Two Fixed-Parameter Algorithms for Vertex Covering by Paths on Trees [1]

Jiong Guo and Rolf Niedermeier and Johannes Uhlmann

Institut für Informatik, Friedrich-Schiller-Universität Jena, Ernst-Abbe-Platz 2, D-07743 Jena, Germany.
{guo,niedermr,uhlmann}@minet.uni-jena.de

Abstract

Vertex Covering by Paths on Trees with applications in machine translation is the task to cover all vertices of a tree $T = (V, E)$ by choosing a minimum-weight subset of given paths in the tree. The problem is NP-hard and has recently been solved by an exact algorithm running in $O(4^C \cdot |V|^2)$ time, where C denotes the maximum number of paths covering a tree vertex. We improve this running time to $O(2^C \cdot C \cdot |V|)$. On the route to this, we introduce the problem Tree-like Weighted Hitting Set which might be of independent interest. In addition, for the unweighted case of Vertex Covering by Paths on Trees, we present an exact algorithm using a search tree of size $O(2^k \cdot k!)$, where k denotes the number of chosen covering paths. Finally, we briefly discuss the existence of a size-$O(k^2)$ problem kernel.

Key words: graph algorithms, combinatorial problems, fixed-parameter tractability, exact algorithms

1 Introduction

The NP-hard Vertex Covering by Paths on Trees (VCPT) has been recently identified as an interesting algorithmic problem with a direct application in machine translation [4,5]:

Input: A tree $T = (V, E)$ and a set L of paths in T, where each path is associated with a positive number as its weight.

[1] Supported by the Deutsche Forschungsgemeinschaft (DFG), Emmy Noether research group PIAF (fixed-parameter algorithms), NI 369/4, and project PABI (parameterized algorithmics for bioinformatics), NI 369/7.

Task: Find a minimum-weight subset $L' \subseteq L$ such that every vertex of T is covered by at least one path from L'.

Note that a path in L can uniquely be specified by its two endpoints and hence can be represented by a vertex pair. VCPT is closely related to connectivity augmentation problems in graphs. For instance, the NP-hard TREE AUGMENTATION problem asks for a minimum-size set of additional edges whose insertion into a tree makes it 2-edge-connected [1]. It is not hard to see that the unit-weight case of VCPT generalizes TREE AUGMENTATION [5]. Indeed, one may interpret this unweighted VCPT as a "vertex version" of TREE AUGMENTATION, that is, whereas unweighted VCPT seeks to *cover* all tree vertices by a minimum number of paths, TREE AUGMENTATION seeks to cover all tree edges by a minimum number of paths.

Lin et al. [5] proved the NP-hardness of VCPT, even for the unweighted case. [2] Moreover, they provided an exact exponential-time algorithm and showed that VCPT can be approximated to a factor 2 in polynomial time. Their exact algorithm has running time $O(4^C \cdot |V|^2)$, where C is the maximum number of paths covering a tree vertex. The algorithm is based on dynamic programming. Lin et al. emphasize that exact algorithms "are desired in the machine translation application" [5, page 75] and that assuming small values for the parameter C "maps to the machine translation application" [5, page 81]. Thus, Lin et al.'s exact algorithm can be seen as a practically important fixed-parameter tractability result with respect to the parameter C, inviting further studies concerning fixed-parameter algorithms [6] for VCPT.

This work makes two contributions concerning the fixed-parameter tractability of VCPT. First, we improve Lin et al.'s $O(4^C \cdot |V|^2)$ time bound to $O(2^C \cdot C \cdot |V|)$. In particular, for the practically relevant case of $C = O(1)$ this means linear instead of quadratic time. We achieve this result by reducing VCPT to the more general problem TREE-LIKE WEIGHTED HITTING SET, and by solving the latter with dynamic programming in the style of tree decomposition based dynamic programming. Our second contribution only applies to the unweighted case of VCPT. Instead of considering the parameter C, here we study the parameter k denoting the size of the solution L', that is, $k := |L'|$. We provide three polynomial-time data reduction rules and, based on these, develop a size-$O(2^k \cdot k!)$ search tree for unweighted VCPT. Finally, pointing to related work [3,9] for the TREE AUGMENTATION problem, we indicate the existence of a size-$O(k^2)$ problem kernel for unweighted VCPT.

[2] Note that the NP-hardness heavily relies on the fact that the path set L in the problem definition is part of the input.

2 Parameter Maximum Number of Paths Covering a Vertex

Taking into account that VCPT is NP-hard, Lin et al. [5] introduced the parameter C denoting the upper bound on the number of paths covering a tree vertex. Here, we improve their $O(4^C \cdot |V|^2)$-time algorithm to an $O(2^C \cdot C \cdot |V|)$-time algorithm. Both algorithms are based on dynamic programming. To present our algorithm, we introduce a special case of the HITTING SET problem, namely the TREE-LIKE WEIGHTED HITTING SET (TWHS) problem, and show that it is a generalization of VCPT. TREE-LIKE WEIGHTED HITTING SET is defined as follows.

Input: A base set $S = \{s_1, s_2, \ldots, s_n\}$, a tree-like subset collection $\mathcal{K}$ of subsets of S, $\mathcal{K} = \{K_1, K_2, \ldots, K_m\}$, and a weight function $\omega : S \to \mathbb{Q}_{>0}$, where $\mathbb{Q}_{>0}$ is the set of positive rational numbers.
Task: Find a *hitting set* $S' \subseteq S$ of minimum weight such that $K_i \cap S' \neq \emptyset$ for all $1 \leq i \leq m$.

Tree-like means that the subsets in $\mathcal{K}$ can be organized in a so-called *subset-tree* $\mathcal{T}$ such that there exists a one-to-one mapping between the vertices of $\mathcal{T}$ and the subsets in a way that for every element $s \in S$ the vertices corresponding to the subsets containing s induce a subtree of $\mathcal{T}$. The subset-tree and the corresponding mapping can be computed in linear time for a given tree-like subset collection [8]. Thus, we may assume that we are given a subset-tree and the mapping for the tree-like subset collection under consideration.

Given a VCPT-instance (T, L), define $\text{vpaths}(v) := \{p \in L \mid v \text{ lies on } p\}$. Moreover, for an edge e let $\text{epaths}(e) := \{p \in L \mid e \text{ lies on } p\}$. Clearly, $C = \max_{v \in V} |\text{vpaths}(v)|$. More importantly, note that for the base set $S := L$ the subset collection $\mathcal{K} := \{\text{vpaths}(v) \mid v \in V\}$ is tree-like—the subset-tree $\mathcal{T}$ and the VCPT input instance tree T are isomorphic. VCPT forms the special case of TWHS where every element in S only maps to a path and not to a subtree of $\mathcal{T}$.

Theorem 1 TREE-LIKE WEIGHTED HITTING SET *can be solved in* $O(2^C \cdot C \cdot m)$ *time, where* $C := \max_{K \in \mathcal{K}} |K|$.

PROOF. Let $\mathcal{T} = (I, F)$ be the subset-tree for the tree-like subset collection $\mathcal{K}$. We use dynamic programming to find an optimal solution for TWHS. To this end, we construct a coloring of the elements of the base set with colors 0 or 1 such that for every subset at least one of the elements that it contains is colored 1, and the sum of the weights of the elements that is colored 1 is minimized. Let $\mathcal{T}$ be rooted at an arbitrary vertex $r \in I$. We use the following notation. For a vertex $v \in I$ let $\mathcal{T}_v = (I_v, F_v)$ denote the subtree of $\mathcal{T}$ rooted at vertex v. If v is a child of u in the rooted tree, then let parent$(v) := u$. Define $K(v)$ to be the subset in $\mathcal{K}$ associated with $v \in I$ and define $\mathcal{K}_v$ to be the set of subsets corresponding to the vertices in $\mathcal{T}_v$, that is, $\mathcal{K}_v := \{K(u) \,|\, u \in I_v\}$, and define $S_v := \bigcup_{u \in I_v} K(u)$. In this way, every vertex $v \in I$ is associated with a sub-instance of TWHS consisting of the subset collection $\mathcal{K}_v$ of the base set S_v. Finally, for a subset $S' \subseteq S$ we define $\omega(S') := \sum_{s \in S'} \omega(s)$.

For $v \in I$ let $K(v) = \{s_{v_1}, s_{v_2}, \ldots, s_{v_{n_v}}\} \subseteq S$ with $n_v := |K(v)|$. For every vertex $v \in I$ we have a dynamic programming table with $n_v + 1$ columns and 2^{n_v} rows, one row for every $c \in \{0,1\}^{n_v}$. Every row represents a coloring $F_v^c : K(v) \to \{0,1\}$ of the elements in $K(v) \subseteq S$ according to $c = (c_1, c_2, \ldots, c_{n_v}) \in \{0,1\}^{n_v}$, that is, $F_v^c(s_{v_j}) = c_j$ for all $1 \le j \le n_v$. Furthermore, $(F_v^c)^{-1}$ denotes the preimage of function F_v^c defined as $(F_v^c)^{-1}(1) := \{s_{v_j} \,|\, F_v^c(s_{v_j}) = 1\}$ and $(F_v^c)^{-1}(0) := \{s_{v_j} \,|\, F_v^c(s_{v_j}) = 0\}$. A coloring F_v^c has the interpretation that exactly those elements from $K(v)$ colored 1 shall be part of the solution. The (n_v+1)-th entry in every row represents the weight $A_v(c)$ for the coloring c. That is, after the dynamic programming has been carried out,

$$\begin{aligned} A_v(c) \;&= \omega((F_v^c)^{-1}(1)) \\ &\quad + \min\{\omega(S') \,|\, S' \subseteq (S_v \setminus K(v)) \;\wedge\; \forall_{K \in \mathcal{K}_v}((S' \cup (F_v^c)^{-1}(1)) \cap K) \neq \emptyset\}. \end{aligned}$$

That is, $A_v(c)$ corresponds to the minimum weight of a solution for the sub-instance consisting of the subset collection $\mathcal{K}_v$ of the base set S_v assuming that the elements in $K(v)$ are chosen according to the coloring F_v^c.

For the root r, we have $\mathcal{K} = \mathcal{K}_r$. Thus, if we succeed in computing $A_r(c)$ in accordance with this interpretation, then the minimum weight for hitting all sets of $\mathcal{K}$ corresponds to the minimum of $A_r(c)$ over all colorings $c \in \{0,1\}^{n_r}$.

We initialize the tables as follows. For every vertex v and for every $c \in \{0,1\}^{n_v}$,

$$A_v(c) := \begin{cases} +\infty, & \text{if } (F_v^c)^{-1}(1) = \emptyset, \\ \omega((F_v^c)^{-1}(1)), & \text{otherwise.} \end{cases}$$

This initialization can be executed in $O(2^C \cdot C \cdot |I|)$ time. Herein, the multiplicative factor C is due to the computation of the weight function ω.

Next, all local solutions are combined into a global solution for the whole instance. The algorithm works bottom up from the leaves to the root. First, the leaves are marked as processed. Then, the dynamic programming table of an inner vertex v is updated by considering the dynamic programming table of each of its children. After the table has been updated by having considered all its children, v is marked as processed, too.

Next, we go into the details of this update step. Let $c' \times c''$ denote the concatenation of the vectors c' and c''. Let $v \in I$ be a vertex not marked as processed yet and let $u \in I$ be a child of v that is already marked as processed. Assume that $K(v) = \{j_1, j_2, \ldots, j_l, i_1, i_2, \ldots, i_q\}$ and $K(u) = \{j_1, j_2, \ldots, j_l, f_1, f_2, \ldots, f_p\}$, that is, $K(v) \cap K(u) = \{j_1, j_2, \ldots, j_l\}$.

For all $c \in \{0,1\}^l$ and for all $c' \in \{0,1\}^q$,

$$\begin{aligned} A_v(c \times c') := \quad & A_v(c \times c') - \left(\textstyle\sum_{s \in \{j_1,\ldots,j_l\} \wedge F_v^{c \times c'}(s)=1} \omega(s) \right) \\ & + \begin{cases} A_u(c), & \text{if } p = 0, \\ \min_{c'' \in \{0,1\}^p} (A_u(c \times c'')), & \text{otherwise.} \end{cases} \end{aligned}$$

For every edge $\{v, u\}$ this update step can be done in $O((2^{n_v} + 2^{n_u}) \cdot C) = O(2^C \cdot C)$ time by sorting the tables adequately using bucket sort. The overall running time is $O(2^C \cdot C \cdot |I|)$ since in the bottom up process every edge of the subset-tree has to be considered exactly once and there are at most $|I| - 1$ edges in the tree. After all vertices are processed the weight of an optimal solution corresponds to $\min_{c \in \{0,1\}^{n_r}} A_r(c)$. The correctness of the algorithm follows from the tree-likeness property of the subset collection guaranteeing the consistency of the dynamic programming. □

Remark: The space consumption of the algorithm in Theorem 1 (and also of the subsequent Theorem 2 and Corollary 3) can easily be bounded by $O(2^C \cdot m \cdot (C + W))$, where W denotes the maximum number of bits needed in the binary encoding of the occurring weights.

Our result for VCPT now follows by giving an efficient translation of a VCPT instance into a TWHS instance and then applying Theorem 1.

Theorem 2 Vertex Covering by Paths on Trees *can be solved in* $O(2^C \cdot C \cdot |V|)$ *time.*

PROOF. Due to Theorem 1 it suffices to show that, given a VCPT instance consisting of the tree $T = (V, E)$ and the path set L, the subset collection $\{\text{vpaths}(v) \mid v \in V\}$ can be constructed in $O(|V| \cdot C \cdot \log(C))$ time.

Let T be rooted at an arbitrary vertex $r \in V$. First, note that $|L| \leq |V| \cdot C$, since for every vertex there are at most C paths covering it. In the following, let $L = \{l_1, l_2, \ldots, l_t\}$. We perform set operations on subsets of L, all of size at most C. Hence, by sorting the paths in these subsets in increasing order of their indices, we can execute these set operations in $O(C \log(C))$ time. Before starting the main algorithm, we do the following initializations. For every vertex $v \in V$ we compute the set $R(v)$ of paths with one endpoint being v. This can clearly be done in $O(|V| \cdot C)$ time. Then, for every leaf v of T, we set $\text{vpaths}(v) := R(v)$ since the paths having v as one endpoint are exactly the paths covering v.

Recall that $\text{epaths}(e)$ for an edge e contains all paths covering e. Clearly, $\text{epaths}(\{v, \text{parent}(v)\}) = R(v)$ for every leaf v. In contrast, for an inner vertex v $\text{epaths}(\{v, \text{parent}(v)\})$ is computed bottom-up based on $\text{epaths}(e)$ for edges e connecting v and its children. Now, we compute the sets $\text{vpaths}(v)$ and $\text{epaths}(\{v, \text{parent}(v)\})$ for every inner vertex v of T. Let $u_1, u_2, \ldots, u_{\delta_v}$ be v's children.

```
1  $A(v) := \emptyset$    // paths covering at least one edge to a child of $v$
2  $B(v) := \emptyset$    // paths covering (exactly) two children of $v$
3  for all $1 \leq j \leq \delta_v$
4      $B(v) := B(v) \cup (A(v) \cap \text{epaths}(\{v, u_j\}))$
5      $A(v) := A(v) \cup \text{epaths}(\{v, u_j\})$
6  $\text{vpaths}(v) := A(v) \cup R(v)$
7  $\text{epaths}(\{v, \text{parent}(v)\}) := (R(v) \setminus A(v)) \cup (A(v) \setminus (B(v) \cup R(v)))$
```

If $v = r$ then line 7 is not executed.

Summing up over all internal vertices in T, we arrive at a total running time of $O(|V| \cdot C \cdot \log(C))$ for computing the subset collection $\{\text{vpaths}(v) \mid v \in V\}$ as an input instance of TWHS, enabling the application of Theorem 1. □

By means of a simple reduction due to Lin et al. [5] from WEIGHTED TREE AUGMENTATION to VCPT we can solve WEIGHTED TREE AUGMENTATION within the same time bounds.

Corollary 3 *The* WEIGHTED TREE AUGMENTATION *problem can be solved in $O(2^C \cdot C \cdot |V|)$ time.*

3 Parameter Solution Size

In this section, we present a result that only applies to unweighted VCPT. We consider the number k of chosen paths as the parameter and show that unweighted VCPT is fixed-parameter tractable with respect to k. This is accomplished by a set of polynomial-time data reduction rules and, based on these, a simple search tree strategy. To state our data reduction rules, we make use of a generalization of VCPT, namely Annotated VCPT (AVCPT):

> **Input:** A tree $T = (V, E)$ with $V = B \uplus W$, a set L of paths in T, and an integer $k \geq 0$.
> **Task:** Find a subset $L' \subseteq L$ with $|L'| \leq k$ such that every vertex of B is covered by at least one path from L'.

The idea behind this annotated version is that the vertex set is partitioned into black (B) and white (W) vertices where the black vertices must be covered and the white vertices are already covered. This is helpful in expressing our subsequent data reduction rules. Clearly, VCPT is the same as AVCPT with only black vertices.

We first introduce some concepts. The *contraction* of an edge $\{v, w\}$ into a white (or black) vertex means that we add a new vertex n_{vw} to W (or B, respectively), connect n_{vw} with the neighbors $(N(v) \cup N(w)) \setminus \{v, w\}$ of v and w, delete v and w from $B \uplus W$, and finally replace every path that has v or w as one endpoint by a path having n_{vw} as new endpoint instead. Note that due to the contraction of an edge, paths consisting of a single vertex may arise. The reason for introducing AVCPT is that if in the course of the data reduction or the search-tree algorithm we will decide to add a path p into the solution, then all vertices on p are covered and hence this path can be repeatedly contracted into a single white vertex reflecting this situation. A path $p \in L$ is called a *shadow* if there exists a path $p' \in L$ which covers a superset of the vertices covered by p. For a vertex v and an edge $e \in E$, let $T_{v,e}$ denote the connected component of $(V, E \setminus \{e\})$ containing v.

We now present three simple data reduction rules.

R1: Delete all shadows from L.

R2: If there exists an edge $e = \{u, v\} \in E$ with $u, v \in W$, then contract e into a white vertex.

R3: Let $v \in W$ be a white leaf in T and let $p \in B$ be its (only) adjacent vertex. Then contract the edge $\{v, p\}$ into a black vertex.

It is straightforward to verify the correctness of these rules. For instance, R1 is correct because it is never better to choose a shadow path than to choose a more extensive path. Clearly, the rules can be executed in polynomial time until a *reduced instance* is reached, that is, an instance of AVCPT where none of the three rules applies. More precisely, it is not hard to derive a running time of $O(|V| \cdot |L|^2)$ [9]. Due to rules R2 and R3 all leaves are black. Let $\ell(T)$ denote the number of leaves in T.

Lemma 4 *(1) A reduced AVCPT instance (T, L, k) can have a solution $L' \subseteq L$ with $|L'| \leq k$ only if $k \geq \lceil \ell(T)/2 \rceil$.*

(2) $|\mathrm{vpaths}(v)| \leq \ell(T) - 1$ for every leaf v in a reduced AVCPT instance.

PROOF. Statement (1) is true since all leaves are black and these need to be covered by paths and each path can cover at most two leaves. To prove statement (2), consider a path P in T from v to a leaf $w \neq v$. There can be at most one path in L with one endpoint being v and the other endpoint being from the vertex set of P, since all shadows (R1) are deleted. This gives the upper bound $\ell(T) - 1$ on the number of paths that may cover v. □

Using the above observations, we now can easily derive a depth-bounded search tree strategy for AVCPT, implying that AVCPT is fixed-parameter tractable with respect to the parameter k.

Theorem 5 Annotated Vertex Covering by paths on Trees *can be solved in $O(2^k \cdot k! \cdot |V| \cdot |L|^2)$ time.*

PROOF. The algorithm works as follows. At every stage, we make sure that the current instance is reduced with respect to the described data reduction rules, taking $O(|V| \cdot |L|^2)$ time according to the previous discussion. According to statement (2) of Lemma 4, in a reduced instance every leaf has at most $\ell(T) - 1$ covering paths. Due to statement (1) of Lemma 4, we have $\ell(T) \leq 2k$. Since every leaf has to be covered by a path, we thus simply branch into all of the at most $2k - 1$ possibilities of choosing a covering path and decrease parameter k by one. Repeating this at most k times gives a search tree of depth k. Observe that at the i-th level of the search-tree we have to consider at most $2(k - i) - 1$ branches. A simple calculation gives a search tree size of $O(2^k \cdot k!)$. Note that if a particular path is chosen, then the vertices on the corresponding path are covered and, hence, these are contracted into a white vertex and the data reduction rules are applied again before the next branching occurs. □

Together with five further rules, one of them being fairly technical and dealing with the reduction of degree-two paths, one can show that the size of a then reduced instance can be upper-bounded by a quadratic function exclusively depending on the parameter value k. We omit any details here because of the quite technical proof which, however, works in analogy to a corresponding "kernelization" result for the Tree Augmentation problem [3,9]. Taking for granted that a size-$O(k^2)$ problem kernel exists, the "interleaving" of branching and data reduction [7] gives the following result.

Corollary 6 ANNOTATED VERTEX COVERING BY PATHS ON TREES *can be solved in* $O(2^k \cdot k! + |V| \cdot |L|^2)$ *time.*

4 Conclusion

We have improved on the time complexity of solving the NP-hard VERTEX COVERING BY PATHS ON TREES by means of exact algorithms. Concerning the parameter "maximum number of paths covering a vertex", the algorithm of Lin et al. [5] and our new, faster one both exhibit dynamic programming as an important design principle for developing fixed-parameter algorithms [6].

Compare the fixed-parameter algorithms for unweighted VCPT and the related MULTICUT IN TREES problem [2] both with respect to the parameter solution size k. In the latter case, one is also given a tree together with a set of paths and the task is to find at most k edges that cut all the given paths. Whereas we have observed a size-$O(2^k \cdot k!)$ search tree and a size-$O(k^2)$ problem kernel for unweighted VCPT, in case of MULTICUT IN TREES we have a (simple) size-$O(2^k)$ search tree but only an exponential-size problem kernel [2]. This somehow opposite behavior in terms of parameterized complexity analysis of two fairly similar problems might stir interest in further research investigating both problems.

Acknowledgements. We thank two anonymous referees for their constructive comments.

References

[1] K. P. Eswaran and R. E. Tarjan. Augmentation problems. *SIAM Journal on Computing*, 5(4):653–665, 1976.

[2] J. Guo and R. Niedermeier. Fixed-parameter tractability results for multicut in trees. *Networks*, 46(3):124–135, 2005.

[3] J. Guo and J. Uhlmann. Kernelization and complexity results for connectivity augmentation problems. In *Proc. WADS 2007*, Lecture Notes in Computer Science 4619, pages 483–494. Springer, 2007.

[4] D. Lin. A path-based transfer model for machine translation. In *Proc. 20th International Conference on Comput. Linguistics*, Geneva, Switzerland, pages 625–630, 2004.

[5] G. Lin, Z. Cai, and D. Lin. Vertex covering by paths on trees with its application in machine translation. *Information Processing Letters*, 97(2):73–81, 2006.

[6] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms.* Oxford University Press, 2006.

[7] R. Niedermeier and P. Rossmanith. A general method to speed up fixed-parameter-tractable algorithms. *Information Processing Letters*, 73(3–4):125–129, 2000.

[8] R. E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing*, 13(3):566–579, 1984. Addendum in *SIAM Journal on Computing*, 14(1): 254-255, 1985.

[9] J. Uhlmann. *Parameterized Algorithmics for Connectivity Augmentation Problems in Networks.* Diploma thesis, WSI für Informatik, Universität Tübingen, Germany, 2007.