# Deleting to Structured Trees

Pratyush Dayal and Neeldhara Misra[(✉)]

Indian Institute of Technology, Gandhinagar, India
{pdayal,neeldhara.m}@iitgn.ac.in
http://www.iitgn.ac.in

**Abstract.** We consider a natural variant of the well-known FEEDBACK VERTEX SET problem, namely the problem of deleting a small subset of vertices or edges to a full binary tree. This version of the problem is motivated by real-world scenarios that are best modeled by full binary trees. We establish that both the edge and vertex deletion variants of the problem are NP-hard. This stands in contrast to the fact that deleting edges to obtain a forest or a tree is equivalent to the problem of finding a minimum cost spanning tree, which can be solved in polynomial time. We also establish that both problems are FPT by the standard parameter.

**Keywords:** Full Binary Trees · Feedback Vertex Set · NP-hardness

## 1 Introduction

The FEEDBACK VERTEX SET (FVS) problem asks for a smallest subset S of vertices in an undirected graph G to be removed such that the graph, G \ S, becomes acyclic. This problem was one of the first problems shown to be NP-complete [6], and has applications to problems that arise in several areas. These applications include, but are not limited to, operating systems, database systems and VLSI chip design. Consequently, the FVS problem has been widely studied in the context of exact, parameterized and approximation algorithms.

Several variations of the FVS theme have also emerged over the years. For instance, one line of work considers the task of "deleting to specialized forests", such as forests of pathwidth one [3,8] or forests whose connected components are stars of bounded degree [5]. In this case, the forests of pathwidth one turn out to be graphs whose connected components are caterpillars.

Meanwhile, another line of work is the TREE DELETION SET (TDS) problem that considers the issue of the connectivity of the structure after the solution has been deleted. In particular, the TDS problem asks for a smallest subset of vertices S such that G \ S is a tree [7,10]. We remark that the NP-completeness of this TDS problem follows from a general result of Yannakakis [12]. To state this result, recall that a *property* π is a class of graphs, and we will say that π

is satisfied by, or is true for, a graph $G$ if $G \in \pi$. A property is said to be *non-trivial* if it is satisfied for at least one graph and false for at least one graph; it is *interesting* if the property is true for arbitrarily large graphs and is *hereditary on induced subgraphs* if the deletion of any node from a graph in $\pi$ always results in a graph that is in $\pi$. The result in question states that the problem of finding a maximum connected subgraph satisfying a property $\pi$ is NP-hard for any non-trivial and interesting property that is hereditary on induced subgraphs.

In this work, we pose a variation of FVS that is in the spirit of a combination of the variations that we have alluded to; here, however, we are looking for a connected object with additional structure. Specifically, we consider the problem of deleting to a full binary tree. We recall that a full binary tree is a tree that has exactly one vertex of degree two and no vertex of degree more than three. Consider the problem of optimally deleting to a full binary tree, posed in the language of the theorem of Yannakakis [12] stated above, which is to find a maximum connected subgraph that satisfies a certain property. Observe that the property in question could be defined as the property of not having cycles, having exactly one vertex of degree two and no vertex of degree more than three. Note that this property is not hereditary on induced subgraphs: in particular, the deletion of a leaf from a graph that has the property will lead to a violation of the property. In our first result, we explicitly establish the NP-hardness of this problem by reducing from a variant of the INDEPENDENT SET problem.

In addition, we also consider the edge deletion version of the question above. Recall that for a given connected graph on $n$ vertices and $m$ edges, deleting a smallest subset of edges to obtain a tree is straightforward: it is clear that we have to remove every edge that does not belong to a spanning tree, so the size of the solution is always $(m - (n - 1))$. In fact, this problem can be solved in polynomial time even when the edges have weights and we seek a subset of edges of smallest total weight, whose removal results in a tree. It is straightforward to see that any such solution is the complement of a maximum spanning tree and thus, can be found in polynomial time.

In a somewhat surprising twist, we show that the problem of deleting a subset of edges of minimum total weight to obtain a full binary tree is, in fact, NP-complete. To establish some intuition for why this is true, we briefly sketch a simple reduction from the problem of EXACT COVER BY 3-SETS to the closely related problem of deleting edges to obtain a full ternary tree.

A ternary tree is a tree where every non-leaf vertex, except the root, is exactly of degree four, while the root has degree three. Let $\mathcal{F} := \{S_1, \ldots, S_p\}$ be a family of sets of size three over the universe $\mathcal{U} := \{x_1, \ldots, x_q\}$. The goal is to find a subfamily of disjoint sets whose union is $\mathcal{U}$. We create a full ternary tree $T$ with $p$ leaves labeled $\{t_1, \ldots, t_p\}$, and set the weight of the edges of $T$ to $B$, a quantity that we will specify later. Then, we introduce for every element $x_i$ in the universe a vertex $v_i$ that is adjacent to $t_j$, if and only if $x_i \in S_j$. The edges between the leaves of $T$ and the vertices corresponding to the elements of $\mathcal{U}$ have unit weights. We also set $B = 3p - q + 1$. It is easy to verify that this graph has a solution of cost $3p - q$ if and only if the system $\mathcal{U}$ has an exact cover, as desired.

It turns out that establishing the hardness of the problem of deleting to full binary trees is non-trivial, and this is one of our main contributions. We reduce from a fairly restrained version of the SATISFIABILITY problem, the hardness of which is inspired by a reduction in [1] and is of independent interest. We note that we deal with the weighted versions of the problems considered, and we also fix a choice of root vertex as part of the input. Finally, we also note that both the problems we propose above are fixed-parameter tractable, when parameterized by the solution size. To this end, we describe a natural branching algorithm and remark that most preprocessing rules that work in a straightforward manner for FEEDBACK VERTEX SET fail when applied as-is to our problem. In particular, it is not trivial to delete degree-one vertices or short-circuit vertices of degree two.

We believe that the problem we propose and the study we undertake has considerable practical motivation. One of the applications of FVS and related problems is to understand noisy datasets. For example, let us say that we expect the data to have a certain structure, but errors in the measurement cause the data at hand not to have the properties expected by said structures. In this context, one approach will be to identify and eliminate the noise - for acyclic structures, that could translate identifying a FVS of small cost. Therefore, for scenarios where the data corresponds to full binary trees, for instances in the case of phylogenetic trees, the problem we present here will be a more relevant model.

## 2    Preliminaries

We follow standard notation and terminology from parameterized complexity [2] and graph theory [4]; we use $[n]$ to denote the set $\{1, 2, \ldots, n\}$. We now turn to the definitions of the problems that we consider.

---

FULL BINARY TREE DELETION BY VERTICES (FBT-DV)
**Input:** A graph $G = (V, E)$, a vertex $r \in V$, vertex weights $w : V \to \mathbb{R}^+$, and $k \in \mathbb{Z}^+$.
**Question:** Does $G$ have a subset $S \subseteq V$ of total weight at most $k$ such that $G \setminus S$ is a full binary tree rooted at $r$?

---

The problems of FULL BINARY TREE DELETION BY EDGES (FBT-DE), COMPLETE BINARY TREE DELETION (by edges or vertices) and BINARY TREE DELETION (by edges or vertices) can be defined analogously. Our focus in this contribution will be mainly on FBT-DV and FBT-DE.

The MULTI-COLORED INDEPENDENT SET problem is the following.

---

MULTI-COLORED INDEPENDENT SET (MCIS)
**Input:** A graph $G = (V, E)$ and a partition of $V = (V_1, \ldots, V_k)$ into $k$ parts.
**Parameter:** $k$
**Question:** Does there exist a subset $S \subseteq V$ such that $S$ is independent in $G$ and for every $i \in [k]$, $|V_i \cap S| = 1$?

---

## 3    NP-hardness

In this section, we establish that the problems of deleting to full binary trees via vertices or edges are NP-complete. We first describe the hardness for the vertex-deletion variant.

**Theorem 1.** FBT-DV *is* NP-*complete.*

*Proof.* We reduce from MULTI-COLORED INDEPENDENT SET [2, Corollary 13.8]. Let $(G, k)$ be an instance of MCIS where $G = (V, E)$ and further, let $V = (V_1, \ldots, V_k)$ denote the partition of the vertex set $V$. We assume, without loss of generality, that $|V_i| = n$ for all $i \in [k]$. Specifically, we denote the vertices of $V_i$ by $\{v_1^i, \ldots, v_n^i\}$. We are now ready to describe the reduced instance of FBT-DV, which we denote by $(H, \ell)$.

To begin with, let $H$ be a complete binary tree with $2nk$ leaves, where a complete binary tree is a full binary tree with $2^w$ vertices at distance $w$ from the root for all $w \in [d-1]$, where $d$ the distance between the root and the leaf furthest away from the root. We denote these leaf vertices as:

$$\left( \cup_{1 \leqslant i \leqslant k} \{a_1^i, \ldots, a_n^i\} \right) \bigcup \left( \cup_{1 \leqslant i \leqslant k} \{b_1^i, \ldots, b_n^i\} \right),$$

where, for all $i \in [k]$ and $j \in [n]$, $a_j^i$ and $b_j^i$ are siblings, and their parent is denoted by $p_j^i$. We refer to this as the *backbone*, to which we will now add more vertices and edges.

For each $i \in [k]$ and $j \in [n]$, we now introduce a third child of $p_j^i$, which we denote by $u_j^i$. We refer to the $u$'s as the *essential* vertices, while its siblings (the $a$'s and the $b$'s) are called *partners*. For all $1 \leqslant i \leqslant k$, we also introduce two *guards*, denoted by $x_i$ and $y_i$, which are adjacent to all the essential vertices of type $i$, that is, all $u_j^i$ for $j \in [n]$. Finally, we ensure that the graph induced on the essential vertices is a copy of $G$, more precisely, we have:

$$(u_i^r, u_j^s) \in E(H) \text{ if and only if } (v_i^r, v_j^s) \in E(G) \text{ for all } i \in [k] \text{ and } j \in [n].$$

We set $\ell = nk$. This completes the construction. We now turn to a proof of equivalence.

*The Forward Direction.* If $S \subseteq V$ is a multi-colored independent set, then consider the subset $S^*$ given by all the essential vertices corresponding to $V \setminus S$, along with the partner vertices $a_j^i$ for each $(i, j)$, for which $v_j^i$ belongs to $S$. It is easy to verify that the proposed set consists of $nk$ vertices. Observe that the deletion of $S^*$ leaves us with a full binary tree where each $p_j^i$ now has two children - either two partner vertices (for vertices not in $S$) or one essential vertex along with one partner vertex (for vertices in $S$). Further, each pair of guards of type $i$ now has an unique parent, which is the essential vertex corresponding to the vertex given by $S \cap V_i$. The essential vertices have degree exactly three because their only other neighbors in $H$ were essential vertices corresponding to neighbors in $G$, but the presence of any such vertex in $H \setminus S^*$ will contradict the fact that $S$ induces an independent set in $G$. This concludes the argument in the forward direction.

*The Reverse Direction.* Let $S^*$ be a subset of $V(H)$ such that $H \backslash S^*$ is a full binary tree. We claim that $S^* \cap \{p_j^i \mid 1 \leqslant i \leqslant k \text{ and } 1 \leqslant j \leqslant n\} = \emptyset$, since the deletion of any parent of a partner vertex will result in the corresponding partner vertex becoming isolated in $H \backslash S^*$— which leads to a contradiction when we account for the budget constraint on $S^*$. Since all the parents of partner vertices survive and have degree four in $H$, it follows that at least one of its neighbors must belong to $S^*$. In particular, we claim that for every $i \in [k]$ and $j \in [n]$, $S^* \cap \{u_j^i, a_j^i, b_j^i\} \neq \emptyset$. Indeed, if this is not the case, then $S^*$ contains the parent of $p_i^j$, and it is easy to verify that this leads to a situation where either $H \setminus S^*$ is disconnected or one of the guard vertices has degree two and is not the root, contradicting the assumption that $H \setminus S^*$ is a full binary tree.

From the discussion above, it is clear that $S^*$ picks at least $n$ vertices of type $i$ for each $1 \leqslant i \leqslant k$, and combined with the fact that $|S^*| \leqslant nk$, we note that $S^*$ does not contain any of the guard vertices. Our next claim is that for all $i \in [k]$, $G \setminus S^*$ contains at least one essential vertex of type $i$. If not, then $S^*$ contains all the neighbors of the guards of type $i$, which makes them isolated in $G \setminus S^*$–a contradiction.

For each $1 \leqslant i \leqslant k$, consider the vertex in $G$ corresponding to the essential vertex that is *not* chosen by $S^*$ (in the event that there are multiple such vertices, we pick one arbitrarily). We denote this collection of vertices by $S$. We claim that $S$ induces an independent set in $G$: indeed, if not, then any edge in $G[S]$ is also present in $H \setminus S^*$ and creates a cycle when combined with the unique path connecting its endpoints via the backbone, which is again a contradiction. This concludes the proof. □

We now turn our attention to the edge-deletion variant. Here, we will find it convenient to reduce from a structured version of exact satisfiability, where the occurrences of the variables are bounded in frequency and also controlled in terms of how they appear. We will turn to a formal description in a moment, noting that here our reduction is similar to the one used to show that Linear-SAT is NP-complete [1].

**Theorem 2.** FBT-DE *is* NP-*complete.*

We first describe the version of SATISFIABILITY that we will reduce from. Our instance consists of $(4p + q)$ clauses which we will typically denote as follows:

$$\mathcal{C} = \{A_1, B_1, A_1', B_1', \cdots, A_p, B_p, A_p', B_p'\} \cup \{C_1, \cdots, C_q\}$$

We refer to the first $4p$ clauses as the *core* clauses, and the remaining clauses as the *auxiliary* clauses. The core clauses have two literals each, and also enjoy the following structure:

$$\forall i \in [p], A_i \cap B_i = \{x_i\} \text{ and } A_i' \cap B_i' = \{\bar{x}_i\}$$

We refer to the $x_i$'s as the *main* variables and the remaining variables that appear among the core clauses as *shadow* variables. The shadow variables occur

exactly once, and have negative polarity among the core clauses. Therefore, using $\ell(\cdot)$ to denote the set of literals occurring amongst a subset of clauses, we have:

$$\left| \ell \left( \bigcup_{i=1}^{p} \{A_i, B_i, A_i', B_i'\} \right) \right| = 6p.$$

The auxiliary clauses have the property that they only contain the shadow variables, which occur exactly once amongst them with positive polarity. Also, every auxiliary clause contains exactly four literals. Note that this also implies, by a double-counting argument, that $q = p$. We say that a collection of clauses is a *chain* if it has all the properties described above. An instance of LINEAR NEAR-EXACT SATISFIABILITY (LNES) is the following: given a set of clauses that constitute a chain, is there an assignment $\tau$ of truth values to the variables such that exactly one literal in every core clause and two literals in every auxiliary clause evaluate to TRUE under $\tau$?

For ease of discussion, given an assignment of truth values $\tau$ we often use the phrase "$\tau$ satisfies a literal" to mean that the literal in question evaluates to true under $\tau$. For instance, the question from the previous paragraph seeks an assignment $\tau$ that satisfies exactly one literal in every core clause and two literals in every auxiliary clause. We also refer to such an assignment as a near-exact satisfying assignment. The following observation is a direct consequence of the definitions above.

**Proposition 1.** *Let $\mathcal{C}$ be a collection of clauses that form a chain. For any assignment of truth values $\tau$, the main variables satisfy exactly two core clauses and the shadow variables satisfy either one core clause or one auxiliary clause.*

We first establish that LNES is NP-complete:

**Lemma 1.** LINEAR NEAR-EXACT SATISFIABILITY *is* NP-*complete.*

*Proof.* We reduce from (2/2/4)-SAT, which is the variant of SATISFIABILITY where every clause has four literals and every literal occurs exactly twice — in other words, every variable occurs in exactly two clauses with positive polarity and in exactly two clauses with negative polarity. The question is, if there exists an assignment $\tau$ of truth values to the variables under which exactly two literals in every clause evaluate to true. This problem is known to be NP-complete [11].

Let $\phi$ be a (2/2/4)-SAT instance over the variables $V = \{x_1, \ldots, x_n\}$ and clauses $\mathcal{C} = \{C_1, \ldots, C_m\}$. For every variable $x_i$, we introduce four new variables: $p_i, r_i$ and $q_i, s_i$. We replace the two positive occurrences of $x_i$ with $p_i$ and $r_i$, and the two negated occurrences of $x_i$ with $q_i$ and $s_i$. We abuse notation and continue to use $\{C_1, \ldots, C_m\}$ to denote the modified clauses. Also, introduce the clauses: $A_i = (x_i, \overline{p}_i), B_i = (x_i, \overline{r}_i), A_i' = (\overline{x}_i, \overline{q}_i), B_i' = (\overline{x}_i, \overline{s}_i)$, for all $1 \leqslant i \leqslant n$. Note that these collection of clauses form a chain, as required. We use $\psi$ to refer to this formula. We now turn to the argument for equivalence.

In the forward direction, let $\tau$ be an assignment that sets exactly two literals of every clause in $\phi$ to true. Consider the assignment $\zeta$ given by:

$$\zeta(x_i) = \tau(x_i), \zeta(p_i) = \zeta(r_i) = \tau(x_i); \zeta(q_i) = \zeta(s_i) = 1 - \tau(x_i),$$

for all $1 \leqslant i \leqslant n$. It is straightforward to verify that $\zeta$ satisfies exactly one literal in every core clause and exactly two literals in every auxiliary clause.

In the reverse direction, let $\zeta$ be an assignment for the variables of $\psi$ that satisfies exactly one literal in every core clause and exactly two literals in every auxiliary clause. Define $\tau$ as the restriction of $\zeta$ on the main variables. Let $C$ be a clause in $\phi$. To see that $\tau$ satisfies exactly two literals of $C$, note that the following:

$$\zeta(p_i) = \zeta(r_i) = \zeta(x_i) = \tau(x_i); \zeta(q_i) = \zeta(s_i) = 1 - \zeta(x_i) = 1 - \tau(x_i)$$

is forced by the requirement that $\zeta$ must satisfy exactly one literal in each core clause. Therefore, if $\tau$ satisfies more or less than two literals of any clause $C$, then that behavior will be reflected exactly in the auxiliary clause corresponding to $C$, which would contradict the assumed behavior of $\zeta$. We make this explicit with an example for the sake of exposition. Let $C$ from $\phi$ be the clause $(x_1, \overline{x}_3, \overline{x}_6, x_7)$, and let the clause constructed in $\psi$ be $(p_1, q_3, q_6, r_7)$. Suppose $\tau(x_1) = \tau(x_7) = \tau(x_6) = 1$ and $\tau(x_3) = 0$. Then we have $\zeta(p_1) = \zeta(r_7) = 1$ and $\zeta(q_6) = 0$, while $\zeta(q_3) = 1$. This demonstrates that $\zeta$ satisfies three literals in the auxiliary clause corresponding to $C$, in one-to-one correspondence with the literals that are satisfied by $\tau$. This completes our argument.  □

We now turn to a proof of Theorem 2. The overall approach is the following. We will introduce a complete binary tree whose leaves will be used to represent variables using variable gadgets which will have obstructions that can be removed in a fixed number of ways, each of which corresponds to a "signal" for whether the variable is to be set to true or false. We will then introduce vertices corresponding to clauses that will be attached to the variable gadgets in such a way that they can only be "absorbed" into the rest of the structure precisely when exactly two of its literals are receiving a signal indicating that they are being satisfied.

*The Shadow Variables.* An instance of the gadget that we construct for the shadow variables is depicted in Fig. 1. We remark here that the notation used for the vertices here is to enable our discussion of how the gadget works and is not to be confused with the notation already used to denote the variables and clauses of the LNES instance.

The vertices $p$ and $q$ are called the *anchors* of the gadget, while the vertices $x, y, a$ and $b$ are called the *drivers* of the gadget. This is because, as we will see, the behavior of the edges incident to these vertices determines the fate of the variable—in terms of whether it "accepts" the vertex corresponding to the core clause or the auxiliary clause to which it belongs. We refer to the vertex $u$ in the gadget as the *negative point of entry*, while the vertex $v$ is called the *positive point of entry*.

We refer to the edges incident on the vertices $x, y, a$ and $b$ as *active* edges and the remaining edges (i.e, $(p, u), (p, v), (q, w), (q, c)$ and $(w, z)$) as *passive* edges. We say that a solution is *nice* if it does not contain any passive edges. We also say that an instance $G$ of FBT-DE contains a clean copy of the gadget
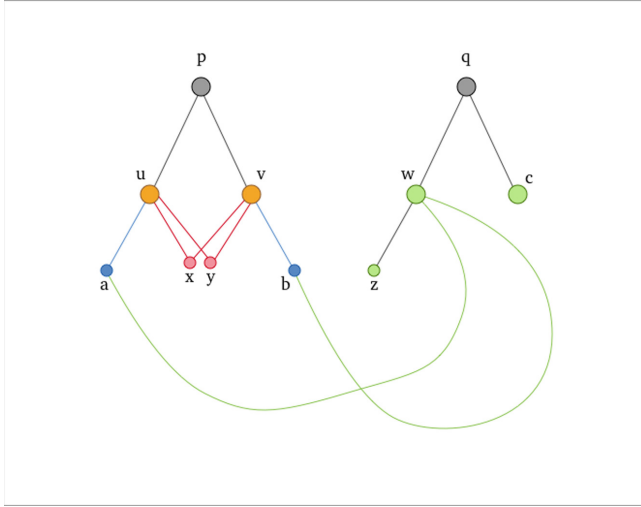
**Fig. 1.** The gadget corresponding to the shadow variables.

H if H appears in G as an induced subgraph, and further, $d_G(x) = d_G(y) = d_G(a) = d_G(b) = 2$ while $d_G(w) = 4$, $d_G(c) = 1$ and none of the vertices of H are chosen to be the target root vertex. We make the following observation about the behavior of this gadget.

*Claim.* Let H be a vertex gadget for a shadow variable as defined above. Let G be an instance of FBT-DE that contains a clean copy of H. Then, any nice solution S contains exactly four edges among the edges of H.

*Proof.* Let F denote the set of active edges in G. Since $d_G(x) = d_G(y) = d_G(a) = d_G(b) = 2$, we claim that any solution S must delete *exactly* four edges from F: in particular, S contains exactly one of the edges incident to each of the vertices. Indeed, if S deletes fewer edges than suggested then $G \setminus S$ contains a degree two vertex different from the root, which is a contradiction. On the other hand, if S contains more than four edges from F, then at least one of these four vertices is isolated in $G \setminus S$, which contradicts our assumption that $G \setminus S$ is connected. This clearly implies the claim, since all edges not considered are passive and a nice solution does not contain these edges by definition.    □

We now analyze the possible behaviors of a solution localized to the gadget in greater detail. We refer the reader to the full version of this paper for the figures associated with this explanation.

The possibilities $(xv, yv, aw, bv)$ and $(xu, yu, au, bw)$ do not arise because employing these deletions causes the entry point vertices to have degree four or more in $G \setminus S$. Further, since the solution S does not involve any of the passive edges, then we also rule out the following possibilities, since they all lead to a situation where the degree of w is four or more in $G \setminus S$:

▷ $xv, yv, au, bv$ ▷ $xu, yu, au, bv$ ▷ $xu, yv, au, bv$ ▷ $xv, yu, au, bv$

Recalling that $d_G(w) = 4$ when H makes a clean appearance in G, we also safely rule out the possibilities: $(xv, yv, aw, bw)$, $(xu, yu, aw, bw)$, $(xu, yv, aw, bw)$, $(xv, yu, aw, bw)$. Note that they result in a situation where the degree of $w$ is exactly two in $G \setminus S$ — since $w$ is not the target root vertex, this is a contradiction as well.

Observe that, given a nice solution S, in all the valid scenarios possible, either $d_{H \setminus S}(u) = 2$ and $d_{H \setminus S}(v) = 3$, or $d_{H \setminus S}(u) = 3$ and $d_{H \setminus S}(v) = 2$. We say that a shadow variable gadget has a *negative signal* in solutions where $d_{H \setminus S}(u) = 2$. Similarly, we say that the gadget has a *positive signal* in the situations where $d_{H \setminus S}(v) = 2$. We refer to the edges $\{(v, x), (v, y), (u, a), (w, b)\}$ as the *negative witness* and the edges $\{(u, x), (u, y), (v, b), (w, a)\}$ as the *positive witness* for the shadow variable gadgets. This concludes the description of the gadget meant for shadow variables.

*The Main Variables.* We now turn our attention to the gadget corresponding to the main variables. Here, we find it convenient to incorporate vertices representing the core clauses that the main variables belong to also as a part of the gadget. The construction of the gadget is depicted in Fig. 2. As before, the notation used for the vertices here is to enable our discussion of how the gadget works. With the exception of $A, B, A', B'$, which indeed are meaningfully associated with the analogously named core clauses, the notation is not to be confused with the notation already used to denote the variables and clauses of the LNES instance.

The edges $(z, u)$ and $(z, v)$ are the *passive* edges of this gadget, while the remaining edges are *active*. The vertex $z$ is called the *anchor* of this gadget. As before, a solution is *nice* if it does not contain any of the passive edges. We say that an instance G of FBT-DE has a clean copy of H if H appears in G as an induced subgraph and, further, $d_G(p) = d_G(q) = d_G(u) = d_G(v) = 3$, $d_G(x) = d_G(y) = 2$, $d_G(B) = d_G(A') = 2$, $d_G(A) = d_G(B') = 3$, and none of the vertices of H are chosen to be the target root vertex.

We reflect briefly on the nature of a nice solution S in instances that have a clean copy of a main variable gadget H. First, since $d_G(x) = 2$ and $x$ is not the target root vertex, we note that exactly one of $(v, x)$ or $(u, x)$ must belong to S. Suppose $(v, x) \in S$. The removal of $(v, x)$ makes $v$ a vertex of degree two, and since $(z, v)$ is a passive edge and S is nice, $(v, q) \in S$ is forced. Along similar lines, we have that $(u, p) \notin S$. Now, we argue that $(q, B') \notin S$. Indeed, if $(q, B') \in S$, then $A'$ has degree two from the deletions so far, and $q$ is a degree-one vertex with $A'$ as its sole neighbor. Recalling that $A'$ is not the target root vertex, we are now forced to delete exactly one of the endpoints incident on $A'$, but both possibilities lead us to a disconnected graph. Therefore, $(q, B') \notin S$. It is easy to see that this forces $(q, A') \in S$ and further, $(A, y) \in S$. A symmetric line of reasoning shows that if $(u, x) \in S$, then $(p, u), (p, B)$ and $(B', y)$ are all in S as well. We refer the reader to the full version of this paper for the figures
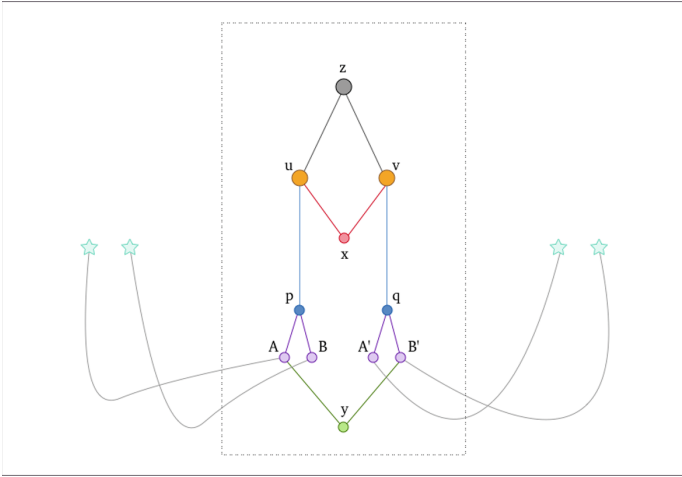
**Fig. 2.** The gadget corresponding to the main variables.

associated with this explanation. These two scenarios motivate the definitions of positive and negative signals that we now make explicit.

With respect to a nice solution $S$, we say that a main variable gadget has a *negative signal* if $d_{H\setminus S}(v) = 2$. Likewise, we say that the gadget has a *positive signal* if $d_{H\setminus S}(u) = 2$. We will also refer to the set of edges $\{(v, x), (v, q), (A', q), (A, y)\}$ as the *positive witness* of this gadget, and the *negative witness* is defined analogously.

We are now ready to discuss the overall construction. Let $\phi$ be an instance of LNES with clauses given by:

$$\mathcal{C} = \{A_1, B_1, A_1', B_1', \cdots, A_p, B_p, A_p', B_p'\} \cup \{C_1, \cdots, C_p\},$$

where the main variable common to $A_i$ and $B_i$ is denoted by $x_i$ and the auxiliary variables in these two clauses are denoted by $p_i$ and $r_i$, while the auxiliary variables in the clauses $A_i'$ and $B_i'$ are denoted $q_i$ and $s_i$. We denote by $\mathcal{I}_\phi := (G, r, w, k)$ the LNES instance that we will now construct based on $\phi$.

First, we construct the smallest complete binary tree with at least $(9p)$ leaves and let $v$ be the root of this tree. We refer to this tree as the *backbone* of $G$. Let the first $9p$ leaves of this tree be denoted by $\ell_1, \ldots, \ell_p; \alpha_1, \beta_1, \ldots, \alpha_{4p}, \beta_{4p}$. For each main variable $x_i$, let $H_i$ be the corresponding gadget. We identify the anchor of $H_i$ with $\ell_i$. For each shadow variable, we introduce a gadget corresponding to it, and identify the first anchor vertex in the gadget with $\alpha_i$ and the second anchor vertex with $\beta_i$. For every core clause $A_i$, we add an edge between the vertex $A$ in the gadget corresponding to $A_i$ and the negative entry point in the gadget for the shadow variable contained in the clause $A_i$. We also do this in an analogous fashion for the core clauses $A_i'$, $B_i$ and $B_i'$.

Finally, for each auxiliary clause $C_i$, we introduce two vertices $\omega_i$ and $\omega_i'$. We connect these vertices with the positive entry point into all gadgets corresponding

to the shadow variables that belong to the clause $C_i$. Note that each of these vertices have degree four. This completes the description of the construction of the graph $G$. We note that all the gadgets present in $G$ are clean by construction. We now define the weight of every edge in the backbone and every passive edge in the gadgets as $(k+1)$, while the weights of the remaining edges are set to be one. Finally, we set $k := 28p$ and let $r = \nu$—this concludes the description of the instance $\mathcal{I}_\phi$. We defer the argument of the equivalence of the instances to the full version of this paper.

## 4    FPT Algorithms

We observe that the problems considered here, namely FBT-DV and FBT-DE are fixed-parameter tractable by the standard parameter. We briefly describe a natural branching algorithm for FBT-DV while noting that an analogous argument works for FBT-DE.

Let $(G = (V, E), k, r, w)$ be an instance of FBT-DV. First, consider a vertex $v$, different from the designated root, that has four or more neighbors. Choose any four neighbors of $v$, say $a, b, c, d$, and branch on the set $\{v, a, b, c, d\}$ and we adjust the remaining budget by subtracting the respective weights of these vertices. The exhaustiveness of this branching rule follows immediately from the definition of a full binary tree. Along similar lines, we can also branch on the designated root along with three of its neighbors at a time, if the root has degree three or more, and also the closed neighborhoods of vertices of degree exactly two. We abort any branches where we have exhausted the budget.

We say that a graph with a designated root vertex is *nice* if it is connected, its maximum degree is three and the root is only vertex of degree two. Note that the depth of the branching thus far is bounded by $O(5^k)$, and we branch appropriately on disconnected instances, noting that only one of the components can "survive". Also, note that all the remaining instances are nice. If any of the remaining graphs are also acyclic, then we are already done.

If not, then we branch on these graphs further as follows. We pre-process vertices of degree three with a degree one neighbor by employing an appropriate short-circuiting rule. We can then start a breadth-first search (BFS) from the root vertex, noting that the depth of the BFS tree is at most $(\log_2 n + 1)$, since every internal vertex in this tree has at least two children. Therefore, we may infer that there exists a cycle of length $O(\log n)$, and we branch on all the vertices of this cycle other than the root vertex. If the deletion of a vertex on the cycle leads to a disconnected graph, then we abort the corresponding branch. Similarly, if the deletion of a vertex on the cycle creates vertices of degree two in the resulting graph, then we branch on the closed neighborhood of these degree two vertices and discard any disconnected graphs until we arrive at a nice instance, at which point we recurse in the fashion described here. The correctness of the overall algorithm follows from the exhaustiveness of the branching rules. The fact that the running time is FPT follows by a well-known argument [9].

**Theorem 3.** *The problems* FBT-DV *and* FBT-DE *are* FPT *with respect to solution size.*

# References

1. Arkin, E.M., et al.: Choice is hard. In: Elbassioni, K., Makino, K. (eds.) ISAAC 2015. LNCS, vol. 9472, pp. 318–328. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48971-0_28
2. Cygan, M., et al.: Parameterized Algorithms. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21275-3
3. Cygan, M., Pilipczuk, M., Pilipczuk, M., Wojtaszczyk, J.O.: An improved FPT algorithm and a quadratic kernel for pathwidth one vertex deletion. Algorithmica **64**(1), 170–188 (2012)
4. Diestel, R.: Graph Theory. Graduate Texts in Mathematics, vol. 173, 4th edn. Springer, New York (2012)
5. Ganian, R., Klute, F., Ordyniak, S.: On structural parameterizations of the bounded-degree vertex deletion problem. In: 35th Symposium on Theoretical Aspects of Computer Science, STACS, pp. 33:1–33:14 (2018)
6. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman, New York (1979)
7. Giannopoulou, A.C., Lokshtanov, D., Saurabh, S., Suchý, O.: Tree deletion set has a polynomial kernel but no opt$^{o(1)}$ approximation. SIAM J. Discrete Math. **30**(3), 1371–1384 (2016)
8. Philip, G., Raman, V., Villanger, Y.: A quartic kernel for pathwidth-one vertex deletion. In: Thilikos, D.M. (ed.) WG 2010. LNCS, vol. 6410, pp. 196–207. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16926-7_19
9. Raman, V., Saurabh, S., Subramanian, C.R.: Faster fixed parameter tractable algorithms for finding feedback vertex sets. ACM Trans. Algorithms **2**(3), 403–415 (2006)
10. Raman, V., Saurabh, S., Suchý, O.: An FPT algorithm for tree deletion set. J. Graph Algorithms Appl. **17**(6), 615–628 (2013)
11. Ratner, D., Warmuth, M.: Finding a shortest solution for the N × N extension of the 15-PUZZLE is intractable. In: Kehler, T.R.S. (ed.) Proceedings of the 5th National Conference on Artificial Intelligence, pp. 168–172
12. Yannakakis, M.: The effect of a connectivity requirement on the complexity of maximum subgraph problems. J. ACM **26**(4), 618–630 (1979)