Odd Cycle Transversal on P_5-free Graphs in Quasi-polynomial Time*

Akanksha Agrawal† Paloma T. Lima‡ Daniel Lokshtanov§ Saket Saurabh¶
Roohani Sharma‖

Abstract

An *independent set* in a graph G is a set of pairwise non-adjacent vertices. A graph G is *bipartite* if its vertex set can be partitioned into two independent sets. In the Odd Cycle Transversal problem, the input is a graph G along with a weight function $\mathtt{w}$ associating a rational weight with each vertex, and the task is to find a smallest weight vertex subset S in G such that $G-S$ is bipartite; the weight of S, $\mathtt{w}(S) = \sum_{v \in S} \mathtt{w}(v)$. We show that Odd Cycle Transversal admits an algorithm with running time $n^{O(\log^2 n)}$ on graphs excluding P_5 (a path on five vertices) as an induced subgraph. The problem was previously known to be polynomial time solvable on P_4-free graphs and NP-hard on P_6-free graphs [Dabrowski, Feghali, Johnson, Paesani, Paulusma and Rzążewski, Algorithmica 2020]. Bonamy, Dabrowski, Feghali, Johnson and Paulusma [Algorithmica 2019] posed the existence of a polynomial time algorithm on P_5-free graphs as an open problem, this was later re-stated by Rzążewski [Dagstuhl Reports, 9(6): 2019] and by Chudnovsky, King, Pilipczuk, Rzążewski, and Spirkl [SIDMA 2021], who gave an algorithm with running time $n^{O(\sqrt{n})}$. While our $n^{O(\log^2 n)}$ time algorithm falls short of completely resolving the complexity status of Odd Cycle Transversal on P_5-free graphs it shows that the problem is not NP-hard unless every problem in NP is solvable in quasi-polynomial time.

1 Introduction

In a vertex deletion problem, the input is a graph G along with a weight function $\mathtt{w} : V(G) \to \mathbb{Q}$, and the task is to find a smallest weight vertex subset S such that removing S from G results in a graph that belongs to a certain target class of graphs; the weight of S being defined as $\mathtt{w}(S) = \sum_{v \in S} \mathtt{w}(v)$. By varying the target graph class we can obtain many classic graph problems, such as deletion to edge-less graphs (Vertex Cover), deletion to acyclic graphs (Feedback Vertex Set), deletion to bipartite graphs (Odd Cycle Transversal), or deletion to planar graphs (Planarization). With the exception of the class of all graphs, for every target class that contains an infinite set of graphs and is closed under vertex deletion, the vertex deletion problem to that graph class is NP-hard [28]. For this reason a substantial research effort has been dedicated to understanding the computational complexity of various vertex deletion problems when the input graph G is required to belong to a restricted graph class as well (see [5] and the companion website [11]).

In this paper, we consider the Odd Cycle Transversal (OCT) problem, that is the vertex deletion problem to *bipartite* graphs. A vertex set S is *independent* if no edge has both its endpoints in S and a graph G is bipartite if its vertex set can be partitioned into two independent sets. A graph is bipartite if and only if it has no odd cycles [12]. The OCT problem is very well studied, and has been considered from the perspective of approximation [17, 13, 26], heuristics [19, 1], exact exponential time [35, 37, 24] and parameterized algorithms [27, 29, 36, 30, 31, 22, 25, 21, 20, 23].

From the viewpoint of restricting the input to specific classes of graphs, OCT is known to be polynomial time solvable on permutation graphs [4] and more generally on graphs of bounded mim-width [6]. More recently, H-free graphs, that is, graph classes defined by one forbidden induced subgraph, received particular attention. Chiarelli et al. [7] showed that OCT is NP-complete on graphs of small

*Funding acknowledgements: Agrawal is supported by SERB Startup Research Grant, no. SRG/2022/000962; Lima is supported by the Independent Research Fund Denmark grant agreement number 2098-00012B; Lokshtanov is supported by NSF grant CCF-2008838; Saurabh is supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme grant agreement number 819416, and by a Swarnajayanti Fellowship (No. DST/SJF/MSA01/2017-18).

†Indian Institute of Technology Madras, Chennai, India, akanksha@cse.iitm.ac.in.
‡IT University of Copenhagen, Copenhagen, Denmark, palt@itu.dk.
§University of California, Santa Barbara, USA daniello@ucsb.edu.
¶Institute of Mathematical Sciences, Chennai, India and University of Bergen, Norway, saket@imsc.res.in.
‖Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany, rsharma@mpi-inf.mpg.de.

fixed girth and line graphs. This implies that if H contains a cycle or a claw, then OCT is NP-hard on H-free graphs. Hence, we can restrict ourselves to the case in which H is a linear forest (i.e., each connected component of H is a path). For P_4-free graphs a polynomial time algorithm follows directly from the algorithm of Brandstädt and Kratsch [4] for permutation graphs. Bonamy et al. [3] posed the question of the existence of a polynomial time algorithm for OCT on P_k-free graphs, for all $k \geq 5$. Subsequently, Okrasa and Rzążewski [33] showed that OCT is NP-hard on P_{13}-free graphs, and shortly thereafter Dabrowski et al. [10] proved that the problem remains NP-hard even on $(P_2 + P_5, P_6)$-free graphs. On the other hand, OCT is known to be polynomial time solvable on sP_2-free graphs [7] and on $(sP_1 + P_3)$-free graphs [10], for every constant $s \geq 1$. Thus, prior to our work, the only *connected* graph H such that the complexity status of OCT on H-free graphs remained unknown was the P_5. For this reason, resolving the complexity status of OCT on P_5-free graphs was posed as an open problem by Rzążewski [9], and by Chudnovsky et al. [8], who gave an algorithm for OCT on P_5-free graphs with running time $n^{O(\sqrt{n})}$. In this paper, we come quite close to resolving the computational complexity of OCT on P_5-free graphs by showing that the problem is unlikely to be NP-hard. Specifically, we prove the following theorem.

THEOREM 1.1. *There is an $n^{O(\log^2 n)}$-time algorithm for* OCT *on P_5-free graphs, where n is the number of vertices in the input graph.*

We now discuss the main ideas of the algorithm of Theorem 1.1. The algorithm follows the framework introduced by Gartland and Lokshtanov [15] for the INDEPENDENT SET problem on P_k-free graphs, or rather the substantially simplified version presented by Pilipczuk et al. [34]. We now briefly review the algorithms in [34] for INDEPENDENT SET. In the INDEPENDENT SET problem the input is a graph G along with a weight function $\mathtt{w} : V(G) \to \mathbb{Q}$, and the task is to find a largest possible weight independent set S. We assume that all the vertex weights are positive, as otherwise, we can remove the respective vertices. Clearly, if the graph G is disconnected we can consider each of the connected components independently. If the graph is connected, we select a vertex v and *branch on* v. In particular, we find the largest independent set in G not containing v by running the algorithm recursively on $G-v$ (this is called the *fail* branch), and the largest independent set in G containing v by running the algorithm recursively on $G - N[v]$ (this is called the *success* branch). Here $N[v]$ is the *closed neighborhood* of v, namely the set of all vertices with an edge to v and v itself. The algorithm terminates when the input graph has at most one vertex, in which case the vertex set $V(G)$ of the graph is a maximum weight independent set in G. The key point of the algorithm in [34] is that, in P_k-free graphs it is always possible to find a vertex v to branch on such that every root-leaf path in the recursion tree of the algorithm has at most $O(\log^c n)$ success branches for some fixed constant c. From here a simple counting argument bounds the size of the recursion tree by $n^{O(\log^c n)}$. Since we spend polynomial time in each node of the recursion tree, we get a quasi-polynomial running time bound for the algorithm.

We would like to use these ideas for OCT on P_5 free graphs. To make the problem formulation closer to INDEPENDENT SET we will work with the MAXIMUM BIPARTITE SUBGRAPH (MAX-BIP-SUB) problem instead. That is, we wish to find a largest possible set S such that $G[S]$ is bipartite. (Notice that OCT is equivalent to MAX-BIP-SUB from the viewpoint of classical complexity.) There is immediately a difficulty that pops up: how can we branch on a vertex? In INDEPENDENT SET, when v is included in S we can immediately discard all of $N(v)$, making a lot of progress towards disconnecting G. In MAX-BIP-SUB, however, when v is in S, many of its neighbors might also be in S, and we do not know which ones.

This is where our main combinatorial insight comes in: we prove that given a P_5 free graph G and a vertex v in G, we can compute in polynomial time a polynomial number of sets $C_1, \ldots, C_t$. Each of these sets contains v and induces a connected bipartite graph in G. Furthermore, out of all vertex sets S containing v such that $G[S]$ is bipartite, there exists a maximum weight such set S such that the (vertex set of the) connected component of $G[S]$ that contains v is contained in C_i, for some $i \leq t$ (and $C_i \subseteq S$). The proof of this lemma heavily relies on the structure of P_5-free graphs, including the classical result that every connected P_5-free graph has a dominating P_3 or dominating clique [2], as well as a somewhat surprising application of the concept of modules [14] (a module in a graph G is a set M of vertices such that every vertex outside of M either is adjacent to all of M or to none of M.)

Armed with the main structural lemma we can now carry out our initial plan: we select a vertex v to branch on using the exact same strategy as Pilipczuk et al. [34]. When we branch on a vertex v, we first need to find the maximum weight set S not containing v such that $G[S]$ is bipartite. We do this by running the algorithm recursively on $G - v$, just as for INDEPENDENT SET. We also need to find

the maximum weight set S containing v such that $G[S]$ is bipartite. By the key lemma, we may assume that the vertex set of the connected component of $G[S]$ that contains v is contained in C_i and $C_i \subseteq S$, for some $i \leq t$. For each choice of $i \leq t$, we try adding C_i to S and call the algorithm recursively on $G - N[C_i]$ (here $N[C_i]$ is the *closed neighborhood* of C_i, namely all vertices in C_i plus all vertices with a neighbor in C_i). We consider each one of these recursive calls a *success branch*.

Since every success branch works with an induced subgraph of $G - N[v]$, the analysis of Pilipczuk et al. [34] showing that every root to leaf path of the recursion tree has at most $O(\log^2 n)$ success branches works out in our setting as well. To translate this to an $n^{O(\log^2 n)}$ running time bound for the algorithm we need to verify that the counting argument showing that the size of the recursion tree is upper bounded by $n^{O(\log^2 n)}$ still works out even when every node of the recursion tree can have $n^{O(1)}$ success branches instead of just one. This was already done by Gartland et al [16] in their algorithm for Feedback Vertex Set on graphs with no short induced cycles, however since the argument is short and instructive we repeat (a variant of) it below.

It is not too difficult to convince oneself that every root-leaf path of the recursion tree is uniquely identified by the following.

1. A set $D \subseteq \{1, \ldots, n\}$ of size at most $O(\log^2 n)$: this is the set of depths at which the path goes to a success branch or graphs with substantially reduced number of vertices.
2. A $|D|$-tuple $i_1, i_2, \ldots, i_{|D|}$ of integers between 1 and $n^{O(1)}$. For each $j \leq |D|$ the integer i_j points to which success branch to follow the j'th time the root-leaf path goes along a success branch, if the graph before the success branch is connected.
3. Every time the root-leaf path of the recursion tree visits a disconnected graph G the corresponding i_j, for $j \leq |D|$, points to the direction where the number of vertices in the instance is reduced at least by half.

There are at most $n^{O(\log^2 n)}$ choices for D and at most $n^{O(\log^2 n)}$ choices for $i_1, i_2, \ldots, i_{|D|}$, leading to the $n^{O(\log^2 n)}$ upper bound on the size of the recursion tree, and therefore also on the running time.

Outline of the Paper. In Section 2 we introduce the necessary notation and review some useful facts about P_5-free graphs. In Section 3 we prove Theorem 1.1 assuming the main combinatorial lemma, while in Section 3.1 we prove the lemma. Finally, we conclude with some open problems in Section 4.

2 Preliminaries

We denote the set of natural numbers and the set of rational numbers by $\mathbb{N} = \{0, 1, 2, \cdots\}$ and $\mathbb{Q}$, respectively, and let $\mathbb{Q}_{\geq 0} = \{x \in \mathbb{Q} \mid x \geq 0\}$. For $c \in \mathbb{N}$, $[c]$ denotes the set $\{1, \cdots, c\}$. Unless stated otherwise, the base of log will be e. For a set X, we denote by 2^X the collection of all subsets of X, by $\binom{X}{i}$ the collection of all i-sized subsets of X, by $\binom{X}{\leq i}$ all subsets of X of size at most i, and by $\binom{X}{j_1 \leq i \leq j_2}$ the collection of all i-sized subsets of X where $j_1 \leq i \leq j_2$.

Consider a graph G. For $v \in V(G)$, $N_G(v)$ denotes the set of neighbours of v in G, and $N_G[v] = N_G(v) \cup \{v\}$. For $X \subseteq V(G)$, $N_G(X) = (\cup_{x \in X} N_G(x)) \setminus X$ and $N_G[X] = N_G(X) \cup X$. For a subgraph H of G, we sometimes write $N_G(H)$ (resp. $N_G[H]$) as a shorthand for $N_G(V(H))$ (resp. $N_G[V(H)]$). For any $X, Y \subseteq V(G)$, $E_G(X, Y)$ denotes the set of edges of G with one endpoint in X and the other in Y. Whenever the graph G is clear from the context, we drop the subscript G from the above notations. For any $X \subseteq V(G)$, $G[X]$ denotes the graph induced by X, i.e., $V(G[X]) = X$ and $E(G[X]) = \{(u, v) \in E(G) \mid u, v \in X\}$. Moreover, by $G - X$ we denote the graph $G[V(G) \setminus X]$. For any $v \in V(G)$, we use $G - v$ as a shorthand for $G - \{v\}$. For two graphs G_1 and G_2 by $G_1 \cup G_2$ we denote the graph with vertex set $V(G_1) \cup V(G_2)$ and edge set $E(G_1) \cup E(G_2)$.

For any $i \in \mathbb{N}$, P_i is a path on i vertices and a P_i-free graph is a graph that has no induced subgraph isomorphic to P_i. A *dominating set* in a graph G is a set of vertices $D \subseteq V(G)$ such that $N[D] = V(G)$.

Proposition 2.1. (Theorem 8, [2]) *Every connected P_5-free graph G has a dominating set D such that $G[D]$ is either a P_3 or a clique.*

3 Algorithm For Max-Bip-Sub on P_5-free Graphs

The Maximum Bipartite Subgraph problem is formally defined below.

Maximum Bipartite Subgraph (Max-Bip-Sub)
Input: An undirected graph G on n vertices and a weight function $\mathtt{w} : V(G) \to \mathbb{Q}$.
Output: Find $S \subseteq V(G)$, such that $G[S]$ is bipartite and $\mathtt{w}(S) = \sum_{v \in S} \mathtt{w}(v)$ is maximized.

A set $S \subseteq V(G)$ such that $G[S]$ is bipartite, is called a *solution* of G. If $\mathtt{w}(S)$ is maximum then we call it an *optimal solution* of $(G, \mathtt{w})$. We will assume that the weight of each vertex is positive, as otherwise, we can remove the vertices with non-positive weight without changing the optimal solution. Recall our main result is Theorem 1.1, restated below.

THEOREM 3.1. *There is an $n^{O(\log^2 n)}$-time algorithm for* OCT *on P_5-free graphs, where n is the number of vertices in the input graph.*

As stated earlier, the key ingredient in the proof of Theorem 1.1 is a structural lemma (to be stated in Lemma 3.1). Roughly speaking, for a given graph G and a vertex v, this result gives a way to construct a polynomial sized family (in polynomial time) of vertex subsets inducing bipartite subgraphs, so that among the solutions containing a vertex v, there is a maximum weight solution S containing v such that the connected component in $G[S]$ containing v is contained in a set C in the computed family, and moreover, $C \subseteq S$. Before moving further, we state our result on the construction of such a family, the proof of which is relegated to Section 3.1.

LEMMA 3.1. *Given a P_5-free graph G on n vertices, a vertex $v \in V(G)$ and a weight function $\mathtt{w} : V(G) \to \mathbb{Q}_{\geq 0}$, there exists a polynomial-time algorithm that outputs a collection $\emptyset \neq \mathcal{C} \subseteq 2^{V(G)}$ of vertex subsets of G such that:*

1. *$|\mathcal{C}| = O(n^6)$,*
2. *for each $C \in \mathcal{C}$, $G[C]$ is bipartite, and*
3. *for every $S \subseteq V(G)$ where $v \in S$, $G[S]$ is bipartite and C is a connected component of $G[S]$ that contains v, there exists $C' \in \mathcal{C}$ such that the following holds for $S' = (S \setminus V(C)) \cup C'$: a) $G[S']$ is bipartite, b) $N(C') \cap S' = \emptyset$, and c) $\mathtt{w}(S') \geq \mathtt{w}(S)$.*

Equipped with the above lemma, we will devise a branching step similar to the one presented in [34]. Roughly speaking, our algorithm for Theorem 1.1 branches on a carefully chosen vertex v as follows: i) either v does not belong to any optimal solution, in which case we delete v from the graph and recurse; or ii) v belongs to an optimal solution, and now using Lemma 3.1 we compute a polynomial-sized family of bipartite vertex subsets of G, such that the connected component of any optimal solution that contains v can be replaced with one of the sets in this family to yield another optimal solution. In the later branch, the algorithm branches and recursively solves the instances $(G - N[C], \mathtt{w})$,[1] for every C in the family outputted by the lemma. We remark that the vertex v will be chosen so that it intersects a fraction of paths for a fraction of pairs of vertices, and this will help us bound the size of the recursion tree.

In the remaining, we give the proof of Lemma 3.1 in Section 3.1 and give the proof of Theorem 1.1 using Lemma 3.1 in Section 3.2.

3.1 Proof of Lemma 3.1

In this section we prove our key lemma restated below.

LEMMA 3.1. *Given a P_5-free graph G on n vertices, a vertex $v \in V(G)$ and a weight function $\mathtt{w} : V(G) \to \mathbb{Q}_{\geq 0}$, there exists a polynomial-time algorithm that outputs a collection $\emptyset \neq \mathcal{C} \subseteq 2^{V(G)}$ of vertex subsets of G such that:*

1. *$|\mathcal{C}| = O(n^6)$,*
2. *for each $C \in \mathcal{C}$, $G[C]$ is bipartite, and*
3. *for every $S \subseteq V(G)$ where $v \in S$, $G[S]$ is bipartite and C is a connected component of $G[S]$ that contains v, there exists $C' \in \mathcal{C}$ such that the following holds for $S' = (S \setminus V(C)) \cup C'$: a) $G[S']$ is bipartite, b) $N(C') \cap S' = \emptyset$, and c) $\mathtt{w}(S') \geq \mathtt{w}(S)$.*

At the heart of our proof for the above lemma, lies a property that we prove: for any $S \subseteq V(G)$ where $G[S]$ is a bipartite graph and C is the connected component of $G[S]$ containing v, there is a small set $\widetilde{D} \subseteq V(G)$, so that (after doing some appropriate cleaning operation of the graph), $N[\widetilde{D}] = N[C]$. Once we have the above property, we will be able to find an appropriate replacement for C to obtain C' with the desired property, by exploiting the known algorithm for computing independent sets on P_5-free graphs. We will now intuitively explain the steps of our algorithm, and we give a concrete pseudo code for it in Algorithm 1.

[1] For simplicity, with a slight abuse of notation here—instead of correctly writing $\mathtt{w}|_{G-N[C]}$, we will simply write $\mathtt{w}$. This convention will be followed in this as well as the next section.

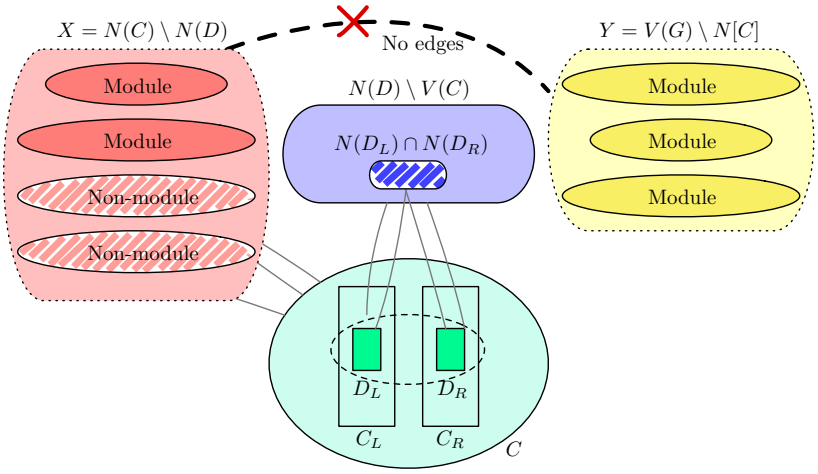

Figure 1: Illustration of various sets and connected components used in the algorithm. The striped ovals denote removal of the corresponding vertices by the algorithm.

In the following, we denote the input graph by G' (instead of G). Consider a set $S \subseteq V(G')$ where $v \in S$ and $G'[S]$ is bipartite, and let C be the connected component of $G'[S]$ that contains v. Our goal is to compute a family $\mathcal{C}$ of vertex subsets, that either contains C itself or a replacement for C which is as good as C in terms of weight. Since we would like to do our computations in polynomial time and we require the family to contain polynomially many sets, we cannot store all such potential Cs, and thus in certain cases we would like to work with a replacement for C from the family $\mathcal{C}$, which does the job as good as C itself.

To cover the trivial case when C has exactly one vertex, we add $\{v\}$ to the set $\mathcal{C}$ in Line 1. Hereafter, we will worry about the case when C has at least 2 vertices. As C is a connected, P_5-free and bipartite graph, from Proposition 2.1, there must exist a dominating induced P_3 or a dominating P_2 for C; let one such P_3 or P_2 be D. Note that $V(C) \subseteq N[D]$. The loop at Line 2 of Algorithm 1 will precisely be iterating over such potential Ds. We will next explain our operations for this fixed D, and since we will delete some vertices from our graph, we initialize $G = G'$ at Line 3. We remark that, at all point of time we will implicitly maintain that $S \subseteq V(G)$, and thus maintain that C is the connected component of $G[S]$, containing v.

We let $C_L \uplus C_R$ be the bipartition of C and $D_L \uplus D_R$ be the bipartition of D such that $D_L \subseteq C_L$ and $D_R \subseteq C_R$ (see Figure 1).[2] Recall that C is the connected component of $G[S]$ containing v. Notice that any vertex $u \in N(D_L) \cap N(D_R)$ must lie outside of S.[3] Thus, Line 5-7 removes such vertices from G.

A set of vertices $A \subseteq V(G)$ is a *module* in G, if for every pair $u, v \in A$, $N(u) \setminus A = N(v) \setminus A$. Note that checking whether a set $A \subseteq V(G)$ is a module can be done in polynomial time.

We let X be the neighbors of C outside $N(D)$, i.e., $X = N(C) \setminus N(D)$. Also, we let Y be the vertices outside of C and its neighborhood, i.e., $Y = V(G) \setminus N[C]$. We will establish the following statements:

(In Claim 3.1) There are no edges between a vertex in X and a vertex in Y, i.e., $E(X, Y) = \emptyset$.

(In Claim 3.2) Each connected component of $G[Y]$ is a module in G.

(In Claim 3.3, stated roughly here) the graph $G[X \cup V(C)]$ has a dominating set $\widetilde{D}$ such that: $D \subseteq \widetilde{D}$ and $|\widetilde{D} \setminus D| \leq 3$.

Recall our assumption that $V(C) \subseteq N[D]$. Using the first property that we establish, we can obtain that for any connected component Z of $G - N[D]$, either $V(Z) \subseteq X$ or $V(Z) \subseteq Y$. Now using the second property stated above, we can obtain that the vertices in a non-module connected component of $G - N[D]$ must belong to X, and thus it cannot contain a vertex from S. This leads us to Line 8-10 of our algorithm, where we remove vertices from all such non-module connected components. We let

[2] Note that both C and D are connected bipartite graphs. Thus, upto switching parts, there are exactly one valid bipartitions of these graphs.

[3] Unless stated explicitly, inside the loop starting at Line 2, we will only be concerned with the graph G, and thus, we omit the graph subscripts from the notations like $N(D_L)$.

Algorithm 1 Isolating a connected Component

Input: An undirected graph G', a vertex $v \in V(G')$ and a weight function $\mathtt{w} : V(G') \to \mathbb{Q}_{\geq 0}$

Output: $\mathcal{C} \subseteq 2^{V(G')}$ satisfying the properties of Lemma 3.1

1: Initialize $\mathcal{C} = \{\{v\}\}$.
2: **for all** $D \subseteq \binom{V(G')}{2 \leq i \leq 3}$, where $G'[D]$ is connected and bipartite **do**
3: Initialize $G = G'$.
4: Fix a bipartition $D = D_L \uplus D_R$.
5: **while** there exists $u \in N(D_L) \cap N(D_R)$ **do**
6: Delete u from G. That is, $G = G - u$.
7: **end while**
8: **while** there exists a connected component Z of $G - N[D]$ such that $V(Z)$ is not a module in G **do**
9: Delete Z from G. That is, $G = G - V(Z)$.
10: **end while**
11: **for all** $D' \subseteq \binom{V(G)}{\leq 3}$ **do**
12: Let $C^*_{D,D'} = N[D \cup D']$.
13: **while** there exists $u \in C^*_{D,D'}$ such that $N_{G'}(u) \setminus C^*_{D,D'} \neq \emptyset$ **do**
14: Delete u from G. That is, $G = G - u$.
15: **end while**
16: Let I_L be a maximum weight independent set obtained by running the algorithm of Proposition 3.1 on input $(G[N(D_R)], \mathtt{w})$.
17: Let I_R be a maximum weight independent set obtained by running the algorithm of Proposition 3.1 on input $(G[N(D_L)], \mathtt{w})$.
18: Let $C_{D,D'} = I_L \cup I_R$.
19: Update $\mathcal{C} = \mathcal{C} \cup \{C_{D,D'}\}$.
20: **end for**
21: **end for**

$R \subseteq X$ be the vertices remaining after the above stated deletion of vertices from X.

The third property will be used to completely identify the vertices of $N[C]$, however, we may not be able to precisely distinguish which vertices from $N[C]$ belong to C. To this end, we will iterate over the potential choices of D' of size at most 3, so that $\widetilde{D} = D \cup D'$ is a dominating set for $G[R \cup V(C)]$, at Line 11. We will argue that for such a correct $\widetilde{D}$, we must have $N[\widetilde{D}] = N[C]$. Thus, knowing $\widetilde{D}$ precisely gives us $N[C]$, and at Line 12 we denote the respective set by $C^*_{D,D'} = N[\widetilde{D}] = N[C]$.

Recall that C is a connected biparitite induced subgraph of $G[S]$, and any vertex from $G[N[\widetilde{D}]] = G[N[C]]$ that has a neighbor from $V(G') \setminus N[\widetilde{D}]$ cannot belong to S (and thus C). Thus, at Line 13-15 we do the cleaning operation by removing such vertices from G.

After this (in the graph resulting after the previous deletions), we obtain that $N[C]$ is a connected component of G (not necessarily bipartite) and $S \subseteq V(G)$. Now instead of finding $V(C)$ inside $N[C]$ exactly, we will find some C' which will be as good as $V(C)$ as follows. We recall that D is a connected bipartite dominating set for C and thus (upto switching parts), D has a unique bipartition $D_L \uplus D_R$, which we have fixed at Line 4, and we have assumed that $D_L \subseteq C_L$ and $D_R \subseteq C_R$. Due to the cleaning operation of G at Line 5-7, D_L and D_R have no common neighbors, i.e., $N[D_L] \cap N[D_R] = \emptyset$. Moreover, as D is a dominating set for the bipartite connected graph C, it must be the case that $C_L \subseteq N[D_R]$ and $C_R \subseteq N[D_L]$. Also, recall due to the operations at Line 13-15 we will be able to conclude that $N[C]$ is a connected component in G and S is an induced subgraph of G. Now instead of finding C_L and C_R precisely, we find maximum weight independent sets I_L and I_R on graphs $G[N[D_R]]$ and $G[N[D_L]]$, respectively. Notice that due to our discussions above, I_L and I_R must be disjoint. We then show that $C_{D,D'} = I_L \uplus I_R$ serves as a replacement for $V(C)$, and thus we add $C_{D,D'}$ to $\mathcal{C}$ at Line 19.

We next state a known result regarding computation of independent sets on P_5-free graphs.

PROPOSITION 3.1. ([32], INDEPENDENT SET ON P_5-FREE) *Given a graph G and a weight function $\mathtt{w} : V(G) \to \mathbb{Q}_{\geq 0}$, there is a polynomial-time algorithm that outputs a set $I \subseteq V(G)$ such that I is an independent set in G and $\mathtt{w}(I) = \sum_{u \in I} \mathtt{w}(u)$ is the maximum.*

We remark that for our algorithm we need the algorithm of Proposition 3.1 to return a maximum

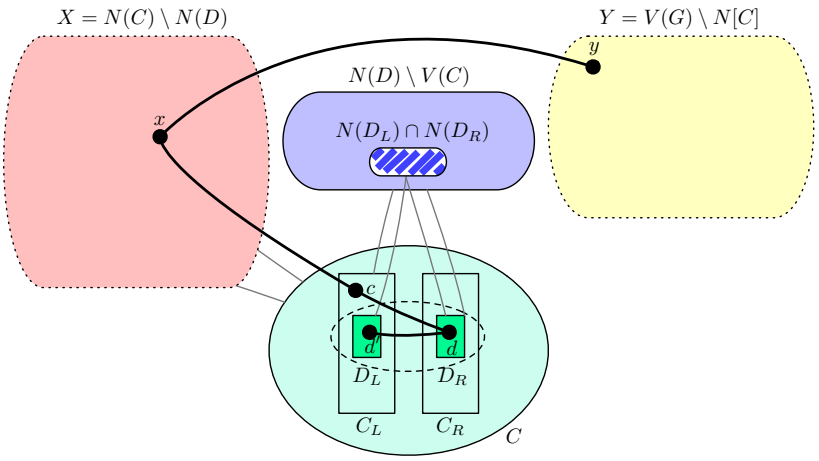

Figure 2: Illustration of various elements in the proof of Claim 3.1.

weight independent set and not just the weight such a set. The arguments in [32] suffice to output such a set. The following observation is immediate from the description of the algorithm and Proposition 3.1.

OBSERVATION 3.1. *Algorithm 1 runs in polynomial time.*

We prove the correctness of our algorithm in the next lemma.

LEMMA 3.2. *Given a P_5-free graph G', vertex $v \in V(G')$ and a weight function $\mathtt{w} : V(G') \to \mathbb{N}$, Algorithm 1 outputs a family $\mathcal{C}$ such that:*

- $|\mathcal{C}| = O(n^6)$,
- *for each $C \in \mathcal{C}$, $G'[S]$ is bipartite, and*
- *for every $S \subseteq V(G')$ where $v \in S$, $G'[S]$ is bipartite and C is the connected component of $G'[S]$ that contains v, there exists $C' \in \mathcal{C}$ such that the graph G' induced by $S' = (S \setminus V(C)) \cup C'$ is bipartite, $N(C') \cap S' = \emptyset$ and $\mathtt{w}(S') \geq \mathtt{w}(S)$.*

Proof. To bound the size of the family $\mathcal{C}$, observe that $\mathcal{C}$ is initialized in Line 1 and updated only in Line 19. At Line 1 the size of $\mathcal{C}$ is 1 and Line 19 is executed at most $O(n^6)$ times: inside the for-loop at Line 2 which contains the for-loop at Line 11.

It is also easy to see that the sets of $\mathcal{C}$ induce bipartite subgraphs of G. Indeed, at Line 1 there is a singleton set and, at Line 19 the set $C_{D,D'}$ added to the family is the disjoint union of two independent sets I_L and I_R (Line 18), therefore $G[C_{D,D'}]$ is bipartite.

We will now show the third property of the lemma. Let $S \subseteq V(G')$ such that $v \in S$ and $G'[S]$ is bipartite. Let C be the connected component of $G'[S]$ that contains v. If C has exactly one vertex (namely, v), then $C \in \mathcal{C}$ from Line 1, and thus the property trivially holds. Therefore assume that $|V(C)| \geq 2$. As C is a connected, P_5-free and bipartite graph on at least 2 vertices (therefore K_3-free[4]), from Proposition 2.1 there exists a dominating set D of C which either induces a P_3 or a P_2. We consider the execution of the for-loop at Line 2 for this D, and let $D_L \uplus D_R$ be the bipartition of D considered at Line 4 of Algorithm 1. Furthermore, let $C_L \uplus C_R$ be the bipartition of C such that $D_L \subseteq C_L$ and $D_R \subseteq C_R$.

[Lines 5-7] Note that any vertex $u \in N(D_L) \cap N(D_R)$ is not in C. Furthermore, such a vertex $u \notin S$ because $u \in N(C)$ (since $D \subseteq V(C)$). Thus, S is an induced bipartite subgraph of G.

[Lines 8-10] We will now show that any connected component of $G - N[D]$ which is not a module, is a subset of $N(C)$. Once this is proved, S also induces a bipartite subgraph in $G - V(Z)$, where Z is a connected component of $G - N[D]$ that is not a module. Let $X = N(C) \setminus N(D)$ and let $Y = V(G) \setminus N[C]$.

CLAIM 3.1. $E(X, Y) = \emptyset$.

[4] K_3 is a clique on three vertices.

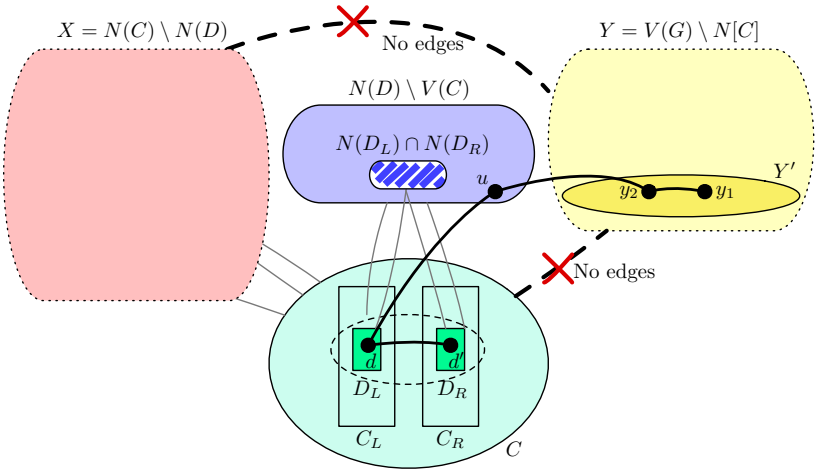

Figure 3: Illustration of various elements in the proof of Claim 3.2.

Proof. Suppose for the sake of contradiction that there exists $y \in Y$ and $x \in X$ such that $(y, x) \in E(G)$ (see Figure 2). Since $x \in X$ and $X = N(C) \setminus N(D)$, there exists $c \in C$ such that $(x, c) \in E(G)$. Also $c \notin D$, as otherwise $x \in N(D)$. Without loss of generality, say $c \in C_L \setminus D$ (the other case is symmetric). Since D is a dominating set of C, there exists $d \in D_R$, such that $(c, d) \in E(G)$. Let $d' \in D_L$, and note that $(d, d') \in E(G)$.

Consider the path $P^* = (y, x, c, d, d')$. We claim P^* is an induced P_5. First observe that all the vertices of P^* except y, x are in C. Also $E(Y, C) = \emptyset$ because $Y \cap N(C) = \emptyset$ by the definition of Y. In particular, $(y, c), (y, d), (y, d') \notin E(G)$. Since $x \in X$ and $X \cap N(D) = \emptyset$, $(x, d), (x, d') \notin E(G)$. Finally, since $c, d' \in C_L$ and C_L is an independent set (because it is one of the parts of the bipartition of C), $(c, d') \notin E(G)$. ∎

From Claim 3.1, for any connected component Z of $G - N[D]$, either $V(Z) \subseteq X$, or $V(Z) \subseteq Y$.

Claim 3.2. *Let Y' be a connected component of $G[Y]$. Then $V(Y')$ is a module in G.*

Proof. First note that, from Claim 3.1, $N_G(Y) \subseteq N(D) \setminus V(C)$. For the sake of contradiction, say $V(Y')$ is not a module, and thus there exists $y_1, y_2 \in V(Y')$ and $u \in N(D) \setminus V(C)$ such that $(y_1, u) \notin E(G)$ and $(y_1, y_2), (y_2, u) \in E(G)$ (see Figure 3). Since $u \in N(D) \setminus V(C)$ and $D \subseteq V(C)$, there exists $d \in D$ such that $(u, d) \in E(G)$. Without loss of generality let $d \in D_L$. Let $d' \in D_R$ (and note that $(d, d') \in E(G)$). Then $(u, d') \notin E(G)$ as otherwise $u \in N(D_L) \cap N(D_R)$, which is a contradiction as such vertices do not exist (from Line 5-7). Then $P^* = (y_1, y_2, u, d, d')$ is an induced P_5 in G, which is a contradiction. ∎

From Claim 3.2 if a connected component Z of $G - N[D]$ is not a module, then $V(Z) \subseteq X$. Since $X \subseteq N(C)$, S is also an induced subgraph of G after the execution of Line 8-10.

[Line 11] Let $R = N(C) \setminus N(D)$, i.e., R are the remaining vertices of $N(C)$ that are not in $N(D)$ (note that R are precisely the vertices of X from the previous discussion that are not deleted at Line 8-10). We now show that there exists a small dominating set of $G[R \cup V(C)]$.

Claim 3.3. *$G[R \cup V(C)]$ has a dominating set $\widetilde{D}$ of size at most 6 such that $D \subseteq \widetilde{D}$ and $|\widetilde{D} \setminus D| \leq 3$.*

Proof. Since $G[R \cup V(C)]$ is a connected and P_5-free graph, from Proposition 2.1, there exists a dominating set of $G[R \cup V(C)]$ which is either a clique or an induced P_3. If D itself is a dominating set of $G[R \cup V(C)]$, then the claim trivially follows. Otherwise, we consider a dominating clique or a dominating induced P_3, D' of $G[R \cup V(C)]$ of minimum possible size. If D' induces a P_3 then, $\widetilde{D} = D' \cup D$ satisfies the requirement of the claim. Now we consider the case when D' is a clique. Using D' we will construct a dominating set of $G[R \cup V(C)]$ with at most 6 vertices, containing the vertices from D. Intuitively speaking, apart from D (whose size is at most 3) we will add vertices from $D' \cap V(C)$ and at most one more vertex. We remark that as D' is a clique and C is a bipartite graph, $|D' \cap V(C)| \leq 2$.

Let $R_1, \cdots, R_p$ be the connected components of $G[R]$. Since D' is a clique, D' intersects at most one R_i, i.e., there exists an $i \in [p]$, such that $D' \cap V(R_j) = \emptyset$, for all $j \in [p] \setminus \{i\}$. Therefore the

vertices of $\cup_{j\in[p]\setminus\{i\}}V(R_i)$ are dominated by the vertices of $D'\cap V(C)$. If $D'\cap V(R_i)=\emptyset$, then notice that $D'\subseteq V(C)$, where $|D'|\leq 2$, and thus $\widetilde{D}=D'\cup D$ satisfies the requirement of the claim. Now suppose that $D'\cap V(R_i)\neq\emptyset$, and consider a vertex $x\in D'\cap V(R_i)$. Recall that $x\in N(C)$ and R_i is a module in G. Thus, there exists a vertex $v'\in V(C)\cap N(x)$, and moreover, we have $V(R_i)\subseteq N(v')$. Note that D dominates each vertex in C, v' dominates each vertex in R_i, and $D'\cap V(C)$ dominates each vertex in $\cup_{j\in[p]\setminus\{i\}}V(R_i)$. Thus, $\widetilde{D}=D\cup\{v'\}\cup(D'\cap V(C))$ dominates each vertex in $G[R\cup V(C)]$ and $|D'\cap V(C)|\leq 2$. This concludes the proof. □

Now consider the execution of the for-loop at Line 11 when $\widetilde{D}=D'\cup D$ is a dominating set for $G[R\cup C]$.

[Line 12] Our objective is to argue that the set $C^*_{D,D'}$ at Line 12 is equal to $N[C]$. To obtain the above, it is enough to show that $N[\widetilde{D}]=N[C]$. To this end, we first obtain that $N[C]\subseteq N[\widetilde{D}]$. Recall that, after removing the vertices from X at Line 8, $N[C]=R\cup V(C)\cup N(D)$. As $\widetilde{D}=D\cup D'$ is a dominating set for $G[R\cup V(C)]$, we have $R\cup V(C)\subseteq N[\widetilde{D}]$. Moreover, as $D\subseteq\widetilde{D}$, we have $N(D)\subseteq N[\widetilde{D}]$. Thus we can conclude that $N[C]\subseteq N[\widetilde{D}]$. We will next argue that $N[\widetilde{D}]\subseteq N[C]$. Recall that from Claim 3.1, $E(R,Y)=\emptyset$. Thus, for any vertex $v\in D'\setminus V(C)$, $N(v)\subseteq N[C]$. In the above, when $v\in D'\setminus V(C)$, without loss of generality we can suppose that $v\in R\subseteq N(C)$, as $\widetilde{D}=D\cup D'$ is a dominating set for $G[R\cup V(C)]$. Thus, for each $v\in D'\setminus V(C)$, $N[v]\subseteq N[C]$. Also, $D\subseteq V(C)$, and thus, $N[D]\subseteq N[C]$. Hence it follows that $N[\widetilde{D}]\subseteq N[C]$. Thus we obtain our claim that, $N[\widetilde{D}]=N[C]$.

[Lines 13-15] Since $C^*_{D,D'}=N[C]$, if there exists $u\in C^*_{D,D'}$ such that u has a neighbour outside $C^*_{D,D'}$, then $u\in N(C)$ and hence $u\notin S$. Thus S induces a bipartite subgraph even in the graph obtained by deleting such vertices. Notice that after execution of these steps, $N[C]$ is a connected component of G, as removing a vertex from $N(C)$ cannot disconnect the graph $N[C]$.

[Lines 16-18] Recall that D is a dominating set of C and $C_L\uplus C_R$ is a bipartition of C, where $C_L\subseteq N(D_R)$ and $C_R\subseteq N(D_L)$. Also, from Line 5-7, $N(D_L)\cap N(D_R)=\emptyset$. Let I_L (resp. I_R) be the maximum weight independent set in $G[N[D_R]]$ (resp. $G[N[D_L]]$) computed at these steps. Then $\mathtt{w}(I_L)\geq\mathtt{w}(C_L)$ (resp. $\mathtt{w}(I_R)\geq\mathtt{w}(C_R)$) because C_L (resp. C_R) is an independent set in $G[N[D_R]]$ (resp. $G[N[D_L]]$). Thus, $\mathtt{w}(C_{D,D'})\geq\mathtt{w}(V(C))$.

Let $S'=(S\setminus V(C))\cup C_{D,D'}$. Then $\mathtt{w}(S')\geq\mathtt{w}(S)$. Also $G[S']$ is bipartite because $G[C_{D,D'}]$ is bipartite and $E(S\setminus V(C),C_{D,D'})=\emptyset$ because $N[C]$ is a connected component of (the reduced graph) G. This concludes the proof. □

The proof of Lemma 3.1 follows from Observation 3.1 and Lemma 3.2.

3.2 Proof of Theorem 1.1

Before giving the formal description of the algorithm for Theorem 1.1, we introduce some notations.

Notations. Consider a P_5-free graph G. For a pair of vertices $x,y\in V(G)$, let $\mathcal{B}^G_{\{x,y\}}$ be the set of induced paths between x and y in G; we will call $\mathcal{B}^G_{\{x,y\}}$ *buckets*. We will skip the superscript G from the above notation when the context is clear. We say that (x,y) is an *active pair* if $\mathcal{B}_{\{x,y\}}\neq\emptyset$. A vertex v *hits* an active pair $(x,y)\in V(G)\times V(G)$, if $N[v]$ intersects at least $|\mathcal{B}_{\{x,y\}}|/10$ many paths from $\mathcal{B}_{\{x,y\}}$, i.e., there is $\mathcal{B}\subseteq\mathcal{B}_{\{x,y\}}$ of size at least $|\mathcal{B}_{\{x,y\}}|/10$, such that for each $P\in\mathcal{B}$, $V(P)\cap N[v]\neq\emptyset$. We will use the below result to choose a vertex for branching.

PROPOSITION 3.2. (LEMMA 2.1, [34]) *Consider any fixed $t\in\mathbb{N}$. Any connected P_t-free graph G has a vertex that hits at least $1/2t^{th}$ fraction of the total number of active pairs in G.*

The Algorithm. The formal description of the algorithm of Theorem 1.1 is given in Algorithm 2. It takes as input a graph G on n vertices and a vertex weight function $\mathtt{w}$, and the goal is to output a set S of vertices of G of maximum weight that induces a bipartite graph. Recall our assumption that for each $u\in V(G)$, $\mathtt{w}(u)\geq 0$, as otherwise, we can remove such vertices. If G is bipartite then the algorithm outputs the entire vertex set in Line 2. Now consider the case when G is disconnected, and let $G_1,G_2,\cdots,G_k$ be the connected components of G such that $|V(G_1)|\geq|V(G_2)|\geq\cdots\geq|V(G_k)|$, and let $i\in[k]$ be the smallest index such that $\sum_{1\leq j\leq i}|V(G_j)|\geq 0.5|V(G)|$. We remark that the operation in Line 5 is only to simplify some of our later arguments, otherwise, we could even directly recurse on each of the connected components. We create the graph H_1 by taking the union of the first i connected components of G, i.e., $H_1=G_1\cup G_2\cup\cdots G_i$. Also at this line we let H_2 to be the graph obtained by taking the union of the remaining components, i.e., $H_2=G_{i+1}\cup G_{i+2}\cup\cdots G_k$. Note that as G is

Algorithm 2 MAX-BIP-SUB

Input: An undirected graph G and a weight function $\mathtt{w} : V(G) \to \mathbb{Q}_{\geq 0}$
Output: $S \subseteq V(G)$ such that $G[S]$ is bipartite and $\mathtt{w}(S)$ is maximized

1: **if** G is bipartite **then**
2: **return** $V(G)$.
3: **end if**
4: **if** G is disconnected **then**
5: Let $G_1, \cdots, G_r$ be the connected components of G, where $|V(G_1)| \geq |V(G_2)| \geq \cdots |V(G_r)|$. Also, let $i \in [r]$ be the smallest index such that $\sum_{1 \leq j \leq i} |V(G_j)| \geq |V(G)|/2$, and $H_1 = G_1 \cup G_2 \cup \cdots G_i$ and $H_2 = G_{i+1} \cup G_{i+2} \cup \cdots G_r$.
6: **return** MAX-BIP-SUB$(H_1, \mathtt{w}) \cup$ MAX-BIP-SUB$(H_2, \mathtt{w})$.
7: **end if**
8: Let v be a vertex that hits the maximum number of active pairs in G.
9: Let $\mathcal{C}$ be the family obtained from Lemma 3.1 on input $(G, v, \mathtt{w})$ (and remove duplicate copies of sets in $\mathcal{C}$, if any).
10: Let $S_0 =$ MAX-BIP-SUB$(G - v, \mathtt{w})$, and for each $C \in \mathcal{C}$, let $S_{C'} = C' \cup$ MAX-BIP-SUB$(G - N[C'], \mathtt{w})$.
11: **return** $\arg\max\{\mathtt{w}(S_0), \max_{C' \in \mathcal{C}} \mathtt{w}(S_{C'})\}$.

disconnected and $|V(G_1)| \geq |V(G_2)| \geq \cdots \geq |V(G_k)|$, it must be the case that $V(H_1), V(H_2) \neq \emptyset$. As an optimal solution to G is simply a disjoint union of the optimal solutions for H_1 and H_2, the algorithm returns an optimal solution at Line 6.

If none of the above holds, then G is connected and non-bipartite, and the algorithm picks a vertex v that hits the most number of active pairs in G in Line 8. Note that, from Proposition 3.2, such a vertex hits at least $1/10^{th}$ of the active pairs. Moreover, such a vertex can be computed in polynomial time, as the number of induced paths between any pair of vertices in a P_5-free graph is at most n^4. The algorithm now invokes Lemma 3.1 for the graph G, vertex v, and weight function $\mathtt{w}$, and computes the family $\mathcal{C}$ using the lemma, in Line 9. Now it then executes the following branching rule in Line 10.

(v is not in our optimal solution.) The algorithm recursively solves the instance $(G - v, \mathtt{w})$. We call this the *failure* branch.

(v is in our optimal solution.) For each $C' \in \mathcal{C}$, (recursively) solve the instance $(G - N[C'], \mathtt{w})$. We call these the *successful* branches.

Finally in Line 11, it returns the best among the above solutions.

Note that, at each recursive call, the number of vertices in the graph strictly reduces, thus the algorithm necessarily terminates. In the next lemma, we prove the correctness of our algorithm.

LEMMA 3.3. *Algorithm 2 on input $(G, \mathtt{w})$, returns a set S such that $G[S]$ is bipartite and $\mathtt{w}(S)$ is maximized.*

Proof. We will prove the lemma by induction on the number of vertices in the graph. The base case (Lines 1) is easy to verify: when the graph is bipartite, the optimal solution is the whole vertex set (recall our positive weight assumption). Note that when $|V(G)| \leq 2$, then trivially our base case occurs.

For induction hypothesis we suppose that for each $n' < n = |V(G)|$, where $n' \geq 2$, for every graph G' on n' vertices, the algorithm is correct. We will now prove the statement for a graph G on n vertices. We follow the notations from the description of the algorithm, without defining them again.

If G is disconnected, then notice that an optimal solution can be obtained by taking the union of optimal solutions for H_1 and H_2. Thus, the correctness for this case follows from the induction hypothesis and Line 6.

Now we suppose that G is connected, and we let v be the vertex selected by Algorithm 2 at Line 8. Let S be an optimal solution for $(G, \mathtt{w})$. Note that by the induction hypothesis, at Line 11 we necessarily return a set that induces a bipartite graph. If $v \notin S$, then notice that S is also an optimal solution for $(G - v, \mathtt{w})$, and thus at Line 11, the algorithm must have returned an optimal solution.

Now we consider the case when $v \in S$, and let C be the connected component of $G[S]$ containing v. Note that $N(C) \cap S = \emptyset$. Recall that $\mathcal{C}$ is the family obtained from Lemma 3.1 in Line 9. From the lemma there exists $C' \in \mathcal{C}$, such that $S' = (S \setminus V(C)) \cup C'$ is also an optimal solution of G and $N(C') \cap S' = \emptyset$. Then, $S' \setminus C'$ is also an optimal solution of $G - N[C']$. Therefore MAX-BIP-SUB$(G - N[C'], \mathtt{w})$ outputs a solution of weight at least $\mathtt{w}(S' \setminus C')$. Thus, MAX-BIP-SUB$(G - N[C'], \mathtt{w}) \cup C'$ is a solution of weight

at least $\mathtt{w}((S' \setminus C') \cup C') = \mathtt{w}(S') \geq \mathtt{w}(S)$ (from Lemma 3.1). Thus at Line 11 the algorithm returns an optimal solution for $(G, \mathtt{w})$. ◻

We will now move towards the runtime analysis of Algorithm 2. To this end, we will bound the size of its recursion tree. The crux of the analysis is that whenever we create a new branch, at all but at most one of the branches we simplify the graph substantially, either in terms of reduction on the number of vertices by a fraction, or by exploiting Proposition 3.2 as follows. Note the proposition guarantees the existence of a vertex v such that $N[v]$ intersects at least $1/10^{th}$ fraction of the total number of induced paths of at least $1/10^{th}$ fraction of the total number of active pairs of G; roughly speaking, due to the above, we reduce the total number of paths between pairs of vertices by a fraction.

To avoid confusion, we will refer to the vertices of the recursion tree as nodes. Note that the recursion tree of the algorithm on input $(G, \mathtt{w})$ is the tree $\mathcal{T}$ where the root node r is labelled by the input instance $(G, \mathtt{w})$. For each recursive call that the algorithm makes, in the recursion tree $\mathcal{T}$, there is a child mode attached to the node from which the recursive call was made. This child node is labelled with the instance at the corresponding recursive call. For each node of the recursion tree, except the root node, we call it a *failure node* if it is created by a failure branch, and we call it a *successful node* if it is created by a successful branch. For the two recursive calls made at Line 6, we call the instance created for the smaller graph, H_2 a *light* node, and the one created for H_1 a *heavy* node.

For simplifying our arguments, we remove those nodes from $\mathcal{T}$ that are (both) not successful and where the base case is applicable (we absorb the runtime incurred for such a node at its parent node). For a node $q \in V(\mathcal{T})$, let G_q be its input graph (along with the weight function $\mathtt{w}$). Towards analysing the size of $\mathcal{T}$, we first show in Lemma 3.4 that, in any root-to-leaf path in $\mathcal{T}$, there are at most $O(\log^2 n)$ successful nodes.

LEMMA 3.4. *Any root-to-leaf path in the recursion tree $\mathcal{T}$ of Algorithm 2 has at most $O(\log^2 n)$ successful nodes.*

Proof. For simplicity we assume that the input graph G is connected, i.e., the graph $G_r = G$, where r is the root of $\mathcal{T}$ (as otherwise we can do the same analysis for each of the connected components). Recall for $x, y \in V(G)$, $\mathcal{B}^G_{\{x,y\}}$ is the set of all induced paths between x and y in G. As G is P_5-free, we have $|\mathcal{B}^G_{\{x,y\}}| \leq n^4$. We say that an active pair (x, y) is *hit* at a successful node, if for some successful node $q \in V(\mathcal{T})$ with parent $p \in V(\mathcal{T})$, at the execution for p, we selected a vertex v at Line 8 such that: (x, y) is an active pair in G_p and v hits (x, y) in G_p. For an $i \in \mathbb{N}$, (x, y) is *hit i times* on a fixed root-to-leaf path of the recursion tree, if there exists i successful nodes on this root-to-leaf path, at which (x, y) is hit.

CLAIM 3.4. *There is a (computable) constant α, such that any active pair (x, y) in G is hit at most $\alpha \log |\mathcal{B}^G_{\{x,y\}}|$ times on a fixed root-to-leaf path $\mathcal{T}$.*

Proof. Each time an active pair is hit, the number of paths between them in the graphs associated with the nodes at the subtrees decreases at least by a factor $1/10^{th}$, hence the claim follows. ◻

For a graph H on at most n vertices, define the measure $\mu(H)$ as follows, where α is the constant from Claim 3.4.

$$\mu(H) := \sum_{(x,y) \in \binom{V(H)}{2}} \alpha \log |\mathcal{B}^H_{\{x,y\}}| \leq \binom{|V(H)|}{2} \cdot 4\alpha \log n$$

Note that $\mu(G)$ is an upper bound on the total number of times active pairs in G can be hit on a root-to-leaf path of $\mathcal{T}$. We will now argue that a root-to-leaf path in $\mathcal{T}$ can have at most $(40\alpha \log |V(G)|)^2$ successful nodes. Towards a contraction, suppose the above statement is not true, and let $q_1, q_2, \cdots, q_k$ be the successful nodes at some root-to-leaf path in $\mathcal{T}$, ordered form root to leaf, where $k \geq (40\alpha \log |V(G)|)^2 + 1$. For each $i \in [k]$, let p_i be the parent of q_i in $\mathcal{T}$ and v_i be the vertex considered at Line 8 during the execution of p_i (along with a family computed at Line 9, to create the node q_i). To simplify our arguments, we let $G_{-1} = G_0 = G$ and for each $i \in \{1, 2, \cdots, k\}$, $G_i = G_{q_i}$. We will first show (by induction) that for each $i \in \{0, 1, 2, \cdots, k\}$, $\mu(G_i) \leq \mu(G_{i-1}) \left(1 - \frac{1}{40\alpha \log |V(G)|}\right)^i$. Clearly, for $i = 0$, the required inequality holds.

We now suppose for some $0 \leq j < k$, the inequality is satisfied for all j', where $0 \leq j' \leq j$, and prove the statement for $i = j + 1$.

Since v_i hits at least $1/10^{th}$ of the active pairs in G_{p_i} (note G_{p_i} must be connected), i.e., $N_{G_{p_i}}[v_i]$ intersects at least $1/10$ fraction of the paths in the buckets of at least $1/10$ active pairs of G_{p_i}, and $G_i = G_{q_i}$ does not contain $N_{G_{p_i}}[v_i]$, $\mu(G_i)$ drops by at least $\frac{1}{10}\binom{|V(G_{p_i})|}{2}$. That is,

$$\begin{aligned}
\mu(G_i) &\leq \mu(G_{p_i}) - \frac{1}{10}\binom{|V(G_{p_i})|}{2} \\
&\leq \mu(G_{p_i}) - \frac{\mu(G_{p_i})}{40\alpha \log |V(G_{p_i})|} && \text{:from the upper bound on } \mu(G_{p_i}) \\
&\leq \mu(G_{p_i})\left(1 - \frac{1}{40\alpha \log |V(G)|}\right) && \text{:because } |V(G)| \geq |V(G_{p_i})| \\
&\leq \mu(G_0)\left(1 - \frac{1}{40\alpha \log |V(G_0)|}\right)^i && \text{:by induction and the fact that } \mu(G_{p_i}) \leq \mu(G_{i-1})
\end{aligned}$$

Thus we have established the required inequality by induction. Now, as $k \geq (40\alpha \log |V(G)|)^2 + 1$ (recall $G_0 = G$), then

$$\begin{aligned}
\mu(G_{k-1}) &\leq \mu(G)\left(1 - \frac{1}{40\alpha \log |V(G)|}\right)^{(40\alpha \log |V(G)|)^2} \\
&\leq \mu(G)\frac{1}{e^{40\alpha \log |V(G)|}} && \text{:}(1-x) \leq e^{-x}, \text{ for any } x \\
&= \frac{\mu(G)}{|V(G)|^{40\alpha}} \\
&\leq \frac{4\alpha \cdot |V(G)|^2 \cdot \log |V(G)|}{|V(G)|^{40\alpha}} && \text{:from the upper bound on } \mu(G) \\
&\leq 0 && .
\end{aligned}$$

Since $\mu(G_{k-1}) \leq \mu(G_k)$, this contradicts that G_k is associated with a successful node. Thus the number of successful nodes in any root-to-leaf path of $\mathcal{T}$ is at most $(40\alpha \log |V(G)|)^2$. ∎

The next simple observation bounds the number of light nodes on a root-to-leaf path in $\mathcal{T}$.

OBSERVATION 3.2. *Any root-to-leaf path in the recursion tree $\mathcal{T}$ of Algorithm 2 has at most $O(\log n)$ light nodes.*

Proof. The proof follows from the fact that, each time such a call is made, the number of vertices in the graphs drops by at least a factor of $1/2$. ∎

Finally we now bound the running time of our algorithm.

LEMMA 3.5. *The running time of Algorithm 2 is $n^{O(\log^2 n)}$.*

Proof. We remark that the depth of $\mathcal{T}$ is bounded by n, as the number of vertices strictly decreases with each recursive call. Thus, to bound the running time of algorithm it is enough to bound the number of leaves in $\mathcal{T}$ by $n^{O(\log^2 n)}$.

To analyse the number of leaves in $\mathcal{T}$, let $c = O(n^6)$ be the maximum cardinality of the family that can be obtained from Lemma 3.1. We remark that for each such family that is computed, we can obtain an ordering of the sets in a family by taking the natural lexicographical ordering (by fixing an arbitrary ordering of vertices to begin with), and we will be working with such orderings. Recall that r is the root node of $\mathcal{T}$. We will now define a labelling function $\ell : V(\mathcal{T}) \setminus \{r\} \to \{0, 1, \cdots, c\}$ as follows. Consider a non-leaf node $p \in V(\mathcal{T})$.

1. If G_p is disconnected, p has exactly two children, say q_1 and q_2, where $|V(G_{q_1})| \geq |V(G_{q_2})|$. We set $\ell(q_1) = 0$ and $\ell(q_2) = 1$.
2. Now consider the case when G_p is connected. At the run of the algorithm for node p we must have executed Line 8 and 9, and we let v be the vertex selected and $\mathcal{C}$ be the family computed at these lines for the execution of p, respectively. If $G_q = G_p - v$, then we set $\ell(q) = 0$. Otherwise, let $i \in \{1, \cdots, c\}$ be the index such that $G_q = G_p - N_{G_p}[C_i]$, where C_i is the i^{th} set of $\mathcal{C}$. Note that i is unique due to duplicate removal operation (see Line 9). We set $\ell(q) = i$.

Using the above labelling we now construct a string associated with each leaf of $\mathcal{T}$ of length exactly $n-1$. Let ϕ be a new symbol. Consider a leaf q of $\mathcal{T}$, and let $P = (r = w_1, w_2, \cdots, w_s = q)$ be the path from r to q in $\mathcal{T}$. Note that $s - 1 \leq n - 1$. We construct the string ξ^q of length exactly $n-1$ as follows. For $i \in \{1, 2, \cdots, s-1\}$, we set the i^{th} character of ξ^q, $\xi^q_i = \ell(w_{i+1})$ (recall we do not have a label for r). Now we set the remaining character of ξ^q as ϕ, i.e., for each $i \in \{1, 2, \cdots, n-1\} \setminus \{1, 2, \cdots, s-1\}$, set $\xi^q_i = \phi$. In the following claim we prove that strings associates with different leaves are distinct.

Claim 3.5. *For distinct leaves q_1, q_2 of $\mathcal{T}$, we have $\xi^{q_1} \neq \xi^{q_2}$.*

Proof. Let p be the least common ancestor of q_1 and q_2 in $\mathcal{T}$. Furthermore, let q_1' and q_2' be the vertices immediately following p in the paths from p to q_1 and q_2 in $\mathcal{T}$, respectively. Let i be the distance of q_1' from r (counting number of edges in the path) in $\mathcal{T}$. Note that i is also the distance of q_2' from r. We will argue that $\xi^{q_1}_i \neq \xi^{q_2}_i$ by considering the following cases.

1. Consider the case when G_p is a disconnected graph. Then, p has exactly two children, namely q_1' and q_2', and by construction we have $\ell(q_1') \neq \ell(q_2')$. Thus, $\xi^{q_1}_i \neq \xi^{q_2}_i$.
2. When G_p is connected, by construction, $\ell(q_1')$ and $\ell(q_2')$ are distinct indices from $\{0, 1, \cdots, c\}$. Thus we can conclude that $\xi^{q_1}_i \neq \xi^{q_2}_i$.

□

Due to the above result, to bound the number of leaves, it is enough to bound the number of such strings that can be generated from a recursion tree of the algorithm. For a leaf $q \in V(\mathcal{T})$, the number of entries in ξ^q that can be different from 0 is bounded by $O(\log^2 n)$ (see Lemma 3.4 and Observation 3.2).

For each string of length exactly $n-1$ that could be generated, there are at most $n-1$ choices of the position from which ϕ starts (and continues till the end). There are at most $n-1$ choose $O(\log^2 n)$ choices of positions on which entries from $\{1, 2, \cdots, c\}$ can be filled, and $c^{O(\log^2 n)}$ ways of filling a fixed set of such positions with non-zero entries. Recall that $c = O(n^6)$. Thus overall, the number of strings that can be generated from the recursion tree is bounded by $n \cdot c^{O(\log^2 n)} \in n^{O(\log^2 n)}$.

Finally, apart from the recursive steps, our algorithm only spends polynomial amount of time, and the number of nodes in the recursion tree can be bounded by $n^{O(\log^2 n)}$ using the bound on the number of string that we obtained, Claim 3.5, and the fact that the depth of the recursion tree is bounded by n. This concludes the proof. □

Algorithm 2 together with Lemma 3.3 and 3.5 gives a proof of Theorem 1.1.

4 Conclusion

We gave an $n^{O(\log^2 n)}$ time algorithm for OCT (or Max-Bip-Sub) on P_5-free graphs. Several interesting problems in this direction remain open.

1. Does OCT have a polynomial-time algorithm on P_5-free graphs?
2. The algorithms for Independent Set on P_k-free graphs [34] also work for counting the number of independent sets of a given size within the same time bound. Because of the "greedy choice" implicit in the statement of Lemma 3.1 our algorithm does not work for counting the number of induced bipartite subgraphs of a given size. Does a polynomial (or quasi-polynomial) time algorithm exist for this problem in P_5-free graphs?
3. For every fixed positive integer ℓ we can determine whether G is ℓ-colorable in polynomial time on P_5-free graphs [18]. Therefore, in light of Theorem 1.1 it makes sense to ask whether for every positive integer ℓ there exists a polynomial or quasi-polynomial time algorithm that takes as input a graph G and outputs a maximum sized set S such that $G[S]$ is ℓ-colorable.
4. Our result completes the classification of all *connected* graphs H into ones such that OCT on H-free graphs is quasi-polynomial time solvable or NP-complete. Such a classification for all graphs H (connected or not) remains open. Even more generally one could hope for a complete classification of the complexity of OCT on $\mathcal{F}$-free graphs for every finite set $\mathcal{F}$.

References

[1] T. Akiba and Y. Iwata, *Branch-and-reduce exponential/fpt algorithms in practice: A case study of vertex cover*, Theor. Comput. Sci., 609 (2016), pp. 211–225.

[2] G. Bacsó and Z. Tuza, *Dominating cliques in P_5-free graphs*, Periodica Mathematica Hungarica, 21 (1990), pp. 303–308.
[3] M. Bonamy, K. K. Dabrowski, C. Feghali, M. Johnson, and D. Paulusma, *Independent feedback vertex set for p_5-free graphs*, Algorithmica, 81 (2019), pp. 1342–1369.
[4] A. Brandstädt and D. Kratsch, *On the restriction of some NP-complete graph problems to permutation graphs*, in Fundamentals of Computation Theory, L. Budach, ed., Berlin, Heidelberg, 1985, Springer Berlin Heidelberg, pp. 53–62.
[5] A. Brandstädt, J. P. Spinrad, et al., *Graph classes: a survey*, no. 3, Siam, 1999.
[6] B.-M. Bui-Xuan, J. A. Telle, and M. Vatshelle, *Fast dynamic programming for locally checkable vertex subset and vertex partitioning problems*, Theoretical Computer Science, 511 (2013), pp. 66–76. Exact and Parameterized Computation.
[7] N. Chiarelli, T. R. Hartinger, M. Johnson, M. Milanič, and D. Paulusma, *Minimum connected transversals in graphs: New hardness results and tractable cases using the price of connectivity*, Theoretical Computer Science, 705 (2018), pp. 75–83.
[8] M. Chudnovsky, J. King, M. Pilipczuk, P. Rzazewski, and S. Spirkl, *Finding large H-colorable subgraphs in hereditary graph classes*, SIAM J. Discret. Math., 35 (2021), pp. 2357–2386.
[9] M. Chudnovsky, D. Paulusma, and O. Schaudt, *Graph Colouring: from Structure to Algorithms (Dagstuhl Seminar 19271)*, Dagstuhl Reports, 9 (2019), pp. 125–142.
[10] K. K. Dabrowski, C. Feghali, M. Johnson, G. Paesani, D. Paulusma, and P. Rzążewski, *On cycle transversals and their connected variants in the absence of a small linear forest*, Algorithmica, 82 (2020), pp. 2841–2866.
[11] H. De Ridder et al., *Information system on graph classes and their inclusions)*, www.graphclasses.org, (2016).
[12] R. Diestel, *Graph Theory, 4th Edition*, vol. 173 of Graduate texts in mathematics, Springer, 2012.
[13] S. Fiorini, N. Hardy, B. A. Reed, and A. Vetta, *Approximate min-max relations for odd cycles in planar graphs*, Math. Program., 110 (2007), pp. 71–91.
[14] T. Gallai, *Transitiv orientierbare graphen*, Acta Mathematica Hungarica, 18 (1967), pp. 25–66.
[15] P. Gartland and D. Lokshtanov, *Independent set on P_k-free graphs in quasi-polynomial time*, in 2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS), 2020, pp. 613–624.
[16] P. Gartland, D. Lokshtanov, M. Pilipczuk, M. Pilipczuk, and P. Rzążewski, *Finding large induced sparse subgraphs in $C_{>t}$-free graphs in quasipolynomial time*, in Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2021, Association for Computing Machinery, 2021, p. 330–341.
[17] M. X. Goemans and D. P. Williamson, *Primal-dual approximation algorithms for feedback problems in planar graphs*, Comb., 18 (1998), pp. 37–59.
[18] C. T. Hoàng, M. Kaminski, V. V. Lozin, J. Sawada, and X. Shu, *Deciding* k*-colorability of P_5-free graphs in polynomial time*, Algorithmica, 57 (2010), pp. 74–81.
[19] F. Hüffner, *Algorithm engineering for optimal graph bipartization*, J. Graph Algorithms Appl., 13 (2009), pp. 77–98.
[20] Y. Iwata, K. Oka, and Y. Yoshida, *Linear-time FPT algorithms via network flow*, 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. 1749–1761.
[21] H. Jacob, T. Bellitto, O. Defrain, and M. Pilipczuk, *Close relatives (of feedback vertex set), revisited*, in 16th International Symposium on Parameterized and Exact Computation, IPEC 2021, September 8-10, 2021, Lisbon, Portugal, vol. 214 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, pp. 21:1–21:15.
[22] B. M. P. Jansen and S. Kratsch, *On polynomial kernels for structural parameterizations of odd cycle transversal*, in Parameterized and Exact Computation - 6th International Symposium, IPEC 2011, Saarbrücken, Germany, September 6-8, 2011. Revised Selected Papers, D. Marx and P. Rossmanith, eds., vol. 7112 of Lecture Notes in Computer Science, Springer, 2011, pp. 132–144.
[23] K. Kawarabayashi and B. A. Reed, *An (almost) linear time algorithm for odd cyles transversal*, in Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010, M. Charikar, ed., SIAM, 2010, pp. 365–378.
[24] S. Khot and V. Raman, *Parameterized complexity of finding subgraphs with hereditary properties*, Theor. Comput. Sci., 289 (2002), pp. 997–1008.
[25] S. Kolay, P. Misra, M. S. Ramanujan, and S. Saurabh, *Faster graph bipartization*, J. Comput. Syst. Sci., 109 (2020), pp. 45–55.
[26] D. Král', J. Sereni, and L. Stacho, *Min-max relations for odd cycles in planar graphs*, SIAM J. Discret. Math., 26 (2012), pp. 884–895.
[27] S. Kratsch and M. Wahlström, *Compression via matroids: A randomized polynomial kernel for odd cycle transversal*, ACM Trans. Algorithms, 10 (2014), pp. 20:1–20:15.
[28] J. M. Lewis and M. Yannakakis, *The node-deletion problem for hereditary properties is NP-complete*,

Journal of Computer and System Sciences, 20 (1980), pp. 219–230.
[29] D. Lokshtanov, N. S. Narayanaswamy, V. Raman, M. S. Ramanujan, and S. Saurabh, *Faster parameterized algorithms using linear programming*, ACM Trans. Algorithms, 11 (2014), pp. 15:1–15:31.
[30] D. Lokshtanov, S. Saurabh, and S. Sikdar, *Simpler parameterized algorithm for OCT*, in Combinatorial Algorithms, 20th International Workshop, IWOCA 2009, Hradec nad Moravicí, Czech Republic, June 28-July 2, 2009, Revised Selected Papers, vol. 5874 of Lecture Notes in Computer Science, Springer, 2009, pp. 380–384.
[31] D. Lokshtanov, S. Saurabh, and M. Wahlström, *Subexponential parameterized odd cycle transversal on planar graphs*, in IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2012, December 15-17, 2012, Hyderabad, India, vol. 18 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012, pp. 424–434.
[32] D. Lokshtanov, 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.
[33] K. Okrasa and P. Rzążewski, *Subexponential algorithms for variants of the homomorphism problem in string graphs*, Journal of Computer and System Sciences, 109 (2020), pp. 126–144.
[34] M. Pilipczuk, M. Pilipczuk, and P. Rzążewski, *Quasi-polynomial-time algorithm for independent set in P_t-free graphs via shrinking the space of induced paths*, in 4th Symposium on Simplicity in Algorithms, SOSA 2021, Virtual Conference, January 11-12, 2021, H. V. Le and V. King, eds., SIAM, 2021, pp. 204–209.
[35] V. Raman, S. Saurabh, and S. Sikdar, *Efficient exact algorithms through enumerating maximal independent sets and other techniques*, Theory Comput. Syst., 41 (2007), pp. 563–587.
[36] B. A. Reed, K. Smith, and A. Vetta, *Finding odd cycle transversals*, Oper. Res. Lett., 32 (2004), pp. 299–301.
[37] Y. Takazawa and S. Mizuno, *On a reduction of the weighted induced bipartite subgraph problem to the weighted independent set problem*, CoRR, abs/1807.10277 (2018).