Contents lists available at ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

Deeper local search for parameterized and approximation algorithms for maximum internal spanning tree[☆]

Wenjun Li[a,b,1], Yixin Cao[c,2], Jianer Chen[d], Jianxin Wang[a,*,1]

[a] School of Information Science and Engineering, Central South University, Changsha, China
[b] School of Computer and Communication Engineering, Hunan Provincial Key Laboratory of Intelligent Processing of Big Data on Transportation, Changsha University of Science and Technology, Changsha, China
[c] Department of Computing, Hong Kong Polytechnic University, Hong Kong, China
[d] Department of Computer Science and Engineering, Texas A&M University, College Station, TX, United States

ARTICLE INFO

Article history:
Received 14 May 2015
Received in revised form 27 September 2016
Available online 14 November 2016

Keywords:
Maximum internal spanning tree
Parameterized computation
Local search

ABSTRACT

The maximum internal spanning tree problem asks for a spanning tree of a given graph that has the maximum number of internal vertices among all spanning trees of this graph. In its parameterized version, we are interested in whether the graph has a spanning tree with at least k internal vertices. Fomin et al. (2013) [4] crafted a very ingenious reduction rule, and showed that a simple application of this rule is sufficient to yield a $3k$-vertex kernel, implying an $O^*(8^k)$-time parameterized algorithm. Using depth-2 local search, Knauer and Spoerhase (2015) [9] developed a (5/3)-approximation algorithm for the optimization version. We try deeper local search: We conduct a thorough combinatorial analysis on the obtained spanning trees and explore their algorithmic consequences. We first observe that from the spanning tree obtained by depth-3 local search, one can easily find a reducible structure and apply the reduction rule of Fomin et al. This gives an improved kernel of $2k$ vertices, and as a by-product, a deterministic algorithm running in time $O^*(4^k)$. We then go even deeper by considering the spanning tree obtained by depth-5 local search. It is shown that the number of internal vertices of this spanning tree is at least 2/3 of the maximum number a spanning tree can have, thereby delivering an improved approximation algorithm with ratio 1.5 for the problem.

1. Introduction

Spanning tree is a fundamental concept in graph theory, and finding a spanning tree of the input graph is a routine step of graph algorithms, though it usually induces no extra cost: Most algorithms will start from exploring the input graph anyway, and both breadth- and depth-first-search procedures produce a spanning tree as a by-product. However, a graph can have an exponential number of spanning trees, of which some might suit a specific application better than others. We are hence asked to find constrained spanning trees, i.e., spanning trees minimizing or maximizing certain objective functions.

[☆] Extended abstracts of this paper appeared in the proceedings of ESA 2014 [13] and WADS 2015 [14].
[*] Corresponding author.
E-mail addresses: liwenjun@csu.edu.cn (W. Li), yixin.cao@polyu.edu.hk (Y. Cao), chen@cse.tamu.edu (J. Chen), jxwang@mail.csu.edu.cn (J. Wang).
[1] Supported by the National Natural Science Foundation of China (NSFC) under grants 61420106009, 61232001, 61472449, 61672536, and 61502054.
[2] Supported in part by the National Natural Science Foundation of China (NSFC) under grant 61572414 and the Hong Kong Research Grants Council (RGC) under grant 252026/15E.

http://dx.doi.org/10.1016/j.ic.2016.11.003
0890-5401/© 2016 Elsevier Inc. All rights reserved.

The most classic example is the minimum weight spanning tree problem (in weighted graphs), which has an equivalent but less known formulation, i.e., maximum weight spanning tree. Other constraints that have received wide attention include minimum diameter spanning tree [7], degree constrained spanning tree [10,11], maximum leaf spanning tree [16], and maximum internal spanning tree [22]. Unlike the minimum weight spanning tree problem [8], most of these constrained versions are NP-hard, even on unweighted graphs [18].

The optimization objective we consider in the present paper is to maximize the number of internal vertices (i.e., non-leaf vertices) of the spanning tree, or equivalently, to minimize the number of its leaves. More formally, the *maximum internal spanning tree* problem asks for a spanning tree of a given graph G that has the maximum number of internal vertices among all spanning trees of G. Containing the Hamiltonian path problem as a special case, it is clearly NP-hard. A very effective approach applied to this problem is local search, or more specifically, by *edge swappings* defined as follows. Let T be a spanning tree of G. If we pick an edge e of the graph that is not used by T and replace by e any other edge in the unique cycle in $T + e$, then we get another spanning tree of G [9,19,23,21]. Some edge swapping may increase the number of internal vertices, which we apply exhaustively. Prieto and Sloper [19] showed that this simple strategy is already sufficient to achieve a 2-approximation algorithm for the maximum internal spanning tree problem.

An edge swapping can be regarded as a basic step for the local search process of *depth*-1 on spanning trees. It is reasonable to consider local search processes of depth-t with $t > 1$, which examine a sequence of up to t consecutive edge swappings to seek a possible improvement on a spanning tree. Intuitively, a local search process of larger depth may result in a better spanning tree. This is indeed the idea used by Knauer and Spoerhase [9], who proved that a local search process of depth-2 leads to a (5/3)-approximation algorithm for maximum internal spanning tree. We also remark that the (7/4)-approximation algorithm on graphs without degree-1 vertices by Salamon [21] uses depth-3. Although a local search process with an even larger depth seems conceivably to lead to further improved spanning trees, it also presents a great (if not formidable) challenge for the analysis to confirm the improvement. To see how the depth complicates the analysis, the reader may compare the analysis (of a few lines) for the depth-1 local search process [19,23] with the analysis (of more than 5 pages) for the depth-2 local search process [9].

We also study the parameterized version of the maximum internal spanning tree problem, which asks whether a given graph G has a spanning tree with at least k internal vertices, and is known as the k-*internal spanning tree* problem. Given an instance (G, k) of k-internal spanning tree, a *kernelization algorithm* produces in polynomial time an equivalent instance (G', k') such that $k' \le k$ and that the *kernel size* (i.e., the number of vertices in G') is upper bounded by some function of k'. Prieto and Sloper [19] presented an $O(k^3)$-vertex kernel for the problem, and improved it to $O(k^2)$ in the journal version [20]. Fomin et al. [4] crafted a very ingenious reduction rule, and showed that a simple application of this rule is sufficient to yield a $3k$-vertex kernel.

A nonempty independent set X (i.e., a subset of vertices that are pairwise nonadjacent in G) as well as its neighborhood are called a *reducible structure* if $|X|$ is at least twice as the cardinality of its neighborhood. To apply the reduction rule one needs a reducible structure. The observation in [4] is that the leaves of a depth-first-search tree T are necessarily independent. Therefore, if the graph has at least $3k - 3$ vertices, then either the problem has been solved (when T has k or more internal vertices), or the set of (at least $2k - 2$) leaves of T will be the required independent set. It is, however, very nontrivial to find a reducible structure when G has less than $3k - 3$ vertices, and this will be the focus of the first part of this paper. We apply first a depth-3 local search process to produce a local-optimal spanning tree T of the input graph. A nontrivial analysis tells us that if T has more leaves than internal vertices, then a subset of the leaves of T and its neighborhood make the reducible structure. This enables us to apply the reduction rule and claim a $2k$-vertex kernel, resolving a question asked in [4].

Theorem 1.1. *The k-internal spanning tree problem has a $2k$-vertex kernel.*

Priesto and Sloper [19,20] also initiated the study of parameterized algorithms (i.e., algorithms running in time $O(f(k) \cdot n^{O(1)})$ for some function f independent of n)[3] for k-internal spanning tree, which have undergone a sequence of improvements. This line of research is closely related to the k-*internal out-branching* problem, which, given a *directed* graph G and a parameter k, asks if G has an out-branching (i.e., a directed spanning tree having exactly one vertex of in-degree 0) with at least k vertices of positive out-degrees. It is known that any $O^*(f(k))$-time algorithm for k-internal out-branching can solve k-internal spanning tree in the same time—replacing every edge by two arcs of opposite directions, calling the algorithm for k-internal out-branching, and then dropping the directions from the obtained out-branching,—but not necessarily the other way. After a successive sequence of studies [6,2,5,25,3], the current best deterministic and randomized parameterized algorithms for k-internal out-branching run in time $O^*(6.86^k)$ and $O^*(4^k)$ respectively, which are also the best known for k-internal spanning tree. Table 1 summarizes the history of this line of research.

The $O^*(4^k)$-time *randomized* algorithm for k-internal out-branching [3, Theorem 180] was obtained using a famous algebraic technique developed by Koutis and Williams [12], which, however, is very unlikely to be derandomized. As a corollary of Theorem 1.1, we obtain an $O^*(4^k)$-time deterministic algorithm for k-internal spanning tree,—it suffices to apply the $O^*(2^n)$-time algorithm of Nederlof [17] to the $2k$-vertex kernel produced by Theorem 1.1,—matching the running

[3] Following convention, we use the $O^*(f(k))$ notation to suppress the polynomial factor $n^{O(1)}$ in the running time.