How to Eliminate a Graph*

Petr A. Golovach[1], Pinar Heggernes[2], Pim van 't Hof[2], Fredrik Manne[2], Daniël Paulusma[1], and Michał Pilipczuk[2]

[1] School of Engineering and Computing Sciences, Durham University, UK
{petr.golovach,daniel.paulusma}@durham.ac.uk
[2] Department of Informatics, University of Bergen, Norway
{pinar.heggernes,pim.vanthof,fredrik.manne,michal.pilipczuk}@ii.uib.no

Abstract. Vertex elimination is a graph operation that turns the neighborhood of a vertex into a clique and removes the vertex itself. It has widely known applications within sparse matrix computations. We define the Elimination problem as follows: given two graphs G and H, decide whether H can be obtained from G by $|V(G)| - |V(H)|$ vertex eliminations. We study the parameterized complexity of the Elimination problem. We show that Elimination is $W[1]$-hard when parameterized by $|V(H)|$, even if both input graphs are split graphs, and $W[2]$-hard when parameterized by $|V(G)| - |V(H)|$, even if H is a complete graph. On the positive side, we show that Elimination admits a kernel with at most $5|V(H)|$ vertices in the case when G is connected and H is a complete graph, which is in sharp contrast to the $W[1]$-hardness of the related Clique problem. We also study the case when either G or H is tree. The computational complexity of the problem depends on which graph is assumed to be a tree: we show that Elimination can be solved in polynomial time when H is a tree, whereas it remains NP-complete when G is a tree.

1 Introduction

Consider the problem of choosing a set S of resilient communication hubs in a network, such that if any subset of the hubs should stop functioning then all the remaining hubs in S can still communicate. Such a set is attractive if the probability of a hub failure is high, or if the network is dynamic and hubs can leave the network. We can formulate this as a graph problem in the following way. Given a graph G and an integer k, is there a set S of k vertices, such that if any subset of S is removed from G, then every pair of remaining vertices in S are still connected via paths in the modified graph. Obviously, choosing S to be a clique of size k would solve the problem, but only allowing for cliques is overly restrictive. A necessary and sufficient condition on S is that for each pair $u, v \in S$, either u and v are adjacent or there is a path between u and v in G not containing any vertex of S except u and v. Thus we can view the described problem as a relaxation of the well-known Clique problem.

* This work is supported by EPSRC (EP/G043434/1) and Royal Society (JP100692), and by the Research Council of Norway (197548/F20).

M.C. Golumbic et al. (Eds.): WG 2012, LNCS 7551, pp. 320–331, 2012.

The above problem can be stated in terms of a well-known graph operation related to Gaussian elimination: vertex elimination [14]. The *elimination* of a vertex v from a graph G is the operation that adds edges to G such that the neighbors of v form a clique, and then removes v from the resulting graph. With this operation, the above problem can be defined as follows: find a set S of size k such that eliminating all vertices of $V(G) \setminus S$ leaves S as a clique. In fact, we state a more general problem: the ELIMINATION problem takes as input two graphs G and H, and asks whether a graph isomorphic to H can be obtained by the elimination of $|V(G)| - |V(H)|$ vertices from G. If this is possible, then we say that H is an *elimination* of G.

The vertex elimination operation described above has long known applications within linear algebra, and it simulates in graphs the elimination of a variable from subsequent rows during Gaussian elimination of symmetric matrices [14]. The resulting *Elimination Game* [14] repeatedly chooses a vertex and eliminates it from the graph until the graph becomes empty. The amount of edges added during the process, called the *fill-in*, is crucial for sparse matrix computations, and a vast amount of results have appeared on this subject during the last 40 years; see e.g., [6, 7, 14, 17]. Our problem ELIMINATION is equivalent to stopping Elimination Game after $|V(G)|-|V(H)|$ steps to see whether the resulting graph at that point is isomorphic to H. A crucial aspect of Elimination Game is the order in which the vertices are chosen, as this influences the fill-in. Note however that, for our problem, only the set of $|V(G)| - |V(H)|$ vertices chosen to be eliminated is important, and not the order in which they are eliminated.

Graph modification problems resulting from operations like vertex deletion, edge deletion, edge contraction, and local complementation are well studied, especially within fixed-parameter tractability; see e.g., [1, 3, 5, 8, 9, 11–13, 15, 19]. Given the wide use of the vertex elimination operation, we find it surprising that the ELIMINATION problem does not seem to have been studied before. The only related study we are aware of is by Samdal [18], who generated all eliminations of the $n \times n$ grids for $n \leq 7$.

Our Contribution. In this paper we study the computational complexity of ELIMINATION. In particular, we show that ELIMINATION is $W[1]$-hard when parameterized by $|V(H)|$ even when both input graphs are split graphs, and $W[2]$-hard when parameterized by $|V(G)|-|V(H)|$ even when H is a complete graph. On the positive side, for the case when H is complete, we show that ELIMINATION is fixed-parameter tractable when parameterized by $|V(H)|$, and has a kernel with at most $5|V(H)|$ vertices on connected graphs, which contrasts the hardness of the CLIQUE problem. We also study the cases when one of the input graphs is a tree. It turns out that the complexity of the problem changes completely depending on which input graph is a tree; we show that if G is a tree then the problem remains NP-complete, whereas if H is a tree then it can be solved in polynomial time. The mentioned kernel result is obtained by proving a combinatorial theorem on the maximum number of leaves in a spanning tree of a graph, similar to a proof by Kleitman and West [10]. We find this a contribution of independent interest.

Notation. All graphs in this paper are undirected, finite, and simple. Let $G = (V, E)$ be a graph. We sometimes use $V(G)$ and $E(G)$ to denote V and E, respectively. The *neighborhood* of a vertex $v \in V$ is the set of its neighbors $N_G(v) = \{w \in V \mid vw \in E\}$, and the *closed neighborhood* of v is the set $N_G[v] = N_G(v) \cup \{v\}$. The *degree* of v is $d_G(v) = |N_G(v)|$. For any subset $A \subseteq V$, we define $N_G[A] = \bigcup_{a \in A} N_G[a]$, $N_G(A) = N_G[A] \setminus A$, and $d_G(A) = |N_G(A)|$. For any subset $A \subseteq V$, $G[A]$ denotes the subgraph of G induced by A. For a subgraph H of G, we write $G \setminus H$ to denote the graph obtained from G by deleting all the vertices of H from G, i.e., $G \setminus H = G[V(G) \setminus V(H)]$.

A *clique* is a set of vertices that are all pairwise adjacent. A vertex v is *simplicial* if $N_G(v)$ is a clique. A graph G is *complete* if $V(G)$ is a clique. The complete graph on k vertices is denoted by K_k. An *independent* set is a set of vertices that are pairwise non-adjacent. If G is a bipartite graph, where (A, B) is a partition of V into two independent sets, then we denote it as $G = (A, B, E)$ and we call (A, B) a *bipartition* of G. A graph is a *split graph* it its vertex set can be partitioned into a clique and an independent set. A vertex is a *cut-vertex* if the removal of the vertex leaves the graph with more connected components than before.

A parameterized problem Q belongs to the class XP if each instance (I, k) can be solved in $f(k)|I|^{g(k)}$ time for some functions f and g that depend only on the *parameter* k, and $|I|$ denotes the size of I. If a problem belongs to XP, then it can be solved in polynomial time for every fixed k. If a parameterized problem can be solved by an algorithm with running time $f(k)\,|I|^{O(1)}$, then we say the problem is *fixed-parameter tractable*. The class of all fixed-parameter tractable problems is denoted FPT. Between FPT and XP is a hierarchy of parameterized complexity classes, $\mathsf{FPT} \subseteq \mathsf{W}[1] \subseteq \mathsf{W}[2] \subseteq \cdots \subseteq \mathsf{W}[\mathsf{P}] \subseteq \mathsf{XP}$, where hardness for one of the W-classes is considered to be strong evidence of intractability with respect to the class FPT. A parameterized problem is said to admit a *kernel* if there is a polynomial-time algorithm that transforms each instance of the problem into an *equivalent* instance whose size and parameter value are bounded from above by $g(k)$ for some (possibly exponential) function g. We refer to the textbook by Downey and Fellows [5] for formal background on parameterized complexity.

In this extended abstract, proofs of some theorems and lemmas, which are marked with the symbol ♠, have been omitted due to page restrictions.

2 Preliminaries and Hardness of Elimination

We start this section with an observation that provides a characterization of graphs that have some fixed graph H as an elimination. Our proofs heavily rely on this observation.

Observation 1 ([17]). *Let G and H be two graphs, where $V(H) = \{u_1, \ldots, u_h\}$. Then H is an elimination of G if and only if there exists a set $S = \{v_1, \ldots, v_h\}$ of h vertices in G that satisfies the following: $u_iu_j \in E(H)$ if and only if $v_iv_j \in E(G)$ or there is a path in G between v_i and v_j whose internal vertices are all in $V(G) \setminus S$, for $1 \leq i < j \leq h$.*

For two input graphs G and H that form an instance of Elimination, we let n denote the number of vertices in G. If G and H form a `yes`-instance, we say that a subset $X \subseteq V(G)$ is a *solution* if H is the resulting graph when all vertices in X are eliminated. By Observation 1, the vertices in X can be eliminated in any order. A vertex which is not eliminated is said to be *saved*. The set $S = V(G) \setminus X$ of saved vertices is called a *witness*.

Since we can check in polynomial time whether a set $S \subseteq V(G)$ of $|V(H)|$ vertices is a witness, Observation 1 immediately implies the following result.

Corollary 1. Elimination *is in* XP *when parameterized by* $|V(H)|$.

Corollary 1 naturally raises the question whether Elimination is FPT when parameterized by $|V(H)|$. The following theorem shows that this is highly unlikely.

Theorem 1 (♠). Elimination *is* $W[1]$*-hard when parameterized by* $|V(H)|$*, even if both* G *and* H *are split graphs.*

Since Elimination is unlikely to be FPT in general as a result of Theorem 1, it is natural to ask whether certain restrictions on G or H make the problem tractable. In Section 3, we restrict H to be a complete graph; note that due to Theorem 1, restricting H to be a split graph does not suffice to guarantee tractability. In Section 4, we study the variant where either G or H is a tree.

Another possible way of achieving tractibility is to investigate a different parameterization of the problem. For instance, instead of choosing the size of the witness as the parameter, we can parameterize Elimination by the size of the solution, i.e., the number of eliminated vertices. The next theorem shows that the problem remains intractable with this parameter.

Theorem 2 (♠). Elimination *is* $W[2]$*-hard when parameterized by* $|V(G)| - |V(H)|$*, even if* H *is a complete graph.*

We point out that the reductions used in the proofs of Theorems 1 and 2 immediately imply that the unparameterized version of Elimination is NP-complete, even if both G and H are split graphs, or if H is a complete graph.

3 Eliminating to a Complete Graph

In this section, we consider a special case of the Elimination problem when H is a complete graph. This corresponds exactly to the problem described in the first paragraph of Section 1. We define the problem Clique Elimination, which takes as input a graph G on n vertices and an integer k, and asks whether the complete graph K_k is an elimination of G. Since Clique Elimination is $W[2]$-hard when parameterized by $|V(G)| - k$ due to Theorem 2, we choose k as the parameter throughout this section.

If G contains a tree T with k leaves as a subgraph, then K_k is an elimination of G, as the leaves of T can serve as a witness. It is easy to observe that G contains a tree with k leaves as a subgraph if and only if G contains $K_{1,k}$, i.e., a star

with k leaves, as a minor. Moreover, by Observation 1, for any fixed graph H, the property that H is an elimination of a graph G can be expressed in monadic second-order logic. Since graphs that exclude $K_{1,k}$ as a minor have bounded treewidth [16], Courcelle's Theorem [4] implies that Clique Elimination is FPT when parameterized by k.

Even though fixed-parameter tractability of Clique Elimination is already established, two interesting questions remain. Does the problem admit a polynomial kernel? Does there exist an algorithm for the problem with single-exponential dependence on k? We provide an affirmative answer to both questions below. In particular, we prove the following result.

Theorem 3. Clique Elimination *admits a kernel with at most* $5k$ *vertices for connected graphs.*

We would like to remark that the assumption that the input graph is connected is probably necessary, as Clique Elimination in general graphs admits a simple composition algorithm that takes the disjoint union of instances, so existence of a polynomial kernel in the general setting would imply that NP $\subseteq$ coNP/poly. We refer an interested reader to the work of Bodlaender et al. [2] for an introduction to the methods of proving implausibility of polynomial kernelization algorithms.

As a result of Theorem 3, an algorithm with single-exponential dependence on k can be obtained by kernelizing every connected component of the input graph separately, and then running a brute-force search on each kernel. This gives us a better running time than the aforementioned combination of meta-theorems.

Corollary 2. Clique Elimination *can be solved in* $\binom{5k}{k} n^{O(1)} \leq 12.21^k\, n^{O(1)}$ *time and polynomial space.*

The remainder of this section is devoted to the proof of Theorem 3. Before presenting the formal proof, we give some intuition behind our approach. Our kernelization algorithm is based on the observation that the max-leaf number of a graph, i.e., the maximum number of leaves a spanning tree of the graph can have, is a lower bound on the size of a complete graph that can be obtained as an elimination. Kleitman and West [10] showed that a connected graph G with minimum degree at least 3 admits a spanning tree with at least $|V(G)|/4 + 2$ leaves. Their result immediately leads to a linear kernel for Clique Elimination provided that the input graph G has minimum degree at least 3. Unfortunately, we are unable to get rid of all vertices of degree at most 2 in our setting. However, we can modify our input graph in polynomial time such that we either can solve the problem directly, or obtain a new graph G^* with no vertices of degree 1 and with no edge between any two vertices of degree 2. We then prove a modified version of the aforementioned result by Kleitman and West [10], namely that such graphs G^* admit a spanning tree with at least $|V(G^*)|/5 + 2$ leaves. This leads to Theorem 3.

We now proceed with the formal proof of Theorem 3. Following Observation 1, we will be looking for a set S that is a witness of cardinality k, i.e., every two non-adjacent vertices of S can be connected by a path all internal vertices of which are outside S.

We start by providing four reduction rules, i.e., polynomial-time algorithms that, given an instance (G, k) of Clique Elimination, output an equivalent instance (G', k'). Each time, we apply the rule with the smallest number among the applicable ones. We argue that if none of the rules is applicable, then the modified graph has no vertices of degree 1 and no edge between any two vertices of degree 2. Recognizing whether a rule can be applied, as well as the application of the rule itself, will trivially be polynomial-time operations. The total number of applications will be bounded by a polynomial in the input size.

Reduction Rule 1. *If $k \leq 3$ or $n \leq 3$, then resolve the instance in polynomial time via a brute-force algorithm, and output a trivial* `yes-` *or* `no-`*instance, depending on the result.*

The safeness of the above rule is obvious.

Reduction Rule 2. *If G contains a vertex v of degree 1, eliminate its sole neighbor v' to obtain a graph G'. Output the instance (G', k).*

Lemma 1. *Reduction Rule 2 is safe.*

Proof. We need to argue that if we can find a witness S, then we can also find a witness S' of the same size that does not contain v'. If $v' \notin S$, then we set $S' = S$. If $v' \in S$, then $v \notin S$. Otherwise, since v is adjacent only to v', $k \leq 2$ and we could have applied Reduction Rule 1. We now set $S' = (S \setminus \{v'\}) \cup \{v\}$ to obtain a witness set of the same cardinality that does not contain v'. □

Reduction Rule 3. *If G contains a triangle v', v_1, v_2 such that v_1, v_2 are of degree 2, then eliminate v' to obtain a graph G'. Output the instance (G', k).*

Lemma 2. *Reduction Rule 3 is safe.*

Proof. Again, we need to argue that if we can find a witness S, then we can also find a witness S' of the same size that does not contain v'. If $v' \notin S$, then we set $S' = S$. Suppose that $v' \in S$. As Reduction Rule 1 was not applicable, we find that $k > 3$. Then neither v_1 nor v_2 belongs to S. We set $S' = (S \setminus \{v'\}) \cup \{v_1\}$ to obtain a witness set of the same cardinality that does not contain v'. □

Reduction Rule 4. *If G has a path v_0, v_1, v_2, v_3 such that v_1, v_2 are of degree 2 and $v_0 \neq v_3$, then eliminate v_0 to obtain a graph G'. Output the instance (G', k).*

Lemma 3. *Reduction Rule 4 is safe.*

Proof. We need to argue that if we can find a witness S, then we can also find a witness S' of the same size that does not contain v_0. If $v_0 \notin S$, then we set $S' = S$. Suppose that $v_0 \in S$. As Reduction Rule 1 was not applicable, we find that $k > 3$. Hence, S contains at most one vertex from the set $\{v_1, v_2, v_3\}$, as otherwise one of them could be connected to at most two other vertices from S via paths avoiding other vertices from S. If $|S \cap \{v_1, v_2, v_3\}| = 0$, then we take $S' = (S \setminus \{v_0\}) \cup \{v_1\}$, while if $|S \cap \{v_1, v_2, v_3\}| = 1$, then we take $S' = (S \setminus \{v_0, v_1, v_2, v_3\}) \cup \{v_1, v_2\}$. It is easy to check that S' defined in this manner is a witness of the same cardinality that does not contain v_0. □

If, after applying our four reduction rules exhaustively, we have not yet solved the problem, then we have obtained a graph G^* with no vertices of degree 1 and no edge between any two vertices of degree 2. If G^* has at most $5k-11$ vertices, then we output the instance as the obtained kernel. Otherwise, i.e., if G^* has at least $5k-10$ vertices, then we can safely return a trivial `yes`-instance due to the next result, which is our modified version of the aforementioned result by Kleitman and West [10]. This concludes the proof of Theorem 3.

Theorem 4. *Let G be a connected graph with minimum degree at least 2 such that no two vertices of degree 2 are adjacent. Then G admits a spanning tree with at least $|V(G)|/5+2$ leaves.*

Proof. We gradually grow a tree T in G keeping track of three parameters:

- n, the number of vertices in T;
- l, the number of leaves in T;
- m, the number of *dead* leaves in T, i.e., leaves that have no neighbor in $G \setminus T$.

The tree will be grown via a number of operations called *expansions*: by an expansion of a vertex $x \in V(T)$ we mean the adding of all the vertices $v \in V(G) \setminus V(T)$ with $xv \in E(G)$ and all the edges $xv \in E(G)$ with $v \notin V(T)$ to the tree T. We start with a tree T such that only leaves of T have neighbors in $G \setminus T$. Therefore, if we only use expansions to grow the tree, at each step of the growth process only the leaves of T are adjacent to $G \setminus T$. A leaf that is not dead, is called *alive*.

For a tree T, let us consider the potential $\phi(T)$ defined as $\phi(T) = 4l + m - n$. The goal is to

(a) find a starting tree T with $\phi(T) \geq 9$;
(b) provide a set of growing rules, such that there is always a rule applicable unless T is a spanning tree, and $\phi(T)$ does not decrease during the application of any rule;
(c) prove that during the whole process the potential increases by at least 1.

If goals (a), (b) and (c) are accomplished, then we can grow T using the rules until it becomes a spanning tree; in this situation we have $l = m$ and $n = |V(G)|$. As the potential increased by at least 1 during the whole process, we infer that $5l \geq |V(G)| + 10$, and hence $l \geq |V(G)|/5 + 2$, as claimed.

Goal (a) can be achieved by a careful case study; we omit the details due to page restrictions.

Having chosen the starting tree T, we can proceed with the growing rules. In order to grow the tree we always choose the rule that has the lowest number among the applicable ones, i.e., when applying a rule, we can always assume that the ones with lower numbers are not applicable. We would like to point out that the first three rules were already used in the original proof of Kleitman and West.

Growing Rule 1. If some leaf of T has at least two neighbors from $G \setminus T$, expand it. The potential $\phi(T)$ increases by at least $4 \cdot (d-1) - d = 3d - 4 \geq 2$, where $d \geq 2$ is the number of the aforementioned neighbors from $G \setminus T$.

Growing Rule 2. If some vertex $v \in V(G \setminus T)$ is adjacent to at least two leaves of T, expand one of these leaves. Observe that, as Rule 1 was not applicable and only leaves of T are adjacent to $G \setminus T$, this expansion results in adding only v to T. Moreover, all the remaining leaves adjacent to v were alive but become dead, so the potential $\phi(T)$ increases by at least $1 - 1 = 0$.

Growing Rule 3. If there is a vertex $v \in V(G \setminus T)$ of degree at least 3 in G that is adjacent to a leaf of T, expand this leaf (which results in adding only v to T, as Rule 1 was not applicable) and then v. The potential increases by at least $4 \cdot (d-2) - d = 3d - 8 \geq 1$, where $d \geq 3$ is the degree of v, as all the other neighbors of v are added to T as leaves, due to Rule 2 not being applicable.

Growing Rule 4. If there is a vertex $v \in V(G \setminus T)$ of degree 2 in G that is adjacent to a leaf of T, expand this leaf (which results only in adding v as a leaf, as Rule 1 was not applicable), then expand v, and then expand the second neighbor v' of v that became a leaf in T during the previous expansion. Note that v' could not be already in T, as otherwise Rule 2 would be triggered on vertex v. Since we assumed that no vertices of degree 2 are adjacent in G, the degree of v' is at least 3 and, as Rule 3 was not applicable, none of the neighbors of v' was in T. Denote by d the degree of v'; therefore, we have added to the tree T exactly $d+1$ vertices (v, v' and $d-1$ other neighbors of v') and increased the number of leaves by exactly $d - 2$. Hence, the increase of the potential is $4(d-2) - (d+1) = 3d - 9 \geq 0$, as $d \geq 3$.

It remains to argue that goal (c) is achieved. It is clear that if Growing Rule 1 or 3 is applied at least once, then the potential increases by at least 1. Suppose only Growing Rules 2 and 4 are applied during the whole process. Let x be a vertex of G that was added to T as a leaf during the very last rule application. Then x is a dead leaf. Since this was not taken into account when we determined a lower bound of 0 on the increase of the potential, the potential increases by

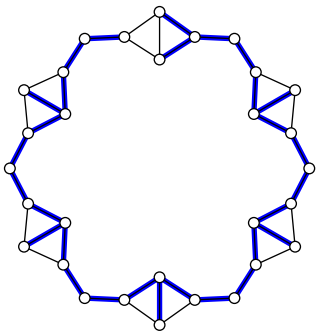

Fig. 1. A graph on 30 vertices for which the maximum possible number of leaves in a spanning tree is exactly 8; the bold (blue) edges indicate a spanning tree with 8 leaves. This example shows that the bound in Theorem 4 is tight.

at least 1. Thus, from the previously described analysis we conclude that using the presented method we are able to grow a tree with at least $|V(G)|/5 + 2$ leaves. □

The bound $|V(G)|/5 + 2$ is best possible. A family of examples with tight inequality can be obtained by connecting a number of diamonds in the way as shown in Figure 1.

4 Elimination on Trees

In this section, we study Elimination when G or H is a tree. When H is a tree, we show that the problem can be solved in polynomial time. Then we show that when G is a tree, the problem is NP-complete.

For a tree H with at least two vertices, we denote by $L(H)$ the set of leaves of H. The remaining set of vertices is denoted by $I(H) = V(H) \setminus L(H)$ and called the *inner vertices*. For a graph G, by $C(G)$ we denote the set of cut-vertices of G. A connected graph is 2-*connected* if it does not contain a cut-vertex. A maximal 2-connected subgraph of G is called a *biconnected component* (*bicomp* for short), and we denote by $\mathcal{B}(G)$ the set of bicomps of G. Consider the bipartite graph $\mathcal{T}_G$ with the vertex set $C(G) \cup \mathcal{B}(G)$, where $(C(G), \mathcal{B}(G))$ is the bipartition, such that $c \in C(G)$ and $B \in \mathcal{B}(G)$ are adjacent if and only if $c \in V(B)$. This graph $\mathcal{T}_G$ is a tree if G is connected, and is called the *bicomp-tree* of G.

Let G and H be an instance of Elimination where H is a tree. Since a graph G can be eliminated to a connected graph H if and only if at least one connected component of G can be eliminated to H, we assume without loss of generality that G is connected. Also it is easy to see that any graph G with at least one vertex can be eliminated to K_1, and K_2 is an elimination of a graph G whenever G has at least one edge. Hence, we can assume that H has at least three vertices. Therefore, $L(H) \neq \emptyset$ and $I(H) \neq \emptyset$.

Suppose that H is an elimination of G. Let $S = \{v_x \mid x \in V(H)\}$ be the witness, where v_x is the vertex of G that corresponds to the vertex x of H, and let $X = V(G) \setminus S$ be the corresponding solution yielding H. The witness S satisfies the structural properties given in the two following lemmas.

Lemma 4 (♠). *For any bicomp $B \in \mathcal{B}(G)$ it holds that $|V(B) \cap S| \leq 2$, and if $v_x, v_y \in V(B) \cap S$ for $x \neq y$, then $xy \in E(H)$.*

Lemma 5 (♠). *For any $x \in I(H)$, $v_x \in C(G)$.*

Now we choose an arbitrary inner vertex z of H and say that it is the *root* of H. The root defines the parent-child relation between any two adjacent vertices of H. For any two vertices $x, y \in V(H)$, we say that y is a *descendant* of x if x lies on the unique path in H from y to the root z. If y is a descendant of x and $xy \in E(H)$, then y is a *child* of x, and x is the *parent* of y. By definition, every vertex $x \in V(H)$ is a descendant of itself. For a vertex $x \in V(H)$, H_x denotes the subtree of H induced by the descendants of x, and for a vertex $x \in V(H)$ with a child y, H_{xy} is the subtree of H induced by x and the descendants of y.

Consider $r = v_z \in V(G)$. We choose r to be the *root* of the bicomp-tree $\mathcal{T}_G$ of G. By Lemma 5, r is a cut-vertex in G. The root r defines the parent-child relation on $\mathcal{T}_G$. Each bicomp B is a child of some inner vertex c in $\mathcal{T}_G$, and we say that the vertices of B are *children* of the corresponding cut-vertex c in G. A vertex $v \in V(G)$ is a *descendant* of a cut-vertex c if v is a child of some descendant of c in $\mathcal{T}_G$. For a cut-vertex c, we write G_c to denote the subgraph of G induced by the descendants of c. For a cut-vertex c and a bicomp B such that B is a child of c in $\mathcal{T}_G$, G_{cB} is the subgraph of G induced by the vertices of B and the descendants of all cut-vertices $c' \in V(B) \setminus \{c\}$.

Now consider two vertices x and y in H, such that neither is a descendant of the other, and let p be their lowest common ancestor. A crucial observation in our algorithm is that v_x and v_y are descendants of v_p in G, but they do not appear in the same subgraph $G_{v_p B}$ for some bicomp B that is a child of v_p. The following lemma formalizes this idea.

Lemma 6 (♠). *For any inner vertex $x \in V(H)$, if $y \in V(H)$ is a descendant of x in H, then v_y is a descendant of v_x in G. Moreover, if $y_1, \ldots, y_l$ are the children of x in H, then there are distinct children $B_1, \ldots, B_l$ of v_x in the bicomp-tree for which the following holds: for each $i \in \{1, \ldots, l\}$, if $y \in V(H_{xy_i})$, then $v_y \in G_{v_x B_i}$.*

We are now ready to describe our algorithm in the proof of the following theorem.

Theorem 5 (♠). Elimination *can be solved in time $O(n^{9/2})$ when H is a tree.*

Proof. Let G and H be an instance of Elimination where H is a tree. Clearly, if $|V(H)| > n$, then we have a `no`-instance of the problem. Hence, we assume that $|V(H)| \leq n$. Recall that it is sufficient to solve the problem for connected graphs G and trees H with at least three vertices. For the tree H, we choose an arbitrary inner vertex z and make it the root of H. For the graph G, we find the set of cut-vertices $C(G)$ and the set of bicomps $\mathcal{B}$, and construct the bicomp-tree $\mathcal{T}_G$. Then we construct a set $U \subseteq V(G)$ as follows: for each bicomp B that is a leaf of $\mathcal{T}_G$, we choose an arbitrary vertex $u \in V(B) \setminus C(G)$ and include it in U. It can be shown that H is an elimination of G if only if G can be eliminated to H with a witness $S \subseteq C(G) \cup U$. A formal proof of this statement requires some additional lemmas; we omit the details here due to page restrictions.

Suppose H is an elimination of G with a witness $S = \{v_x \mid x \in V(H)\}$. Since we chose z to be an inner vertex of H, the vertex v_z is a cut-vertex of G due to Lemma 5. Hence, by Lemma 6 there is a cut-vertex r in G such that if y is a descendant of x in H rooted at z, then v_y is a descendant of v_x in G rooted at r. We check all cut-vertices $r \in C(G)$, and for each r, we root G at r and try to find a witness that satisfies this condition. Clearly, H is an elimination of G if and only if we find such a witness for some r, and we have a `no`-instance of Elimination otherwise.

From now on, we assume that the root vertex r of G is fixed, and we construct a dynamic programming algorithm. For each vertex $u \in C(G) \cup U$, the algorithm will create a set $R_u \subseteq V(H)$ such that:

- for any $u \in U$, $R_u = L(H)$;
- for any $u \in C(G)$, R_u is the set of all vertices x of H such that H_x is an elimination of G_u with the property that for any $y, y' \in V(H_x)$, if y' is a descendant of y in H_x, then $v_{y'}$ is a descendant of v_y in G_r, where $v_y, v_{y'}$ are the saved vertices in G_u corresponding to y, y'.

The algorithm returns `yes` if R_r contains z, and `no` otherwise.

Notice that the sets for $u \in U$ are already defined. The sets R_u for cut-vertices u are constructed as follows. Denote by $B_1, \ldots, B_k$ the bicomps of G that are children of the cut-vertex u in the bicomp-tree $\mathcal{T}_G$. Let D_u be the set of all vertices $w \in C(G) \cup U$ other than u that are descendants of u and are contained in some bicomp together with u. In other words, $D_u = (C(G) \cup U \setminus \{u\}) \cap \bigcup_{i=1}^{k} V(B_i)$. Suppose that the sets R_w have already been constructed for all $w \in D_u$. We then create R_u in two steps.

Step 1. All the vertices that are in R_w for some $w \in D_u$ are included in R_u.

Step 2. Let $T_i = \cup_{w \in D_u \cap V(B_i)} R_w$ for $i \in \{1, \ldots, k\}$. A vertex $x \in V(H)$ with children $y_1, \ldots, y_l$ is included in R_u if there is a set $\{i_1, \ldots, i_l\} \subseteq \{1, \ldots, k\}$ such that $y_j \in T_{i_j}$ for $j \in \{1, \ldots, l\}$.

In order to perform Step 2, whose correctness is guaranteed by Lemma 6, we need to solve a matching problem on an auxiliary graph. The full proof of correctness and the running time analysis of our algorithm will appear in the journal version of this paper. □

Finally, we consider the case when G is a tree and H is an arbitrary graph. First, we make the following observation. A connected graph is called a *block graph* if each of its bicomps is a complete graph. Observe that if G is a block graph, then elimination of any vertex v results in another block graph, because this operation unites all maximal cliques that contain v into a single clique and then removes v. Since trees are block graphs, it gives us the following proposition.

Proposition 1. *If H is an elimination of a tree G, then H is a block graph.*

Despite the fact that graphs that are eliminations of trees have relatively simple structure, it turns out that Elimination remains intractable when G is assumed to be a tree.

Theorem 6 (♠). Elimination *is* NP*-complete, even if G is restricted to be a tree.*

Acknowledgements. We would like to thank Łukasz Kowalik for an inspiring discussion on the theorem of Kleitman and West.

References

1. van Bevern, R., Komusiewicz, C., Moser, H., Niedermeier, R.: Measuring Indifference: Unit Interval Vertex Deletion. In: Thilikos, D.M. (ed.) WG 2010. LNCS, vol. 6410, pp. 232–243. Springer, Heidelberg (2010)

2. Bodlaender, H.L., Downey, R.G., Fellows, M.R., Hermelin, D.: On problems without polynomial kernels. J. Comput. Syst. Sci. 75(8), 423–434 (2009)
3. Cai, L.: Fixed-parameter tractability of graph modification problems for hereditary properties. Inform. Process. Lett. 58(4), 171–176 (1996)
4. Courcelle, B.: The monadic second-order logic of graphs III: Tree-decompositions, minor and complexity issues. ITA 26, 257–286 (1992)
5. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer (1999)
6. George, J.A., Liu, J.W.H.: Computer Solution of Large Sparse Positive Definite Systems. Prentice-Hall Inc. (1981)
7. Heggernes, P.: Minimal triangulations of graphs: A survey. Discrete Mathematics 306, 297–317 (2006)
8. Heggernes, P., van 't Hof, P., Jansen, B.M.P., Kratsch, S., Villanger, Y.: Parameterized Complexity of Vertex Deletion into Perfect Graph Classes. In: Owe, O., Steffen, M., Telle, J.A. (eds.) FCT 2011. LNCS, vol. 6914, pp. 240–251. Springer, Heidelberg (2011)
9. Kawarabayashi, K., Reed, B.A.: An (almost) linear time algorithm for odd cyles transversal. In: Charikar, M. (ed.) SODA 2010, pp. 365–378. SIAM (2010)
10. Kleitman, D.J., West, D.B.: Spanning trees with many leaves. SIAM J. Discrete Math. 4, 99–106 (1991)
11. Marx, D.: Chordal deletion is fixed-parameter tractable. Algorithmica 57(4), 747–768 (2010)
12. Marx, D., Schlotter, I.: Obtaining a planar graph by vertex deletion. Algorithmica 62(3-4), 807–822 (2012)
13. Natanzon, A., Shamir, R., Sharan, R.: Complexity classification of some edge modification problems. Discrete Appl. Math. 113, 109–128 (2001)
14. Parter, S.: The use of linear graphs in Gauss elimination. SIAM Review 3, 119–130 (1961)
15. 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)
16. Robertson, N., Seymour, P.D., Thomas, R.: Quickly excluding a planar graph. J. Comb. Theory B 62, 323–348 (1994)
17. Rose, D.J., Tarjan, R.E., Lueker, G.S.: Algorithmic aspects of vertex elimination on graphs. SIAM J. Comput. 5, 266–283 (1976)
18. Samdal, E.: Minimum Fill-in Five Point Finite Element Graphs. Master's thesis, Department of Informatics, University of Bergen, Norway (2003)
19. Yannakakis, M.: Edge-deletion problems. SIAM J. Comput. 10, 297–309 (1981)