Journal of Discrete Algorithms, Vol. 4(4), pp. 608-622, 2006

Exact Algorithms and Applications for Tree-Like Weighted Set Cover [1]

Jiong Guo and Rolf Niedermeier

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

Abstract

We introduce an NP-complete special case of the Weighted Set Cover problem and show its fixed-parameter tractability with respect to the maximum subset size, a parameter that appears to be small in relevant applications. More precisely, in this practically relevant variant we require that the given collection C of subsets of a some base set S should be "tree-like." That is, the subsets in C can be organized in a tree T such that every subset one-to-one corresponds to a tree node and, for each element s of S, the nodes corresponding to the subsets containing s induce a subtree of T. This is equivalent to the problem of finding a minimum edge cover in an edge-weighted acyclic hypergraph. Our main result is an algorithm running in $O(3^k \cdot mn)$ time where k denotes the maximum subset size, $n := |S|$, and $m := |C|$. The algorithm also implies a fixed-parameter tractability result for the NP-complete Multicut in Trees problem, complementing previous approximation results. Our results find applications in computational biology in phylogenomics and for saving memory in tree decomposition based graph algorithms.

Keywords: NP-hard problems, (Weighted) Set Cover, Multicut in Trees, Minimum Weighted Edge Cover on acyclic hypergraphs, fixed-parameter tractability.

1 Introduction

Efficiently finding feasible solutions for NP-hard problems is a central topic of algorithmic research. One line of investigations in these studies deals with exact algorithms, i.e., fast exponential-time solutions. In this paper we deal

[1] Supported by the Deutsche Forschungsgemeinschaft (DFG), Emmy Noether research group PIAF (fixed-parameter algorithms), NI 369/4. Parts of this work were done while the authors were with Universität Tübingen.

with fixed-parameter algorithms that emerge from the field of parameterized complexity analysis [9,24]. The point here is to detect relevant (and hopefully small) parameters in instances of hard problems and to try to restrict the seemingly unavoidable combinatorial explosions in the running times of the solving algorithms to these parameters. For small parameter values—as occur in many problem instances of practical interest—this leads to efficient algorithms; see [8,12,13,23] for surveys.

Here we report on progress in dealing with a practically relevant special case of the Weighted Set Cover problem. Set Cover is one of the most important problems in combinatorial optimization with numerous applications in various fields. Roughly speaking, the task is to cover a given base set S with a selection of a given set of subsets of S as cheaply as possible–see Section 2 for a precise definition. Unfortunately, Set Cover appears to be very hard from an algorithmic point of view. Unless NP has slightly super-polynomial time algorithms, the best polynomial-time approximation algorithm achieves approximation factor $\Theta(\ln n)$ [11], where n denotes the base set size. In addition, from the viewpoint of parameterized complexity, the problem is known to be W[2]-complete [9] with respect to the parameter "number of chosen covering sets" which excludes fixed-parameter tractability in this respect.

Hence, a prospective way out of this misery seems to be to further explore "structure" in the problem which, if available, makes the problem more amenable to sound algorithmic solutions. In this paper we study such a practically relevant special case of the Weighted Set Cover problem and, although the problem remains NP-complete, we propose a fixed-parameter exact algorithm that is efficient for some realistic application cases.

The problem we study is called Tree-like Weighted Set Cover (TWSC). Here we are given a *tree-like* collection C of subsets (each having some positive real weight) of some base set S and the task is to "cover" S by a minimum weight choice of subsets from C. A subset collection C is called *tree-like* if the subsets in C can be organized in a tree T such that every subset one-to-one corresponds to a node of T and, for each element s in S, the nodes of T corresponding to the subsets containing s induce a subtree of T. In particular, "tree-like" refers to the fact that the subset collection has to fulfill an acyclicity property (called consistency property later) known from the famous concept of tree decompositions of graphs [27,20,4,5]. It is not very hard to see that a subset collection fulfills the "tree-like property" iff it can be interpreted as an acyclic hypergraph as studied in a somewhat different context by Tarjan and Yannakakis [29]. Hence, the Tree-like Weighted Set Cover problem is equivalent to the Minimum Weighted Edge Cover problem on acyclic hypergraphs. The tree-likeness property of a set system can be checked in linear time using the test for acyclicity of hypergraphs developed by Tarjan and Yannakakis [29]. Refer to Gottlob, Leone, and Scarcello [16] for a survey on

measures of hypergraph cyclicity and how to make use of "bounded cyclicity".

The special case of TWSC where T should only be a path has been studied under the name Set Cover with Consecutive Ones Property (SCC1P) [19,22]. A Set Cover instance can also be represented by a binary coefficient matrix M over $\{0, 1\}$ where the rows correspond to the elements in S, the columns correspond to the subsets in C, and an entry is set to "1" if the corresponding element is contained in the corresponding subset; otherwise, it is set to "0." A matrix M has the *consecutive ones property* if the "1"s in each row appear consecutively. In the SCC1P problem, we deal with Set Cover instances whose coefficient matrices have the consecutive ones property. SCC1P can be solved in polynomial time [19,22].

Besides the interest on its own, TWSC is motivated by the following two concrete applications in practice. The first application deals with dynamic programming on tree decompositions. It is well known that graphs with small treewidth allow for efficient solutions of otherwise hard graph problems [4]. The core tool is dynamic programming on tree decompositions. As further discussed in [2], the main bottleneck of this approach is memory space consumption which is exponential with respect to the treewidth. To attack this problem, one can try to minimize the redundancy of information stored by avoiding to keep all dynamic programming tables in memory. This can be formulated as TWSC, where the tree decomposition serves as the subset collection and each tree node is assigned a positive weight equal to the size of the dynamic programming table associated with it. With the help of the TWSC formalization, experiments showed memory savings of around 80 to 90 % [2].

The second application arises in computational molecular biology. In their work on vertebrate phylogenomics, Page and Cotton [25] formulate the problem of locating gene duplications as Tree-like Unweighted Set Cover: given a so-called species tree in which each node has a set of gene duplications associated with it, find the smallest set of nodes whose union includes all gene duplications. Assigning each node a positive weight equal to the minimum number of distinct "episodes" of duplications at this node and solving Tree-like Weighted Set Cover on this tree seems to be a prospective way to answer one of their open questions.

Whereas we observe that Tree-like Weighted Set Cover is NP-complete, our first simple observation is that the unweighted case is easily polynomial-time solvable. Then, we demonstrate that TWSC can be solved in $O(n \cdot m \cdot 3^k)$ worst-case time, where n denotes the size of the base set S, m denotes the number of subsets in the collection C, and k is the maximum subset size[2] —that is, TWSC is fixed-parameter tractable with respect to the parameter k. Note

[2] This parameter is also known as the rank of a hypergraph.

that our basic technique in a sense is similar in spirit to dynamic programming on tree decompositions as exhibited in [30,1]. We emphasize that the trivial solution for (Tree-like) Weighted Set Cover (not considering parameter k) has running time (ignoring polynomial factors) $O(2^m)$, where $m = O(2^n)$ is possible. By way of contrast, for problem instances in the application concerned with dynamic programming on tree decomposition of graphs the values of parameter k realistically are in the range between 5 and 20 [2] and in the phylogenomics application described in [25] we always encountered parameter values smaller than 50. Note that the parameter "maximum subset size" also played an important role in developing polynomial-time approximation algorithms for unweighted Set Cover [10,18], after a series of papers the best approximation factor for subset size 3 now being 4/3 [10]. It is an open question of Halldórsson [18], however, to obtain similar results for the weighted version of Set Cover with restricted subset size. We remark that "maximum subset size" might be the only natural parameterization for TWSC that is of interest for fixed-parameter tractability studies. The two other parameterizations "total weight of the solution" and "maximum number of occurrences of an element in the given subsets" result in fixed-parameter intractability (see Section 2.3).

As an application of our fixed-parameter tractability result for TWSC, we can also show that the so-called Multicut in Trees problem (also known as Weighted Set Cover with tree-representable set system and being different from TWSC) [15] is solvable in $O(m \cdot n \cdot 3^k)$ time, where m denotes the number of node pairs, n denotes the number of tree nodes, and k is the maximum number of paths passing through some tree node. Multicut in Trees, which contains as special cases classical optimization problems such as Matching and Vertex Cover, has been shown to be NP-complete and MAXSNP-hard by Garg, Vazirani, and Yannakakis [15] and, on the positive side, they have given a factor-2 polynomial-time approximation algorithm. By way of contrast, our above fixed-parameter algorithm provides an *optimal* solution and it is efficient whenever k is not too big, a reasonable assumption for several applications.

2 Preliminaries and Basic Facts

We assume familiarity with the basic notions of algorithms and complexity. Fixed-parameter tractability is a core concept in this work. A "parameterized problem" with input instance I and parameter k is called fixed-parameter tractable iff it can be solved in $f(k) \cdot |I|^{O(1)}$ time, where f is an arbitrary computable function only depending on k (see [9,8,12,13,23,24] for more details). For instance, the best known fixed-parameter algorithm for the Vertex Cover problem is known to run in less than $O(kn+1.3^k)$ worst-case time [6,7],

where n denotes the number of graph vertices and parameter k denotes the size of the vertex cover set asked for.

2.1 Tree-Like Weighted Set Cover

(Weighted) Set Cover is one of the most prominent NP-complete problems [14]. The basic Set Cover problem (optimization version) is defined as follows:

Input: A base set $S = \{s_1, s_2, \ldots, s_n\}$ and a collection C of subsets of S, $C = \{c_1, c_2, \ldots, c_m\}$, $c_i \subseteq S$ for $1 \leq i \leq m$, and $\bigcup_{1 \leq i \leq m} c_i = S$.
Task: Find a subset C' of C with minimal cardinality which covers all elements in S, i.e., $\bigcup_{c \in C'} c = S$.

Assigning weights to the subsets and minimizing total weight of the collection C' instead of its cardinality, one naturally obtains the Weighted Set Cover problem. We call C' the *minimum set cover* of S resp. the *minimum weight set cover*. Define the *occurrence* of an element $s \in S$ in C as the number of the subsets in C which contain s. An element with occurrence of one is called *unique*. Set Cover remains NP-complete even if the occurrence of each element is bounded by 2 [26].

Definition 1 (Tree-like subset collection)
A base set $S = \{s_1, s_2, \ldots, s_n\}$ and a collection C of subsets of S, $C = \{c_1, c_2, \ldots, c_m\}$, $c_i \subseteq S$ for $1 \leq i \leq m$. We say that C is a tree-like *subset collection of S if we can organize the subsets in C in an unrooted tree T such that every subset one-to-one corresponds to a node of T and, for each element $s_j \in S$, $1 \leq j \leq n$, all nodes in T corresponding to the subsets containing s_j induce a subtree of T.*

We call T the underlying *subset tree* and the property of T that, for each $s \in S$, the nodes containing s induce a subtree of T is called the *consistency property* of T. Observe that the consistency property also is of central importance in Robertson and Seymour's famous notion of tree decompositions of graphs [27,20,4,5]. By results of Tarjan and Yannakakis [29], we can test whether a subset collection is a tree-like subset collection and, if yes, we can construct a subset tree for it in linear time. Therefore, in the following, we always assume that the subset collection is given in form of a subset tree. For convenience, we denote the nodes of the subset tree by their corresponding subsets. We define the *height* of a subset tree as the minimum height of all rooted trees which can be obtained by taking one node of the subset tree as the root.

Example: For $S = \{s_1, s_2, s_3\}$, the subset collection $C = \{c_1, c_2, c_3\}$ where $c_1 = \{s_1, s_2\}$, $c_2 = \{s_2, s_3\}$, and $c_3 = \{s_1, s_3\}$ is not a tree-like subset collection. These three subsets can only be organized in a triangle. By way of contrast, if $c_1 = \{s_1, s_2, s_3\}$ instead, then we can construct a subset tree (actually a path) with these three nodes and two edges, one between c_1 and c_2 and one between c_1 and c_3.

We now define the central problem of this paper.

Tree-like Weighted Set Cover (TWSC):
Input: A base set $S = \{s_1, s_2, \ldots, s_n\}$ and a tree-like collection C of subsets of S, $C = \{c_1, c_2, \ldots, c_m\}$, $c_i \subseteq S$ for $1 \leq i \leq m$, and $\bigcup_{1 \leq i \leq m} c_i = S$. Each subset in C has a positive real weight $w(c_i) > 0$ for $1 \leq i \leq m$. The weight of a subset collection is the sum of the weights of all subsets in it.
Task: Find a subset C' of C with minimum weight which covers all elements in S, i.e., $\bigcup_{c \in C'} c = S$.

The following easy to prove observation will be helpful in developing our algorithm.

Lemma 1 *Given a tree-like subset collection C of S together with its underlying subset tree T, then each leaf of T is either a subset of its parent node or it has a unique element.*

2.2 A Simple Solution for the Unweighted Case

Unlike the general unweighted Set Cover which is NP-complete, Tree-like Unweighted Set Cover can be solved by a simple polynomial-time algorithm:

We process the subset tree T in a bottom-up manner, i.e., we begin with the leaves. By Lemma 1, each leaf of T is either a subset of its parent node or it contains a unique element from S. If a leaf c contains a unique element, the only choice to cover this element is to put c into the set cover. Then, we delete c from T and c's elements from other subsets. If c is completely contained in its parent node, it is never better to take c into the set cover than to take its parent node. Hence, we can safely delete it from the subset tree. After processing its children, an internal node becomes a leaf and we can iterate the described process. Thus, we obtain the following result.

Proposition 2 Tree-like Unweighted Set Cover *can be solved in $O(mn)$ time.*

Page and Cotton [25] in their work on phylogenomics implicitly dealt with

Tree-like Unweighted Set Cover only giving a heuristic solution seemingly unaware of the simple polynomial-time solvability.

2.3 Hardness of Tree-Like Weighted Set Cover

The key idea of the above algorithm for Tree-like Unweighted Set Cover is that we never put a set into the desired subset collection C' which is a subset of other subsets in the collection C. However, if we associate each subset with an arbitrary positive weight and ask for the set cover with minimum weight, then this strategy is no longer valid.

Proposition 3 *The decision version of* Tree-like Weighted Set Cover *is NP-complete.*

PROOF. TWSC is clearly in NP. To show its NP-hardness, we reduce the general (unweighted) Set Cover problem to it. Given a Set Cover instance with the base set S and the subset collection C, we construct a new subset collection $C' := C \cup \{c_{m+1}\}$ with an additional subset $c_{m+1} := S$. All subsets in C' except c_{m+1} have unit weight, $w(c_i) = 1$ for $1 \leq i \leq m$, and $w(c_{m+1}) = m+1$. Then, the base set S and the new subset collection C' form an instance of TWSC, the underlying tree being a star with center c_{m+1}. □

Since this reduction is gap-preserving, by Feige's result for Set Cover [11] it directly follows that the best polynomial-time approximation for TWSC is $\Theta(\ln n)$ unless NP has slightly super-polynomial time algorithms. Moreover, because this reduction also preserves the total weight of the optimal solution in the sense of parameterized complexity theory [9], we also can infer W[2]-hardness for TWSC with respect to the parameter "total weight of the solution" of the set cover. This excludes fixed-parameter tractability for this parameterization [9]. The reduction above shows also that TWSC remains NP-complete even if the subset tree has height one. In the following, we will show that several other relevant variations of TWSC are also NP-complete. The first corollary follows from the NP-completeness of the variant of Set Cover where the occurrence of elements is bounded by 2 [26] and the reduction used above.

Corollary 4 *The decision version of* Tree-like Weighted Set Cover *is NP-complete if the occurrence of each element from S in the subsets of the collection C is at most* 3.

Note that, in this way, we can also conclude that parameterization by "occurrence number" is not useful concerning fixed-parameter tractability studies

for TWSC. The following result is of particular relevance in the context of the application of TWSC for minimizing memory consumption in dynamic programming on tree decompositions [2] where we typically have exponential weights for the subsets.

Corollary 5 *The decision version of* Tree-like Weighted Set Cover *is NP-complete if the weight of a subset c_i is exponential in its size, i.e., $w(c_i) = \alpha^{|c_i|}$ with $\alpha \geq 2$ and $1 \leq i \leq m$.*

PROOF. The reduction is from an NP-complete variant of Set Cover, so-called 3-Set Cover, where each subset contains exactly three elements [14] and it works in the same way as used for Proposition 3 with the exception that the weights of the subsets are exponential in their sizes. The subset c_{m+1} introduced by the reduction has weight α^n and all other subsets have the same weight α^3. Since we need at most $(n-2)$ pairwise different 3-element subsets to cover an n-element base set and $\alpha^n \geq (n-2) \cdot \alpha^3$ for $n > 0$ and $\alpha \geq 2$, the subset c_{m+1} cannot be in the minimum weight set cover. Hence, the one-to-one correspondence between the solutions is preserved. □

From Proposition 3 we see that trees with star structure keep TWSC NP-complete. The same is true with binary, balanced trees.

Corollary 6 *The decision version of* Tree-like Weighted Set Cover *is NP-complete if the underlying subset tree is a balanced binary tree.*

PROOF. The reduction from Set Cover differs from the one used for Proposition 3 only in the construction of the underlying subset tree. Instead of a star, here we construct an arbitrary balanced binary tree with the subsets in C being the leaves. All internal nodes are set equal to S and have a weight of $m + 1$. □

3 Algorithm for Tree-Like Weighted Set Cover

We present a fixed-parameter algorithm for TWSC with respect to the parameter maximal subset size k, i.e., $k := \max_{c \in C}\{\, |c| \,\}$. This implies that the problem can be efficiently solved for small values of k, a realistic assumption for several applications. To facilitate the presentation of the algorithm, we will describe, in the first subsection, how to solve the problem for binary subset trees, an also NP-complete special case (cf. Corollary 6), and then, in the second subsection, we extend the described algorithm to arbitrary trees.

3.1 Tree-Like Weighted Set Cover with Binary Subset Tree

Our dynamic programming algorithm processes the underlying subset tree bottom-up, i.e., first the leaves, then the nodes having leaves as their children, and finally the root. For a given tree-like subset collection C with its underlying subset tree T, we define for each node c_i of T a set $A(c_i)$ which contains all elements occurring in the nodes of the subtree with c_i at the root:

$$A(c_i) := \bigcup_{c \in T[c_i]} c\,,$$

where $T[c_i]$ denotes the node set of the subtree of T rooted at c_i.

Moreover, we associate with each node c of T a table D_c. Table D_c has three columns, the first two corresponding to the two children of c and the third to c. The rows of the table correspond to the elements of the power set of c, i.e., there are $2^{k'}$ rows if $c = \{s_1, s_2, \ldots, s_{k'}\}$, $k' \leq k$. Figure 1 illustrates the structure of table D_c for a node c having two children c' and c''. Table D_c has $3 \cdot 2^{k'} = O(2^k)$ entries. Entry $D_c(x, y)$ is defined as follows:

> $D_c(x, y) :=$ the minimum weight to cover the elements in $x \cup (A(y) \setminus c)$ by using the subsets in the subtree $T[y]$ for $y \in \{c, c', c''\}$ and $x \subseteq c$.

During the bottom-up process, the algorithm fills out such a table for each node. For an internal node c, the entries of the columns corresponding to c' and c'' can be directly retrieved from $D_{c'}$ and $D_{c''}$, which have been already computed before we arrive at node c.[3] Using the values from the first two columns, we then compute the entries in the column of c. After D_r for the root r of the subset tree is computed, the minimum weight to cover all elements in S is the value $D_r(r, r)$. In the following, we describe the subtle details how to fill out the table for a node in the tree. We distinguish three cases:

Case 1: Node $c := \{s_1, s_2, \ldots, s_{k'}\}$ is a leaf:

Since c has no child, columns c' and c'' are empty. We can easily compute the third column:

$$D_c(x, c) := \begin{cases} 0, & \text{if } x = \emptyset; \\ w(c), & \text{otherwise.} \end{cases} \tag{1}$$

Case 2: Node $c := \{s_1, s_2, \ldots, s_{k'}\}$ has only one child c':

[3] Note that these two columns are only needed to make the description of the computation of the last column more simple. For the purpose of implementation, the table D_c needs only the column corresponding to c.

D_c	c'	c''	c
$\emptyset$			
$\{s_1\}$			
$\{s_2\}$			
$\vdots$			
$c := \{s_1, s_2, \ldots, s_{k'}\}$			

Fig. 1. Table D_c for node $c := \{s_1, s_2, \ldots, s_{k'}\}$ with $k' \leq k$ having two children c' and c''

The column c'' of D_c is empty. The first step to fill out the table is to get the values of the first column from the table $D_{c'}$. If there is one element s_j, $1 \leq j \leq k'$, in set x which does not occur in $T[c']$, i.e., $x \nsubseteq A(c')$, then it is impossible to cover $x \cup (A(c') \setminus c)$ by using only the subsets in $T[c']$. The entry $D_c(x, c')$ is then set to ∞. Otherwise, i.e., $x \subseteq A(c')$, to determine the value of $D_c(x, c')$, we have to find the (uniquely determined) row in table $D_{c'}$ which corresponds to the subset x' of c' satisfying $x' \cup (A(c') \setminus c') = x \cup (A(c') \setminus c)$. Due to the consistency property of tree-like subset collections, each element in c also occurring in $T[c']$ is an element of c'. Hence, we get

$$x \cup (A(c') \setminus c) = x \cup (c' \setminus c) \cup (A(c') \setminus c').$$

We set $x' := x \cup (c' \setminus c)$. Since $x \subseteq A(c')$ and $x \subseteq c$, it follows that $x \subseteq c'$. Therefore, also $x' \subseteq c'$ and there is a row in $D_{c'}$ corresponding to x'. Thus, $D_c(x, c')$ is set equal to $D_{c'}(x', c')$. Altogether, we have:

$$D_c(x, c') := \begin{cases} \infty, \text{ if } x \nsubseteq c'; \\ D_{c'}(x \cup (c' \setminus c), c'), \text{ if } x \subseteq c'. \end{cases} \tag{2}$$

The second step is to compute the last column of D_c using the values from the column for c'. For each row corresponding to a subset x of c, we have to compare the two possibilities to cover the elements of $x \cup (A(c) \setminus c)$, either using c to cover elements in x and using some subsets in $T[c']$ to cover the remaining elements or using solely subsets in $T[c']$ to cover all elements:

$$D_c(x, c) := \min\{w(c) + D_c(\emptyset, c'),\ D_c(x, c')\}. \tag{3}$$

Case 3: Node $c := \{s_1, s_2, \ldots, s_{k'}\}$ has two children c' and c'':

In this case, the first step can be done in the same way as in Case 2, i.e., retrieving the values of the columns c' and c'' of D_c from tables $D_{c'}$ and $D_{c''}$.

In order to compute the value of $D_c(x, c)$, for a row x corresponding to a subset

of c, we also compare the two possibilities to cover $x \cup (A(c) \setminus c)$, either using c to cover x or not. In this case, however, we have two subtrees $T[c']$ and $T[c'']$ and, hence, we have more than one alternative to cover $x \cup (A(c) \setminus c)$ by only using subsets in $T[c']$ and $T[c'']$. As a simple example consider a subset $x \subseteq c$ that has only two elements, i.e., $x = \{s'_1, s'_2\}$. We can cover it by using only subsets in $T[c']$, only subsets in $T[c'']$, or a subset in $T[c']$ to cover $\{s'_1\}$ and a subset in $T[c'']$ to cover $\{s'_2\}$ or vice versa. Therefore, for $x := \{s'_1, s'_2, \ldots, s'_{k''}\} \subseteq c$ with $k'' \leq k'$,

$$D_c(x, c) := \min \left\{ \begin{array}{l} w(c) + D_c(\emptyset, c') + D_c(\emptyset, c''), \\ D_c(\emptyset, c') + D_c(x, c''), \\ D_c(\{s'_1\}, c') + D_c(x \setminus \{s'_1\}, c''), \\ D_c(\{s'_2\}, c') + D_c(x \setminus \{s'_2\}, c''), \\ \vdots \\ D_c(x \setminus \{s'_2\}, c') + D_c(\{s'_2\}, c''), \\ D_c(x \setminus \{s'_1\}, c') + D_c(\{s'_1\}, c''), \\ D_c(x, c') + D_c(\emptyset, c'') \end{array} \right\}. \tag{4}$$

With these three cases, we can fill out D_c for all nodes c. The entry $D_r(r, r)$ stores the minimum weight to cover all elements where r denotes the root of the subset tree. In order to construct the minimum weight set cover, using table D_r we find out whether the computed minimum weight is achieved by taking r into the minimum weight set cover or not. Then, doing a traceback from the root to the leaves, we recursively determine the subsets in the minimum weight set cover. Note that, if we only want to know the minimum weight, we may discard the tables $D_{c'}$ and $D_{c''}$ after filling out D_c, for each internal node c with children c' and c'', to reduce the required memory space from $O(m2^k)$ to $O(2^k)$.

Theorem 7 Tree-like Weighted Set Cover *with an underlying binary subset tree can be solved in* $O(m \cdot n \cdot 3^k)$ *time, where* k *denotes the maximum subset size, i.e.,* $k := \max_{c \in C} |c|$.

PROOF. The correctness of the above algorithm directly follows from the above description.

Concerning the running time of the algorithm, the size of table D_c is bounded by $3 \cdot 2^k$ for each node c, since $|c| \leq k$. Using a proper data structure, such as a hash table, the retrieval of a value from one of the tables corresponding to the children can be done in constant time. Thus, the two columns of D_c

corresponding to the two children c' and c'' can be filled out in $O(2^k)$ time. To compute an entry in the column c, which corresponds to a subset x of c, the algorithm compares all possibilities to cover some elements of x by the subsets in $T[c']$. There can be only $2^{|x|}$ such possibilities. Hence, it needs $O(2^{|x|})$ steps to compute $D_c(x, c)$ for each subset x of c. Since all set operations needed between two sets with maximum size of n can be done in $O(n)$ time, the running time for computing D_c is

$$n \cdot (\sum_{j=1}^{|c|} \binom{|c|}{j} O(2^j)) + O(2^{|c|}) = O(n \cdot 3^{|c|}).$$

Therefore, the computation of the tables of all nodes can be done in $O(m{\cdot}n{\cdot}3^k)$ time. During the traceback, we visit, from the root to leaves, each node only once and, at each node, can in constant time find out whether to put this node into the set cover and with which entries in the tables of the children to continue the traceback. Thus, the traceback works in $O(m)$ time. □

3.2 Tree-Like Weighted Set Cover with Arbitrary Subset Tree

Our subsequent main result gives a fixed-parameter algorithm that solves TWSC (on arbitrary subset trees).

Theorem 8 Tree-like Weighted Set Cover *can be solved in* $O(m{\cdot}n{\cdot}3^k)$ *time, where* k *denotes the maximum subset size, i.e.,* $k := \max_{c \in C} |c|$.

PROOF. We can transform an arbitrary tree to a binary tree. For an internal node c with $l > 2$ children nodes, c^1, c^2, ..., c^l, we add $l - 2$ new nodes c^{12}, c^{123}, ..., $c^{1\cdots l-1}$ into the subset tree as illustrated in Figure 2. All newly added nodes are set equal to c and have weight $w(c) + 1$. Observe that the nodes $c^{12}, c^{123}, \ldots, c^{1\cdots l-1}$ can never be in an optimal set cover since they cover the same elements as c but have higher weight. Hence, there is a one-to-one correspondence between the solution for the arbitrary subset tree and the solution for the binary subset tree. Thus, we can apply the algorithm from Section 3.1 to the constructed binary tree which contains at most $2n$ nodes. □

4 Application to Multicut in Trees

The Multicut in trees problem is defined as follows:

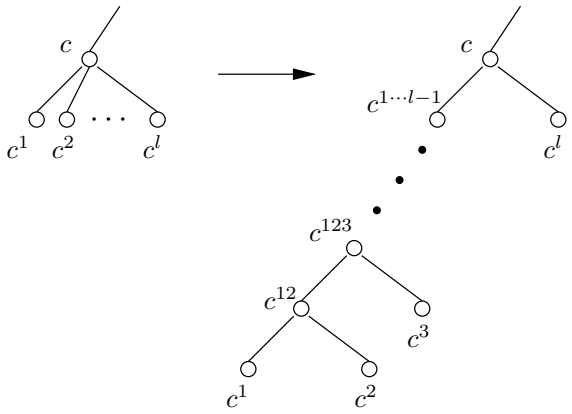

Fig. 2. Transformation of an arbitrary subset tree to a binary subset tree

Input: An undirected tree $T = (V, E)$, $n := |V|$, and a collection H of m pairs of nodes in V, $H = \{(v_i, u_i) \mid v_i, u_i \in V, v_i \neq u_i, 1 \leq i \leq m\}$. Each edge $e \in E$ has a positive real weight $w(e) > 0$.
Task: Find a subset E' of E with minimum total weight whose removal separates each pair of nodes in C.

Such a subset of edges is called a *multicut*. This problem was shown to be NP-complete even for an input tree having height one and unweighted edges [15]. Garg, Vazirani, and Yannakakis [15] have given a factor-2 approximation algorithm for the general case. In the following, we show that Multicut in Trees can be reduced to Tree-like Weighted Set Cover in polynomial time. In this way, the fixed-parameter algorithm presented in Section 3 can also be used to solve Multicut in Trees, giving a potentially practical exact algorithm for it.

Theorem 9 Multicut in Trees *can be solved in* $O(m \cdot n \cdot 3^k)$ *time, where* k *denotes the maximum number of paths passing through a node or an edge,* m *denotes the number of node pairs, and* n *denotes the number of tree nodes.*

PROOF. Given an instance of Multicut in Trees, we create a new tree T' by adding some new nodes to the input tree $T = (V, E)$, $n := |V|$. We replace each edge $e = \{u, v\} \in E$ with a new node w_e and we connect it by two edges with u and v. The set of these new nodes is denoted as V'. The new tree T' has $2n - 1$ nodes and $2n - 2$ edges. Then, we create a set P containing the paths in T which connect the node pairs in H. For a node pair $(u_i, v_i) \in H$, there is a unique path p in T connecting u_i and v_i. We can determine p in $O(n)$ time and we put it into P. Furthermore, we create a set P_e for each $e \in E$ which contains the paths in P passing through e, and a set P_v for each $v \in V$ containing the paths in P passing through v. Note that P_v shall *not* contain the paths starting or ending at v. The Tree-like Weighted Set Cover instance then consists of the base set P and the subset collection $C := C_1 \cup C_2$,

where $C_1 := \{P_e \mid e \in E\}$ and $C_2 := \{P_v \mid v \in V\}$. We have $\bigcup_{P_e \in C_1} P_e = P$. Each subset P_e in C_1 is defined to have the same weight as its corresponding edge e, i.e., $w(P_e) := w(e)$. Since Multicut in Trees asks for a subset of the edge set, we have to give the subsets P_v in C_2 a weight such that none of them will be in the minimum weight set cover: $w(P_v) := \sum_{e \in E} w(e) + 1$ for all $v \in V$. It is clear that C is a tree-like subset collection: The underlying subset tree is T' by associating the subsets $P_e \in C_1$ with the nodes $w_e \in V'$ and the subsets $P_v \in C_2$ with the nodes $v \in V$. The maximum subset size corresponds to the maximum number of paths passing through a node or an edge [4].

It is easy to see that an optimal solution $\{P_{e'_1}, P_{e'_2}, \ldots, P_{e'_l}\}$ for the TWSC instance corresponds to an optimal solution $\{e'_1, e'_2, \ldots, e'_l\}$ for the Multicut in Trees instance and vice versa. □

We remark that in a related work we have shown that *unweighted* Multicut in Trees is fixed-parameter tractable with respect to the size of the solution set E' [17].

Garg et al. [15] have shown that Multicut in Trees is equivalent to the so-called Tree-representable Set Cover problem. A (weighted) Set Cover instance (S, C) is called a *tree-representable set system* if there is a tree T in which each edge is associated with a subset in C such that, for each element s in S, the edges corresponding to the subsets containing s induce a path in T. There are almost linear-time algorithms to decide whether a given Set Cover instance is tree-representable [3]. The problem of deciding whether a given Set Cover instance is tree-representable has been extensively studied in different contexts [31].

Compare Tree-representable Set Cover with Tree-like Set Cover. Both problems have an underlying tree and, in both problems, the subsets containing an element should induce a connected substructure. In the tree-representable case, however, these subsets induce only a path whereas in the tree-like case they induce a subtree. Furthermore, the subsets in the subset collection are associated with the edges of the tree in the tree-representable case and the subsets are associated with the nodes of the tree-like case. Concerning their complexity, for the unweighted case, Tree-representable Set Cover is NP-complete, since Multicut in Trees is NP-complete for unit weights [15], while Tree-like Set Cover can be easily solved in polynomial

[4] Note that the set of the paths passing through a node does not contain such paths which start or end at the node. Therefore, the number of the paths passing through an edge can be greater than each of the two numbers of the paths passing through its endpoints.

time as shown in Section 2. By way of contrast, due to the equivalence between Multicut in Trees and Tree-representable Set Cover, Tree-representable Weighted Set Cover can be reduced to TWSC. The reverse reduction, if valid, seems to be hard to show. In this sense, we have the "paradoxical situation" that, whereas the *unweighted* case of TWSC is much easier than Tree-representable Set Cover, the *weighted* case of TWSC seems harder than Tree-representable Weighted Set Cover.

5 Conclusion

In a way continuing work of Tarjan and Yannakakis [29] and Garg, Vazirani, and Yannakakis [15], we have identified (fixed-parameter) tractable, nontrivial special cases of the Weighted Set Cover problem. As a by-product, we provided a fixed-parameter tractability [5] result for the Multicut in Trees problem which was studied by Garg et al. [15] from the viewpoint of polynomial-time approximability. They provided a factor-2 approximation. Our exact algorithms are based on dynamic programming.

Concerning applications, our original motivation to study TWSC came from efforts to reduce the space requirement of dynamic programming in determining optimal solutions (for problems such as Vertex Cover, Dominating Set etc.) on tree decompositions of graphs, where parameter k (which corresponds to treewidth there) is typically between 5 and 20, underpinning the practical relevance of our parameterization. Using TWSC, memory savings of up to 90 % and more have been achieved in this way [2]. An interesting potential for further applications appears in a recent work of Mecke and Wagner [21] in the context of railway optimization problems. Other applications of TWSC are conceivable with respect to Multicut in Trees [15], acyclic hypergraphs and their applications for relational databases [29], and several other fields with set covering applications to be explored in future research.

We conjecture that it is hard to significantly improve on our exponential term 3^k. Moreover, since our parameter k is not directly related to the size of the solution we search for, there seems to be no point in asking for a reduction to a problem kernel [9,13,23,24] with respect to parameter k. Finally, it is an interesting task for future research to investigate the relation between tree-like set covering and set covering with almost consecutive ones property as introduced by Ruf and Schöbel [28].

[5] The corresponding parameters are the maximum subset size and the maximum number of paths passing through a node or an edge, respectively. See [17] for a fixed-parameter tractability result for Multicut in Trees with respect to another parameterization.

Acknowledgment We are grateful to Jochen Alber, Nadja Betzler, and Johannes Uhlmann (all from Tübingen) for inspiring this research. We thank Roderic D. M. Page (Glasgow) for explaining the application in computational biology. We thank an editor of *Journal of Discrete Algorithms* for pointing to the connection with set covering with consecutive ones property.

References

[1] J. Alber, H. L. Bodlaender, H. Fernau, T. Kloks, and R. Niedermeier. Fixed parameter algorithms for Dominating Set and related problems on planar graphs. *Algorithmica*, 33(4):461–493, 2002. 4

[2] N. Betzler, R. Niedermeier, and J. Uhlmann. Tree decompositions of graphs: saving memory in dynamic programming. In *Proc. of CTW04*, volume 17 of *Electronic Notes in Discrete Mathematics*, 2004, Elsevier. 3, 4, 8, 15

[3] R. E. Bixby and D. K. Wagner. An almost linear time algorithm for graph realization. *Math. Oper. Res.*, 13:99–123, 1988. 14

[4] H. L. Bodlaender. Treewidth: algorithmic techniques and results. In *Proc. of 22nd MFCS*, volume 1295 of *LNCS*, pages 19–36, 1997, Springer. 2, 3, 5

[5] H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209:1–45, 1998. 2, 5

[6] L. S. Chandran and F. Grandoni. Refined memorisation for vertex cover. *Information Processing Letters*, 93(3):125–131, 2005. 4

[7] J. Chen, I. A. Kanj, and G. Xia. Simplicity is beauty: improved upper bounds for Vertex Cover. Manuscript, April 2005. 4

[8] R. G. Downey. Parameterized complexity for the skeptic. In *Proc. of 18th IEEE Annual Conference on Computational Complexity*, pages 147–169, 2003. 2, 4

[9] R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Monographs in Computer Science. Springer, 1999. 2, 4, 7, 15

[10] R. C. Duh and M. Fürer. Approximation of k-Set Cover by semi-local optimization. In *Proc. of 29th STOC*, pages 256–264, 1997, ACM Press. 4

[11] U. Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45:634–652, 1998. 2, 7

[12] M. R. Fellows. New directions and new challenges in algorithm design and complexity, parameterized (invited paper). In *Proc. of 8th WADS*, volume 2748 of *LNCS*, pages 505–519, 2003, Springer. 2, 4

[13] M. R. Fellows. Blow-ups, win/win's, and crown rules: Some new directions in FPT. In *Proc. of 29th WG*, volume 2880 of *LNCS*, pages 1–12, 2003, Springer. 2, 4, 15

[14] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. 5, 8

[15] N. Garg, V. V. Vazirani, and M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18:3–30, 1997. 4, 13, 14, 15

[16] G. Gottlob, N. Leone, and F. Scarcello. Hypertree decompositions: a survey. In *Proc. of 26th MFCS*, volume 2136 of *LNCS*, pages 37–57. Springer, 2001. 2

[17] J. Guo and R. Niedermeier. Fixed-parameter tractability and data reduction for Multicut in Trees. *Networks*, to appear, 2005. 14, 15

[18] M. M. Halldórsson. Approximating k-Set Cover and complementary graph coloring. In *Proc. of 5th IPCO*, volume 1084 of *LNCS*, pages 118–131, 1996, Springer. 4

[19] A. F. Veinott Jr. and H. M. Wagner. Optimal capacity scheduling. *Operation Research*, 10:518–532, 1962. 3

[20] T. Kloks. *Treewidth: Computations and Approximations*. Volume 842 of *LNCS*, 1994, Springer. 2, 5

[21] S. Mecke and D. Wagner. Solving geometric covering problems by data reduction. In *Proc. of 12th ESA*, volume 3221 of *LNCS*, pages 760–771, 2004, Springer. 15

[22] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Wiley, 1988. 3

[23] R. Niedermeier. Ubiquitous parameterization—invitation to fixed-parameter algorithms. In *Proc. of 29th MFCS*, volume 3153 of *LNCS*, pages 84–103, 2004, Springer. 2, 4, 15

[24] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, forthcoming, 2005. 2, 4, 15

[25] R. D. M. Page and J. A. Cotton. Vertebrate phylogenomics: reconciled trees and gene duplications. In *Proc. of Pacific Symposium on Biocomputing 2002*, pages 536–547, 2002. 3, 4, 6

[26] C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43:425–440, 1991. 5, 7

[27] N. Robertson and P. D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7:309–322, 1986. 2, 5

[28] N. Ruf and A. Schöbel. Set covering with almost consecutive ones property. Technical Report, number 91/2003, Georg-August-Universität Göttingen, 2003. 15

[29] 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 J. Comput.*, 13(3):566–579, 1984. Addendum in *SIAM J. Comput.*, 14(1): 254-255, 1985. 2, 5, 15

[30] J. A. Telle and A. Proskurowski. Practical algorithms on partial k-trees with an application to domination-like problems. In *Proc. of 3rd WADS*, volume 709 of *LNCS*, pages 610–621. Springer, 1993. 4

[31] W. T. Tutte. An algorithm for determining whether a given binary matroid is graphic. In *Proc. Amer. Math. Soc.*, 11:905–917, 1960. 14