The Set Cover Conjecture and Subgraph Isomorphism with a Tree Pattern

Robert Krauthgamer
Weizmann Institute of Science, Rehovot, Israel
robert.krauthgamer@weizmann.ac.il

Ohad Trabelsi
Weizmann Institute of Science, Rehovot, Israel
ohad.trabelsi@weizmann.ac.il

Abstract

In the Set Cover problem, the input is a ground set of n elements and a collection of m sets, and the goal is to find the smallest sub-collection of sets whose union is the entire ground set. The fastest algorithm known runs in time $O(mn2^n)$ [Fomin et al., WG 2004], and the Set Cover Conjecture (SeCoCo) [Cygan et al., TALG 2016] asserts that for every fixed $\varepsilon > 0$, no algorithm can solve Set Cover in time $2^{(1-\varepsilon)n}\mathsf{poly}(m)$, even if set sizes are bounded by $\Delta = \Delta(\varepsilon)$. We show strong connections between this problem and kTree, a special case of Subgraph Isomorphism where the input is an n-node graph G and a k-node tree T, and the goal is to determine whether G has a subgraph isomorphic to T.

First, we propose a weaker conjecture Log-SeCoCo, that allows input sets of size $\Delta = O(1/\varepsilon \cdot \log n)$, and show that an algorithm breaking Log-SeCoCo would imply a faster algorithm than the currently known $2^n\mathsf{poly}(n)$-time algorithm [Koutis and Williams, TALG 2016] for Directed nTree, which is kTree with $k = n$ and arbitrary directions to the edges of G and T. This would also improve the running time for Directed Hamiltonicity, for which no algorithm significantly faster than $2^n\mathsf{poly}(n)$ is known despite extensive research.

Second, we prove that if p-Partial Cover, a parameterized version of Set Cover that requires covering at least p elements, cannot be solved significantly faster than $2^n\mathsf{poly}(m)$ (an assumption even weaker than Log-SeCoCo) then kTree cannot be computed significantly faster than $2^k\mathsf{poly}(n)$, the running time of the Koutis and Williams' algorithm.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Discrete optimization; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Conditional lower bounds, Hardness in P, Set Cover Conjecture, Subgraph Isomorphism

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.45

Related Version This paper is based on two preliminary versions arXiv:1711.08041 and arXiv:1708.07591.

Funding *Robert Krauthgamer*: Work supported in part by the Israel Science Foundation grant #1086/18, ONR Award N00014-18-1-2364, a Minerva Foundation grant, and a Google Faculty Research Award. Part of this work was done while was visiting the Simons Institute for the Theory of Computing.

1 Introduction

Set Cover and Subgraph Isomorphism are two of the most well-researched problems in theoretical computer science. In this paper we show a strong connection between their time complexity. We first discuss each, and then show our results.

Set Cover

In the Set Cover problem, the input is a ground set $[n] = \{1, ..., n\}$ and a collection of m sets, and the goal is to find the smallest sub-collection of sets whose union is the entire ground set. An exhaustive search takes $O(n2^m)$ time, and a dynamic-programming algorithm has running time $O(mn2^n)$ [15], which is faster when $m > n$, a common assumption that we will make throughout. In spite of extensive effort, no algorithm that runs in time $O^*(2^{(1-\varepsilon)n})$ is known, although some improvements are known in special cases [22, 9, 30, 10]. Here and throughout, $O^*(\cdot)$ hides polynomial factors in the instance size, and unless stated otherwise, $\varepsilon > 0$ denotes a fixed constant (and similarly ε'). Thus, it was conjectured that the above running time is optimal [12], even if the input sets are small. To state this more formally, let Δ-Set Cover denote the Set Cover problem where all sets have size at most $\Delta > 0$.

▶ **Conjecture 1.1** (Set Cover Conjecture (SeCoCo) [12]). *For every fixed $\varepsilon > 0$ there is $\Delta(\varepsilon) > 0$, such that no algorithm (even randomized) solves Δ-Set Cover in time $O^*(2^{(1-\varepsilon)n})$.*

This conjecture clearly implies that for every $\Delta = \omega(1)$, no algorithm solves Δ-Set Cover in time $O^*(2^{(1-\varepsilon)n})$. Several conditional lower bounds were based on this conjecture (by reducing Set Cover to it) in the recent decade, including for Steiner Tree, Set Partitioning, and more [12, 11, 8, 24, 25]. The authors of [12] asked whether the problems they reduce Set Cover to can be reduced *back* to Set Cover, so that their running time complexity would stand and fall with SeCoCo. They believed it would be hard to do, since it would probably provide for those problems an alternative algorithm with running time that matches the currently fastest one, which is very complex and took decades to achieve for some (e.g., for Steiner Tree).

Connection to SETH

No formal connection is known to date between the SeCoCo conjecture and the Strong Exponential Time Hypothesis (SETH) of [18], which asserts that for every $\varepsilon > 0$ there exists $k(\varepsilon)$, such that kSAT on N variables and M clauses cannot be solved in time $O^*(2^{(1-\varepsilon)N})$. Cygan et al. [12] provided a partial answer by showing a SETH-based lower bound for a certain variant of Set Cover (that counts the number of solutions). It is known that the weaker assumption ETH implies a $2^{\Omega(n)}$ time lower bound for Set Cover, even if $\Delta = O(1)$, and that SAT can be solved in time $O^*(2^{(1-\varepsilon)N})$ if and only if Set Cover can be solved in time $O^*(2^{(1-\varepsilon')m})$, see [12]. Some researchers hesitate to rely on SeCoCo as a conjecture, and prefer other, more popular conjectures such as SETH. For example, a running time lower bound for Subset Sum was recently shown [1] based on SETH, even though a lower bound based on SeCoCo was already known [12].

We address the necessity of SeCoCo by proposing a weaker assumption, and showing an independent justification for it. Our conjecture deals with Δ-Set Cover for $\Delta = O(\log n)$, as follows.

▶ **Conjecture 1.2** (Logarithmic Set Cover Conjecture (Log-SeCoCo)). *For every fixed $\varepsilon > 0$, there is $\Delta(\varepsilon, n) = O(1/\varepsilon \cdot \log n)$ such that no algorithm (even randomized) solves Δ-Set Cover in time $O^*(2^{(1-\varepsilon)n})$.*

The fastest algorithm known for Δ-Set Cover runs in time $O^*(2^{n\lambda_\Delta})$ [22] for $\lambda_\Delta = (2\Delta - 2)/\sqrt{(2\Delta-1)^2 - 2\ln(2)} \leq 1 - 1/(2\Delta)$, where the inequality assumes $\Delta \geq 2$, hence this running time is slightly faster than for general Set Cover. All known hardness results that are based on SeCoCo can be based also on our conjecture, with appropriate adjustments related to the set sizes in Set Cover parameterized by the universe size plus the solution size [12] and in Parity of Set Covers [8].

Subgraph Isomorphism with a tree pattern

The Subgraph Isomorphism problem asks whether a host graph G contains a copy of a pattern graph H as a subgraph. It is well known to be NP-hard since it generalizes hard problems such as Maximum Clique and Hamiltonicity [21], but unlike many natural NP-hard problems, it requires $N^{\Omega(N)}$ time where $N = |V(G)| + |V(H)|$ is the total number of vertices, assuming the exponential time hypothesis (ETH) [13]. Hence, most past research addressed its special cases that are in P, including the case where the pattern graph is of constant size [28], or when both graphs are trees [2], biconnected outerplanar graphs [26], two-connected series-parallel graphs [27], and more [14, 29]. We will focus on a version called kTree, where the pattern is a tree T on k nodes. In the directed version of the problem, denoted Directed kTree, the edges of G and T are oriented, allowing also anti-parallel edges in G[1]. Throughout, unless accompanied with the word *directed*, kTree and nTree refer to their undirected versions. Directed kTree can only be harder than kTree - even when the directed tree T is an arborescence, as one can reduce the undirected version to it with essentially no loss[2]. A couple of different techniques were used in order to design algorithms for Directed kTree. The color-coding method, designed by Alon, Yuster, and Zwick [3], yields an algorithm with running time $O^*((2e)^k)$. Later, a new method utilized kMLD (stands for k Multilinear Monomial Detection – the problem of detecting multilinear monomials of degree k in polynomials presented as circuits) to design a Directed kTree algorithm with running time $O^*(2^k)$ [23].

Our Results

The first result connects our conjecture to the Directed nTree problem (see Figure 1), which is Directed kTree with $k = n$. This problem includes as a special case the well known Directed Hamiltonicity problem, which asks to determine whether a directed graph G contains a simple path (or cycle) that visits all the nodes (the Hamiltonian cycle and path problems are easily reducible to each other with only small overhead). Next, we show that an algorithm that breaks Log-SeCoCo implies a fast algorithm for Directed nTree.

▶ **Theorem 1.3.** *Suppose Log-SeCoCo fails, namely, there is $\varepsilon > 0$ such that for every $\Delta = O(1/\varepsilon \cdot \log n)$, Δ-Set Cover can be solved in time $O^*(2^{(1-\varepsilon)n})$. Then for some $\delta(\varepsilon) > 0$, Directed nTree on $\tilde{n}$ nodes can be solved in time $O^*(2^{(1-\delta)\tilde{n}})$. This holds even when in Δ-Set Cover, every optimal solution is of size $O(\varepsilon n/\log n)$ and consists of disjoint sets.*

In the special case of Directed Hamiltonicity, we actually reduce to rather constrained instances of Set Cover.

▶ **Theorem 1.4.** *Suppose Log-SeCoCo fails, namely, there is $\varepsilon > 0$ such that for every $\Delta = O(1/\varepsilon \cdot \log n)$, Δ-Set Cover can be solved in time $O^*(2^{(1-\varepsilon)n})$. Then for some $\delta(\varepsilon) > 0$, Directed Hamiltonicity on $\tilde{n}$ nodes can be solved in time $O^*(2^{(1-\delta)\tilde{n}})$. This holds even when in Δ-Set Cover, all sets are of the same size and every optimal solution is of size $O(\varepsilon n/\log n)$ and consists of disjoint sets.*

---

[1] T need not be an arborescence, only its underlying undirected graph is a tree.

[2] This could be done in the following way. Define the host graph G' to be G with edges in both directions, and direct the edges in T away from an arbitrary vertex $v \in T$ to create the directed tree T', which is thus an arborescence. Clearly, the directed instance is a yes-instance if and only if the undirected instance also is.

We can also show that even moderate improvements to the fastest known running time for Δ-Set Cover, namely, to the $O^*(2^{(1-1/2\Delta)n})$ time algorithm of [22], implies improvements for Directed nTree and for Directed Hamiltonicity (Section 4).

A Map of New and Known Reductions

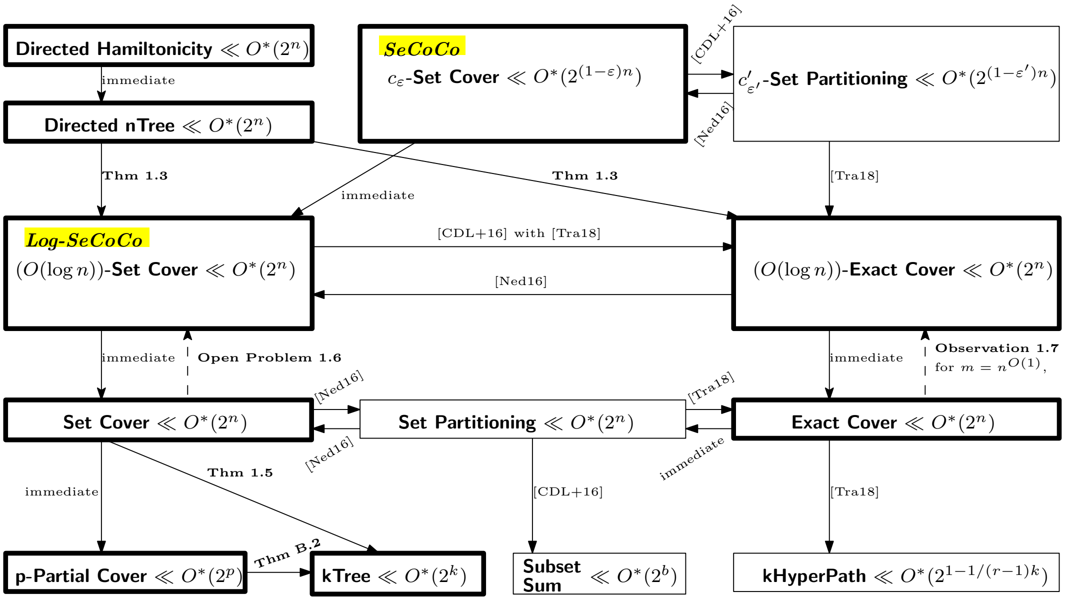

Figure 1 An arrow from a box with $A \ll O^*(2^{n_A})$ to $B \ll O^*(2^{n_B})$ represents a reduction from problem A to problem B, such that if B can be solved in time $O^*(2^{(1-\varepsilon)n_B})$ then A can be solved in time $O^*(2^{(1-\varepsilon'(\varepsilon))n_A})$. We denote by b the number of bits required to represent the integers in Subset Sum, and by r the uniformity parameter in kHyperPath. The problems we focus on are drawn in thick frames.

Our next result, whose proof appears in Section 3, shows that the $2^k\mathsf{poly}(n)$ running time of kTree by [23] is actually optimal (up to exponential improvements) even when considering the undirected version, assuming SeCoCo or even weaker hypotheses such as Log-SeCoCo.

▶ **Theorem 1.5.** *If for some fixed $\varepsilon > 0$, kTree can be solved in time $O^*((2-\varepsilon)^k)$, then for some $\delta(\varepsilon) > 0$, Set Cover on n elements and m sets can be solved in time $O^*((2-\delta)^n)$.*

In fact, our reduction also works from the more general p-Partial Cover problem, whose input is similar to the Set Cover problem but with an additional integer p, and the goal is to find the smallest sub-collection of sets whose union contains at least p elements (rather than all elements). For simplicity, we first present the reduction from Set Cover to kTree (Section 3), and then we show how to adjust it to be from p-Partial Cover (Subsection 3.1).

Discussion

Our first result (Theorem 1.3) supports the validity of Log-SeCoCo based on the Directed nTree problem, which we believe does not admit an $O^*(2^{(1-\varepsilon)n})$-time algorithm, for two reasons. First, this problem includes the well-known Directed Hamiltonicity problem, and in the last 50 years no algorithm significantly faster than $O^*(2^n)$-time was found for it, despite extensive efforts [4, 17, 5, 33] and in contrast to progress on its undirected version [6]. Second, for a generalization of nTree and kTree variants, namely, for Subgraph Isomorphism where the pattern is an arbitrary graph of arbitrary size, a time lower bound $n^{\Omega(n)}$ is known assuming ETH [13], even when the host and pattern graphs have the same number of nodes. We see it as evidence that also Directed kTree does not become easier as the size k of the pattern graph increases all the way to $k = n$, which would imply that the conditional lower bound in Theorem 1.5 which shows that kTree cannot be solved in time $O^*(2^{(1-\varepsilon)k})$, extends to $k = n$. If true, then by our results, solving Set Cover significantly faster than $O^*(2^n)$-time is equivalent to achieving the same running time in the special case of Δ-Set Cover with $\Delta = O(\log n)$, which can be seen as an analogue to the SETH sparsification lemma [19]. Another interesting consequence of our results is that if kTree can be solved significantly faster than $O^*(2^k)$ than Directed nTree can be solved significantly faster than $O^*(2^n)$. Such a reduction from a directed problem to its undirected version is not obvious, even when the latter has extra freedom in the form of parameterization. A potentially interesting conclusion from the special instances of Δ-Set Cover produced in Theorem 1.4, where the goal could be stated as finding a sub-collection of disjoint sets that covers the entire ground set, which we call Exact Cover, is that Directed Hamiltonicity could be more closely related to Exact Cover than to Set Cover. This is despite the fact that Set Cover and Exact Cover were shown to be equivalent with respect to solvability in $O^*(2^{(1-\varepsilon)n})$ time [30, 32], as there is an exponential blowup in the number of sets in the reduction from Set Cover to Exact Cover. As we observe below, Exact Cover with polynomially many sets can indeed be solved significantly faster than $O^*(2^n)$. See Figure 1 for an overview of new and known reductions, where problem A being drawn above problem B implies that there is a path, and a reduction, from A to B. The following open problem formalizes the foregoing discussion.

▶ **Open Problem 1.6.** *Does an $O^*(2^{(1-\varepsilon)n})$-time algorithm for Δ-Set Cover with $\Delta = O(\log n)$ imply an $O^*(2^{(1-\varepsilon'(\varepsilon))n})$-time algorithm for Set Cover?*

Perhaps surprisingly, we can resolve the Exact Cover analogue of Open Problem 1.6 in the special but common case $m = n^{O(1)}$, as follows. Here, $O(c \log n)$-Exact Cover is Exact Cover with sets of size bounded by $O(c \log n)$.

▶ **Observation 1.7.** *If for some fixed $\varepsilon > 0$ and $c > 0$, $O(c \log n)$-Exact Cover can be solved in time $O^*(2^{(1-\varepsilon)n})$, then for some $\delta(\varepsilon) > 0$, Exact Cover with $m = O(n^c)$ can be solved in time $O^*(2^{(1-\delta)n})$.*

To see this, simply guess which sets of size larger than Δ participate in an optimal solution, using an exhaustive search over at most $n \cdot \binom{m}{n/\Delta}$ choices, and then apply the assumed algorithm for the remaining sets.

We note that the results can be easily generalized to weighted Directed Hamiltonicity (i.e., TSP) and Directed nTree by using a generalized conjecture about the weighted version of Set Cover, whose input is similar to the Set Cover only with a positive weight for each set, and the goal is to find a minimum-weight sub-collection whose union is the entire ground set. The generalized conjecture then states that for every fixed $\varepsilon > 0$, weighted Set Cover with the cardinality of every set bounded by $O(1/\varepsilon \cdot \log n)$ cannot be solved in time $O^*(2^{(1-\varepsilon)n})$.

Prior Work

Relevant state-of-the-art algorithms to Set Cover and Subgraph Isomorphism variants are as follows. Set Cover can be solved in time $(m + 2^n)\mathsf{poly}(n)$ [9], which for $m = n^{\omega(1)}$ is faster than the aforementioned $O(mn2^n)$ algorithm of [15]. The case where all sets are of size q and the goal is to determine whether p pairwise-disjoint sets can be packed, can be solved in time $O^*(2^{(1-\varepsilon)pq})$ for $\varepsilon(q) > 0$ [10]. Determining whether a Set Cover instance has a solution of size at most σn can be done in time $O^*(2^{(1-\Omega(\sigma^4))n})$ [30]. The fastest known running time for Directed Hamiltonicity is $O^*(2^{n-\Theta(\sqrt{n/\log n})})$ [7]. Finally, several problems, including Directed Hamiltonicity and Set Cover, were shown to belong to the class EPNL, defined as all problems that can be solved by a non-deterministic turing machine with space $n + O(\log n)$ bits [20].

Techniques

To demonstrate our basic technique for Theorems 1.3 and 1.4, let us present an extremely simple reduction from Directed Hamiltonicity to Δ-Set Cover with $\Delta = O(\log n)$. Given a directed graph G, first guess (by exhaustive search) a relatively small set of nodes ("representatives"), and an ordering for them $z_1, z_2, \ldots$ in a potential Hamiltonian cycle. Then construct a Set Cover instance whose ground set is the nodes of G and has the following sets: for every possible path of length Δ in G from some z_i to z_{i+1} that does not visit any representative in between, there is a set that contains all the nodes in this path except for z_{i+1}. A Hamiltonian cycle in G clearly corresponds to a set cover using exactly n/Δ sets, and vice versa. The main challenge we deal with when reducing from the more general Directed nTree is that the pattern tree does not decompose easily into appropriate subgraphs.

The intuition for Theorem 1.5 is as follows. In the reduction from Set Cover to kTree we first guess a partition of n (the number of elements) that represents how an optimal solution covers the elements, by exhaustive search over $2^{O(\sqrt{n})}$ unordered partitions of n. Then, we represent the Set Cover instance using a Subgraph Isomorphism instance, whose pattern tree T succinctly reflects the guessed partition of n, and the idea is that this tree is isomorphic to a subgraph of the Set Cover graph if and only if the Set Cover instance has a solution that agrees with our guess. The main difficulty here is that we reduce to the undirected version of kTree, and thus additional attention is required to make the tree fit only in specific locations in the host graph.

2 Reduction from Directed nTree to Set Cover

In this section we prove Theorem 1.3. The heart of the proof is actually the following lemma.

▶ **Lemma 2.1.** *Directed nTree on $\tilde{n}$ nodes can be reduced, for every $\Delta \in [\tilde{n}]$, to $O(\tilde{n}^{9\tilde{n}/\Delta})$ instances of Δ-Set Cover, each with $n \le \tilde{n} + 9\tilde{n}/\Delta$ elements, in time $O(\tilde{n}^{\Delta+1} + \tilde{n}^{9\tilde{n}/\Delta})$.*

Proof of Theorem 1.3. Assume there is an algorithm for Δ-Set Cover on n elements and $\Delta = O(1/\varepsilon \cdot \log n)$ that runs in time $O^*(2^{(1-\varepsilon)n})$. Given an instance of Directed nTree on $\tilde{n}$ nodes, apply Lemma 2.1 with

$$\Delta = 81/\varepsilon \cdot \log \tilde{n} = O(1/\varepsilon \cdot \log n),$$

and then solve each of the resulting $O(\tilde{n}^{9\tilde{n}/\Delta}) = O^*(2^{\varepsilon\tilde{n}/9})$ instances of Δ-Set Cover, using the assumed algorithm, in time

$$O^*(2^{(1-\varepsilon)n}) \le O^*(2^{(1-\varepsilon)(\tilde{n}+9\tilde{n}/(81/\varepsilon \cdot \log \tilde{n}))}) \le O^*(2^{(1-\varepsilon)(\tilde{n}+\varepsilon\tilde{n}/(9 \cdot \log \tilde{n}))}).$$

The total running time is

$$O^*(2^{81/\varepsilon \cdot \log^2 \tilde{n} + \log \tilde{n} + \varepsilon\tilde{n}/9 + (1-\varepsilon)(\tilde{n}+\varepsilon\tilde{n}/(9\cdot\log\tilde{n}))}) \le O^*(2^{\tilde{n}-\varepsilon\tilde{n}/2}),$$

which concludes the proof for $\delta(\varepsilon) = \varepsilon/2$. ◀

It remains to prove Lemma 2.1, and we start with an overview of this proof. Consider an instance (G, T) of Directed nTree, and for this overview, assume that the tree T is rooted at some node r, and all edges are directed away from it. The idea is to create roughly $\tilde{n}^{9\tilde{n}/\Delta}$ instances of Δ-Set Cover on $n \leq \tilde{n} + 9\tilde{n}/\Delta$ elements each, such that at least one of them has a solution of size $t \leq 9\tilde{n}/\Delta$ if and only if the instance (G, T) has a solution. The first step is to cover the tree T with t small subtrees, each of size at most Δ, such that the union of their node sets is T and they may intersect only at their roots (the root of a subtree is the node closest to r). Then guess, by enumerating over all possible choices, how the solution to (G, T) maps the root of each subtree to a node in G, and create a corresponding an instance of Δ-Set Cover. For every such instance, perform an inner enumeration to further guess, what is the (unordered) set of nodes in G that each subtree is mapped to, and add a corresponding set to the Δ-Set Cover instance, but only if this guess does not violate the local and global structure of T. That is, taking into account the edges within and between the subtrees, by testing whether the set can be an isomorphic copy of the subtree, testing for the edges between roots, respectively. For the correctness, we need to show that a solution of size t to the Δ-Set Cover instance implies a one-to-one correspondence between the t sets and the roots of the subtrees, and hence a copy of T in G. The general case where the edges of T are orientated arbitrarily is similar, except that the edge orientations are taken into account when comparing subtrees but not when computing a cover of T by small subtrees.

We proceed to the algorithm that computes the aforementioned cover of T by small subtrees. This algorithm traverses the tree using DFS and add subtrees to the cover whenever the DFS accumulates enough nodes, see Algorithm 1 for full details. Its output is a set S, where each $s \in S$ is a connected subset of the nodes of T, and thus we can refer to each such s as a subtree of T, and let $r(s)$ denote its root, i.e., its node that is closest to r in T. The following lemma describes the guarantees of this algorithm and will be later used to prove Lemma 2.1.

▶ **Lemma 2.2.** *Given a tree T with root r on $\tilde{n}$ nodes and an integer $l \leq \tilde{n}$, Algorithm 1 finds in polynomial time a collection S of subtrees of T such that:*

a. *the number of nodes in each subtree is at most $2(l-1)$;*
b. *every node in T is in some subtree;*
c. *two subtrees in S may only intersect in their roots; and*
d. *the number of subtrees is $|S| \leq \frac{3\tilde{n}}{l-1}$.*

Proof of Lemma 2.2. We first show that items (a)–(c) are satisfied by the output of Algorithm 1. Since in the worst case Algorithm 1 adds a subtree in the first time the accumulated number of nodes exceeds l, the number of nodes of each subtree is bounded by $2(l-1)$. In addition, every node v appears in some subtree, since at some point during the DFS it will be the child, and then it will be passed up the tree and eventually added to S. To see why the last requirement holds, observe that whenever an accumulated set is passed up the tree and encounters an existing root, this set will be added to S.

To prove item (d), denote denote by S_{big} the collection of sets in S of size at least l (added in line 6), and by S_{sml} the collection of sets in S of size smaller than l (added in lines 11 and 13). A set $s \in S_{\text{sml}}$ was created only if $r(s)$ at the time of its creation was the root of at least one (other) set in S_{big} (line 11) or was the last traversed node in the DFS (line 13). Together with the fact that each root has at most one set from S_{sml}, we conclude that each set $s \in S_{\text{sml}}$ excluding at most one, can be associated with a distinct set in S_{big}, one that contains $r(s)$. Hence, $|S_{\text{sml}}| - 1 \leq |S_{\text{big}}|$. The big sets have size at least l, and

Algorithm 1

Input: tree T rooted at r and size parameter $l \in [n]$
Output: cover S of T by subtrees of size at most $2(l-1)$

$S \leftarrow \emptyset$
for all $v \in V$ **do** $s(u) \leftarrow \{u\}$
traverse T using a DFS from r, and whenever returning from a node v to its parent p in T, do the following:
    let $s(p) \leftarrow s(p) \cup s(v)$
    if $|s(p)| \geq l$ **then**
        add $s(p)$ to S
        if p has unvisited children **then**
            let $s(p) \leftarrow \{p\}$
        else let $s(p) \leftarrow \emptyset$
    else if p has no unvisited children and $p \in s$ for some $s \in S$ **then**
        add $s(p)$ to S and let $s(p) \leftarrow \emptyset$
    else if p is the last node traversed in the tree **then**
        add $s(p)$ to S
return S

except for their roots they have distinct vertices, hence $|S_{\text{big}}| \leq \frac{\tilde{n}}{l-1}$. We conclude that

$$|S| = |S_{\text{sml}}| + |S_{\text{big}}| \leq 2|S_{\text{big}}| + 1 \leq \frac{2\tilde{n}}{l-1} + 1 \leq \frac{3\tilde{n}}{l-1},$$

which completes the proof of Lemma 2.2. ◀

Proof of Lemma 2.1. We describe the reduction in stages.

- Apply the aforementioned Algorithm 1 for partition T into subtrees that satisfy the conditions in Lemma 2.2. By picking $l = \Delta/3 + 1$, we obtain that each set is bounded by Δ and that $|S| \leq 9\tilde{n}/\Delta$. Hence, the cardinality of $R := \{r(s)\}_{s \in S}$ is bounded by $9\tilde{n}/\Delta$. For S returned by Algorithm 1, let $R_T = \{r(s) : s \in S\}$ (note that $|R_T|$ may be smaller than $|S|$).
- Then, guess $|R_T|$ nodes in G that will function as the image of the nodes in R_T in a potential subgraph isomorphism function and denote them by R_G, and then guess a bijection f from R_T to R_G. The guessing is done by exhaustive search over $\binom{\tilde{n}}{|R_T|}$ choices of nodes, and together with the number of ways to choose a bijection it can be done in time $\binom{\tilde{n}}{|R_T|}|R_T|!$.
- Finally, enumerate all sets s' of nodes of size at most Δ in G, and denote by $G(s')$ the graph induced from each on G. For every subtree $s \in S$, look by brute force for an isomorphic copy of s in subgraphs $G(s')$ that contain $f(r(s))$ as a root and no other node in R_G, and that satisfy $|s'| = |s|$. For each one that was found, add to the constructed Set Cover instance a set s'_G with the root r' labeled r'_s where s corresponds to the subtree s of T whose copy found to be in $G(s')$. Note that the number of elements in the Set Cover instance is exactly $\tilde{n} - |R_T| + |S|$, and that the time spent per each subgraph isomorphism test is at most $|s|! \leq \Delta!$, and thus the total time spent in this step is $|S|\binom{\tilde{n}}{\Delta}\Delta! = |S|\tilde{n} \cdot (\tilde{n}-1) \cdots (\tilde{n}-\Delta+1) \leq 9\tilde{n}/\Delta \cdot \tilde{n}^{\Delta} \leq \tilde{n}^{\Delta+1}$.

Now we show that the size constraints follow. As $|R_T| \leq 9\tilde{n}/\Delta$, similar to before, the number of Set Cover instances is bounded by

$$\binom{\tilde{n}}{9\tilde{n}/\Delta}(9\tilde{n}/\Delta)! = \tilde{n} \cdot (\tilde{n}-1) \cdots (\tilde{n} - 9\tilde{n}/\Delta + 1) \leq \tilde{n}^{9\tilde{n}/\Delta}$$

as required.

We now prove that at least one of the Set Cover instances has solution of size at most $|S|$ (in fact exactly $|S|$ as no smaller solutions available) if and only if the Directed nTree instance is a yes instance. For the first direction, assume that the Directed nTree instance is a yes instance. Considering the isomorphic copy of T in G, its $|S|$ subtrees as Algorithm 1 outputs on T will be sets in the Set Cover instance the reduction outputs, and so it has a solution of size at most $|S|$. For the second direction, if a Set Cover instance has a solution I of size at most $|S|$ and since the number of labeled roots is $|S|$, it must be that for each subtree $s \in S$ its labeled root is in exactly one set in I, and so $|I| = |S|$. Since I is a legal solution and S covers all the nodes, no node in $V(G) \setminus R_G$ appears twice in I. The conclusion is that these sets together form the required tree, concluding the proof of Lemma 2.1. ◀

We note that in the case of Theorem 1.4 for Directed Hamiltonicity, we do not have to use Algorithm 1, but simply guess n/Δ representative nodes in G and their ordering in the potential cycle, and then enumerate all paths of size Δ to represent paths between consecutive representatives. Hence we obtain a Δ-Set Cover instance with the additional constraints of Theorem 1.4.

3 Reduction from Set Cover to kTree

In this section we prove Theorem 1.5. In order to make the proof simpler, we will have an assumption regarding the Set Cover instance, as follows. For a constant $g > 0$ to be determined later, we can assume that all the sets in the Set Cover instance are of size at most n/g^2, as otherwise such instance can already be solved significantly faster than $O^*(2^n)$, proving the theorem in a degenerate manner. We formalize it as follows.

▶ **Assumption 3.1.** *All the sets in the Set Cover instance are of size at most n/g^2.*

To justify this assumption, notice that one can remove all sets of size more than n/g^2 from the Set Cover instance. Indeed, if some optimal solution for the Set Cover instance contains a set of size at least n/g^2, such optimal solution can be found by simply guessing one set of at least this size (using exhaustive search over at most m choices) and then applying the known dynamic programming algorithm on the still uncovered elements (at most $n - n/g^2$ of them), and return the optimal solution in total time $O^*(2^{(1-1/g^2)n})$. We continue to the following lemma, which is the heart of the proof.

▶ **Lemma 3.2.** *For every fixed $\varepsilon > 0$, Set Cover on a ground set $N = [n]$ and a collection M of m sets that satisfies assumption 3.1, can be reduced to $2^{O(\sqrt{n})}$ instances of kTree with $k = (1+\varepsilon)n + O(1)$.*

We will use this lemma to prove Theorem 1.5, the proof of Lemma 3.2 will be given after.

▶ **Theorem 1.5** (*restated*). *If for some fixed $\varepsilon > 0$, kTree can be solved in time $O^*((2-\varepsilon)^k)$, then for some $\delta(\varepsilon) > 0$, Set Cover on n elements and m sets can be solved in time $O^*((2-\delta)^n)$.*

Proof of Theorem 1.5. Assume that for some $\varepsilon' \in (0,1)$, kTree can be solved in time $O^*((2-\varepsilon')^k) \le O^*(2^{(1-\varepsilon'/2)k})$. We reduce the Set Cover instance by applying Lemma 3.2 with $\varepsilon = \varepsilon'/4$, and then solve each of the $2^{c_1\sqrt{n}}$ instances of kTree in the assumed time of $O^*(2^{(1-\varepsilon'/2)((1+\varepsilon)n+c_2)})$, where $c_1, c_2 > 0$ are the constants implicit in the terms $2^{O(\sqrt{n})}$ and $O(1)$ in the lemma, respectively. The total running time is $O^*(2^{(1-\varepsilon'/2)(1+\varepsilon)n+c_1\sqrt{n}}) = O^*(2^{(1-\varepsilon'/4-\varepsilon'^2/8)n+c_1\sqrt{n}}) \le O^*(2^{(1-\varepsilon'/4)n}) \le O^*((2-\varepsilon'/4)^n)$, which concludes the proof for $\delta(\varepsilon') = \varepsilon'/4$. ◀

To outline the proof of Lemma 3.2, we will need the following definition. For an integer $a > 0$, let $p(a)$ be the set of all unordered partitions of a, where a *partition* of a is a way of writing a as a sum of positive integers, and *unordered* means that the order of the summands is insignificant. The asymptotic behaviour of $|p(a)|$ (as a tends to infinity) is known [16] to be

$$e^{\pi\sqrt{2a/3}}/(4a\sqrt{3}) = 2^{O(\sqrt{a})}.$$

It is possible to enumerate all the partitions of a with constant delay between two consecutive partitions, exclusive of the output [31, Chapter 9].

Now the intuition for our reduction of Set Cover to kTree is to first guess a partition of n (the number of elements) that represents how an optimal solution covers the elements, as follows. Associate each element arbitrarily with one of the sets that contain it (so in effect, we assume each element is covered only once) and count how many elements are covered by each set in the optimal solution. This guessing is done by exhaustive search over $p(n) \le 2^{O(\sqrt{n})}$ partitions of n. Then, we represent the Set Cover instance using a Subgraph Isomorphism instance, whose pattern tree T succinctly reflects the guessed partition of n. The idea is that the tree is isomorphic to a subgraph of the Set Cover graph if and only if the Set Cover instance has a solution that agrees with our guess.

► **Lemma 3.2** (*restated*). *For every fixed $\varepsilon > 0$, Set Cover on a ground set $N = [n]$ and a collection M of m sets that satisfies assumptions 3.1, can be reduced to $2^{O(\sqrt{n})}$ instances of kTree with $k = (1+\varepsilon)n + O(1)$.*

Proof of Lemma 3.2. Given a Set Cover instance on n elements $N = \{n_i : i \in [n]\}$ and m sets $M = \{S_i\}_{i\in[m]}$ and an $\varepsilon > 0$, construct $2^{O(\sqrt{n})}$ instances of kTree as follows. For a constant $g(\varepsilon)$ to be determined later, the host graph $G_g = (V_g, E_g)$ is the same for all the instances, and is built on the bipartite graph representation of the Set Cover instance, with some additions. This is done in a way that a constructed tree will fit in G_g if and only if the Set Cover instance has a solution that corresponds to the structure of the tree, as follows (see Figure 2). The set of nodes is $V_g = N \cup M \cup M_g \cup R \cup \{r_g, r_1, r_2, r\}$, where $M_g = \{X \subseteq M : |X| = g\}$ and $R = \{v_j^i : i \in [4], j \in [n/(g/2)]\}$. Intuitively, the role of M_g is to keep the size of the trees small by representing multiple vertices in M (multiple sets in Set Cover) at once as the "powering" technique for Set Cover done in [12][3], and the role of R and $\{r_g, r_1, r_2, r\}$ is to enforce that the trees the reduction constructs will fit only in certain ways.

The set of edges is constructed as follows. Edges between N and M are the usual bipartite graph representation of Set Cover (i.e., connect vertices $n_j \in N$ and $S_i \in M$ whenever $n_j \in S_i$). Also, connect vertex $X \in M_g$ to vertex $n_j \in N$ if at least one of the sets

[3] We can slightly simplify this step in the construction by using the equivalence from [12] between solving Set Cover in time $O^*(2^{(1-\varepsilon)n})$ and in time $O^*(2^{(1-\varepsilon')(n+t)})$ where t is the solution size. However, we opted to reduce directly from Set Cover for compatibility with our parameters and for sake of generality.

in X contains n_j. Additionally, add edges between r_g and every vertex in M_g, and $v_j^4 \in R$ for $j \in [n/(g/2)]$, between r_i and v_j^i for every $i \in \{1,2\}$ and $j \in [n/(g/2)]$, and finally between r and every vertex $v \in \{r_g, r_1, r_2\}$, $S_i \in M$, and $v_j^3 \in R$ for $j \in [n/(g/2)]$.

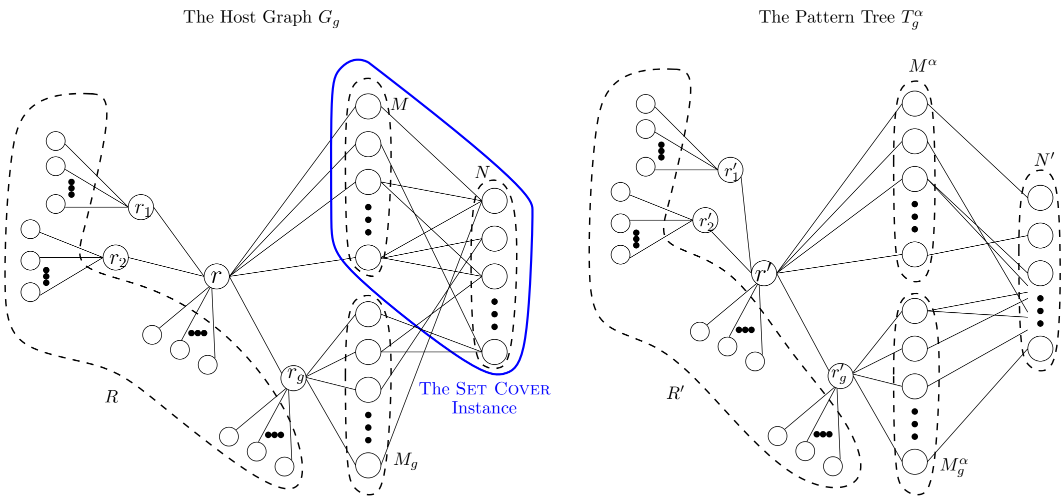

Figure 2 An illustration of part of the reduction. The Set Cover instance is depicted in blue, and sets of vertices are indicated by dashed curves.

Next, construct $2^{O(\sqrt{n})}$ trees such that identifying those that are isomorphic to a subgraph of G_g will determine the optimum of the Set Cover instance.

For every partition $\alpha = (p_1, p_2, ..., p_l) \in p(n)$ (with possible repetitions) where $p(n)$ is as defined above, construct a tree $T_g^\alpha = (V_g^\alpha, E_g^\alpha)$. This tree has the same set of edges and vertices as G_g, except for the vertices in $M \cup M_g$ and the edges incident to them, which are replaced by a set of new vertices $M^\alpha \cup M_g^\alpha$, and these new vertices are connected to the rest in a way that the resulting graph is a tree. In more detail, $V_g^\alpha = N' \cup M^\alpha \cup M_g^\alpha \cup R' \cup \{r'_g, r'_1, r'_2, r'\}$ where $N', R', r'_g, r'_1, r'_2, r'$ are tagged copies of the originals, and M^α, M_g^α are initialized to be $\emptyset$.

We define α_g to be a partition of n which is also a shrinked representation of α by partitioning α into sums of g numbers for a total of $\lfloor l/g \rfloor$ such sums, and a remaining of less than g numbers. Formally,

$$\alpha_g = (\sum_{i=1}^{g} p_i, \sum_{i=g+1}^{2g} p_i, ..., \sum_{i=(g-1)\cdot\lfloor l/g \rfloor+1}^{g\lfloor l/g \rfloor} p_i, p_{g\lfloor l/g \rfloor+1}, ..., p_l)$$

Note that all the numbers in α_g are a sum of g numbers in α, except (maybe) for the last $g' := l - g\lfloor l/g \rfloor < g$ numbers in α_g, a (multi)set which we denote $s(\alpha_g)$. For every $i \in \alpha_g$ (with possible repetitions), add a star on $i+1$ vertices to the constructed tree T_g^α. If $i \in \alpha_g \setminus s(\alpha_g)$, add the center vertex to M_g^α, connect it to r'_g, and add the rest i vertices to N'. Else, if $i \in s(\alpha_g)$, add the center vertex to M^α, connect it to r', and again add the rest i vertices to N'. Return the minimum cardinality of α for which (G_g, T_g^α) is a yes-instance. To see that this construction is small enough, note that the size of G_g is at most $4 + 4 \cdot n/(g/2) + m^g + m + n$ which is polynomial in m, and the size of the tree T_g^α is at most

$$4 + 4 \cdot n/(g/2) + n/g + g + n = n \cdot (1 + 9/g) + O(1) = n \cdot (1 + \varepsilon) + O(1)$$

where the last equality holds for $g = 9/\varepsilon$, and so the size constraint follows.

We now prove that at least one of the trees T_g^α returns yes and satisfies $|\alpha| \leq d$, if and only if the Set Cover instance has a solution of size at most d. For the first direction, assume that the Set Cover instance has a solution I with $|I| \leq d$. Consider a partition $\alpha_I \in p(n)$ of n that corresponds to I in the following way. Associate every element with exactly one of the sets in I that contains it, and then consider the list of sizes of the sets in I according to this association (eliminating zeroes). Clearly, $(G_g, T_g^{\alpha_I})$ is a yes-instance and so the reduction will return a number that is at most $|I|$.

For the second direction, assume that every solution to the Set Cover instance is of size at least $d+1$. We need to prove that for every tree T_g^α with $|\alpha| \leq d$, (G_g, T_g^α) is a no-instance. Assume for the contrary that there exists such α for which (G_g, T_g^α) is a yes-instance with the isomorphism function f from T_g^α to G_g. We will show that the only way f is feasible is if $f(r') = r$, $f(M^\alpha) \subseteq M$, $f(M_g^\alpha) \subseteq M_g$, and also $f(N') = N$, which together allows us to extract a corresponding solution for the Set Cover instance, leading to a contradiction. We start with the vertex $r' \in T_g^\alpha$. Since its degree is at least $n/(g/2) + 3$ and by Assumption 3.1 and the construction of G_g, it holds that $f(r') \notin \{r_1, r_2\} \cup R \cup M \cup M_g$. Moreover, if it was the case that $f(r') \in \{r_g\} \cup N$ then $\{f(r_1'), f(r_2')\} \cap (M \cup M_g) \neq \emptyset$, however, the degree of r_1' and r_2' in T_g^α is $n/(g/2)$, and the degree of the vertices in $M \cup M_g$ in G_g is at most $g \cdot n/g^2 = n/g$, so it must be that $f(r) = r$. Our next claim is that $f(r_g') = r_g$. Observe that Assumption 3.1 implies that every solution for the Set Cover instance is of size at least g^2 and so $M_g^\alpha \neq \emptyset$, which means r_g' in the tree has vertices in distance 2 from it and away from r', a structural constraint that cannot be satisfied by any vertex in $\{r_1, r_2\} \cup R$. Furthermore, the degree of r_g' is at least $n/(g/2)$ and so again by Assumption 3.1 it is also impossible that $f(r_g') \in M^\alpha$, and hence it must be that $f(r_g') = r_g$. Finally, by the same Assumption and the degrees of r_1 and r_2, $f(r_1')$ and $f(r_2')$ must be in $\{r_1, r_2\}$. Altogether, it must be that $f(M_g^\alpha) \subseteq M_g$, $f(M^\alpha) \subseteq M$ and that $f(N') = N$, and therefore it is possible to extract a feasible solution to the Set Cover instance that has at most d sets in it, which is a contradiction, concluding the proof of Lemma 3.2.

3.1 Reduction from p-Partial Cover

In this subsection we show that Theorem 1.5 is correct also assuming a weaker conjecture, that p-Partial Cover cannot be solved significantly faster than $O^*(2^p)$. Notice that p-Partial Cover can be solved in time $O^*(2^p)$ by a simple application of the method in [23], as pointed out to us by Cornelius Brand and anonymous referees. We now reduce from p-Partial Cover to Directed kTree by following Lemma 3.2 with the following adjustments.

Instead of enumerating over all the partitions of n, do it only for p and hence the number of partitions is $2^{O(\sqrt{p})}$ with each partition α inducing a tree T_g^α in a similar way to Lemma 3.2, of size at most $2p/g + p$. Note that Assumption 3.1 adjusted to the p-Partial Cover case hold also here, since it is possible to use the $O^*(2^p)$-time algorithm for p-Partial Cover mentioned above after removing large sets of size $\geq p/g^2$. From here onwards, the proof of correctness is similar to Lemma 3.2, and thus we omit it. Regarding running time, assume that for some $\varepsilon' \in (0,1)$, kTree can be solved in time $O^*((2-\varepsilon')^k) \leq O^*(2^{(1-\varepsilon'/2)k})$. Setting $g = 8/\varepsilon'$ for $\varepsilon' = 64(1 - \log_2(2-\varepsilon))$ (without loss of generality, assume that ε' is small enough), we get a total running time of

$$\begin{aligned}
&O(m^{c_1 g} 2^{p-p/g^2} + 2^{(p+2p/g)(1-\varepsilon'/2)+c_2\sqrt{p}} \cdot m^{c_3 g}) \\
&\qquad = O(m^{c_1 8/\varepsilon'} 2^{p-\varepsilon' p/64} + 2^{p+\varepsilon'/4\cdot p-\varepsilon'/2\cdot p-\varepsilon'^2/8\cdot p+c_1\sqrt{p}} \cdot m^{c_3 4/\varepsilon'}) \\
&\qquad \leq O(2^{(1-\varepsilon'/64)p} \cdot m^{c_1 4/\varepsilon'}) \\
&\qquad \leq O((2-\varepsilon)^p \cdot m^{c_1 4/\varepsilon}),
\end{aligned}$$

where c_1 is the constant derived from the method of [23], c_2 is the constant implicit in the term $2^{O(\sqrt{p})}$, and c_3 is the constant in the exponent of m implicit in the term $O^*(2^{(1-)p})$, as required.

▶ **Lemma 3.3.** *For every fixed $\varepsilon > 0$, p-Partial Cover on a ground set $N = [n]$ and a collection M of m sets can be reduced to $2^{O(\sqrt{p})}$ instances of kTree with $k = (1+\varepsilon)p + O(1)$.*

We thus proved the following theorem.

▶ **Theorem 3.4.** *If for some fixed $\varepsilon > 0$, kTree can be solved in time $O^*((2-\varepsilon)^k)$, then for some $\delta(\varepsilon) > 0$, p-Partial Cover on n elements and m sets can be solved in time $O^*((2-\delta)^p)$.*

4 Moderate Improvements to Δ-Set Cover Imply New Algorithms for Directed nTree and Directed Hamiltonicity

In this section we show how moderate improvements for variants of Set Cover imply new algorithms for Directed nTree. Given any algorithm for Δ-Set Cover with runtime $f(n, m, \Delta)$, by Lemma 2.1 Directed nTree admits an algorithm with running time $O(\tilde{n}^{\Delta} + \tilde{n}^{\tilde{n}/\Delta} f(n, m, \Delta))$. We now demonstrate how this algorithm behaves with different regimes of Δ.

If there exists $\varepsilon > 0$ such that for every $\Delta = \mathsf{poly}(\log n)$, $f(n, m, \Delta) = O^*(2^{(1-1/\Delta^{1-\varepsilon})n})$ then by considering $\Delta = \log^{(1+\varepsilon')/\varepsilon} n = \mathsf{poly}(\log n)$ for $\varepsilon' > 0$, Directed nTree has an algorithm with runtime

$$O(2^{\log^{(1+\varepsilon')/\varepsilon+1} \tilde{n}}) + O^*(2^{\tilde{n}/\log^{(1+\varepsilon')/\varepsilon-1} \tilde{n}} \cdot 2^{(1-1/(\log^{(1+\varepsilon')/\varepsilon} \tilde{n})^{1-\varepsilon})\tilde{n}})$$
$$= O^*(2^{(1-1/(\log^{(1+\varepsilon')/\varepsilon-2} \tilde{n}))\tilde{n}})$$

Considering larger regimes, if for some fixed $\varepsilon > 0$, $\delta \in (0, 1/2)$, and $\Delta = O(n^{\delta})$, $f(n, m, \Delta) = O^*(2^{(1-\frac{(1+\varepsilon)\log\Delta}{\delta\Delta})n})$ then Directed nTree can be solved in time

$$2^{\tilde{n}^{\delta} \log \tilde{n}} + 2^{\tilde{n}^{1-\delta} \log \tilde{n}} \cdot O^*(2^{(1-\frac{(1+\varepsilon)\log\Delta}{\delta\Delta})\tilde{n}}) = O^*(2^{(1-\varepsilon/\tilde{n}^{\delta})\tilde{n}}) = 2^{\tilde{n}-\Theta(\tilde{n}^{1-\delta})}$$

Note that to break the fastest known $2^{\tilde{n}-\Theta(\sqrt{\tilde{n}/\log\tilde{n}})}$ algorithm for Directed Hamiltonicity by [7], it is enough to have either $f(n, m, \Delta) = O^*(2^{(1-\frac{(2+\varepsilon)\log\Delta}{\Delta})n})$ for $\Delta = n^{1/2-\delta'}$ with every fixed $\delta' > 0$, or $f(n, m, \Delta) = O(m \cdot 2^{(1-\frac{(4+\varepsilon)\log\Delta}{\Delta})n})$ for $\Delta = \sqrt{n}$, taking into account that most algorithms for variants of Set Cover that have the factor m in their runtime, do not have it with higher power than one. ◀

References

1 Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. SETH-Based Lower Bounds for Subset Sum and Bicriteria Path. In *30th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '19, pages 41–57, 2019. doi:10.1137/1.9781611975482.3.

2 Amir Abboud, Virginia Vassilevska-Williams, and Huacheng Yu. Matching Triangles and Basing Hardness on an Extremely Popular Conjecture. In *Proceedings of the Forty-seventh Annual ACM Symposium on Theory of Computing*, STOC '15, pages 41–50. ACM, 2015. doi:10.1145/2746539.2746594.

3 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, July 1995. doi:10.1145/210332.210337.

4 Richard Bellman. Combinatorial processes and dynamic programming. In *Combinatorial analysis*, Proceedings of Symposia in Applied Mathematics, pages 217–249. American Mathematical Society, 1960. doi:10.1090/psapm/010.

5 Richard Bellman. Dynamic Programming Treatment of the Travelling Salesman Problem. *J. ACM*, 9(1):61–63, 1962. `doi:10.1145/321105.321111`.

6 Andreas Bjorklund. Determinant sums for undirected hamiltonicity. *SIAM Journal on Computing*, 43(1):280–299, 2014. `doi:10.1137/110839229`.

7 Andreas Björklund. Below All Subsets for Some Permutational Counting Problems . In *15th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2016)*, volume 53 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 17:1–17:11, 2016. `doi:10.4230/LIPIcs.SWAT.2016.17`.

8 Andreas Björklund, Dell Holger, and Thore Husfeldt. The Parity of Set Systems under Random Restrictions with Applications to Exponential Time Problems. In *42nd International Colloquium on Automata, Languages and Programming (ICALP 2015)*, volume 9134, pages 231–242. Springer, 2015. `doi:10.1007/978-3-662-47672-7_19`.

9 Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set Partitioning via Inclusion-Exclusion. *SIAM J. Comput.*, 39(2):546–563, July 2009. `doi:10.1137/070683933`.

10 Andreas Björklund, Thore Husfeldt, Kaski Ptteri, and Mikko Koivisto. Narrow Sieves for Parameterized Paths and Packings. *Journal of Computer and System Sciences*, 87:119–139, 2017. `doi:10.1016/j.jcss.2017.03.003`.

11 Andreas Björklund, Petteri Kaski, and Lukasz Kowalik. Constrained Multilinear Detection and Generalized Graph Motifs. *Algorithmica*, 74(2):947–967, 2016. `doi:10.1007/s00453-015-9981-1`.

12 Marek Cygan, Holger Dell, Daniel Lokshtanov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. On Problems As Hard As CNF-SAT. *ACM Transactions on Algorithms*, 12(3):41:1–41:24, 2016. `doi:10.1145/2925416`.

13 Marek Cygan, Fedor V. Fomin, Alexander Golovnev, Alexander S. Kulikov, Ivan Mihajlin, Jakub Pachocki, and Arkadiusz Socała. Tight Bounds for Graph Homomorphism and Subgraph Isomorphism. In *27th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '16, pages 1643–1649. SIAM, 2016. `doi:10.1137/1.9781611974331.ch112`.

14 Anders Dessmark, Andrzej Lingas, and Andrzej Proskurowski. Faster Algorithms for Subgraph Isomorphism of k-Connected Partial k-Trees. *Algorithmica*, 27(3):337–347, January 2000. `doi:10.1007/s004530010023`.

15 Fedor V. Fomin, Dieter Kratsch, and Gerhard J. Woeginger. Exact (Exponential) Algorithms for the Dominating Set Problem. In *30th International Conference on Graph-Theoretic Concepts in Computer Science*, WG'04, pages 245–256. Springer-Verlag, 2004. `doi:10.1007/978-3-540-30559-0_21`.

16 Godfrey H. Hardy and Srinivasa Ramanujan. Asymptotic Formulæ in Combinatory Analysis. *Proceedings of the London Mathematical Society*, s2-17(1):75–115, 1918. `doi:10.1112/plms/s2-17.1.75`.

17 Michael Held and Richard M. Karp. A Dynamic Programming Approach to Sequencing Problems. In *Proceedings of 16th ACM National Meeting*, ACM '61, pages 71.201–71.204. ACM, 1961. `doi:10.1145/800029.808532`.

18 Russell Impagliazzo and Ramamohan Paturi. On the Complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, March 2001. `doi:10.1006/jcss.2000.1727`.

19 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which Problems Have Strongly Exponential Complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001. `doi:10.1006/jcss.2001.1774`.

20 Yoichi Iwata and Yuichi Yoshida. On the Equivalence among Problems of Bounded Width. In *23rd Annual European Symposium on Algorithms (ESA 2015)*, pages 754–765. Springer, 2015. `doi:10.1007/978-3-662-48350-3_63`.

21 Richard M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. The IBM Research Symposia Series. Springer US, 1972. `doi:10.1007/978-1-4684-2001-2_9`.

22 Mikko Koivisto. Partitioning into Sets of Bounded Cardinality. In *Parameterized and Exact Computation (IWPEC 2009)*, volume 5917 of *Lecture Notes in Computer Science*, pages 258–263. Springer-Verlag, 2009. doi:10.1007/978-3-642-11269-0_21.

23 Ioannis Koutis and Ryan Williams. LIMITS and applications of group algebras for parameterized problems. *ACM Trans. Algorithms*, 12(3):31:1–31:18, May 2016. doi:10.1145/2885499.

24 Lukasz Kowalik and Juho Lauri. On Finding Rainbow and Colorful Paths. *Theoretical Computer Science*, 628(C):110–114, 2016. doi:10.1016/j.tcs.2016.03.017.

25 R. Krithika, Abhishek Sahu, and Prafullkumar Tale. Dynamic Parameterized Problems. In *11th International Symposium on Parameterized and Exact Computation (IPEC 2016)*, volume 63 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 19:1–19:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.IPEC.2016.19.

26 Andrzej Lingas. Subgraph Isomorphism for Biconnected Outerplanar Graphs in Cubic Time. *Theoretical Computer Science*, 63(3):295–302, 1989. doi:10.1016/0304-3975(89)90011-X.

27 László Lovász and Michael D Plummer. *Matching theory*, volume 367. American Mathematical Society, 2009.

28 Dániel Marx and Michal Pilipczuk. Everything you always wanted to know about the parameterized complexity of Subgraph Isomorphism (but were afraid to ask). In *31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014)*, volume 25 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 542–553. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2014. doi:10.4230/LIPIcs.STACS.2014.542.

29 Jiří Matoušek and Robin Thomas. On the complexity of finding iso- and other morphisms for partial k-trees. *Discrete Mathematics*, 108(1):343–364, 1992. doi:10.1016/0012-365X(92)90687-B.

30 Jesper Nederlof. Finding Large Set Covers Faster via the Representation Method. In *24th Annual European Symposium on Algorithms (ESA 2016)*, volume 57 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 69:1–69:15. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.ESA.2016.69.

31 Albert Nijenhuis and Herbert S. Will. *Combinatorial Algorithms: For Computers and Hard Calculators*. Academic Press, 2nd edition, 1978.

32 Ohad Trabelsi. Nearly Optimal Time Bounds for kPath in Hypergraphs. *CoRR*, 2018. URL: http://arxiv.org/abs/1803.04940.

33 Gerhard J. Woeginger. Exact Algorithms for NP-hard Problems: A Survey. In Michael Jünger, Gerhard Reinelt, and Giovanni Rinaldi, editors, *Combinatorial Optimization - Eureka, You Shrink!*, pages 185–207. Springer-Verlag, 2003. doi:10.1007/3-540-36478-1.