Independent Set on P_k-Free Graphs in Quasi-Polynomial Time

Peter Gartland
Department of Computer Science
University of California, Santa Barbara
Goleta, USA
petergartland@ucsb.edu

Daniel Lokshtanov
Department of Computer Science
University of California, Santa Barbara
Goleta, USA
daniello@ucsb.edu

***Abstract*—We present an algorithm that takes as input a graph G with weights on the vertices, and computes a maximum weight independent set S of G. If the input graph G excludes a path P_k on k vertices as an induced subgraph, the algorithm runs in time $n^{O(k^2 \log^3 n)}$. Hence, for every fixed k our algorithm runs in quasi-polynomial time. This resolves in the affirmative an open problem of [Thomassé, SODA'20 invited presentation]. Previous to this work, polynomial time algorithms were only known for P_4-free graphs [Corneil et al., DAM'81], P_5-free graphs [Lokshtanov et al., SODA'14], and P_6-free graphs [Grzesik et al., SODA'19]. For larger values of t, only $2^{O(\sqrt{kn \log n})}$ time algorithms [Bacsó et al., Algorithmica'19] and quasi-polynomial time approximation schemes [Chudnovsky et al., SODA'20] were known. Thus, our work is the first to offer conclusive evidence that INDEPENDENT SET on P_k-free graphs is not NP-complete for any integer k.**

Additionally we show that for every graph H, if there exists a quasi-polynomial time algorithm for INDEPENDENT SET on C-free graphs for every connected component C of H, then there also exists a quasi-polynomial time algorithm for INDEPENDENT SET on H-free graphs. This lifts our quasi-polynomial time algorithm to T_k-free graphs, where T_k has one component that is a P_k, and $k-1$ components isomorphic to a *fork* (the unique 5-vertex tree with a degree 3 vertex).

***Index Terms*—graph algorithm, independent set, P_k-free graphs**

I. INTRODUCTION

An *independent set* (also known as a *stable set*) in a graph G is a vertex set S such that no pair of distinct vertices in S are adjacent in G. In the INDEPENDENT SET problem the input is a graph G on n vertices and integer k, the task is to determine whether G contains an independent set S of size at least k. INDEPENDENT SET is a well-studied and fundamental graph problem which is NP-complete [1], [2] and intractable within most frameworks for coping with NP-hardness. Indeed, INDEPENDENT SET was one of the very first problems to be shown to be NP-hard to approximate [3], [4], one of the first intractable problems from the perspective of parameterized complexity [5], one of the first problems to be shown not to have a $2^{o(n)}$ time algorithm assuming the Exponential Time Hypothesis (ETH) [6], and one of the very first problems whose hardness of parameterized approximation, assuming the Gap-ETH, was established [7].

With the above in mind, it is natural that a significant research effort has been devoted to identifying classes of input graphs for which the INDEPENDENT SET problem is substantially easier than on general graphs. Of particular interest are the classes where INDEPENDENT SET becomes polynomial time solvable. Most famously the problem becomes polynomial time solvable on Perfect graphs [8], other examples of polynomial time solvable cases include $k \times K_2$-free graphs [9] and graphs of bounded cliquewidth [10]. For an extensive list, see [11] and the companion website [12]. On the other hand the problem remains NP-complete even on planar graphs of maximum degree 3 [13], unit disc graphs [14], triangle-free graphs [15] and AT-free graphs [16].

This paper fits in a long line of work to precisely classify the complexity of INDEPENDENT SET on all *hereditary* graph classes defined by a single forbidden induced subgraph H (and more generally, by a finite set $\mathcal{H}$ of forbidden induced subgraphs). A graph G is said to be *H-free* if G does not contain a copy of H as an induced subgraph. For a set $\mathcal{H}$ of graphs, G is *$\mathcal{H}$-free* if G is H-free for all $H \in \mathcal{H}$. The ultimate goal of this research direction is to establish a dichotomy theorem that for every finite set $\mathcal{H}$ of graphs determines whether INDEPENDENT SET on $\mathcal{H}$-free graphs is polynomial time solvable, or NP-complete [1].

In 1982 Alekseev [18] observed that INDEPENDENT SET remains NP-complete on the class of $\mathcal{H}$-free graphs for every finite set $\mathcal{H}$ that does not include a graph H whose every connected component is a path or a

[1]There is of course the possibility that INDEPENDENT SET on $\mathcal{H}$-free graphs has NP-intermediate complexity for some choice of $\mathcal{H}$. We believe this is unlikely, however that is pure speculation.

Full version of this paper is available at arxiv.org [17].

Daniel Lokshtanov is supported by United States - Israel Binational Science Foundation grant no. 2018302.

2575-8454/20/$31.00 ©2020 IEEE
DOI 10.1109/FOCS46700.2020.00063

subdivision of the claw ($K_{1,3}$). Since then, no new NP-completeness results for INDEPENDENT SET on $\mathcal{H}$-free graphs have been found for any other finite set $\mathcal{H}$. Thus, it is consistent with current knowledge that INDEPENDENT SET is polynomial time solvable on $\mathcal{H}$-free graphs for all other finite sets $\mathcal{H}$. At the same time, progress on algorithms has been embarrassingly slow. The only connected graphs H for which NP-completness of INDEPENDENT SET does not follow from Alekseev's result are paths and subdivisions of the claw. Polynomial time algorithms for INDEPENDENT SET on claw-free graphs were found independently by Sbihi [19] and Minty [20] in 1980. A polynomial time algorithm on *fork*-free graphs (a fork is a claw with one subdivided edge) was found by Alekseev [21]. Subsequently, Lozin and Milanic [22] gave an algorithm for WEIGHTED INDEPENDENT SET on fork-free graphs. For paths, INDEPENDENT SET on P_4-free graphs was shown to be polynomial time solvable by Corneil et al. [23] in 1981. After a series of papers giving polynomial time algorithms for various subclasses of P_5-free graphs [24]–[29], in 2014 Lokshtanov et al. [30] gave a polynomial time algorithm on P_5 free graphs. Two years later, Lokshtanov et al. [31] devised a quasi-polynomial time algorithm on P_6-free graphs, before Grzesik et al. [32] designed a polynomial time algorithm for P_6-free graphs in 2019. This summarizes the state-of-the-art for polynomial time solvability of INDEPENDENT SET on H-free graphs.

It appears that the currently known techniques are very far from being able to yield polynomial time algorithms for INDEPENDENT SET on P_k-free graphs for $k = 8$, let alone for all values of k. More concretely, the polynomial time algorithms for P_5-free graphs of Lokshtanov et al. [30] and for P_6-free graphs of Grzesik et al. [32] are based on the same method. First, from a sample of two articles the complexity of applying this method seems to grow exponentially with k. Second, and more importantly, in a recent manuscript Grzesik et al. [33] show that a crucial component of this method fails completely on P_k-free graphs for $k \geq 8$.

The slow progress on polynomial time algorithms have prompted researchers to look for weaker forms of tractability of INDEPENDENT SET on P_k-free graphs. Bacsó et al. [34] provided $2^{O(\sqrt{kn \log n})}$ time algorithms for INDEPENDENT SET on P_k-free graphs (see also [35], [36]). Finally, Chudnovsky et al. [37] obtained quasi-polynomial time approximation schemes for P_k-free graphs for all k. In fact their result is much more general - they obtain quasi-polynomial time approximation schemes on $\mathcal{H}$-free graphs for all sets $\mathcal{H}$ for which NP-hardness does not follow from Alekseev's [18] observation. While the results above are general, they are consistent with INDEPENDENT SET being NP-complete on all $\mathcal{H}$-free classes of graphs on which polynomial time algorithms are not already known. In this paper we obtain a quasi-polynomial time algorithm for WEIGHTED INDEPENDENT SET on P_k-free graphs for every k. In particular we prove the following theorem.

Theorem 1. *There exists an algorithm that given a graph G and weight function $w : V(G) \to \mathbb{N}$ outputs the weight of a maximum weight independent set of G. If G is P_k-free then the algorithm runs in $n^{O(k^2 \log^3 n)}$ time.*

Theorem 1 implies that unless NP $\subseteq$ QP, INDEPENDENT SET on P_k-free graphs is not NP-complete for any k. This is the first conclusive evidence against NP-completeness for any $k \geq 7$. The running time of the algorithm of Theorem 1 matches that of Chudnovsky et al. [37], but computes optimal solutions instead of $(1 - \epsilon)$-approximate ones. It is also worth mentioning that our algorithm and analysis is substantially simpler than the quasi-polynomial time algorithm of Lokshtanov et al. [31] for the special case of P_6-free graphs. We have been unsuccessful in generalizing Theorem 1 to a quasi-polynomial time algorithms for H-free graphs where H is a subdivision of a claw. However, the techniques used to prove Theorem 1 can be used to show that such an algorithm would automatically generalize to all classes of $\mathcal{H}$-free graphs for which NP-hardness is not already known. More concretely, for a graph H let O_H be an oracle that takes as input an H-free graph G and outputs the weight of a maximum weight independent set in G. Further, let $\mathcal{CC}(H)$ denote the set of connected components of H. Our second result is the following.

Theorem 2. *There exists an algorithm that given as input a graph H, a graph G, and weight function $w : V(G) \to \mathbb{N}$ as well as access to oracles O_{H_i} for all $H_i \in \mathcal{CC}(H)$, outputs the weight of a maximum weight independent set of G. If G is H-free then the algorithm uses at most $n^{O(|H|^2 |\mathcal{CC}(H)| \log^3(n))}$ operations and oracle calls on induced subgraphs of G.*

Theorem 2 has two immediate consequences. First, coupled with Theorem 1 and the polynomial time algorithm for WEIGHTED INDEPENDENT SET on fork-free graphs, Theorem 2 yields a quasi-polynomial time algorithm for WEIGHTED INDEPENDENT SET on T_k-free graphs, where T_k is the graph with k connected components the first of which is a P_k and each of the remaining $k - 1$ are isomorphic to a fork. Second, Theorem 2 implies that if WEIGHTED INDEPENDENT SET has a quasi-polynomial time algorithm on H-free graphs for every subdivided claw H, then WEIGHTED INDEPENDENT SET also has a quasi-polynomial time algorithm on all $\mathcal{H}$-free classes of graphs, for finite sets $\mathcal{H}$, for which NP-hardness does not follow from

Alekseev's result. Or, stated more poetically, the buck stops at the (subdivided) claw.

a) Methods.: The starting point for our algorithm is the $2^{O(\sqrt{n \log n})}$ time algorithm for P_k-free graphs of Bacsó et al. [34]. The algorithm of Bacsó et al. [34] is simple enough that we can give a quite detailed overview here. It combines two methods - "*degree reduction*" and "*balanced separation*".

The "degree reduction" approach can be summarised as follows. As long as the input graph G contains a vertex v of sufficiently high degree (degree $\geq d$) then *branch on* v. That is, find the best solution avoiding v by a recursive call on $G - v$, and the best solution containing v by adding v to the solution obtained from a recursive call on $G - N[v]$. Output the best of these two solutions. A simple recurrence analysis shows that this reduces the problem to solving $2^{O(\frac{n \log n}{d})}$ instances in which no vertex has degree at least d. Bacsó et al. [34] set $d = \sqrt{n \log n}$ and obtain $2^{O(\sqrt{n \log n})}$ instances with maximum degree $\sqrt{n \log n}$.

The "balanced separation" technique is based on the classic "Gyárfás path" argument [38] for proving that P_k-free graphs are χ-bounded. A simple lemma (Lemma 2 of Bacsó et al. [34]), whose proof spans less than a page, shows that in every P_k free graph G there exists a vertex set X_1 of size at most $k-1$, such that every connected component of $G - N[X_1]$ has at most $n/2$ vertices. Bacsó et al. [34] apply this result to instances output by the degree reduction procedure above. In such instances, $|N[X_1]| \leq O(\sqrt{n \log n})$, assuming k is a constant. Then, after guessing the intersection of the optimal solution with $N[X_1]$ (there are at most $2^{|N[X_1]|} \leq 2^{\sqrt{n \log n}}$ such guesses) the connected components of $G - N[X_1]$ become independent sub-instances of size at most $n/2$, on which the algorithm may be called recursively. Thus, solving a single instance on n vertices reduces to solving $2^{O(\sqrt{n \log n})}$ instances on at most $n/2$ vertices. Analyzing the corresponding recurrence shows that the total running time of the algorithm is upper bounded by $2^{O(\sqrt{n \log n})}$.

If we wish to improve the running time from $2^{O(\sqrt{n \log n})}$ to quasi-polynomial, we may only apply degree reduction with $d = \Omega(\frac{n}{\log^{O(1)} n})$, and we can not afford to guess the intersection of the balanced separator $N[X_1]$ with an optimal solution. At this point we apply a slight generalization of degree reduction, to degree reduction relative to a vertex set S. Here we branch on vertices v that have at least d' neighbors in S (the vertex v itself does not have to be in S). A simple recursion analysis shows that this will reduce a single instance to $n^{|S|/d'}$ instances where every vertex has at most d' neighbors in S. We apply degree reduction on the balanced separator $N[X_1]$ with $d' = |N[X_1]|/c$ for some constant c (possibly depending on k). Thus, the initial degree reduction, followed by the degree reduction on $N[X_1]$, reduces the task of solving a single instance G to that of solving the problem on $2^{\log^{O(1)} n}$ instances in which every vertex has degree at most $n/\log^{O(1)} n$ and furthermore has at most $|N[X_1]|/c$ neighbors in the set $N[X_1]$. Here we are working with induced subgraphs of the original graph G, so when we say $N[X_1]$ we really mean what remains of the set $N[X_1]$ (with the neighborhood taken in the graph G) in the subgraph of G that is currently being considered.

The route above is perhaps the most natural one to try to obtain a quasi-polynomial time algorithm. Indeed, it is also the engine in the quasi-polynomial time algorithm of Lokshtanov et al. [31] for P_6-free graphs. However it is not at all clear how to deal with the instances output by this degree reduction. For P_6-free graphs, Lokshtanov et al. [31] (essentially) show that if the balanced separator $N[X_1]$ is chosen very carefully, then the degree reduction procedure never gets stuck: as long as $N[X_1]$ is non-empty some vertex is a neighbor to a constant fraction of $N[X_1]$. Thus the algorithm will make quasi-polynomially many calls on instances where the balanced separator $N[X_1]$ has been reduced to the empty set, in which case each connected component of the graph is substantially smaller than the original graph. This leads to a recurrence that solves to quasi-polynomial time. We are not able to prove an analogous statement for P_k-free graphs for higher values of k, and so we are faced with the problem of how to deal with the degree-reduced instances described above.

The key insight of our algorithm is the following: *if we re-apply the "Gyárfás path" argument of Bacsó et al. [34] on the degree-reduced instances to obtain a new balanced separator $N[X_2]$, then $N[X_2]$ can not have large intersection with $N[X_1]$.* This is because $N[X_2]$ is the neighbor set of a constant size set (X_2) and no vertex in the degree-reduced instance has many neighbors in $N[X_1]$. We now apply the degree reduction procedure again, this time on $N[X_2]$. If this reduction procedure completely reduces X_1 or X_2 to the empty set, or disconnects the graph into connected components so that the largest one has at most $0.9n$ vertices, then we have won, because the connected components of our instances are substantially smaller than on the original graph. If the procedure gets stuck then we obtain yet another balanced separator X_3, observe that X_3 has small intersection with X_2 and X_1, and do degree-reduction on X_3. And this keeps going, we keep adding new balanced separators into the mix until the degree-reduction procedure sufficiently disconnects the graph (i.e the largest connected component of the instances becomes sufficiently smaller than the original graph. The hard part of the analysis is to prove that the graph does become substantially disconnected by the time at most $O(\log n)$ separators have been added to the instance.

The actual final form of the algorithm is slightly different from what we describe above. Indeed, based on the ideas in the previous paragraph we can get an algorithm with running time $O(2^{n^\epsilon})$ for every $\epsilon > 0$, however to obtain quasi-polynomial time we need to be slightly more careful. The main difference is that we do not do degree reduction on each individual separator $N[X_i]$. Instead we define *level sets*. Level i is the set of all vertices that appear in at least i of the separators $N[X_1], \ldots, N[X_t]$ that we have constructed so far. We will maintain that throughout the course of the algorithm the size of level i drops exponentially with i. Thus there will only be $O(\log n)$ levels, and we can afford to run degree reduction so that for each level, no vertex sees more than a $(\frac{1}{k \log n})^{O(1)}$ fraction of that level. Then, when we add a new separator, because it is the neighbor set of only a constant number of vertices, each level will increase by at most a factor of $1 + (\frac{1}{k \log n})^{O(1)}$ of the size of the previous level. Thus, such a process may continue to depth $(k \log n)^{O(1)}$ while maintaining the invariant that the size of the level i drops exponentially with i.

If recursion depth $\Omega(k \log n)$ is reached without sufficiently disconnecting the graph (i.e the largest connected component C of the graph still has size at least $N/2$, where N is the number of vertices in the original graph) this means that we have found $\Omega(k \log n)$ balanced separators for the graph such that no vertex is contained in more than $O(\log n)$ of them. A simple counting argument then shows that the average distance between pairs of vertices in the component C has to be at least $\frac{k \log n}{\log n} \geq k$, contradicting that G is P_k-free. This means that after recursion depth $O(k \log n)$, the graph has already been disconnected! At this point running the algorithm from scratch on each of the connected components yields at most n instances of size at most $n/2$ which solves to quasi-polynomial time.

Our algorithm for Theorem 2 follows the same template as the algorithm for Theorem 1. The key difference is that instead of growing a sequence of balanced separators we grow a sequence of (neighborhoods of) induced copies in G of connected components of H. Again the sequence has the property that the sets in the sequence do not overlap too much, so if we can grow the sequence to length $\Omega(|H|^{O(1)} \log n)$ then we can find an induced copy of H in G.

II. Preliminaries

All graphs in this paper are assumed to be simple, undirected graphs. We denote the edge set of a graph G by $E(G)$ and the vertex set of a graph by $V(G)$. If $v \in V(G)$, then we use $N[v]$ to denote the closed neighborhood of v, i.e. the set of all neighbors of v together with v itself. We use $N(v)$ to denote the set $N[v] - \{v\}$. If $X \subseteq V(G)$, then $N[X] = \bigcup_{x \in X} N[x]$ and $N(X) = N[X] - X$. We use $\mathcal{CC}(G)$ to denote the set of connected components of G. If G_1, G_2,..., G_m are graphs, then we use $G_1 + G_2 + ... + G_m$ to denote the graph that that has vertex set $V(G_1) \cup V(G_2) \cup ... \cup V(G_m)$, and edge set $E(G_1) \cup E(G_2) \cup ... \cup E(G_m)$.

Given a weight function $w : V(G) \to \mathbb{N}$ the weight of a vertex set S is defined as $w(S) = \sum_{v \in S} w(v)$. An *independent set* in G is a vertex set S such that no pair of vertices in S have an edge between them. We define $\mathsf{mwis}(G)$ to be the weight of the maximum weight independent set in G. The $length$ of a path is the number of vertices in the path and we denote by P_k the path of length k. If $X \subseteq V(G)$ then we will use $G(X)$ to denote the the graph induced by the vertex set X, and if it is clear from the context we will use $G - X$ to denote the graph $G(V(G) - X)$.

Given a positive number c and a graph G, we call a set $S \subset V(G)$ a c-$balanced\ separator$ if no connected component of $G - S$ has over c vertices. A *vertex multi-family* $\mathcal{F}$ is a collection of vertex sets that allows for multiple instances of its vertex sets. If $\mathcal{F} = \{S_1, S_2, \ldots, S_n\}$ and X is a set of vertices, then $\mathcal{F} - X$ is the vertex multi-family $\{S_1 - X, S_2 - X, \ldots, S_n - X\}$. For two vertex multi-families $\mathcal{A}$ and $\mathcal{B}$ their *union* is denoted by $\mathcal{A} \cup \mathcal{B}$ and is defined by the vertex multi-family that contains all elements of $\mathcal{A}$ and of $\mathcal{B}$. The multiplicity of an element X in $\mathcal{A} \cup \mathcal{B}$ is its multiplicity in $\mathcal{A}$ plus its multiplicity in $\mathcal{B}$. We will use $\log(x)$ to denote the function $\max(\lceil \log_2(x) \rceil, 1)$ throughout this paper. The proofs of Lemmas and Observations marked with a ($\star$) have been omitted due to space restrictions and can be found in the full version of the paper [17].

III. Quasi-Polynomial Time Algorithm for P_k-Free Graphs

In this section we prove Theorem 1. We will make use of the following balanced separator lemma from Bacsó et al. [34].

Lemma 1. *[34] There exists an algorithm that given a graph G runs in polynomial time and outputs an induced path P in G such that $N(V(P))$ is a $\frac{|V(G)|}{2}$-balanced separator of G.*

We begin by proving a slight strengthening of Lemma 1.

Lemma 2. $\star$ *There exists an algorithm that takes as input a graph G, and a positive integer i such that $2^i < |V(G)|$, runs in polynomial time and outputs a set X such that $N[X]$ is a $\frac{|V(G)|}{2^i}$-balanced separator in G. Furthermore, if G is P_k-free then $|X| \leq 2^{i+1} \cdot k$.*

To describe the algorithm of Theorem 1 we first need to define the notion of *level sets* relative to a vertex multi-family $\mathcal{F}$.

Definition 1. Given a graph G and a vertex multi-family $\mathcal{F}$ consisting of vertex sets of G, for positive integers i, the *i^{th} level relative to $\mathcal{F}$* is denoted by $L(\mathcal{F}, i)$ and defined as follows

$$L(\mathcal{F}, i) = \{v \in V(G) \ : \ |\{S \in \mathcal{F} \ : \ v \in S\}| \geq i\}$$

In other words $L(\mathcal{F}, i)$ is a vertex set containing all vertices of G that are contained in at least i sets in $\mathcal{F}$. Our algorithm will also make use of a number N, this number will be approximately equal to the number of vertices in the input graph G.

Definition 2. The i^{th} *branch threshold* is denoted by Δ_i and is defined as $\Delta_i = N/2^i$. Given a multi-family $\mathcal{F}$, a vertex $v \in V(G)$ is a *branchable vertex* if there exists an $i \geq 1$ such that $|N[v] \cap L(\mathcal{F}, i)| \geq \Delta_i$.

In the following G is always a graph, w is a weight function $w : V(G) \to \mathbb{N}$, N is an integer, and $\mathcal{F}$ is a multi-family of subsets of $V(G)$. We now describe the main subroutine ALG_1 in the algorithm of Theorem 1. The algorithm takes as input G, w, N and $\mathcal{F}$ and (as we will prove) outputs the weight of a maximum weight independent set in G. The algorithm of Theorem 1 will call ALG_1 with parameters G, $N = |V(G)|$, w, and $\mathcal{F} = \emptyset$. ALG_1 is a recursive branching algorithm with only four rules. First, if G has at most one vertex, then return $V(G)$. Second, if the largest component of G has at most $|N|/2$ vertices then solve the problem recursively on each component and return the sum. Third, if there exists a branchable vertex v, then branch on v (i.e solve the problem with v forced in to the independent set, and v forced out). Finally, if none of the previous rules apply then *add* a new balanced separator $N[X]$ (obtained by Lemma 2) to $\mathcal{F}$. In other words, make a recursive call on the instance $(G, w, N, \mathcal{F} \cup \{N[X]\})$.

ALG_1 is very similar to well known exact exponential time branching algorithms for INDEPENDENT SET [39]. The key differences are that we use the multi-family $\mathcal{F}$ of balanced separators to guide which vertex to branch on, that when no rules apply we add a separator to the family $\mathcal{F}$ (at a glance this appears to make no progress at all, but it increases the size of the level sets, making more vertices branchable), and that we wait with recursing on connected components until the size of the largest component becomes less than $N/2$ (this is primarily to simplify the analysis).

ALG_1

1: **Input:** G, w, N, $\mathcal{F}$.
2: **Output:** $\mathsf{mwis}(G)$.
3: **if** $|V(G)| \leq 1$ **then**
4: **return** $w(V(G))$
5: **else if** $(\max_{C \in \mathcal{CC}(G)} |V(C)|) \leq N/2$ **then**
6: **return** $\sum_{C \in \mathcal{CC}(G)} \text{ALG}_1(C, w, |V(C)|, \emptyset)$
7: **else if** exists branchable vertex v **then**
8: **return** $\max(\text{ALG}_1(G - v, w, N, \mathcal{F} - \{v\}), \text{ALG}_1(G - N[v], w, N, \mathcal{F} - N[v]) + w(v))$
9: **end if**
10: **obtain** X by applying Lemma 2 on G with $i = 2$
11: **return** $\text{ALG}_1(G, w, N, \mathcal{F} \cup \{N[X]\})$

We will distinguish between the three different kinds of recursive calls that ALG_1 can make. If the **else if** condition on line 5 holds, then the algorithm makes the recursive calls on line 6. In this case we say that ALG_1 *recurses on connected components*. If the **else if** condition on line 7 holds, then the algorithm makes the recursive calls on line 8. In this case we say that ALG_1 *branches on a branchable vertex*. Otherwise the algorithm makes the recursive call on line 10. In this case we say that ALG_1 *adds a balanced separator*. We will frequently need to refer to parts of the execution of the algorithm. For disambiguation, we collect the terminology here.

An *instance* is a four-tuple $(G, w, N, \mathcal{F})$. A *run* of the algorithm refers to the entire execution of the algorithm on an instance. A *call* $\text{ALG}_1(G, w, N, \mathcal{F})$ refers to the computation done in the root node of the recursion tree of the run $\text{ALG}_1(G, w, N, \mathcal{F})$. We remark that parameters G, w, N, and $\mathcal{F}$ never change during the call $\text{ALG}_1(G, w, N, \mathcal{F})$. When a run or a call $\text{ALG}_1(G, w, N, \mathcal{F})$ recursively calls ALG_1 on the instance $(G', w, N', \mathcal{F}')$ we say the run or the call *executes* a run or a call on $(G', w, N', \mathcal{F}')$. This will sometimes be referred to as *makes a recursive call* $\text{ALG}_1(G', w, N', \mathcal{F}')$. A run of $\text{ALG}_1(G, w, N, \mathcal{F})$ is called a *k-$fair\ run$* if G is a P_k-free graph, $N = |V(G)|$, $\mathcal{F} = \emptyset$, and w is a weight function. A call $\text{ALG}_1(G, w, N, \mathcal{F})$ is called a *k-$fair\ call$* if it is executed during the course of a k-fair run. An instance $(G, w, N, \mathcal{F})$ is called a *k-fair* instance if $\text{ALG}_1(G, w, N, \mathcal{F})$ is a k-fair call. Note that $N \geq |V(G)|$ for every fair instance.

Lemma 3. $\star$ *$ALG_1(G, w, N, \mathcal{F})$ terminates on every input.*

Lemma 4. $\star$ *A run $ALG_1(G, w, N, \mathcal{F})$ always returns the weight of a maximum weight independent set of G under the weight function w.*

We have now proved that ALG_1 always terminates and that it always outputs the correct answer. The remainder of the section is devoted to the running time analysis. We will now prove some lemmas to help us bound the run time of ALG_1 on k-fair runs. First, in Observation 1 we will prove that $\mathcal{F}$ remains a multi-family of balanced

separators of G throughout the execution of the algorithm. In Observation 2 we will show that no vertex appears in many (more than $\log N$) sets in $\mathcal{F}$. This will ensure that $\mathcal{F}$ can never grow too large, because, as we will show in Lemma 5, a connected P_k-free graph can not contain a large fractional packing of balanced separators.

Observation 1. *$\star$ Let $(G, w, N, \mathcal{F})$ be a k-fair instance. Then every set $S \in \mathcal{F}$ is a $\frac{N}{4}$-balanced separator of G.*

Observation 2. *For every k-fair instance $(G, w, N, \mathcal{F})$, we have that $L(\mathcal{F}, \log(N) + 1) = \emptyset$.*

Proof. Consider a k-fair call $\text{ALG}_1(G, w, N, \mathcal{F})$. We will prove the statement by induction on the depth the call $\text{ALG}_1(G, w, N, \mathcal{F})$ in the recursion tree of a run $\text{ALG}_1(G^*, w, |V(G^*)|, \emptyset)$ which executes $\text{ALG}_1(G, w, N, \mathcal{F})$.

If $\mathcal{F} = \emptyset$ then the result is trivially true. Suppose now that $\mathcal{F} \neq \emptyset$, it follows ALG_1 executes $\text{ALG}_1(G, w, N, \mathcal{F})$ during a k-fair call $\text{ALG}_1(G', w, N, \mathcal{F}')$ by branching on a branchable vertex or by adding a balanced separator $N[X]$. In the first case $\mathcal{F} = \mathcal{F}' - S$ for some vertex set S. By the induction hypothesis $L(\mathcal{F}', \log(N) + 1) = \emptyset$ and hence $L(\mathcal{F}, \log(N) + 1) = \emptyset$. In the second case, $\text{ALG}_1(G, w, N, \mathcal{F}')$ does not branch on a branchable vertex, so we have that $L(\mathcal{F}', \log(N)) = \emptyset$ since every vertex in $L(\mathcal{F}', \log(N))$ is branchable. It follows that $L(\mathcal{F}, \log(N) + 1) = L(\mathcal{F}' \cup \{N[X]\}, \log(N) + 1) = \emptyset$. □

Lemma 5. *For every k-fair instance $(G, w, N, \mathcal{F})$ it holds that $|\mathcal{F}| \leq 10k \cdot \log(N)$.*

Proof. Consider a k-fair instance $(G, w, N, \mathcal{F})$. We will prove the result by induction on the depth of the call $\text{ALG}_1(G, w, N, \mathcal{F})$ in the recursion tree of a run $\text{ALG}_1(G^*, w, |V(G^*)|, \emptyset)$ which executes $\text{ALG}_1(G, w, N, \mathcal{F})$.

In the base case $\mathcal{F} = \emptyset$, and the claim of the lemma holds trivially, so assume $\mathcal{F} \neq \emptyset$. Thus the call $\text{ALG}_1(G, w, N, \mathcal{F})$ is executed by a k-fair call $\text{ALG}_1(G', w, N', \mathcal{F}')$. By the induction hypothesis $|\mathcal{F}'| \leq 10k \cdot \log(N)$ ($N = N'$ since $\mathcal{F} \neq \emptyset$). Thus, unless $\text{ALG}_1(G', w, N', \mathcal{F}')$ recurses by adding a balanced separator we have that $|\mathcal{F}| \leq 10k \cdot \log(N)$ as well. So assume that $\text{ALG}_1(G', w, N', \mathcal{F}')$ adds a balanced separator $N[X]$ and that therefore $G' = G$, $N' = N$ and $\mathcal{F} = \mathcal{F}' \cup \{N[X]\}$. We prove that $|\mathcal{F}'| < 10k \cdot \log(N)$, then the result follows since $|\mathcal{F}| = |\mathcal{F}'| + 1$.

Suppose for contradiction that $|\mathcal{F}'| > 10k \cdot \log(N)$, we will now produce an induced path of length k in G, contradicting that G is P_k-free. The call $\text{ALG}_1(G', w, N', \mathcal{F}') = \text{ALG}_1(G, w, N, \mathcal{F}')$ added a balanced separator, and so the size of the largest connected component, C, in G is greater than $\frac{N}{2}$. This, together with Observation 1 then gives that every set $S \in \mathcal{F}$ is a $\frac{|V(C)|}{2}$-balanced separator for C. Consider the following random process. Uniformly at random, select vertices x and y in C. For all $S \in \mathcal{F}$, let X_S denote the random variable that is 1 if x and y are not in the same connected component of $C - S$ and 0 otherwise. Since S is a $\frac{|V(C)|}{2}$-balanced separator for C, the probability that x and y are in the same connected component of $C - S$ is at most $\frac{1}{2}$. Thus $X_S = 1$ with probability at least $\frac{1}{2}$. We denote by $\mathcal{F}_{x,y}$ all sets $S \in \mathcal{F}$ such that x and y are not in the same component of $C - S$, again including multiplicity. By linearity of expectation we have that

$$E[|\mathcal{F}_{x,y}|] = \sum_{S \in \mathcal{F}} E[X_S] \geq |\mathcal{F}|/2 > 5k \cdot \log(N).$$

It follows there exists vertices a and b in C such that $|\mathcal{F}_{a,b}| > 5k \cdot \log(N)$. Let P be a shortest path connecting a and b in C. Since G is P_k-free, P has at most $k - 1$ vertices. By Observation 2, each of these vertices is contained in at most $\log(N)$ sets in $\mathcal{F}_{a,b}$. But then there exists a set $S \in \mathcal{F}_{a,b}$ disjoint from $V(P)$ contradicting that a and b are not in the same component of $C - S$. □

The following observation shows that the level sets do not grow a lot in each successive recursive call, and that they therefore never get very large. Note in particular that the size of level set i drops exponentially with i.

Observation 3. *For every k-fair call $\text{ALG}_1(G, w, N, \mathcal{F})$ that adds a balanced separator $N[X]$ and every i,*

$$|L(\mathcal{F} \cup \{N[X]\}, i)| \leq \Delta_{i-1} \cdot 8k + |L(\mathcal{F}, i)|.$$

Furthermore, for every k-fair instance $(G', w, N', \mathcal{F}')$,

$$|L(\mathcal{F}', i)| \leq \Delta_{i-1} \cdot 8k \cdot |\mathcal{F}'|.$$

Proof. Consider a k-fair call $\text{ALG}_1(G, w, N, \mathcal{F})$ that adds a balanced separator $N[X]$. Let X_j denote the set of vertices in $L(\mathcal{F}, j) \cap \{N[X]\}$, then we can see that $|L(\mathcal{F} \cup \{N[X]\}, j)| \leq L(\mathcal{F}, j) + |X_{j-1}|$. Since the call $\text{ALG}_1(G, w, N, \mathcal{F})$ adds a balanced separator, $N[X]$, there are no branchable vertices. So, we have that for all $v \in G$, $|N[v] \cap L(\mathcal{F}, j)| \leq \Delta_j$. Furthermore, by Lemma 2, since $N[X]$ is generated as an $\frac{N}{4}$-balanced separator and therefore a $\frac{|G|}{4}$-balanced separator for G, X consists of most $8k$ vertices, hence $|X_{j-1}| \leq \Delta_{j-1} \cdot 8k$ and the result $|L(\mathcal{F} \cup \{N[X]\}, i)| \leq \Delta_{i-1} \cdot 8k + |L(\mathcal{F}, i)|$ follows.

The second statement follows by combining induction, the first part of this observation, and the fact that if the call $\text{ALG}_1(G, w, N, \mathcal{F})$ executes $\text{ALG}_1(G', w, N', \mathcal{F}')$, then $|\mathcal{F}| < |\mathcal{F}'|$ if and only if the call $\text{ALG}_1(G, w, N, \mathcal{F})$ adds a balanced separator.

For k-fair instances $(G, w, N, \mathcal{F})$ we define a measure:

$$\mu_k(G, w, N, \mathcal{F}) = 400k^2 \cdot \log^2(N) \cdot (N + |V(G)|) + \sum_i \left(|L(\mathcal{F}, i)| \cdot \frac{N}{\Delta_{i-1}} \right) + 16k \cdot N \cdot \log(N) \cdot (10k \cdot \log(N) - |\mathcal{F}|)$$

If $(G, w, N, \mathcal{F})$ is not a k-fair instance, then $\mu_k(G, w, N, \mathcal{F})$ is undefined. Note that $\mu_k(G, w, N, \mathcal{F})$ must always be an integer, and that it is independent of the weight function w. We will say that two instances $(G, w, N, \mathcal{F})$ and $(G', w', N', \mathcal{F}')$ are *essentially different* if $G' \neq G$, $N' \neq N$ or $\mathcal{F}' \neq \mathcal{F}$.

Lemma 6. ⋆ *For every positive integer μ, the number of essentially different k-fair instances $(G, w, N, \mathcal{F})$ such that $\mu_k(G, w, N, \mathcal{F}) = \mu$ is finite. In addition, for every k-fair instance it holds that $\mu_k(G, w, N, \mathcal{F}) \geq 0$.*

Lemma 7. ⋆ *For every k-fair instance $(G, w, N, \mathcal{F})$ it holds that $\mu_k(G, w, N, \mathcal{F}) \leq 1050k^2 \cdot N \cdot \log^2(N)$*

We define $T_k(G, w, N, \mathcal{F})$ to be the running time of a k-fair run of ALG_1 starting with the inputs $(G, w, N, \mathcal{F})$. We also define

$$T_k(\mu) = \max_{\substack{G,N,\mathcal{F} \text{ s.t.} \\ \mu_k(G,w,N,\mathcal{F}) \leq \mu}} \text{T}_k(G, w, N, \mathcal{F}).$$

When we analyze run time we assume that arithmetic (addition, subtraction, comparisons) on weights of vertices and vertex sets is constant time. Thus, both the running time of ALG_1 and the measure of an instance $(G, w, N, \mathcal{F})$ are independent of the weight function w. Thus, by Lemma 6, $T_k(\mu)$ is well defined.

Lemma 8. *If $\mu > 1$ then $T_k(\mu)$ satisfies the following recurrence:*

$$T_k(\mu) \leq \mu^{O(1)} + max \begin{cases} \mu T_k(.95\mu) \\ (T_k(\mu - 1) + \\ T_k(\mu[1 - 1/(2100k^2 \cdot \log^2(\mu))])) \\ T_k(\mu[1 - 1/(200k \cdot \log(\mu))]) \end{cases}$$

Proof. Let $(G, w, N, \mathcal{F})$ be a k-fair instance such that $\mu_k(G, w, N, \emptyset) = \mu > 1$ and $T_k(\mu)$ is the run time of $\text{ALG}_1(G, w, N, \mathcal{F})$. If the call $\text{ALG}_1(G, w, N, \mathcal{F})$ recurses on connected components, then it makes at most $|V(G)|$ recursive calls on instances of the form $(G', w, N', \emptyset)$, where $|V(G')| \leq |V(G)|$ and $N' \leq \frac{N}{2}$. It follows that for each of these recursive calls we have

$$\begin{aligned} &\mu_k(G', w, N', \emptyset) = 400k^2 \cdot \log^2(N')(N' + |V(G')|) \\ &\quad + 160k^2 \cdot N' \cdot \log^2(N') \\ &\leq 400k^2 \cdot \log^2(N)(\frac{N}{2} + |V(G)|) + 80k^2 \cdot N \cdot \log^2(N) \\ &\leq 400k^2 \cdot \log^2(N)(N + |V(G)|) - 100k^2 \cdot N \cdot \log^2(N) \\ &\leq \mu - 100k^2 \cdot N \cdot \log^2(N) \\ &\leq .95\mu \qquad \text{(by Lemma 7)} \end{aligned}$$

Also, the algorithm does $|V(G)|^{O(1)} = \mu^{O(1)}$ work in a call where it recurses on connected components. Therefore, if the instance $\text{ALG}_1(G, w, N, \mathcal{F})$ recurses on connected components, we must have that $T_k(\mu) \leq \mu^{O(1)} + |V(G)| \cdot T_k(.95\mu) \leq \mu^{O(1)} + \mu \cdot T_k(.95\mu)$.

If the call $\text{ALG}_1(G, w, N, \mathcal{F})$ branches on a branchable vertex, v, then it makes two recursive calls, one execution $\text{ALG}_1(G - \{v\}, w, N, \mathcal{F} - \{v\})$, where the instance $(G - \{v\}, w, N, \mathcal{F} - \{v\})$ has measure $\mu_k(G - \{v\}, w, N, \mathcal{F} - \{v\}) \leq \mu - 1$, and the other execution is $\text{ALG}_1(G - N[v], w, N, \mathcal{F} - N[v])$. Note that for a branchable vertex, v, we have that

$$\sum_i (|L(\mathcal{F} - N[v], i)| \cdot \frac{N}{\Delta_{i-1}} \leq \sum_i (|L(\mathcal{F}, i)| \cdot \frac{N}{\Delta_{i-1}}) - \frac{N}{2},$$

since for at least one level i we have that $|N[v] \cap L(\mathcal{F}, i)| \geq \Delta_i$ and $\frac{\Delta_i}{\Delta_{i-1}}$ = 1/2. It follows that

$$\begin{aligned} &\mu_k(G - N[v], w, N, \mathcal{F} - N[v]) = 400k^2 \cdot \log^2(N) \cdot (N \\ &\quad + |V(G) - N[v]|) + \sum_i \left(|L(\mathcal{F} - N[v], i)| \cdot \frac{N}{\Delta_{i-1}} \right) \\ &\quad + 16k \cdot N \cdot \log(N) \cdot (10k \cdot \log(N) - |\mathcal{F}|) \\ &\leq 400k^2 \log^2(N) \cdot (N + |V(G)|) \\ &\quad + \sum_i \left(|L(\mathcal{F}, i)| \cdot \frac{N}{\Delta_{i-1}} \right) \\ &\quad + 16k \cdot N \cdot \log(N) \cdot (10k \cdot \log(N) - |\mathcal{F}|) - \frac{N}{2} \\ &\leq \mu - \frac{N}{2} \\ &\leq \mu \left(1 - \frac{1}{2100k^2 \cdot \log^2(N)} \right) \qquad \text{(by Lemma 7)} \\ &\leq \mu \left(1 - \frac{1}{2100k^2 \cdot \log^2(\mu)} \right) \end{aligned}$$

The algorithm does $|V(G)|^{O(1)} = \mu^{O(1)}$ work in a call where it branches on a branchable vertex. Therefore, if the call $\text{ALG}_1(G, w, N, \mathcal{F})$ branches on a branchable vertex, then we have that $T_k(\mu) \leq \mu^{O(1)} + T_k(\mu - 1) + T_k(\mu[1 - \frac{1}{2100k^2 \cdot \log^2(\mu)}])$.

Finally, if the call $\text{ALG}_1(G, w, N, \mathcal{F})$ adds a balanced separator, $N[X]$, then it makes a single recursive

call $\text{ALG}_1(G, w, N, \mathcal{F} \cup N[X])$. By Observation 3 and Lemma 7 we obtain the following.

$$\mu_k(G, w, N, \mathcal{F} \cup \{N[X]\}) < \mu + 8k \cdot N \cdot \log(n) - 16k \cdot N \cdot \log(N) < \mu \left([1 - \frac{1}{200k \cdot \log(\mu)}] \right)$$

The algorithm does $|V(G)|^{O(1)} = \mu^{O(1)}$ work in a call where adds a balanced separator. Therefore, if the call $\text{ALG}_1(G, w, N, \mathcal{F})$ adds a balanced separator, then $T_k(\mu) \leq \mu^{O(1)} + T_k(\mu[1 - \frac{1}{200k \cdot \log(\mu)}])$.

The result now follows from the observation that $\text{ALG}_1(G, w, N, \mathcal{F})$ always recurses on connected components, branches on a branchable vertex, adds a balanced separator, or returns without making further recursive calls. □

Since $T_k(\mu)$ is a non negative, non decreasing function, by adding the three possibilities in the max of Lemma 8 we immediately obtain the following simplified recurrence.

Corollary 1. $T_k(\mu) \leq \mu^{O(1)} + \mu T_k(.95\mu) + T_k(\mu - 1) + T_k(\mu[1 - \frac{1}{2100k^2 \cdot \log^2(\mu)}]) + T_k(\mu[1 - \frac{1}{200k \cdot \log(\mu)}]) < T_k(\mu - 1) + \mu^{O(1)} + 3\mu \cdot T_k(\mu[1 - \frac{1}{2100k^2 \cdot \log^2(\mu)}])$

Lemma 9. $T_k(\mu) = \mu^{O(k^2 \cdot \log^3(\mu))}$

Proof. For the sake of being able to apply induction to prove the bound, it will be beneficial for us to define the function $T'_k(\mu) = T_k(\lfloor \mu \rfloor)$. We will prove that $T'_k(\mu) = \mu^{O(k^2 \cdot \log^3(\mu))}$, then it will follow that $T_k(\mu) = \mu^{O(k^2 \cdot \log^3(\mu))}$ since $T_k(\mu) \leq T'_k(\mu+1) = \mu^{O(k^2 \cdot \log^3(\mu))}$. Since T'_k rounds down its input and Lemma 6 shows the measure μ is always non-negative for any fair instance, it suffices to prove the desired bound on T'_k by induction on the natural numbers. We may also assume μ is an integer. The base case is established by observing that the runtime of algorithm on all such fair instances where its measure μ is at most 1 is bounded by a constant. Now, for the inductive hypothesis, assume that there exists a number c such that for all integers, a, less than $\mu > 1$, $T'_k(a) \leq a^{ck^2 \cdot \log^3(a)}$. By Corollary 1 we have the inequality $T'_k(\mu) \leq T'_k(\mu - 1) + \mu^{O(1)} + 3\mu T'_k(\mu[1 - \frac{1}{2100k^2 \cdot \log^2(\mu)}])$ and repeatedly applying the inequality to the first term on the right hand side, gives $T'_k(\mu) \leq \mu^{O(1)} + 3\mu^2 \cdot T'_k(\mu[1 - \frac{1}{2100k^2 \cdot \log^2(\mu)}])$. By the inductive hypothesis then, we have the following:

$$\begin{aligned}
T'_k(\mu) &\leq \mu^{O(1)} + 3\mu^2 \lfloor \mu[1 - \frac{1}{2100k^2 \cdot \log^2(\mu)}] \rfloor^{ck^2 \cdot \log^3(\mu)} \\
&\leq \mu^{O(1)} \\
&\quad + 3\mu^2 \cdot \mu^{ck^2 \cdot \log^3(\mu)} [1 - \frac{1}{2100k^2 \cdot \log^2(\mu)}]^{ck^2 \cdot \log^3(\mu)} \\
&\leq \mu^{O(1)} + 3\mu^2 \cdot \mu^{ck^2 \cdot \log^3(\mu)} \cdot e^{-\frac{ck^2 \cdot \log^3(\mu)}{2100k^2 \cdot \log^2(\mu)}} \\
&\quad (\text{ since } 1 - x \leq e^{-x} \text{ }) \\
&\leq \mu^{O(1)} + 3\mu^2 \cdot \mu^{ck^2 \cdot \log^3(\mu)} \cdot e^{-\frac{c \log(\mu)}{2100}} \\
&\leq \mu^{ck^2 \cdot \log^3(\mu)} \qquad (\text{ for } c \text{ chosen large enough })
\end{aligned}$$

□

We are now ready to prove Theorem 1.

Proof of Theorem 1. The algorithm returns the answer of $\text{ALG}_1(G, |V(G)|, w, \emptyset)$. By Lemma 3 ALG_1 terminates, by Lemma 4 ALG_1 returns return the weight of a maximum weighted independent set. For the running time, observe that $(G, w, N, \emptyset)$ is a k-fair instance and let $\mu = \mu_k(G, w, N, \emptyset)$. By Lemma 7 we have that $\mu < 1050k^2 \cdot N \cdot \log^2(N) = n^{O(1)}$. Hence, by Lemma 9 it follows that $T(G, w, N, \emptyset) \leq T(\mu) = \mu^{O(k^2 \cdot \log^3(\mu))} = n^{O(k^2 \cdot \log^3(n))}$. □

IV. Disconnected Forbidden Induced Subgraphs

Let H be a graph. We denote by O_H an oracle that takes an H-free graph G as input and outputs the weight of a maximum weight independent set in G. In this section we present a quasi-polynomial time algorithm for Maximum Weight Independent Set in H-free graphs, assuming we have access to the oracles O_C for all $C \in \mathcal{CC}(H)$. Specifically we will prove Theorem 2.

In the following, $H = H_0 + H_1 + ... + H_{c-1}$ is a graph, G is a graph, w is a weight function on the vertices of G, N is a positive integer, and $\mathcal{F}$ is a vertex multi-family of subsets of $V(G)$. We now present the algorithm ALG_2 of Theorem 2. The algorithm is very similar to the algorithm ALG_1 for P_k free graphs, the main difference is that instead of packing balanced separators in the family $\mathcal{F}$, the algorithm "packs" (neighborhoods of) copies of induced H_i's.

ALG_2

1: **input:** $H, G, w, N, \mathcal{F}$.
2: **output:** $\mathsf{mwis}(G)$.
3: $i = |\mathcal{F}| \mod c$
4: **if** exists branchable vertex v **then**
5: **return** $\max(\text{ALG}_2(H, G - v, w, N, \mathcal{F} - \{v\}), \text{ALG}_2(H, G - N[v], w, N, \mathcal{F} - N[v]) + w(v))$
6: **else if** exists induced H_i **then**
7: **obtain** $X \leftarrow$ induced H_i in G

8: **return** $\text{ALG}_2(H, G, w, N, \mathcal{F} \cup \{N[X]\})$
9: **end if**
10: **return** $O_{H_i}(G)$

The proof of correctness and running time analysis for ALG_2 closely follows that of ALG_1. The main difference is in the proof of why the family $\mathcal{F}$ can not grow beyond size $\log N$ (Lemmata 12 and 13). The other parts are just minor modifications of corresponding results from Section III.

We will distinguish between the two different kinds of recursive calls that ALG_2 can make. If the **if** condition of line 4 holds, then the algorithm makes the recursive calls on line 5. In this case we say that ALG_2 *branches on a branchable vertex*. If the **else if** condition of line 6 holds, then the algorithm makes the recursive call in line 8. In this case we say that ALG_2 *adds a neighborhood*. We define instances, runs, calls, execution and making a recursive call similarly as for ALG_1. Just as for ALG_1, a run of $\text{ALG}_2(H, G, w, N, \mathcal{F})$ is called a $fair\ run$ if G is an H-free graph, $N = |V(G)|$, $\mathcal{F} = \emptyset$, and w is a weight function. A call $\text{ALG}_2(H, G, w, N, \mathcal{F})$ is called a $fair\ call$ if it is executed during the course of a fair run. An instance $(H, G, w, N, \mathcal{F})$ is a *fair instance* if $\text{ALG}_2(H, G, w, N, \mathcal{F})$ is a fair call.

Lemma 10. $\star$ *$ALG_2(H, G, w, N, \mathcal{F})$ terminates on every input.*

Lemma 11. $\star$ *A run $ALG_2(H, G, w, N, \mathcal{F})$ returns the weight of a maximum weight independent set of G.*

Observation 4. $\star$ *For every fair instance $(H, G, w, N, \mathcal{F})$, we have that $L(\mathcal{F}, \log(N) + 1) = \emptyset$.*

Lemma 12. *Let G be a graph, N an integer greater than 1, and let $H = H_0 + H_1 + ... + H_{c-1}$ be a graph. If there exists a sequence of subsets of $V(G)$, $\{X_m\} = X_0, X_1, ..., X_{c\cdot|H|\cdot\log(N)-1}$ such that for all i, $X_i \subset V(G)$, the subgraph induced by X_i is isomorphic to $H_{i \pmod c}$, and for all $v \in X_i$ we have that $\{v\} \cap N[X_j] \neq \emptyset$ for at most $\log(N)$ X_j's where $j < i$, then there exists a subset $I \subseteq \{0, 1, 2, \ldots, c\cdot|H|\cdot\log(N)-1\}$ such that $X_I = \bigcup_{i\in I} X_i$ forms an induced H in G.*

Proof. Let G and H be graphs, N an integer greater than 1, and $X_0, X_1, ..., X_{c\cdot|H|\cdot\log(N)-1}$ a sequence of sets of vertices with the properties given in the statement of the lemma. Given an X_j, set $i = j - (j \pmod c)$. We will refer to the segment X_i, X_{i+1},..., X_{i+c-1} as X_j's block.

The proof is by induction on c. If $c = 1$ then the statement is trivially true. Assume now that $c > 1$ and that the statement is true for all smaller values. There are at most $|H_{c-1}| \cdot \log(N)$ X_j's such that some vertex of $X_{c\cdot|H|\cdot\log(N)-1}$ belongs to X_j, $j \neq c \cdot |H| \cdot \log(N) - 1$. Remove from the sequence each such X_j along with all other vertex sets in X_j's block, as well as all X_t's such that $c - 1 \equiv t \pmod c$. After these deletions, re-name the sets X_j in the updated sequence so that the index j of each set X_j is equal to the position of X_j in the sequence (starting with X_0).

Let $H' = H - H_{c-1}$. There are at least $\log(N) \cdot (c \cdot |H| - c\cdot|H_{c-1}| - |H| + |H_{c-1}|) - 1 = \log(N)\cdot(c-1)\cdot|H'| - 1$ remaining vertex sets in the updated sequence, and this new sequence along with H' and G satisfies the condition of the inductive hypothesis. It follows that there exists a set X'_I such that $G[X'_I] = H'$ and X'_I is the union of sets in the (updated) sequence. Since $X_{c\cdot|H|\cdot\log(N)-1}$ does not belong to the neighborhood of any of the vertex sets in the new sequence, $X_{c\cdot|H|\cdot\log(N)-1}$ is disjoint from $N[X'_I]$, and hence $X_I = X'_I \cup X_{c\cdot|H|\cdot\log(N)-1}$ induces H in G, completing the proof. □

Lemma 13. *For every fair instance $(H, G, w, N, \mathcal{F})$ with $H = H_0 + H_1 + ... + H_{c-1}$, it holds that $|\mathcal{F}| < c \cdot |H| \cdot \log(N)$*

Proof. Let the fair instance $(H, G, w, N, \mathcal{F})$ be as in the statement of the lemma, furthermore let G' be the graph used in the initial input of ALG_2 of the fair run that produces the instance $(H, G, w, N, \mathcal{F})$. Assume to the contrary, that $|\mathcal{F}| \geq c \cdot |H| \cdot \log(N)$. In the fair run that executes the call $\text{ALG}_2(H, G, w, N, \mathcal{F})$, consider the sequence of recursive calls (ordered by when the call occurs) that lead to the call $\text{ALG}_2(H, G, w, N, \mathcal{F})$. In particular, consider the subsequence: $\text{ALG}_2^0(H, G^0, w, N, \mathcal{F}^0), \text{ALG}_2^1(H, G^1, w, N, \mathcal{F}^1), \ldots,$ $\text{ALG}_2^{c\cdot|H|\cdot\log(N)-1}(H, G^{c\cdot|H|\cdot\log(N)-1}, w, N,$ $\mathcal{F}^{c\cdot|H|\cdot\log(N)-1})$

such that the call $\text{ALG}_2^i(H, G^i, w, N, \mathcal{F}^i)$ is the $(i+1)^{th}$ call to add a neighborhood $N[X_i]$. By Observation 4, we can see that for all X_i, and for all vertices $v \in X_i$, $\{v\} \cap N[X_j] \neq \emptyset$ for at most $\log(N)$ X_j's with $j < i$. The result follows now by observing that G', H, N, and the sequence $X_0, X_1, ..., X_{c\cdot|H|\cdot\log(N)-1}$ satisfy the hypothesis of Lemma 12, contradicting that G' is H-free. □

Observation 5. $\star$ *For every fair call $ALG_2(H, G, w, N, \mathcal{F})$ that recurses by adding a neighborhood $N[X]$ and for every i,*

$$|L(\mathcal{F} \cup N[X], i)| \leq \Delta_{i-1} \cdot |H| + |L(\mathcal{F}, i)|$$

Furthermore, for every fair instance $(H, G', w, N', \mathcal{F}')$,

$$|L(\mathcal{F}', i)| \leq \Delta_{i-1} \cdot |H| \cdot |\mathcal{F}'|$$

For fair instances $(H, G, w, N, \mathcal{F})$ we define the measure

$$\mu_H(H, G, w, N, \mathcal{F}) = |V(G)| + \sum_i \left(|L(\mathcal{F}, i)| \cdot \frac{N}{\Delta_{i-1}} \right)$$
$$+ 2|H| \cdot N \cdot \log(N) \cdot (|H| \cdot |\mathcal{CC}(H)| \cdot \log(N) - |\mathcal{F}|)$$

If $(H, G, w, N, \mathcal{F})$ is not a fair instance, then $\mu_H(H, G, w, N, \mathcal{F})$ is undefined. Note that $\mu_H(H, G, w, N, \mathcal{F})$ must always be an integer and that it is independent of the weight function w. We will say that two instances $(H, G, w, N, \mathcal{F})$ and $(H, G', w', N', \mathcal{F}')$ are *essentially different* if $G' \neq G$, $N' \neq N$ or $\mathcal{F}' \neq \mathcal{F}$.

If $N = 1$ then a fair run $\text{ALG}_2(H, G, w, N, \mathcal{F})$ clearly terminates after a constant number of steps (since in a fair run, $|V(G)| \leq N$) regardless of the other inputs, so from now on we will assume $N > 1$.

Lemma 14. ⋆ *For every positive integer μ, the number of essentially different fair instances $(H, G, w, N, \mathcal{F})$ such that $\mu_H(H, G, w, N, \mathcal{F}) = \mu$ is finite. In addition, for every fair instance $\mu(H, G, w, N, \mathcal{F}) \geq 0$.*

Lemma 15. ⋆ $\mu_H(H, G, w, N, \mathcal{F}) \leq 4|H|^2 \cdot |\mathcal{CC}(H)| \cdot N \cdot \log^2(N)$ *for every fair instance* $(H, G, w, N, \mathcal{F})$.

We define $T_H(H, G, w, N, \mathcal{F})$ to be the running time (including the *number* of oracle calls) of ALG_2 starting with the inputs $(H, G, w, N, \mathcal{F})$. We also define

$$T_H(\mu) = \max_{\substack{G,N,\mathcal{F} \text{ s.t.} \\ \mu_H(H,G,w,N,\mathcal{F}) \leq \mu}} \text{T}_H(H, G, w, N, \mathcal{F})$$

Just as for ALG_1, when we analyze run time we assume that arithmetic on weights takes constant time. Thus, both the running time of ALG_2 and the measure of an instance $(H, G, w, N, \mathcal{F})$ are independent of the weight function w, and so by Lemma 14, $T_H(\mu)$ is well defined.

Lemma 16. *If $\mu > 1$ then $T_H(\mu)$ satisfies the following recurrence:*

$$T_H(\mu) \leq \mu^{O(1)} + max \begin{cases} (T_H(\mu - 1) + \\ T_H(\mu[1 - \frac{1}{8|H|^2 \cdot |\mathcal{CC}(H)| \cdot \log^2(\mu)}])) \\ T_H(\mu[1 - \frac{1}{4|H| \cdot \log(\mu)}]) \end{cases}$$

Proof. Let $(H, G, w, N, \mathcal{F})$ be a fair instance such that $\mu_H(H, G, w, N, \mathcal{F})) = \mu > 1$ and $T_H(\mu)$ is the run time of $\text{ALG}_2(H, G, w, N, \mathcal{F})$. If the call $\text{ALG}_2(H, G, w, N, \mathcal{F})$ branches on a branchable vertex, v, then it makes two recursive calls, one execution on $(H, G - \{v\}, w, N, \mathcal{F} - \{v\})$, which has measure at most $\mu - 1$. The other execution is on the instance $(H, G - N[v], w, N, \mathcal{F} - N[v])$. Note that for a branchable vertex, v, we have that:
$\sum_i \left(|L(\mathcal{F} - N[v], i)| \cdot \frac{N}{\Delta_{i-1}} \right)$
$\leq \sum_i \left(|L(\mathcal{F}, i)| \cdot \frac{N}{\Delta_{i-1}} \right) - \frac{N}{2}$
since for at least one level i we have that $|N[v] \cap L(\mathcal{F}, i)| \geq \Delta_i$ and $\frac{\Delta_i}{\Delta_{i-1}} = 1/2$.
Hence,

$$\begin{aligned}
&\mu_H(H, G - N[v], w, N, \mathcal{F} - N[v]) = \\
&\quad |V(G) - N[v]| + \Sigma_i(|L(\mathcal{F} - N[v], i)| \cdot \frac{N}{\Delta_{i-1}}) \\
&\quad + 2|H| \cdot N \cdot \log(N) \cdot (|H| \cdot |\mathcal{CC}(H)| \cdot \log(N) \\
&\quad - |\mathcal{F} - N[v]|) \\
&\leq |V(G)| + \sum_i \left(|L(\mathcal{F}, i)| \cdot \frac{N}{\Delta_{i-1}} \right) \\
&\quad + 2|H| \cdot N \cdot \log(N) \cdot (|H| \cdot |\mathcal{CC}(H)| \cdot \log(N) - |\mathcal{F}|) \\
&\quad - \frac{N}{2} \\
&= \mu - \frac{N}{2} \\
&\leq \mu \left(1 - \frac{1}{8|H|^2 \cdot |\mathcal{CC}(H)| \cdot \log^2(N)} \right) (by\ Lemma\ 15) \\
&\leq \mu \left(1 - \frac{1}{8|H|^2 \cdot |\mathcal{CC}(H)| \cdot \log^2(\mu)} \right)
\end{aligned}$$

Also, the algorithm does $|V(G)|^{O(1)} = \mu^{O(1)}$ work in a call where it branches on a branchable vertex. Thus, if the call $\text{ALG}_2(H, G, w, N, \mathcal{F})$ branches on a branchable vertex, then we have that:
$T_H(\mu) \leq T_H(\mu - 1) + T_H\left(\mu[1 - \frac{1}{8|H|^2 \cdot |\mathcal{CC}(H)| \cdot \log^2(\mu)}]\right)$

If $\text{ALG}_2(H, G, w, N, \mathcal{F})$ adds a neighborhood, $N[X]$, it makes a single call $\text{ALG}_2(H, G, w, N, \mathcal{F} \cup \{N[X]\})$. By Observation 5 and Lemma 15 we get the following:
$\mu(H, G, w, N, \mathcal{F} \cup \{N[X]\}) < \mu + |H| \cdot N \cdot \log(n)$
$-2|H| \cdot N \cdot \log(N) < \mu\left([1 - \frac{1}{4|H| \cdot |\mathcal{CC}(H)| \cdot \log(\mu)}]\right)$

Also, the algorithm does $|V(G)|^{O(1)} = \mu^{O(1)}$ work in a call where it adds a neighborhood. Thus, if the call $\text{ALG}_2(H, G, w, N, \mathcal{F})$ adds a neighborhood, then $T_H(\mu) \leq T_H(\mu([1 - \frac{1}{4|H| \cdot |\mathcal{CC}(H)| \cdot \log(\mu)}])$. The result now follows from the observation that $\text{ALG}_2(H, G, w, N, \mathcal{F})$ only does $|V(G)|^{O(1)} = \mu^{O(1)}$ work in a given call and always branches on a branchable vertex, adds a balanced separator, or immediately returns a value without making further recursive calls. □

Since $T_H(\mu)$ is a non negative, non decreasing function, by adding the two possibilities in the max of Lemma 16 we immediately obtain the following simplified recurrence.

Corollary 2. $T_H(\mu) \leq \mu^{O(1)} + T_H(\mu[1 - \frac{1}{4|H| \cdot \log(\mu)}]) + T_H(\mu - 1) + T_H(\mu[1 - \frac{1}{8|H|^2 \cdot |\mathcal{CC}(H)| \cdot \log^2(\mu)}]) < T_H(\mu - 1) + \mu^{O(1)} + 2T_H(\mu[1 - \frac{1}{8|H|^2 \cdot |\mathcal{CC}(H)| \cdot \log^2(\mu)}])$

Lemma 17. $T_H(\mu) = \mu^{O(|H|^2 \cdot |\mathcal{CC}(H)| \cdot \log^3(\mu))}$

Proof. For the sake of being able to apply induction to prove the bound, it will be beneficial for us to define the function $T'_H(\mu) = T_H(\lfloor \mu \rfloor)$. We will prove that

$T'_H(\mu) = \mu^{O(|H|^2 \cdot |\mathcal{CC}(H)| \cdot \log^3(\mu))}$, then it will follow that $T_H(\mu) = \mu^{O(|H|^2 \cdot |\mathcal{CC}(H)| \cdot \log^3(\mu))}$ since $T_H(\mu) \leq T'_H(\mu+1) = \mu^{O(|H|^2 \cdot |\mathcal{CC}(H)| \cdot \log^3(\mu))}$. Since T'_H rounds down its input and Lemma 14 shows the measure μ is always non-negative for any fair instance, it suffices to prove the desired bound on T'_H by induction on the natural numbers. We may also assume μ is an integer. The base case is established by observing that the runtime of algorithm on all such fair instances where its measure μ is at most 1 is bounded by a constant. Now, for the inductive hypothesis, assume that there exists a number c such that for all integers, a, less than $\mu > 1$, $T'_H(a) \leq a^{O(|H|^2 \cdot |\mathcal{CC}(H)| \cdot \log^3(a))}$. By Corollary 2 we have the inequality $T'_H(\mu) \leq T'_H(\mu-1) + \mu^{O(1)} + 2T'_H(\mu[1 - \frac{1}{8|H|^2 \cdot |\mathcal{CC}(H)| \cdot \log^2(\mu)}])$ and repeatedly applying the inequality to the first term on the right hand side, gives $T'_H(\mu) \leq \mu^{O(1)} + 2\mu T'_H(\mu[1 - \frac{1}{8|H|^2 \cdot |\mathcal{CC}(H)| \cdot \log^2(\mu)}])$. By the inductive hypothesis then, we have the following:

$$
\begin{aligned}
&T'_H(\mu) \leq \mu^{O(1)} \\
&+ 2\mu \lfloor \mu(1 - \frac{1}{8|H|^2 |\mathcal{CC}(H)| \log^2(\mu)}) \rfloor^{c|H|^2 \cdot |\mathcal{CC}(H)| \cdot \log^3(\mu)} \\
&\leq \mu^{O(1)} \\
&+ 2\mu\mu^{c|H|^2 \cdot |\mathcal{CC}(H)| \cdot \log^3(\mu)} (1 \\
&- \frac{1}{8|H|^2 \cdot |\mathcal{CC}(H)| \cdot \log^2(\mu)})^{c|H|^2 \cdot |\mathcal{CC}(H)| \cdot \log^3(\mu)} \\
&\leq \mu^{O(1)} + 2\mu\mu^{c|H|^2 \cdot |\mathcal{CC}(H)| \cdot \log^3(\mu)} e^{-\frac{c|H|^2 \cdot |\mathcal{CC}(H)| \cdot \log^3(\mu)}{8|H|^2 \cdot |\mathcal{CC}(H)| \cdot \log^2(\mu)}} \\
&\quad (\text{ since } (1-x) \leq e^{-x} \text{ }) \\
&\leq \mu^{O(1)} + 2\mu\mu^{c|H|^2 \cdot |\mathcal{CC}(H)| \cdot \log^3(\mu)} \cdot e^{-\frac{c \log(\mu)}{8}} \\
&\leq \mu^{c|H|^2 \cdot |\mathcal{CC}(H)| \cdot \log^3(\mu)} \text{ (for } c \text{ chosen large enough)}
\end{aligned}
$$

□

We are now ready to prove Theorem 2.

Proof of Theorem 2. The algorithm returns the answer of $\text{ALG}_2(H, G, w, |V(G)|, \emptyset)$. By Lemmata 10 and 11, ALG_2 will always terminate and return the weight of a maximum weight independent set in G. For the running time analysis, observe that $(H, G, w, |V(G)|, \emptyset)$ is a fair instance and let $\mu = \mu_H(H, G, w, N, \mathcal{F})$. We assume that $|H| \leq N$, since the run time bound follows trivially if $|H| > N$. By Lemma 15 we have that $\mu < 4|H|^2 \cdot |\mathcal{CC}(H)| \cdot N \cdot \log^2(N)$. Let $n = N = |V(G)|$, then it follows that $T_H(H, G, w, N, \mathcal{F}) \leq T_H(\mu) = \mu^{O(|H|^2 \cdot |\mathcal{CC}(H)| \cdot \log^3(\mu))} = n^{O(|H|^2 \cdot |\mathcal{CC}(H)| \cdot \log^3(n))}$. This completes the proof. □

Theorem 2 sligthly increases the current reach of Theorem 1. In particular, let T_k be the graph with k connected components the first of which is a path P_k on k vertices and the remaining $k-1$ are forks (a fork is a path on four vertices plus a single vertex adjacent to the second vertex of the path). Lozin and Milanic [22] gave a polynomial time algorithm for WEIGHTED INDEPENDENT SET on fork-free graphs. Theorem 2 implies that WEIGHTED INDEPENDENT SET on T_k free graphs can be solved by making $n^{O(k^3 \log^3(n))}$ oracle calls to the polynomial time algorithm of Lozin and Milanic [22] or the algorithm of Theorem 1. Thus we obtain the following result.

Theorem 3. *There exists an algorithm that given a T_k-free graph G and weight function $w : V(G) \to \mathbb{N}$, runs in $n^{O(k^3 \log^3 n)}$ time, and outputs the weight of a maximum weight independent set of G.*

V. CONCLUSION

In this paper we gave a quasipolynomial time algorithm for WEIGHTED INDEPENDENT SET on P_k-free graphs for all integers k. The dependence on k in the exponent is $O(k^2)$ and so our algorithm is quasi-polynomial even for $k = \log^{O(1)} n$ and sub-exponential for $k = n^{\frac{1}{2}-\epsilon}$ for $\epsilon > 0$. In light of our algorithm it is tempting to conjecture that (WEIGHTED) INDEPENDENT SET on P_k-free graphs can be solved in polynomial time for every k. Given how dependent our algorithms are on branching on high degree vertices it looks unlikely that our techniques can lead to polynomial time algorithms for P_k-free graphs. Nevertheless it may be possible to extract structural insights from our algorithms that could eventually lead to polynomial time algorithms.

Our second main result (Theorem 2) implies that if there exists a quasi-polynomial time algorithm for H-free graphs for every subdivided claw H then there exists a quasi-polynomial time algorithm for every finite family $\mathcal{H}$ such that NP-completeness of INDEPENDENT SET on $\mathcal{H}$-free graphs does not follow from Alekseev's result [18]. Thus, a quasi-polynomial time algorithm for subdivided-claw-free graphs would complete a dichotomy for the complexity of INDEPENDENT SET on $\mathcal{H}$-free graphs for every finite family H: every case is either quasi-polynomial time solvable or NP-complete.

REFERENCES

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

[2] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*, 1972, pp. 85–103.

[3] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy, "Interactive proofs and the hardness of approximating cliques," *J. ACM*, vol. 43, no. 2, pp. 268–292, 1996.

[4] D. Zuckerman, "Linear degree extractors and the inapproximability of max clique and chromatic number," *Theory of Computing*, vol. 3, no. 1, pp. 103–128, 2007.

[5] R. G. Downey and M. R. Fellows, *Parameterized Complexity*. New York: Springer-Verlag, 1999.

[6] R. Impagliazzo, R. Paturi, and F. Zane, "Which problems have strongly exponential complexity?" *J. Comput. Syst. Sci.*, vol. 63, no. 4, pp. 512–530, 2001.

[7] P. Chalermsook, M. Cygan, G. Kortsarz, B. Laekhanukit, P. Manurangsi, D. Nanongkai, and L. Trevisan, "From gap-eth to fpt-inapproximability: Clique, dominating set, and more," in *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, C. Umans, Ed. IEEE Computer Society, 2017, pp. 743–754.

[8] M. Grötschel, L. Lovász, and A. Schrijver, "The ellipsoid method and its consequences in combinatorial optimization." *Combinatorica*, vol. 1, pp. 169–197, 1981.

[9] E. Balas and C. S. Yu, "On graphs with polynomially solvable maximal-weight clique problem," *Networks*, vol. 19, pp. 247—-253, 1989.

[10] B. Courcelle, J. A. Makowsky, and U. Rotics, "Linear time solvable optimization problems on graphs of bounded clique-width," *Theory Comput. Syst.*, vol. 33, no. 2, pp. 125–150, 2000.

[11] A. Brandstädt, J. P. Spinrad *et al.*, *Graph classes: a survey*. Siam, 1999, no. 3.

[12] H. De Ridder *et al.*, "Information system on graph classes and their inclusions)," *www.graphclasses.org*, 2016.

[13] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, ser. Series of Books in the Mathematical Sciences. W. H. Freeman and Co., 1979.

[14] B. N. Clark, C. J. Colbourn, and D. S. Johnson, "Unit disk graphs," *Discrete Math.*, vol. 86, no. 1 – 3, pp. 165 – 177, 1990.

[15] S. Poljak, "A note on stable sets and colorings of graphs," *Commentationes Mathematicae Universitatis Carolinae*, vol. 15, no. 2, pp. 307 – 309, 1974.

[16] H. Broersma, T. Kloks, D. Kratsch, and H. Müller, "Independent sets in asteroidal triple-free graphs," *SIAM J. Discrete Math.*, vol. 12, no. 2, pp. 276–287, 1999.

[17] P. Gartland and D. Lokshtanov, "Independent set on P_k-free graphs in quasi-polynomial time," *ArXiv*, vol. abs/2005.00690, 2020.

[18] V. Alekseev, "The effect of local constraints on the complexity of determination of the graph independence number," *Combinatorial-algebraic methods in applied mathematics*, pp. 3–13, 1982, (in Russian).

[19] N. Sbihi, "Algorithme de recherche d'un stable de cardinalite maximum dans un graphe sans etoile," *Discrete Mathematics*, vol. 29, no. 1, pp. 53 – 76, 1980.

[20] G. J. Minty, "On maximal independent sets of vertices in claw-free graphs," *Journal of Combinatorial Theory, Series B*, vol. 28, no. 3, pp. 284 – 304, 1980. [Online]. Available: http://www.sciencedirect.com/science/article/pii/009589568090074X

[21] V. E. Alekseev, "Polynomial algorithm for finding the largest independent sets in graphs without forks," *Discrete Applied Mathematics*, vol. 135, no. 1-3, pp. 3–16, 2004.

[22] V. V. Lozin and M. Milanic, "A polynomial algorithm to find an independent set of maximum weight in a fork-free graph," *J. Discrete Algorithms*, vol. 6, no. 4, pp. 595–604, 2008.

[23] D. Corneil, H. Lerchs, and L. Burlingham, "Complement reducible graphs," *Discrete Applied Mathematics*, vol. 3, no. 3, pp. 163 – 174, 1981.

[24] R. Boliac and V. V. Lozin, "An augmenting graph approach to the stable set problem in P_5-free graphs," *Discrete Applied Mathematics*, vol. 131, no. 3, pp. 567 – 575, 2003. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0166218X0300221X

[25] A. Brandstädt and R. Mosca, "On the structure and stability number of P_5- and co-chair-free graphs," *Discrete Applied Mathematics*, vol. 132, no. 1–3, pp. 47 – 65, 2003, stability in Graphs and Related Topics. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0166218X03003895

[26] M. U. Gerber and V. V. Lozin, "On the stable set problem in special P_5-free graphs," *Discrete Applied Mathematics*, vol. 125, no. 2-3, pp. 215–224, 2003.

[27] V. V. Lozin and R. Mosca, "Independent sets in extensions of $2k_2$-free graphs," *Discrete Applied Mathematics*, vol. 146, no. 1, pp. 74 – 80, 2005. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0166218X04002902

[28] R. Mosca, "Some observations on maximum weight stable sets in certain P_5-free graphs," *European Journal of Operational Research*, vol. 184, no. 3, pp. 849 – 859, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0377221706012318

[29] B. Randerath and I. Schiermeyer, "On maximum independent sets in P_5-free graphs," *Discrete Applied Mathematics*, vol. 158, pp. 1041–1044, 2010.

[30] D. Lokshantov, M. Vatshelle, and Y. Villanger, "Independent set in P_5-free graphs in polynomial time," in *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, C. Chekuri, Ed. SIAM, 2014, pp. 570–581.

[31] D. Lokshtanov, M. Pilipczuk, and E. J. van Leeuwen, "Independence and efficient domination on P_6-free graphs," *ACM Trans. Algorithms*, vol. 14, no. 1, pp. 3:1–3:30, 2018.

[32] A. Grzesik, T. Klimosova, M. Pilipczuk, and M. Pilipczuk, "Polynomial-time algorithm for maximum weight independent set on P_6-free graphs," in *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, T. M. Chan, Ed. SIAM, 2019, pp. 1257–1271.

[33] A. Grzesik, T. Klimosová, M. Pilipczuk, and M. Pilipczuk, "Covering minimal separators and potential maximal cliques in P_t-free graphs," *CoRR*, vol. abs/2003.12345, 2020.

[34] G. Bacsó, D. Lokshtanov, D. Marx, M. Pilipczuk, Z. Tuza, and E. J. van Leeuwen, "Subexponential-time algorithms for maximum independent set in P_t-free and broom-free graphs," *Algorithmica*, vol. 81, no. 2, pp. 421–438, 2019.

[35] C. Brause, "A subexponential-time algorithm for the maximum independent set problem in P_t-free graphs," *Discret. Appl. Math.*, vol. 231, pp. 113–118, 2017.

[36] C. Groenland, K. Okrasa, P. Rzazewski, A. D. Scott, P. D. Seymour, and S. Spirkl, "H-colouring P_t-free graphs in subexponential time," *Discret. Appl. Math.*, vol. 267, pp. 184–189, 2019.

[37] M. Chudnovsky, M. Pilipczuk, M. Pilipczuk, and S. Thomassé, "Quasi-polynomial time approximation schemes for the maximum weight independent set problem in H-free graphs," in *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, S. Chawla, Ed. SIAM, 2020, pp. 2260–2278.

[38] A. Gyárfás, "Problems from the world surrounding perfect graphs," *Applicationes Mathematicae*, vol. 3, no. 19, pp. 413–441, 1987.

[39] F. V. Fomin and D. Kratsch, *Exact Exponential Algorithms*, ser. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2010.