$ também é. Em especial, esse passo também vale enquanto $|N_{\overline{T}}(u)| = 1$, resultando em uma sequência de árvores inner-maximais $(T, R, B) \prec (T', R', B') \prec ... \prec (T^{(l)}, R^{(l)}, B^{(l)})$. Logo, na Linha 13, a árvore do argumento de MaxLeaf também é inner-maximal.

MaxLeaf(G, T, R, B, k) só retorna "SIM"se $|leaves(T)| = |R| + |B| \geq k$, logo G contém árvore de k-folhas e o algoritmo nunca responde SIM para instâncias-não. Caso contrário, se G contém uma árvore de k-folhas enraizada em r, mostraremos que o MaxLeaf é chamado com uma árvore de k-folhas eventualmente usando indução sob a seguinte hipótese: se (T, R, B) é inner-maximal tal que existe árvore de k-folhas T' que estende T, então provamos que: Ou $T = T'$; ou existe árvore de k-folhas (T''', R''', B'''), uma árvore rotulada (T'', R'', B'') tal que $(T''', R''', B''') \succeq (T'', R'', B'') \succ (T, R, B)$ e MaxLeaf é chamada com (T'', R'', B''). Como G é finito, eventualmente obtemos uma árvore de k-folhas.

Seja r a raiz de uma árvore de k-folhas T em G. Como $k > 2$, podemos assumir que $r \in inner(T)$ (mesmo se o grafo for não-direcionado). Considere $T' = (\{r\}, \emptyset)$, então $(T', \emptyset, \{r\})$ é uma árvore rotulada e $T \succ (T', \emptyset, \{r\})$. Logo pelo Lema 3.4 existe uma árvore de k-folhas $T'' \succ (T_r, \emptyset, N(r))$.

Seja agora (T, R, B) inner-maximal a árvore do argumento de MaxLeaf tal que existe $T' \succeq (T, R, B)$. Se $|leaves(T)| = |R| + |B| \geq k$, então o algoritmo responde "SIM"corretamente.

Caso contrário $B \neq \emptyset$ (Lema 3.1), pois (T,R,B) é inner-maximal. Fixe um $u \in B$ e, segundo o Lema 3.4, temos dois casos: (1) existe árvore de k-folhas $T''' \succeq (T, R \cup \{u\}, B - \{u\})$ ou (2) existe árvore de k-folhas $T''' \succeq (T + \{(u, y) : y \in N_{\overline{T}}(u)\}, R, N_{\overline{T}}(u) \cup B - \{u\})$.

> Se caso (1): Então $T''' \succeq (T, R \cup \{u\}, B - \{u\}) \succ (T, R, B)$ e temos que por H.I. MaxLeaf$(G, T, R \cup \{u\}, B - \{u\}, k)$ retorna "SIM".
>
> Se caso (2): Como existe T' que estende (T,R,B), então existe uma árvore T''' que estende $(T + \{(u, y) : y \in N_{\overline{T}}(u)\}, R, N_{\overline{T}}(u) \cup B - \{u\})$ (Lema 3.4). Para concluir esse caso, notemos o *while* da linha 8. Segundo o Lema 3.5, existe uma sequência de vértices $u = v_0, v_1, ..., v_l$ e de árvores rotuladas $(T, R, B) = (T_0, R_0, B_0), ..., (T_l, R_l, B_l)$ tal que:
>
> A: $(T_{i+1}, R_{i+1}, B_{i+1}) = (T_i + (v_i, v_{i+1}), R_i, B_i \cup N_{\overline{T_i}}(v_i) - \{v_i\})$,
> B: $N_{\overline{T_i}}(v_i) = \{v_{i+1}\}$ para $0 \leq i \leq l$,
> C: $|N_{\overline{T_i}}(v_i)| \neq 1$,
> D: Para cada $o \leq i \leq l$, existe árvore de k-folhas $T'_i \succeq (T_i, R_i, B_i)$.
>
> Segundo o Corolário 3.6, temos que $N_{\overline{T_i}}(v_i) \neq \emptyset$, logo o algoritmo não retorna "NAO". Logo MaxLeaf(G, T_l, R_l, B_l, k) retorna a resposta certa pois (T_l, R_l, B_l) satisfaz a hipótese.

□

Lema 4.2. *Seja $G = (V, E)$ um grafo e $v \in V$. O número de chamadas recursivas do algoritmo* MaxLeaf *com a entrada $(G, T_v, \emptyset, N(v), k)$ é da ordem de $O(2^{2k-|N(v)|})$.*

Demonstração. Seja a função potencial $\phi(k, R, B) := 2k-2|R|-|B|$. Quando chamado com uma árvore rotulada (T, R, B), o algoritmo se invoca recursivamente no máximo duas vezes. Note que a chamada da linha 4 reduz o potencial em $\phi(k, R, B)-\phi(k, R\cup\{u\}, B-\{u\}) = 1$ e a chamada da linha 13 (se caso alcançada temos $|N| \geq 2$) reduz o potencial em pelo menos $\phi(k, R, B) - \phi(k, R, N \cup B - \{u\}) \geq 1$.

Como $\phi(k, R, B) \leq 0$ implica em $|R + B| \geq k$ e temos que o potencial diminui em pelo menos 1 a cada chamada recursiva, a altura da árvore de busca é de no máximo $\phi(k, R, B) \leq 2k$.

Para uma árvore inner-maximal (T, R, B), o número de chamadas recursivas é então da ordem de $O(2^{\phi(k,R,B)})$. Como na primeira chamada do algoritmo temos explicitamente que $|B| = |N(v)|$, temos que a complexidade do algoritmo é da ordem de $O(2^{\phi(k,\emptyset,N(v))}) = O(2^{2k-|N(v)|}) \subseteq O(4^k)$. □

Teorema 4.3. *MLST pode ser resolvido em $O(poly(n) + 4^k k^2)$.*

Demonstração. Seja $G = (V, E)$ um grafo não-direcionado. Utilizando o kernel de Estivill-Castro conseguimos, em tempo polinomial, limitar n em $3.75k \in O(k)$ [7].

Sem perda de generalidade, assumimos que o grafo é conexo e que $k > 2$. Apesar de não sabermos qual $v \in V$ é raiz de uma árvore de k-folhas, basta notar que ou v é a raiz de uma das tais árvores ou algum $u \in N(v)$ é raiz de uma dessas árvores (assumindo que a existência de uma dessas árvores). Seja então $v \in V$ o vértice de menor grau em G ($grau(v) = d$), executamos MaxLeaf$(G, T_u, \emptyset, N(u), k)$ para cada $u \in N[v]$. Pelo Lema 4.1, pelo menos uma das execuções retorna "SIM"sss existir uma árvore de k-folhas em G e, pelo Lema 4.2, o número total de chamadas recursivas é limitado por

$$O(2^{\phi(k,\emptyset,N(v))}) + \sum_{u\in N(v)} O(2^{\phi(k,\emptyset,N(u))}) \subseteq O((d+1)2^{2k-d}) = O\left(4^k \frac{d+1}{2^d}\right) \subseteq O(4^k)$$

Por causa do kernel, as Linhas 5-8 gastam $O(k)$; cada execução do *while* gasta $O(1)$ e é executado no máximo k vezes; e a concatenação de B e N na Linha 13 gasta $O(k)$.

Até esse ponto o algoritmo encontra uma árvore T de k-folhas em G em tempo $O(poly(n)+ 4^k k^2)$. Para encontrarmos uma árvore de k-folhas geradora, é suficiente executarmos um BFS para crescer T - o Lema 3.2 garante a existência dessa árvore. O BFS gasta mais $O(k^2)$ na sua execução, logo obtemos um tempo total de $O(poly(n) + 4^k k^2 + k^2) = O(poly(n) + 4^k k^2)$. □

Teorema 4.4. *DMLOT e DMLST podem ser resolvidos em $O(4^k nm)$.*

Demonstração. Seja $G = (V, E)$ um digrafo. Existe uma árvore T de k-folhas enraizada em $r \in V$ sss MaxLeaf$(G, T_r, \emptyset, N(r), k)$ retorna "SIM". Como não sabemos qual vértice é a raiz de T, executamos o algoritmo para cada $v \in V$:

$$\sum_{v\in V} O(2^{\phi(k,\emptyset,N(v))}) \subseteq O(n2^{2k}) = O(4^k n)$$

Como a cada chamada do algoritmo visitamos no máximo cada umas das m arestas, temos que a complexidade total é de $O(4^k nm)$ para o DMLOT.

Para resolvermos o DMLST tentarmos expandir cada uma das soluções encontradas para o DMLOT usando um BFS. O Lema 3.3 garante a corretude desse processo e aumenta a complexidade em mais $O(n(n + m)) = O(nm)$. Logo a complexidade total é de $O(4^k nm + nm) = O(4^k nm)$. □

5 Pergunta/Trívia

1. (☠☠☠) Se incluíssemos multigrafos nas instâncias do MLST e do DMLOT, como poderíamos modificar (se necessário) o algoritmo apresentado para que ele os resolva e qual seria a sua nova complexidade? (Dica: Essa questão não possui resposta única e lembre-se que você pode considerar estruturas de dados convenientes!)

Referências

[1] H.L. Bodlaender. On linear time minor tests with depth-first search. *Journal of Algorithms*, 14(1):1 – 23, 1993.

[2] Paul Bonsma and Frederic Dorn. *Tight Bounds and a Fast FPT Algorithm for Directed Max-Leaf Spanning Tree*, pages 222–233. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[3] Paul Bonsma and Florian Zickfeld. A 3/2-approximation algorithm for finding spanning trees with many leaves in cubic graphs. *SIAM Journal on Discrete Mathematics*, 25(4):1652–1666, 2011.

[4] Paul Bonsma and Florian Zickfeld. Improved bounds for spanning trees with many leaves. *Discrete Mathematics*, 312(6):1178 – 1194, 2012.

[5] Jean Daligault, Gregory Gutin, Eun Jung Kim, and Anders Yeo. Fpt algorithms and kernels for the directed k-leaf problem. *Journal of Computer and System Sciences*, 76(2):144 – 152, 2010.

[6] Rodney G. Downey and Michael R. Fellows. *Parameterized Computational Feasibility*, pages 219–244. Birkhäuser Boston, Boston, MA, 1995.

[7] Vladimir Estivill-Castro, Michael Fellows, Michael Langston, and Frances Rosamond. Fpt is p-time extremal structure i. In *Algorithms and Complexity in Durham 2005, Proceedings of the first ACiD Workshop, volume 4 of Texts in Algorithmics*, pages 1–41. King's College Publications, 2005.

[8] G. Galbiati, F. Maffioli, and A. Morzenti. A short note on the approximability of the maximum leaves spanning tree problem. *Information Processing Letters*, 52(1):45 – 49, 1994.

[9] Joachim Kneis, Alexander Langer, and Peter Rossmanith. *A New Algorithm for Finding Trees with Many Leaves*, pages 270–281. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[10] Hsueh-I Lu and R Ravi. Approximating maximum leaf spanning trees in almost linear time. *Journal of Algorithms*, 29(1):132 – 141, 1998.

[11] R. Fellows Michael, Catherine McCartin, A. Rosamond Frances, and Ulrike Stege. *Coordinatized Kernels and Catalytic Reductions: An Improved FPT Algorithm for Max Leaf Spanning Tree and Other Problems*, pages 240–251. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000.

[12] Roberto Solis-Oba, Paul Bonsma, and Stefanie Lowski. A 2-approximation algorithm for finding a spanning tree with maximum number of leaves. *Algorithmica*, pages 1–15, 2015.