On Making Directed Graphs Transitive[☆,☆☆]

Mathias Weller[3,*], Christian Komusiewicz[1], Rolf Niedermeier, Johannes Uhlmann[2]

Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany

Abstract

We present a first thorough theoretical analysis of the Transitivity Editing problem on digraphs. Herein, the task is to make a given digraph transitive by a minimum number of arc insertions or deletions. Transitivity Editing has applications in for the detection of hierarchical structure in molecular characteristics of diseases.

We demonstrate that if the input digraph does not contain "diamonds", then there is an optimal solution that performs only arc deletions. This fact helps us construct a first proof of NP-hardness, which also extends to the restricted cases in which the input digraph is acyclic or has maximum degree three.

By providing an $O(k^2)$-vertex problem kernel, we answer an open question from the literature. In case of digraphs with maximum degree d, an $O(k \cdot d)$-vertex problem kernel can be shown. Moreover, we improve previous fixed-parameter algorithms, now achieving a running time of $O(2.57^k + n^3)$ for an n-vertex digraph if k arc modifications are sufficient to make it transitive.

Our hardness as well as algorithmic results transfer to Transitivity Deletion, where only arc deletions are allowed.

Keywords: Graph modification problem, NP-hardness, Hierarchical structure detection, Fixed-parameter tractability, Kernelization and data reduction

1. Introduction

A directed graph (digraph for short) $D = (V, A)$ is called *transitive* if $(u, v) \in A$ and $(v, w) \in A$ imply $(u, w) \in A$ (also cf. [1, Section 4.3]).

To make a digraph transitive by a minimum number of arc modifications has recently been identified to have important applications in detecting hierarchical structure in molecular characteristics of diseases [20, 3].

☆A preliminary version of this work appeared in the proceedings of the 11th Algorithms and Data Structures Symposium (WADS'09), volume 5664 in LNCS, pages 542–553, Springer 2009.

☆☆The main part of this work was done while the authors were with the Friedrich-Schiller-Universität Jena.

*Corresponding author

Email addresses: mathias.weller@tu-berlin.de (Mathias Weller), christian.komusiewicz@tu-berlin.de (Christian Komusiewicz), rolf.niedermeier@tu-berlin.de (Rolf Niedermeier), johannes.uhlmann@campus.tu-berlin.de (Johannes Uhlmann)

[1]Supported by a PhD fellowship of the Carl-Zeiss-Stiftung and the DFG, research project PABI, NI 369/7.

[2]Supported by the DFG, research project PABI, NI 369/7.

[3]Supported by the DFG, research project DARE, GU 1023/1, NI 369/11.

Here, a group of patients is analyzed and a hierarchical classification of diseases in a scheme of sub-diseases based on molecular characteristics is extracted [18]. Due to measurement errors and noise in this data, the resulting relation is often not completely correct. By restoring transitivity to the relation, one hopes to reconstruct the real relation to a fair extent. Hence, the task is to find a consistent disease hierarchy that is closest to the measured data. Obviously, one must assume the error to be small, otherwise, one may reconstruct almost any hierarchical structure from the data. One interprets the data as a directed graph and inserts and deletes arcs until transitivity is achieved. The vertices of the graph are diseases and there is an arc (a, b) from vertex a to vertex b if the experimental data suggests that disease b is a sub-disease of disease a. The central problem under consideration, Transitivity Editing, thus asks whether a given digraph can be transformed into a transitive digraph by inserting or deleting no more than some given number k of arcs.

We provide a first thorough theoretical study of Transitivity Editing, complementing previous work that focused on heuristics, integer linear programming, and simple fixed-parameter algorithms [20, 3]. We also study the special case where only arc deletions (Transitivity Deletion) are allowed and restricted classes of digraphs (acyclic and bounded-degree). Note that Transitivity Completion (where only arc insertions are allowed) is nothing but the well-studied problem of computing the transitive closure of a digraph; this is solvable in polynomial time [23].

Previous work.. Transitivity Editing can be seen as the "directed counterpart" of the so far much more extensively studied problem Cluster Editing on undirected graphs (see [2, 4, 5, 10, 13, 14, 15, 21, 26]). Indeed, both problems are also referred to as Transitive Approximation problem on directed and undirected graphs, respectively. Unfortunately, this was perhaps a reason why the NP-hardness of Transitivity Editing has erroneously been claimed to be proven [20, 3] by referring to work that only considers problems on undirected graphs, including Cluster Editing. On the positive side, however, the close correspondence between Cluster Editing and Transitivity Editing helped Böcker et al. [3] transfer their previous results for Cluster Editing [4] to Transitivity Editing, delivering the currently fastest implementations that exactly solve Transitivity Editing (by means of integer linear programming and fixed-parameter algorithms). In particular, their computational experiments demonstrate that their exact algorithms are by far more efficient in practice than the previously used purely heuristic approach by Jacob et al. [20].

Our contributions.. We start by deriving the helpful observation that any digraph that does not contain a so-called "diamond" has an optimal solution for Transitivity Editing that only deletes arcs. Hence, in these cases, Transitivity Editing and Transitivity Deletion coincide. This observation is useful for both algorithmic and hardness results.

We continue by *proving* the so far only *claimed* NP-hardness of Transitivity Editing, also extending this result to Transitivity Deletion. Moreover, we show that both problems remain NP-hard when restricted to acyclic digraphs or digraphs whose underlying undirected graphs have maximum vertex degree three. These proofs also provide exponential lower bounds on the running time for algorithms solving Transitivity Editing or Transitivity Deletion.

Eventually, we provide a polynomial-time data reduction that yields an $O(k^2)$-vertex problem kernel for Transitivity Editing and Transitivity Deletion. This answers an open question of Böcker et al. [3]. In the special case of digraphs with maximum vertex degree d, we show an $O(k \cdot d)$-vertex kernel. In addition, we develop an

improved search tree for Transitivity Editing. That is, whereas the fixed-parameter algorithm of Böcker et al. [3] runs in $O(3^k \cdot n^3)$ time on n-vertex digraphs, our new algorithm runs in $O(2.57^k + n^3)$ time (note that in our algorithm the cubic term n^3 is additive instead of multiplicative due to our kernelization result). Finally, we observe that Transitivity Deletion can be solved in $O(2^k + n^3)$ time.

Organization of the paper.. In Section 2, after agreeing on some necessary preliminaries, we make a very helpful observation about digraphs excluding certain substructures. Section 3 is the most technical section containing our NP-hardness results. We split Section 3 into two parts, dealing with digraphs whose underlying undirected graph has bounded degree and acyclic digraphs, respectively. With these NP-hardness proofs established, we present our algorithmic results in Section 4: We first show a kernelization for both Transitivity Editing and Transitivity Deletion and then present a way of improving the standard search tree algorithm for Transitivity Editing. Finally, in the concluding Section 5, we summarize our findings and pose open questions for future work.

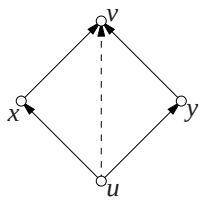
2. Preliminaries and a Structural Result

Parameterized complexity.. Our algorithmic results are in the context of parameterized complexity, which is a two-dimensional framework for studying the computational complexity of problems [9, 11, 24]. One dimension is the input size n (as in classical complexity theory), and the other one is the *parameter* k. A problem is called *fixed-parameter tractable* (fpt) if it can be solved in $f(k) \cdot n^{O(1)}$ time, where f is a computable function only depending on k. This means that when solving a combinatorial problem that is fpt, the combinatorial explosion can be confined to the parameter. A core tool in the development of fixed-parameter algorithms is polynomial-time preprocessing by *data reduction*. Here, the goal is for a given problem instance x with parameter k to transform it into a new instance x' with parameter $k' \leq k$ such that the size of x' is upper-bounded by some function only depending on k and the instance (x, k) is a yes-instance if and only if (x', k') is a yes-instance. The reduced instance, which must be computable in polynomial time, is called a *problem kernel*, and the whole process is called *reduction to a problem kernel* or simply *kernelization* (see [6, 16] for surveys).

Graph-theoretic concepts.. A *directed graph* or *digraph* is a pair $D = (V, A)$ with $A \subseteq V \times V$. The set V contains the *vertices* of the digraph, while A contains the *arcs*. Unless stated otherwise, let $n := |V|$. For $V' \subseteq V$, let $D[V'] := (V', A \cap (V' \times V'))$ denotes the subgraph of D that is *induced* by V'. Furthermore, we write $D - u$ for $D[V \setminus \{u\}]$. The *symmetric difference* of two arc sets A and A' is $A \Delta A' := (A \cup A') \setminus (A \cap A')$. In this work, we only consider simple digraphs, that is digraphs without self-loops and double arcs.

For any $u \in V$, $\text{pred}_A(u) := \{v \in V \mid (v, u) \in A\}$ denotes the set of *predecessors* of u with respect to A (the number of predecessors is called *indegree*), while $\text{succ}_A(u) := \{v \in V \mid (u, v) \in A\}$ denotes its *successors* (the number of successors is called *outdegree*). The vertices in $\text{pred}_A(u) \cup \text{succ}_A(u)$ are said to be *adjacent to* u. The transitive and reflexive closure of the successor relation is the *reachability* relation.

The *degree* of a vertex is the sum of its indegree and its outdegree and the degree of a digraph is the maximum over the degrees of its vertices.



	u	v	x	y
u	-	0	1	1
v	*	-	*	*
x	*	1	-	*
y	*	1	*	-

Figure 1: The diamond structure and its adjacency matrix. According to the definition of diamonds, the solid arcs must be present and the dashed arc must be absent. All other arcs may or may not be present. In the adjacency matrix, the endpoints of each vertex' outgoing arcs are determined by its row. Asterisks represent wildcards, that is, these entries do not matter for the definition.

Definition 1. A digraph $D = (V, A)$ is called *transitive* if

$$\forall u, v, w \in V \ ((u, v) \in A \wedge (v, w) \in A) \Rightarrow (u, w) \in A.$$

In other words, D is transitive if A is a transitive relation on $(V \times V)$. The central problem of this work is defined as follows.

Transitivity Editing:
Input: A digraph $D = (V, A)$ and an integer $k \geq 0$.
Question: Is there a digraph $D' = (V, A')$ that is transitive and $|A \Delta A'| \leq k$?

Analogously, Transitivity Deletion is defined by disallowing arc insertions. Also note that, although we focus on the decision variants of the problems, our algorithms can also solve the corresponding minimization problems.

To derive our results, we make use of the fact that transitive digraphs can be characterized by "forbidden P_3s" [3]. In our setting, the P_3s of a digraph are all vertex triples (u, v, w) such that $(u, v) \in A$, $(v, w) \in A$, and $(u, w) \notin A$. When saying that a digraph D "contains" a P_3 (u, v, w), we mean that u, v, w are vertices in D and the arcs (u, v) and (v, w) are present in D while (u, w) is not. Note that this differs from both the notions of induced subgraphs and subgraphs since a subgraph characterization does not enable us to forbid arcs of the host graph and an induced subgraph characterization cannot have "don't care"-arcs (arcs marked with an asterisk in the table in Figure 1).

meaning a slight abuse of standard notation. We say that the P_3 (u, v, w) *contains* the arcs (u, v) and (v, w) and the vertices u, v, and w. As also noted by Böcker et al. [3], transitive digraphs can be characterized as the digraphs without P_3s, that is, a digraph is transitive if and only if it does not contain a P_3.

Lemma 1 (Folklore)**.** *A digraph $D = (V, A)$ is transitive if and only if it does not contain a P_3.*

Diamonds in digraphs.. Many of our combinatorial studies are based on the consideration of "diamonds". The absence of diamonds in a given digraph simplifies the Transitivity Editing problem and helps us in proving both NP-hardness and our algorithmic results. A *diamond* in a digraph $D = (V, A)$ is a triple $(u, \{x, y\}, v)$, where $u, x, y, v \in V$, $(u, v) \notin A$, and $(u, z), (z, v) \in A$ for $z \in \{x, y\}$ (see Figure 1).[4] If D does not contain a diamond, then it is said to be *diamond-free*.

[4]This is not a standard definition and should not be mixed up for instance with diamonds in undirected graphs.

A set $S \subseteq V \times V$ is a *solution set* of Transitivity Editing for the digraph (V, A) if $(V, A \Delta S)$ is transitive. A solution set S is *optimal* if there is no solution set S' with $|S'| < |S|$. For each solution set S we consider its two-partition $S = S_{\text{DEL}} \uplus S_{\text{INS}}$, where $S_{\text{DEL}} := S \cap A$ denotes the set of arc deletions and S_{INS} denotes the set of arc insertions.

The following lemma shows that the property of being diamond-free is preserved by deleting the arcs of a solution set.

Lemma 2. *Let $D = (V, A)$ be a diamond-free digraph and let S be a solution set for D. Then $D_{\text{DEL}} := (V, A \setminus S_{\text{DEL}})$ is diamond-free.*

Proof. Suppose that D_{DEL} contains a diamond $(u, \{x, y\}, v)$. Note that, since D is diamond-free, $(u, v) \in S_{\text{DEL}}$. Since S_{INS} is a solution set for D_{DEL}, both the P_3s (u, x, v) and (u, y, v) are destroyed by arc insertions.

Hence, $(u, v) \in S_{\text{INS}}$, contradicting $S_{\text{INS}} \cap S_{\text{DEL}} = \emptyset$. □

The following result shows that when solving Transitivity Editing on diamond-free digraphs, it is optimal to only perform arc deletions.

Lemma 3. *Let (D, k) with $D = (V, A)$ be a diamond-free input instance of* Transitivity Editing. *Then, there is an optimal solution set S for D that inserts no arc, that is, $S = S_{\text{DEL}}$.*

Proof. Let S' be any optimal solution set for D. By Lemma 2, we can apply all arc deletions of a given solution set without destroying diamond-freeness. Hence, we assume the solution set S' to only consist of arc insertions. We construct S from S':

$$S := \{(a, b) \mid \exists c \in V : \ (a, c) \in S' \wedge (a, b) \in A \wedge (b, c) \in A\}.$$

Since D is diamond-free, for each $(a, c) \notin A$, there is at most one $b \in V$ meeting the criteria $(a, b) \in A$ and $(b, c) \in A$. Therefore, for each arc $(a, c) \in S'$, there is at most one arc $(a, b) \in S$ and hence $|S| \leq |S'|$.

Let $D^* := (V, A^*)$ with $A^* := A \setminus S$. We show that S is a solution set for D by proving that D^* is transitive: Assume that there is a P_3 $p = (x, y, z)$ in D^*. Since $S = S_{\text{DEL}}$, we know that $(x, y) \in A$ and $(y, z) \in A$ and, since S' is a solution set for D, we know that p is not a P_3 in $(V, A \Delta S')$, implying either $(x, z) \in S'$ or $(x, z) \in S$. However, $(x, z) \notin S'$, because otherwise $(x, y) \in S$, contradicting p being a P_3 in D^*. Hence, $(x, z) \in A$ and $(x, z) \in S$. By definition of S, this implies that there is a vertex $v \in V$ with $(z, v) \in A$ and $(x, v) \in S'$. Also, $(y, v) \notin A$, since, otherwise, (x, z, v) and (x, y, v) would form a diamond in D. Hence, $q = (y, z, v)$ is a P_3 in D. As p, also q cannot be a P_3 in $(V, A \Delta S')$. However, S' does only contain insert operations, which implies $(y, v) \in S'$. Since $(y, z) \in A$ and $(z, v) \in A$, this implies $(y, z) \in S$, contradicting p being a P_3 in D^*. □

3. NP-Hardness Results

In this section, we prove the NP-hardness of Transitivity Editing and Transitivity Deletion in degree-three digraphs and in acyclic digraphs. On the one hand, it seems not very surprising that both problems are NP-hard, since their undirected "sisters" Cluster Editing and Cluster Deletion have been shown to be NP-hard (see, e.g., [22, 28]). On the other hand, the hardness proofs for the undirected problems do not carry over to digraphs so easily (in fact, we were unable to salvage anything from these

proofs). It is also worth mentioning that we show NP-hardness for very restricted classes of digraphs. Essential ideas of our hardness proofs have already been reused to prove NP-hardness of MaxDiCut, Maximum Tree Orientation, and Cluster Editing for very restricted cases as well [8, 21].

3.1. NP-hardness on Bounded-Degree Digraphs

Our NP-hardness result for bounded-degree digraphs is derived by a reduction from 3SAT.

> 3SAT:
> **Input**: A Boolean formula φ in conjunctive normal form with n variables $x_0, \ldots, x_{n-1}$ and m clauses $C_0, \ldots, C_{m-1}$, each consisting of three literals.
> **Question**: Is there a truth assignment for all n variables such that φ evaluates to true?

Construction.. Given an input instance φ of 3SAT in which, without loss of generality, every clause contains each variable at most once, we construct in polynomial time an equivalent instance of Transitivity Editing as follows. For each of the n Boolean variables in φ, we construct a *variable cycle*, that is, a directed cycle of length $8m$, with m being the number of clauses in φ. More specifically, for each variable x_i, the corresponding variable cycle consists of the vertices $V_i^{\text{var}} := \{i_0, \ldots, i_{8m-1}\}$. The vertices in V_i^{var} are connected to form a cycle by adding the arcs $A_i^{\text{var}} := \{(i_p, i_{p+1}) \mid 0 \leq p \leq 8m-1\}$ (for the ease of presentation, let $i_{8m} = i_0$). Each variable cycle can be partitioned into m consecutive subpaths of eight vertices each. We call i_{8j}, $0 \leq j < m$, a *positive j-connection vertex* and i_{8j+1} the *negative j-connection vertex*.

Depending on whether x_i appears negated in the clause C_j or not, we use either the negative or the positive j-connection vertex to connect the variable cycle to the clause gadget of clause C_j.

The collection of all variable cycles is then referred to as $(V^{\text{var}}, A^{\text{var}})$ with $V^{\text{var}} := \bigcup_{i=0}^{n-1} V_i^{\text{var}}$ and $A^{\text{var}} := \bigcup_{i=0}^{n-1} A_i^{\text{var}}$. In the following, we refer to the arcs $(i_0, i_1), (i_2, i_3), \ldots, (i_{8m-2}, i_{8m-1})$ as *positive arcs* and to the remaining arcs in the variable cycle as *negative arcs*.

The basic idea behind the construction is as follows. Since a variable cycle contains $4m$ arc-disjoint P_3s, making it transitive requires at least $4m$ modifications. If we are restricted to arc deletions (we will show that there is an optimal solution that only deletes arcs), this is clearly possible *only* if we delete every second arc. Hence, in this case, there are exactly two ways of making a variable cycle transitive with at most $4m$ arc modifications, namely either deleting all positive or all negative arcs.

Observation 1. *Making a variable cycle transitive by deleting arcs requires at least $4m$ arc deletions. This can only be achieved by either deleting all positive arcs or all negative arcs.*

We use these two optimal solutions to represent the truth value of the corresponding variable and vice versa. If the variable cycle for x_i is made transitive by deleting all positive arcs, then x_i is considered to be assigned true, otherwise x_i is considered to be assigned false.

Next, consider a clause C_j containing the variables x_p, x_q, and x_r. For each $i \in \{p, q, r\}$, let

$$\text{dock}(i, j) := \begin{cases} 8j & \text{if } x_i \text{ occurs negated in } C_j, \\ 8j+1 & \text{if } x_i \text{ occurs nonnegated in } C_j. \end{cases}$$

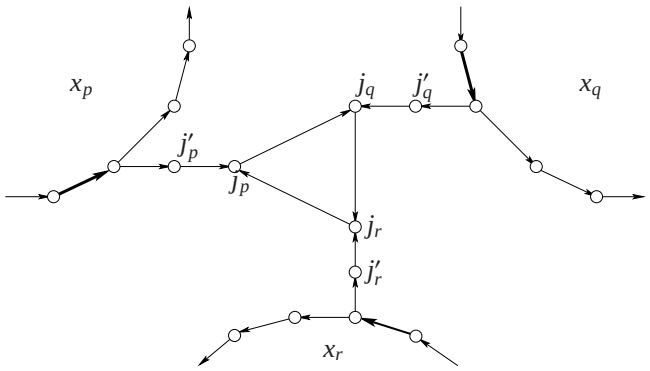


Figure 2: The clause gadget of clause C_j containing the variables x_p, x_q, and x_r. It consists of a length-three cycle that is connected to the variable cycles via P_3s. Bold arcs represent docking arcs.

For each clause C_j and each variable x_i that occurs in C_j, we define the *docking arc* $\alpha_{i,j}$ of the clause C_j in the variable gadget of x_i as the uniquely determined arc that is incoming to $i_{\text{dock}(i,j)}$. Note that the vertex $i_{\text{dock}(i,j)}$ is the negative or positive j-connection vertex of x_i, depending on whether x_i occurs negated in C_j or not.

Observation 2. *Let C_j be a clause of φ, and let x_i be a variable that occurs in C_j. The docking arc $\alpha_{i,j}$ is a positive arc if and only if x_i occurs nonnegated in C_j.*

We continue the construction by adding a *clause gadget* that consists of a length-three cycle (j_p, j_q, j_r) which we connect to the variable cycles of x_p, x_q, and x_r by adding the P_3 $(i_{\text{dock}(i,j)}, j'_i, j_i)$ for all $i \in \{p, q, r\}$. We refer to the arcs in this clause gadget as A_j^{cls} and to the arcs of all clause gadgets as A^{cls}. For an illustration, see Figure 2. In the following paragraph, we show the correctness of the presented construction, that is, the constructed instance of Transitivity Editing is a yes-instance if and only if the original instance of 3SAT is a yes-instance.

Correctness.. Let $D(\varphi)$ denote the digraph that results from this construction. First, observe that two vertices of a variable cycle that are contained in different clause cycles have distance at least seven. Therefore, the constructed digraph is diamond-free. Second, observe that $D(\varphi)$ is a degree-threehas maximum degree three.

Consider a clause gadget. Obviously, a cycle of length three can be made transitive with two arc deletions. Since each clause gadget contains such a cycle and three additional P_3s, it is clear that we need at least five arc deletions for each clause gadget.

Observation 3. *Let S denote a solution set for $D(\varphi)$ with $S = S_{\text{DEL}}$. Then, for each clause C_j of φ, it holds that $|S \cap A_j^{cls}| \geq 5$.*

If the three docking arcs of a gadget are not deleted, then this clause gadget together with these docking arcs contains six arc-disjoint P_3s. Hence, six modifications are required to make this structure transitive in this case (see Figure 3).

Note that the docking arcs are chosen such that this occurs if and only if all literals in the clause corresponding to this gadget evaluate to false. In the following, "making the clause gadget of C_j transitive" refers to destroying all P_3s that contain at least one arc of A_j^{cls}.

Lemma 4. *Let S^{var} denote a solution for (V, A^{var}), let C_j be some clause in φ, and let x_p, x_q, and x_r denote the variables occurring in C_j. Then, the clause gadget of C_j*

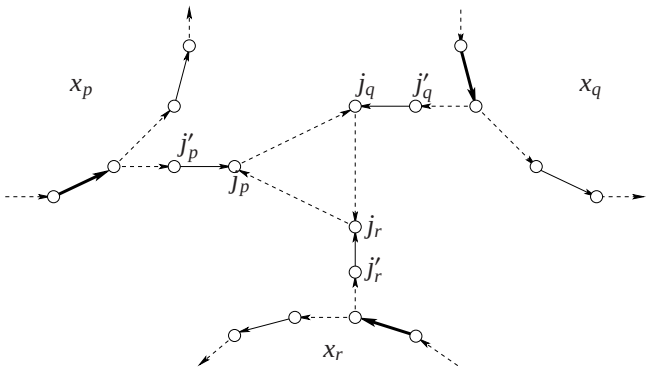


Figure 3: If none of the three docking arcs is deleted, then six arc deletions are required to make this structure transitive. Here, bold arcs are docking arcs and dashed arcs are deleted.

can be made transitive by deleting five arcs in A_j^{cls} if and only if there is some $i \in \{p, q, r\}$ with $\alpha_{i,j} \in S^{var}$.

Proof. Let j_i^+ denote the successor of j_i in the clause gadget of C_j.

"⇒": We show the contraposition. Assume that all docking arcs of C_j are not in S^{var}. Then, for each $i \in \{p, q, r\}$, the P_3s $(i_{\text{dock}(i,j)-1}, i_{\text{dock}(i,j)}, j'_i)$ and (j'_i, j_i, j_i^+) are destroyed by deleting arcs of A_j^{cls}. Since these six P_3s are arc-disjoint, at least six arc modifications are necessary (see Figure 3).

"⇐": Without loss of generality, let $\alpha_{r,j} \in S^{\text{var}}$. Then, we can make the clause gadget of C_j transitive by deleting the arcs $(p_{\text{dock}(p,j)}, j'_p)$, (j_p, j_q), $(q_{\text{dock}(q,j)}, j'_q)$, (j_q, j_r), and (j'_r, j_r) (see Figure 4). □

With these observations at hand, we now show the NP-completeness of Transitivity Editing, even if the maximum degree of the input is three and there are neither sources nor sinks, that is, the indegree and outdegree of each vertex is either one or two.

Theorem 1. Transitivity Editing *on degree-three digraphs is NP-complete.*

Proof. Obviously, one can verify in polynomial time that a digraph is transitive. This implies that Transitivity Editing is in NP. We now show that it is NP-hard by reducing from 3SAT. Let $D(\varphi) = (V, A^{\text{var}} \cup A^{\text{cls}})$ be a digraph constructed as described above from a given instance φ of 3SAT. Clearly, the construction can be performed in polynomial time. We show that

> φ is satisfiable $\Leftrightarrow$ $(D(\varphi), 5m + 4mn)$ is a yes-instance for Transitivity Editing.

"⇒": Let β be a satisfying assignment of φ. Then, we can make $D(\varphi)$ transitive in the following way: First, for each variable x_i, if $\beta(x_i) = \text{true}$, then we delete all negative arcs of the variable cycle of x_i. Otherwise, we delete all positive arcs of the variable cycle of x_i. All in all, by deleting $4m$ arcs for each of the n variable cycles (which is a total of $4mn$ arc deletions), we destroyed all P_3s in variable cycles.

It remains to destroy the P_3s in clause gadgets. To this end, consider an arbitrary clause C_j, and let x_p, x_q, and x_r denote the variables occurring in C_j. Since β is a satisfying assignment for φ, some literal of C_j evaluates to true. Let x_k be the variable

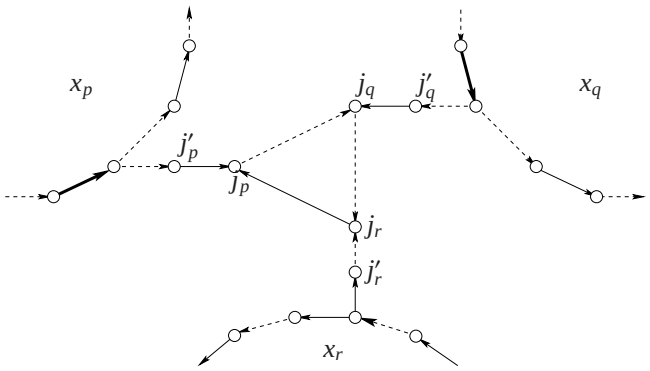


Figure 4: A solution for the clause gadget of clause C_j containing the variables x_p, x_q, and x_r. This solution deletes only five arcs from A_j^{cls} which is only possible if at least one of the corresponding docking arcs is deleted (in this case, $\alpha_{r,j}$). This situation represents that the literal corresponding to x_r in C_j evaluates to true and, thus, that C_j evaluates to true. Here, bold arcs are docking arcs and dashed arcs are deleted.

corresponding to this literal. Then $\alpha_{k,j}$ is deleted and, thus, Lemma 4 implies that the clause gadget can be made transitive with five arc deletions. For all clauses, this requires $5m$ arc deletions in total. In summary, it is possible to make $D(\varphi)$ transitive with $5m + 4mn$ arc deletions.

"$\Leftarrow$": Let S be a solution set for $D(\varphi)$ such that $|S| \leq 5m + 4mn$. Since $D(\varphi)$ is diamond-free, Lemma 3 allows us to assume that $S \subseteq A^{\text{var}} \cup A^{\text{cls}}$. We show that S contains exactly five arcs of each clause gadget and $4m$ arcs from each variable cycle. First, by Observation 3, making all m clause cycles transitive requires at least $5m$ arc deletions. Second, by Observation 1 making a variable cycle transitive requires at least $4m$ arc deletions. Since the variable cycles and clause gadgets are arc-disjoint, S contains exactly five arcs from each clause gadget and $4m$ arcs from each variable cycle. Moreover, by Observation 1, either all $4m$ positive arcs or all $4m$ negative arcs are in S.

In the following, we show that β with

$$\beta(x_i) := \begin{cases} \text{true} & \text{if all positive arcs of the variable cycle of } x_i \text{ are in } S \\ \text{false} & \text{otherwise} \end{cases}$$

is a satisfying assignment for φ. For the sake of contradiction, assume that there is some clause C_j that evaluates to false. Let x_p, x_q, and x_r denote the variables occurring in C_j. Since all literals of C_j evaluate to false, the definition of β and Observation 2 imply that none of the docking arcs of C_j are in S. Then, however, Lemma 4 implies $|S \cap A_j^{\text{cls}}| \geq 6$, a contradiction. □

In the above proof, we never employ arc insertions. This implies that Transitivity Deletion is also NP-complete.

Corollary 1. Transitivity Deletion *on degree-three digraphs is NP-complete.*

By slightly modifying the construction of the Transitivity Editing instance, we can also obtain exponential-time lower bounds. Consider an instance of Transitivity Editing that is constructed as described above with the following exceptions: Instead of creating for each variable x_i a cycle of length $8m$, we create a variable cycle of length $8\#(x_i)$ with $\#(x_i)$ denoting the number of clauses that contain x_i. Furthermore,

for the docking of the clauses, we assume that for each variable x_i there is an arbitrary but fixed ordering of the clauses that contain x_i. Let $\text{pos}(i, j)$ denote the position of clause C_j containing x_i in this ordering. Then, we define $\text{dock}(i, j) = 8 \cdot \text{pos}(i, j)$ if x_i occurs negated in C_j, and $\text{dock}(i, j) = 8 \cdot \text{pos}(i, j) + 1$ if x_i occurs nonnegated in C_j. Finally, we set $k := 5m + 4 \cdot \sum_{i=0}^{n-1} \#(x_i)$. In complete analogy to the proof of Theorem 1, we can show the equivalence of the 3SAT and Transitivity Editing instances.

Observe that in the constructed instance $((V, A), k)$ we have $k = O(m)$ and also $|V| = O(m)$ since $\sum_{i=0}^{n-1} \#(x_i) = 3m$ (as each clause contains exactly three variables). Hence, an algorithm with running time $2^{o(k)} \cdot \text{poly}(|V|)$ or $O(2^{o(|V|)})$ for Transitivity Editing implies an $O(2^{o(m)})$ time algorithm for solving 3SAT instances with m variables. The existence of such an algorithm implies subexponential-time algorithms for many other NP-hard problems as well [19]. It is therefore conjectured that 3SAT cannot be solved in this running time; this conjecture is commonly referred to as *exponential-time hypothesis*.

Theorem 2. Transitivity Editing *on degree-three digraphs cannot be solved in* $2^{o(k)} \cdot \text{poly}(|V|)$ *time or* $O(2^{o(|V|)})$ *time unless the exponential-time hypothesis fails.*

As in the case of the NP-hardness, our results also transfer to Transitivity Deletion.

Corollary 2. Transitivity Deletion *on degree-three digraphs cannot be solved in* $2^{o(k)} \cdot \text{poly}(|V|)$ *time or* $O(2^{o(|V|)})$ *time unless the exponential-time hypothesis fails.*

3.2. NP-Hardness on Acyclic Digraphs

Transitivity Editing's undirected "sister" problem Cluster Editing becomes polynomial-time solvable when the input is a forest, that is, acyclic.[5] It is thus natural to study the complexity of Transitivity Editing on acyclic digraphs. Somewhat surprisingly, we find that Transitivity Editing remains NP-hard for acyclic digraphs, unlike for example Disjoint Paths,

which is NP-hard in general [12], but polynomial-time solvable on acyclic digraphs [29].

The construction in Section 3.1 relies heavily on variable cycles and there are also cycles in the clause gadgets. To replace the variable cycles, we have to find acyclic gadgets that have exactly two optimal ways of being made transitive. Furthermore, we have to come up with replacements for the clause gadgets that need more modifications if the corresponding clause evaluates to false. Unfortunately, we could not realize these gadgets without giving up the bounded-degree constraint of the previous construction.

In the following, we present a many-one reduction from the NP-complete Positive-Not-All-Equal-3SAT problem [27].

> Positive-Not-All-Equal-3SAT (PNAE-3SAT):
> **Input**: A Boolean formula φ with n variables $x_0, \ldots, x_{n-1}$ which is a conjunction of m clauses $C_0, \ldots, C_{m-1}$, each consisting of three positive literals.
> **Question**: Is there a truth assignment for all n variables such that for each clause C_i exactly one or two of its variables are assigned true, that is, for no clause the truth values of its variables are all equal?

[5] In the context of Cluster Editing, it is common knowledge that assuming each vertex to be adjacent (in the input graph) to at least half of the vertices in its cluster is valid. Hence, in trees, each cluster contains at most two elements, implying that Cluster Editing in trees degenerates to maximum matching.

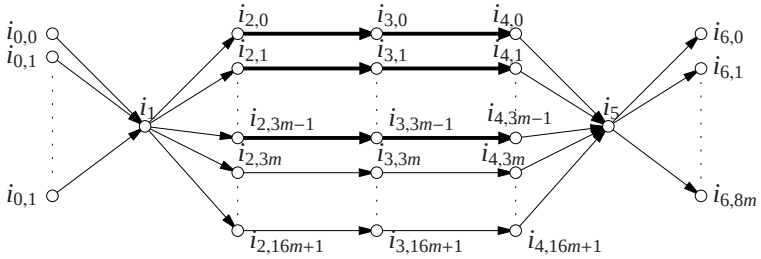


Figure 5: The variable gadget of x_i. The bold arcs show potential docking arcs (more details can be found in Figures 7 and 8), while the additional arc-disjoint P_3s $(i_1, i_{2,\ell}, i_{3,\ell})$ and $(i_{3,\ell}, i_{4,\ell}, i_5)$ with $\ell \geq 3m$ ensure that optimally making this structure transitive requires the deletion of either $(i_{0,\ell}, i_1)$ for each $0 \leq \ell \leq 8m$ or $(i_5, i_{6,\ell})$ for each $0 \leq \ell \leq 8m$.

Similarly to the approach described in Section 3.1, we construct variable gadgets that can optimally be made transitive in exactly two ways and clause gadgets that require more modifications if and only if the corresponding clause evaluates to false under a certain assignment. In the following, we present a formal description of the reduction.

Construction.. Given an instance φ of PNAE-3SAT, we construct a directed acyclic graph $D(\varphi) := (V^{\text{var}} \cup V^{\text{cls}}, A^{\text{var}} \cup A^{\text{cls}})$ as follows. For each of the n Boolean variables of φ, we construct a *variable gadget* (see Figure 5) that has exactly two ways of being made transitive using at most $40m+5$ arc modifications, which will represent assigning true or false, respectively, to the corresponding variable.

For each Boolean variable x_i, we construct the vertex set

$$V_i^{\text{var}} := \{i_1, i_5\} \cup \bigcup_{j=0}^{8m} \{i_{0,j}, i_{6,j}\} \cup \bigcup_{j=0}^{16m+1} \{i_{2,j}, i_{3,j}, i_{4,j}\}$$

and connect these vertices with the arcs

$$A_i^{\text{var}} := \bigcup_{\ell=0}^{16m+1} A_{i,\ell}^{\text{inner}} \cup \bigcup_{\ell=0}^{8m} A_{i,\ell}^{\text{outer}}, \text{ where}$$

$$\begin{aligned} A_{i,\ell}^{\text{inner}} &:= \{(i_1, i_{2,\ell}), (i_{2,\ell}, i_{3,\ell}), (i_{3,\ell}, i_{4,\ell}), (i_{4,\ell}, i_5)\} \text{ and} \\ A_{i,\ell}^{\text{outer}} &:= \{(i_{0,\ell}, i_1), (i_5, i_{6,\ell})\}. \end{aligned}$$

The collection of all variable gadgets is $(V^{\text{var}}, A^{\text{var}}) := (\bigcup_{\ell=0}^{n-1} V_i^{\text{var}}, \bigcup_{\ell=0}^{n-1} A_i^{\text{var}})$. The following arc-disjoint P_3s are contained in each variable gadget $(V_i^{\text{var}}, A_i^{\text{var}})$:

1. $(i_{0,\ell}, i_1, i_{2,\ell}), (i_{2,\ell}, i_{3,\ell}, i_{4,\ell}), (i_{4,\ell}, i_5, i_{6,\ell})$ for all $0 \leq \ell \leq 8m$, and
2. $(i_1, i_{2,\ell}, i_{3,\ell}), (i_{3,\ell}, i_{4,\ell}, i_5)$ for all $8m < \ell \leq 16m+1$.

All in all, there are $3 \cdot (8m+1) + 2 \cdot (8m+1) = 40m+5$ arc-disjoint P_3s in each variable gadget.

Observation 4. *For each variable gadget, at least* $40m + 5$ *arc modifications are required to make it transitive.*

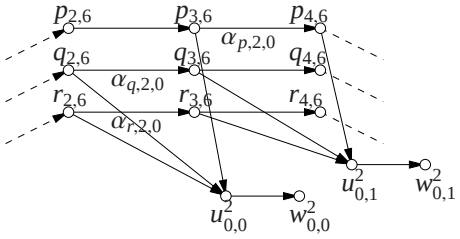


Figure 6: Part 0 of the clause gadget of clause $C_2 = \{x_p, x_q, x_r\}$. The three docking arcs are $(p_{3,6}, p_{4,6})$, $(q_{2,6}, q_{3,6})$, and $(r_{2,6}, r_{3,6})$.

We introduce the following notation for the vertices and arcs of the variable-gadgets. For each variable x_i, the vertices i_1, i_5 and $i_{3,\ell}$ with $\ell \leq 16m + 1$ are *negative vertices*. All other vertices in V^{var} are *positive vertices*. We refer to an arc (u, v) as *negative arc* if u is negative, otherwise (u, v) is called a *positive arc*. In analogy to Section 3.1, deleting all positive or all negative arcs corresponds to assigning true or false, respectively, to x_i.

In the following, we construct the *clause gadgets*: For each clause $C_j = \{x_p, x_q, x_r\}$ in the formula φ, we construct three gadget parts (part 0, 1, and 2) such that exactly one of them can be made transitive with exactly four arc modifications if and only if the variable gadgets of x_p, x_q, and x_r are not made transitive in the same way. Let $\text{part}_j(p) := 0$, $\text{part}_j(q) := 1$, and $\text{part}_j(r) := 2$. For part ℓ of the clause gadget of clause C_j, we construct the vertex set $V^{\text{cls}}_{j,\ell} := \{u^j_{\ell,0}, w^j_{\ell,0}, u^j_{\ell,1}, w^j_{\ell,1}\}$. The two vertices $u^j_{\ell,0}$ and $u^j_{\ell,1}$ are then connected to the variable gadgets, depending on ℓ:

$$A^{\text{cls}}_{i,j,\ell} := \begin{cases} \{(i_{3,3j+\ell}, u^j_{\ell,0}), (i_{4,3j+\ell}, u^j_{\ell,1})\}, & \text{if } \text{part}_j(i) = \ell \\ \{(i_{2,3j+\ell}, u^j_{\ell,0}), (i_{3,3j+\ell}, u^j_{\ell,1})\}, & \text{if } \text{part}_j(i) \neq \ell. \end{cases}$$

Similarly, we define the *docking arc* of part ℓ of the clause gadget of C_j in the variable gadget of x_i as $(i_{3,3j+\ell}, i_{4,3j+\ell})$ if $\text{part}_j(i) = \ell$, and as $(i_{2,3j+\ell}, i_{3,3j+\ell})$ otherwise, and denote it by $\alpha_{i,j,\ell}$ (see Figure 6 for an example).

Informally, the docking arcs are the three arcs of the corresponding variable gadgets (one arc for each variable gadget), whose start- and endpoint are connected to the clause gadget part. Furthermore, for each $i \in \{p, q, r\}$, the arc $\gamma_{i,j,\ell}$ denotes the arc in A^{var}_i that is incoming to the vertex that $\alpha_{i,j,\ell}$ is outgoing from (that is, the arc that precedes $\alpha_{i,j,\ell}$ in the variable gadget). Part ℓ of the clause gadget is completed by adding the two arcs in $A^{\text{cls}}_{j,\ell} := \{(u^j_{\ell,0}, w^j_{\ell,0}), (u^j_{\ell,1}, w^j_{\ell,1})\}$. For each clause C_j, we thus have the arc set

$$A^{\text{cls}}_j := \bigcup_{\ell=0}^{2} \left(A^{\text{cls}}_{j,\ell} \cup \bigcup_{i \in \{p,q,r\}} A^{\text{cls}}_{i,j,\ell} \right).$$

The idea behind the construction of the clause gadget parts is that we save one arc modification in a clause gadget part if and only if all or none of its docking arcs are deleted in the variable gadgets. If this is possible for one part of the clause gadget, then the three variable gadgets corresponding to the variables in the corresponding clause are not made transitive in the same way, hinting at the variables not being assigned equal values.

With $(V^{\text{cls}}, A^{\text{cls}}) := (\bigcup_{j=0}^{m-1} \bigcup_{\ell=0}^{2} V^{\text{cls}}_{j,\ell}, \bigcup_{j=0}^{m-1} A^{\text{cls}}_j)$ denoting the collection of all clause gadgets, we finally set $D(\varphi) := (V^{\text{var}} \cup V^{\text{cls}}, A^{\text{var}} \cup A^{\text{cls}})$. Note that $D(\varphi)$ is acyclic and

diamond-free.

Correctness.. By the construction of the clause gadgets, it is clear that each gadget part requires at least two arc modifications to be made transitive.

Observation 5. *For each clause gadget, at least six arc modifications are required to make it transitive.*

Furthermore, by the construction of $A^{\text{cls}}_{i,j,\ell}$, each part of a clause gadget docks over the negative arc $(i_{3,3j+\ell}, i_{4,3j+\ell})$ in case $\text{part}_j(i) = \ell$ and over the positive arc $(i_{2,3j+\ell}, i_{3,3j+\ell})$, otherwise.

Observation 6. *Each clause gadget part docks over one negative arc and two positive arcs.*

In order to show that the construction of $D(\varphi)$ yields a many-one reduction, we need the following lemmas.

First, we show that in order to make each variable gadget transitive without using too many arc modifications, either all arcs that are incoming to i_1 or all arcs that are outgoing from i_5 must be deleted, but not both.

Lemma 5. *Let S be a solution set for $D(\varphi)$ with $|S| \leq n\cdot(40m+5)+14m$ and $S = S_{\text{DEL}}$. Then for each variable x_i of φ, either*

$$\forall 0 \leq \ell \leq 8m\ (i_{0,\ell}, i_1) \in S_i \text{ or } \forall 0 \leq \ell \leq 8m\ (i_5, i_{6,\ell}) \in S_i,$$

where $S_i := S \cap A^{var}_i$.

Proof. Let D' denote the result of applying S to $D(\varphi)$.

First, we show that at most one of the two statements in the lemma is true. For the sake of contradiction, assume that

$$\forall 0 \leq \ell \leq 8m\ \{(i_{0,\ell}, i_1), (i_5, i_{6,\ell})\} \subseteq S_i.$$

This leaves $2\cdot(16m+2)$ arc-disjoint P_3s in the center of the gadget. Hence, $2\cdot(8m+1)+2\cdot(16m+2) = 48m+6$ arc deletions are then required for this gadget. By Observation 4, we need at least $40m+5$ arc deletions for each of the other $n-1$ variable gadgets and by Observation 5, we need at least six arc deletions for each of the m clause gadgets. The overall number of arc deletions needed is thus at least $(n-1)\cdot(40m+5)+48m+6+6m = n\cdot(40m+5)+14m+1$, a contradiction.

Next, we show that at least one of the two statements is true. For the sake of contradiction, assume that

$$\exists 0 \leq \ell \leq 8m (i_{0,\ell}, i_1) \notin S_i \text{ and } \exists 0 \leq \ell \leq 8m (i_5, i_{6,\ell}) \notin S_i.$$

For each $0 \leq \ell \leq 16m+1$, we have $(i_1, i_{2,\ell}) \in S_i$ since otherwise there is a P_3 $(i_{0,\ell}, i_1, i_{2,\ell})$ in D'. Similarly, the arcs $(i_{4,\ell}, i_5)$, $0 \leq \ell < 16m+1$, are deleted. This requires already $2\cdot(16m+2)$ modifications and leaves $16m+2$ arc-disjoint P_3s in the center of the gadget. Hence, $2\cdot(16m+2)+16m+2 = 48m+6$ arc deletions are required for this gadget. Again, this leads to $|S| \geq (n-1)\cdot(40m+5)+48m+6+6m = n\cdot(40m+5)+14m+1$, a contradiction. □

Since there is either an arc incoming to i_1 that is not deleted or there is an arc outgoing from i_5 that is not deleted, we know that either all arcs that are outgoing from i_1 are deleted or all arcs incoming to i_5 are deleted.

For the following consideration, we introduce the notion of "proper" solution sets and show that we can assume for an optimal solution that it is proper. We call a solution set S for $D(\varphi)$ *proper*, if for all variable gadgets, S contains either all positive arcs and none of the negative arcs or vice versa.

Lemma 6. *If there is an optimal solution set S for $D(\varphi)$ with $|S| \leq n \cdot (40m + 5) + 14m$, then there is also a proper optimal solution set for $D(\varphi)$.*

Proof. Since $D(\varphi)$ is diamond-free, Lemma 3 allows us to assume that S contains only arc deletions. Let D' denote the result of applying S to $D(\varphi)$.

If S is proper, then we are done. Hence, assume that there is a variable x_i of φ such that S does not delete exactly the positive arcs or the negative arcs of the variable gadget of x_i. We show that S can be modified such that either the positive arcs or the negative arcs of the variable gadget are deleted without increasing the size of the solution set. By Lemma 5, either all $(i_{0,\ell}, i_1)$ or all $(i_5, i_{6,\ell})$ with $0 \leq \ell \leq 8m$ are deleted. Since the proof works analogously for both cases, we only consider the case $(i_{0,\ell}, i_1) \in S$ for all $0 \leq \ell \leq 8m$ and $(i_5, i_{6,\ell^*}) \notin S$ for some $0 \leq \ell^* \leq 8m$. More precisely, we show that in this case, there is an optimal solution that deletes exactly the positive arcs of the variable gadget of x_i.

Clearly, $(i_{4,\ell}, i_5) \in S$ for all $0 \leq \ell \leq 16m + 1$ since otherwise $(i_{4,\ell}, i_5, i_{6,\ell^*})$ is a P_3 in D'.

In the following, we show that for each clause C_j and each part ℓ of the clause gadget of C_j, we can modify S such that it is optimal and

$$S \cap A^{\text{inner}}_{i,3j+\ell} = \{(i_{2,3j+\ell}, i_{3,3j+\ell}), (i_{4,3j+\ell}, i_5)\}. \tag{1}$$

Recall that the docking arc $\alpha_{i,j,\ell}$ of part ℓ of the clause gadget of clause C_j is either the arc $(i_{2,3j+\ell}, i_{3,3j+\ell})$ or the arc $(i_{3,3j+\ell}, i_{4,3j+\ell})$ and consider both cases:

Case 1: $\alpha_{i,j,\ell} = (i_{2,3j+\ell}, i_{3,3j+\ell})$.
First, suppose that $(i_{2,3j+\ell}, i_{3,3j+\ell}) \notin S$. Then, $(i_{3,3j+\ell}, i_{4,3j+\ell}) \in S$. Note that the arc $(i_{3,3j+\ell}, i_{4,3j+\ell})$ is only contained in two P_3s in $D(\varphi)$, one of which is destroyed by the deletion of $(i_{4,3j+\ell}, i_5)$. Hence, we can replace $(i_{3,3j+\ell}, i_{4,3j+\ell})$ with $(i_{2,3j+\ell}, i_{3,3j+\ell})$ in S without creating a P_3 in D'. This lets us assume that S contains $(i_{2,3j+\ell}, i_{3,3j+\ell})$.

It remains to show that there is an optimal solution that does not delete $(i_1, i_{2,3j+\ell})$. Since i_1 is a source in D' and $(i_{2,3j+\ell}, i_{3,3j+\ell}) \in S$, we can delete $(i_{2,3j+\ell}, u^j_{\ell,0})$ instead of $(i_1, i_{2,3j+\ell})$, satisfying (1) in this case.

Case 2: $\alpha_{i,j,\ell} = (i_{3,3j+\ell}, i_{4,3j+\ell})$.
In this case, we can assume that $(i_{2,3j+\ell}, i_{3,3j+\ell}) \in S$, since otherwise, we can replace $(i_1, i_{2,3j+\ell})$ with $(i_{2,3j+\ell}, i_{3,3j+\ell})$ in S because i_1 is a source in D' and $i_{2,3j+\ell}$ has only one outgoing arc in $D(\varphi)$. Then, however, deleting $(i_{4,3j+\ell}, i_5)$ and $(i_{2,3j+\ell}, i_{3,3j+\ell})$ already destroys two of the three P_3s containing $(i_{3,3j+\ell}, i_{4,3j+\ell})$. Hence, if $(i_{3,3j+\ell}, i_{4,3j+\ell}) \in S$, then we delete $(i_{4,3j+\ell}, u^j_{\ell,1})$ instead of $(i_{3,3j+\ell}, i_{4,3j+\ell})$, satisfying (1) in this case.

Finally, note that if we delete all positive arcs of a variable gadget, then it becomes transitive and, hence, additional arc deletions imply a contradiction to the optimality of S.

Since the presented modifications of S do not modify any other variable gadgets or arcs that are incident to other variable gadgets, we can repeatedly apply these modifications and eventually obtain a proper optimal solution set. □

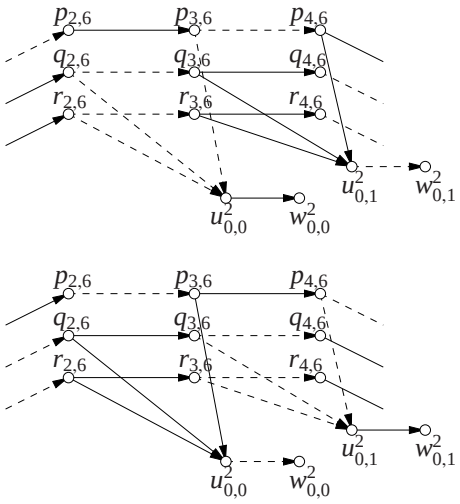


Figure 7: Part 0 of the clause gadget of clause $C_2 = \{x_p, x_q, x_r\}$ with all or no docking arcs being deleted. Dashed lines indicate arcs that are deleted. Here, four arc deletions suffice. This situation corresponds to $\beta(x_p) \neq \beta(x_q) = \beta(x_r)$ for an assignment β.

Having established this knowledge about variable gadgets, we continue by considering clause gadgets. In particular, we show that for making each clause gadget transitive without using too many arc modifications, there must be a part ℓ such that either all or none of the docking arcs of part ℓ are deleted.

Lemma 7. *Let S be an optimal solution set for $D(\varphi)$, let $C_j = \{x_p, x_q, x_r\}$ be a clause of φ and let ℓ be a part of its clause gadget. Furthermore, for each $i \in \{p, q, r\}$ let exactly one of $\alpha_{i,j,\ell}$ and $\gamma_{i,j,\ell}$ be in S. If*

$$\forall i \in \{p, q, r\}\ \alpha_{i,j,\ell} \in S \text{ or } \forall i \in \{p, q, r\}\ \gamma_{i,j,\ell} \in S,$$

then $|S \cap A^{cls}_{j,\ell}| = 4$. Otherwise, $|S \cap A^{cls}_{j,\ell}| = 5$.

Proof. Since $D(\varphi)$ is diamond-free, Lemma 3 allows us to assume that S contains only arc deletions.

Suppose that the premise is true, that is, ℓ is a gadget part of the clause gadget corresponding to C_j such that all $\alpha_{i,j,\ell}$ or all $\gamma_{i,j,\ell}$ are deleted. We show only the case that all $\alpha_{i,j,\ell}$ are deleted, the case that all $\gamma_{i,j,\ell}$ are deleted can be shown analogously. Since for each $i \in \{p, q, r\}$, exactly one of the arcs $\alpha_{i,j,\ell}$ and $\gamma_{i,j,\ell}$ is deleted, we know that $\gamma_{i,j,\ell}$ is not deleted for each $i \in \{p, q, r\}$. Figure 7 shows that it is possible to make part ℓ of the clause gadget corresponding to clause C_j transitive with four arc deletions. As also shown in Figure 7, applying $S \setminus (A^{\text{cls}}_{j,\ell} \cup \bigcup_{i\in\{p,q,r\}} A^{\text{cls}}_{i,j,\ell})$ to $D(\varphi)$ leaves four arc-disjoint P_3s. Hence, four arc deletions are also required.

Suppose that the premise is false, that is, ℓ is a gadget part for which there is some $\alpha_{i,j,\ell}$ that is not deleted and there is also some $\gamma_{l,j,\ell}$, $l \neq i$ that is not deleted. Figure 8 shows that it is possible to make the gadget part ℓ of the clause gadget corresponding to clause C_j transitive with five arc deletions. As also shown in Figure 8, applying $S \setminus (A^{\text{cls}}_{j,\ell} \cup \bigcup_{i\in\{p,q,r\}} A^{\text{cls}}_{i,j,\ell})$ to $D(\varphi)$ leaves five arc-disjoint P_3s. Hence, at least five arc deletions are required. □

Next, we use Lemma 7 to make a similar statement for clause gadgets as a whole. Namely, we can observe that if there is some part of a clause gadget that can be made

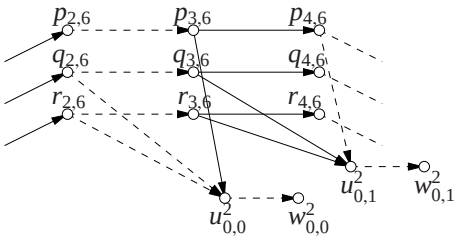


Figure 8: Part 0 of the clause gadget of clause $C_2 = \{x_p, x_q, x_r\}$ with some but not all docking arcs being deleted. Dashed lines indicate arcs that are deleted. Here, at least five arc deletions are required. This situation corresponds to $\beta(x_p) = \beta(x_q) = \beta(x_r)$ for an assignment β of all variables.

transitive with four arc deletions, then we need five arc deletions for each of the other two parts.

Lemma 8. *Let S be a proper optimal solution set for $D(\varphi)$ and let $C_j = \{x_p, x_q, x_r\}$ be a clause of φ. If there is a gadget part ℓ such that $|S \cap A^{cls}_{j,\ell}| = 4$,*

then $|S \cap A^{cls}_j| = 14$. Otherwise, $|S \cap A^{cls}_j| = 15$.

Proof. Since $D(\varphi)$ is diamond-free, Lemma 3 allows us to assume that S contains only arc deletions.

Suppose that the premise is true, that is, there is a gadget part ℓ such that all its docking arcs are deleted or none of its docking arcs are deleted. By Lemma 7, four arc deletions are required to make part ℓ transitive. Since two parts of any clause gadget share exactly one docking arc and differ in two docking arcs, there can be no second part ℓ' of the same gadget such that either all or none of the docking arcs of ℓ' are deleted. By Lemma 7, we thus need five arc deletions each for making the other two parts transitive. Overall, the total number of required arc deletions is 14.

Now suppose that the premise is false, that is, for each gadget part ℓ, there is some $i \in \{p, q, r\}$ and some $i' \in \{p, q, r\}$, $i' \neq i$, such that $\alpha_{i,j,\ell}, \gamma_{i',j,\ell} \notin S$. Then, by Lemma 7, each of the three parts requires five arc deletions, thus all three parts require a total of 15 arc deletions. □

With Lemmas 6 and 8 at hand, we can show that the construction described above is indeed a many-one reduction from PNAE-3SAT to Transitivity Editing.

Theorem 3. Transitivity Editing *on acyclic digraphs is NP-complete.*

Proof. Obviously, one can verify in polynomial time that a digraph is transitive and, thus, containment in NP is clear.

Next, we show that Transitivity Editing is also NP-hard. Let φ be an instance of PNAE-3SAT and let $D(\varphi)$ be as described above. Clearly, the construction of $D(\varphi)$ runs in polynomial time and $D(\varphi)$ is acyclic. It is thus sufficient to show the following:

> φ is a yes-instance of PNAE-3SAT $\Leftrightarrow$ $(D(\varphi), n \cdot (40m + 5) + 14m)$ is a yes-instance of Transitivity Editing.

"$\Rightarrow$": Let β be a satisfying assignment, that is, an assignment to the variables of φ such that there is no clause whose variables are all assigned the same truth value. From this assignment, we can construct a solution of the Transitivity Editing instance as follows. First, for each variable gadget $(V^{\text{var}}_i, A^{\text{var}}_i)$, we delete all positive arcs of the variable gadget if $\beta(x_i) =$ true, and all negative arcs if $\beta(x_i) =$ false. This clearly makes

each variable gadget transitive and requires $n \cdot (40m + 5)$ arc modifications overall. It remains to make the clause gadgets transitive. Since β is a satisfying assignment, there is no clause whose variables are assigned the same truth value. Thus, each clause gadget docks to at least one variable gadget whose negative arcs are deleted and at least one variable gadget whose positive arcs are deleted. Hence, there is exactly one part of each clause gadget for which either all or none of its docking arcs are deleted. By Lemma 8, we can make each clause gadget transitive with 14 arc deletions. Overall, we can thus make $D(\varphi)$ transitive with a total of $n \cdot (40m + 5) + 14m$ arc deletions.

"$\Leftarrow$": Let S be an optimal solution set for $D(\varphi)$ such that $|S| \leq n \cdot (40m + 5) + 14m$. Since $D(\varphi)$ is diamond-free, Lemma 3 allows us to assume that S contains only arc deletions. Let D' denote the result of applying S to $D(\varphi)$. By Lemma 6, we can assume S to be proper.

In the following, we construct a satisfying assignment β for the variables of the given formula φ from S:

$$\beta(x_i) := \begin{cases} \text{true}, & \text{if } (i_{0,0}, i_1) \in S \\ \text{false}, & \text{otherwise.} \end{cases}$$

Since $|S| \leq n \cdot (40m + 5) + 14m$, Observation 4 and Lemma 8 imply that $|S \cap A_j^{\text{cls}}| = 14$ for all $0 \leq j < m$ and, thus, that for each clause gadget there is some part such that all or none of its docking arcs are deleted. Since, in each variable gadget, either all negative arcs or all positive arcs are deleted, Observation 6 implies that out of the three variable gadgets that the clause gadget is connected to, there is one having all its positive arcs in S and one having all its negative arcs in S. Hence, β is a satisfying assignment for the variables of φ.

All in all, the given instance of PNAE-3SAT is a yes-instance if and only if $(D(\varphi), n \cdot (40m + 5) + 14m)$ is a yes-instance of Transitivity Editing. □

In the proof of Theorem 3, we never employ arc insertions which implies that it can be used to prove that Transitivity Deletion is NP-complete on dags.

Corollary 3. Transitivity Deletion *on acyclic digraphs is NP-complete.*

Note that the construction employed here does not allow deriving subexponential lower bounds on the running time for Transitivity Editing and Transitivity Deletion on acyclic digraphs, as we did for bounded-degree digraphs in Section 3.1.

4. Fixed-Parameter Tractability Results

In this section, we complement the NP-hardness results of the previous section with encouraging algorithmic results. Böcker et al. [3] observed that in many applications the input graphs are "almost transitive". Consequently, as Böcker et al. [3], we study how the parameter k (denoting the number of arc modifications) influences the computational complexity. We deliver improved fixed-parameter tractability results; in particular, we positively answer Böcker et al.'s [3] question for the existence of a polynomial-size problem kernel. In the following, we first develop kernelization results, and then we present an improved search tree strategy, altogether yielding the so far fastest fixed-parameter algorithm for Transitivity Editing. Furthermore, we also provide similar results for Transitivity Deletion. In this section, we use n and m to denote $|V|$ and $|A|$, respectively.

First, observe that Transitivity Editing is easily classified as fixed-parameter tractable with respect to the parameter k: The task is simply to destroy all P_3s. Note that, by inspecting all pairs $(u,(v,w)) \in V \times A$ and testing for $(u,v) \in A$ and $(u,w) \notin A$, all P_3s can be found in $O(nm)$ time .

Once a P_3 is found, there are exactly three possibilities to destroy a P_3: either delete one of the two arcs or insert the "missing" one. This yields a search tree of size $O(3^k)$ (cf. [3]), which indeed can be used to enumerate *all* minimal solutions of size at most k because it exhaustively tries all possibilities to destroy P_3s.

4.1. Kernelization

In this section, we show a problem kernel for Transitivity Editing consisting of $O(k^2)$ vertices on general graphs and a problem kernel of $O(k)$ vertices on digraphs with bounded degree. In the latter case, already the following data reduction rule suffices.

Rule 1. *Let $(D = (V,A),k)$ be an input instance of* Transitivity Editing. *If there is a vertex $u \in V$ that does not take part in any P_3 in D, then delete u and all arcs that are incident to u.*

In order to prove the correctness of Rule 1, we observe that for each arc (u,v) that is inserted by an optimal solution set, the vertex v is reachable from u in the original digraph.

Lemma 9. *Let $D = (V,A)$ be a digraph and let S be an optimal solution set for D such that there is an arc $(u,v) \in S \setminus A$. Then, v is reachable from u in D.*

Proof. Let $V_u \subseteq V$ denote the vertices that are reachable from u in D (including u itself). For the sake of contradiction, assume that $v \notin V_u$. Since $(u,v) \in S$ and S is optimal, we know that $S' := S \setminus (V_u \times (V \setminus V_u))$ is not a solution set for D. Hence, there is a P_3 (x,y,z) in $(V, A \Delta S')$ that is not in $(V, A \Delta S)$, implying $x \in V_u$ and $z \in V \setminus V_u$.

This is a contradiction to the fact that $(A \Delta S') \cap (V_u \times (V \setminus V_u)) = \emptyset$.

□

With this lemma, we can now prove the correctness of Rule 1.

Lemma 10. *Rule 1 is correct and can be exhaustively applied in $O(nm)$ time.*

Proof. Let $D = (V,A)$ be a digraph and let $D' = (V',A')$ denote the result of applying Rule 1 to D. In the following, we show that

$$(D,k) \text{ is a yes-instance} \Leftrightarrow (D',k) \text{ is a yes-instance.}$$

"$\Rightarrow$": This direction follows directly from the fact that D' is an induced subgraph of D and that transitivity is a hereditary property, that is, it is closed under vertex deletion.

"$\Leftarrow$": Let S denote an optimal solution set for D'. We show that there is also a size-$|S|$ solution set for D. If S is a solution set for D, then we are done. Otherwise, there is a P_3 (u,v,w) in $(V, A \Delta S)$ that is not in $(V', A' \Delta S)$. Hence, applying Rule 1 deleted either u, v, or w. We consider these cases individually.

Case 1: Rule 1 deleted u.
Since u is not a vertex of D', we can assume that $(u,w) \notin S$ and since $(u,w) \notin A \Delta S$, it is clear that $(u,w) \notin A$. Likewise, $(u,v) \in A$ and since (u,v,w) is not a P_3 in D, we know that $(v,w) \in S \setminus A$. By Lemma 9, we thus know that w is reachable from u in D.

However, u not taking part in any P_3 in D implies $(u, w) \in A$, a contradiction. Note that the case that w is deleted by Rule 1 is completely analogous to this case and is therefore omitted.

Case 2: Rule 1 deleted v.

Then, v not taking part in any P_3 in D implies that, in D, all vertices from which v is reachable are predecessors of all vertices that are reachable from v. By Lemma 9, this extends from D to $(V, A \cup S_{\text{INS}})$. Let S' denote the result of deleting from S all arcs from predecessors of v to successors of v in D. Since $(u, w) \in S_{\text{DEL}}$, we know that $|S'| < |S|$ and since S is optimal, there is a P_3 (x, y, z) in $(V, A \Delta S')$ where x is a predecessor of v and z is a successor of v in D. This, however, implies that $(x, z) \in A \setminus S'$, a contradiction.

Finally, the running time can be seen as follows. We enumerate all pairs $(u, (v, w)) \in V \times A$ and mark u, v, and w if (u, v, w) is a P_3. Afterwards, we delete all unmarked vertices. This procedure can be performed in $O(nm)$ time. □

Clearly, the proof of Lemma 10 also works without Lemma 9 if $S_{\text{INS}} = \emptyset$, proving correctness of Rule 1 also for Transitivity Deletion.

In the following, we show that Rule 1 already implies a problem kernel with a linear number of vertices if the maximum degree of the given digraph is constant.

Theorem 4. *For* Transitivity Editing *restricted to degree-d digraphs, we can compute admits a problem kernel containing at most $2k \cdot (d + 1)$ vertices that can be computed in $O(nm)$ time.*

Proof. Let $D = (V, A)$ be a digraph that is reduced with respect to Rule 1 and let S be a solution set for D with $|S| \leq k$. We show that $|V| \leq 2k(d+1)$. Consider the two-partition of V into $Y := \{v \in V \mid \exists u \in V\ (u, v) \in S \vee (v, u) \in S\}$ and $X := V \setminus Y$. Since $|S| \leq k$, we have $|Y| \leq 2k$. Note that, since D is reduced with respect to Rule 1, every $x \in X$ is contained in a P_3 q. It is clear that the other two vertices of q are in Y and thus every $x \in X$ is adjacent to at least one vertex in Y. However, each vertex in Y has at most d neighbors and thus $|X| \leq d|Y|$, implying $|V| = |X|+|Y| \leq 2k+d \cdot 2k = 2k(d+1)$. □

In the proof of the kernel bound, we actually only need that each remaining vertex is in some P_3, and since Rule 1 is also correct for Transitivity Deletion, the bound still holds for Transitivity Deletion.

Corollary 4. Transitivity Deletion *restricted to degree-d digraphs admits a problem kernel containing at most $2k \cdot (d + 1)$ vertices that can be computed in $O(nm)$ time.*

Next, we prove an $O(k^2)$-vertex kernel for general digraphs.

The following data reduction rule roughly follows an idea for Cluster Editing [14]: If there is some vertex pair (u, v) such that not modifying (u, v) results in a solution size of at least $k + 1$, then every solution of size at most k must contain (u, v).

Rule 2. *Let $(D = (V, A), k)$ be an input instance of* Transitivity Editing.

1. *Let $(u, v) \notin A$ and let $Z := \text{succ}_A(u) \cap \text{pred}_A(v)$. If $|Z| > k$, then insert (u, v) into A and decrease k by one.*

2. *Let $(u, v) \in A$, let $Z_u := \text{pred}_A(u) \setminus \text{pred}_A(v)$ and let $Z_v := \text{succ}_A(v) \setminus \text{succ}_A(u)$. If $|Z_u| + |Z_v| > k$, then delete (u, v) from A and decrease k by one.*

Lemma 11. *Rule 2 is correct and can be exhaustively applied in $O(n^3)$ time.*

Proof. Let $(D^*, k-1)$ with $D^* = (V, A^*)$ denote the instance that is obtained by applying Rule 2 to the given instance (D, k) with $D = (V, A)$. Furthermore, let (u, v) be the arc that is modified by Rule 2. For the correctness of the rule, we show that every solution set S for D with $|S| \leq k$ contains (u, v). Assume that there is a solution set S for D with $(u, v) \notin S$. We show that $|S| > k$.

If (u, v) is an arc insertion, then for $Z := \text{succ}_A(u) \cap \text{pred}_A(v)$ we have $|Z| > k$. Hence, for each $w \in Z$, (a, w, b) is a P_3 in D. Since $(a, b) \notin S$, at least one of the two arcs (a, w) and (w, b) is modified for each $w \in Z$. Since $|Z| > k$, we have $|S| > k$.

If (u, v) is an arc deletion, then we have $|Z_u| + |Z_v| > k$ for $Z_u := \text{pred}_A(u) \setminus \text{pred}_A(v)$ and $Z_v := \text{succ}_A(v) \setminus \text{succ}_A(u)$. Hence, for all $w \in Z_u$, (w, u, v) is a P_3 in D and for all $z \in Z_v$, (u, v, z) is a P_3 in D. Since $(u, v) \notin S$, for each $w \in Z_u \cup Z_v$, at least one arc incident to w has to be inserted or deleted. Since $|Z_u| + |Z_v| > k$ and since Z_u and Z_v are disjoint, $|S| > k$ follows.

It remains to show the running time. We show that, given any pair of vertices, we can execute Rule 2 in $O(n)$ time. Let $u, v \in V$. If $(u, v) \notin A$, then we compute the size of $\text{succ}_A(u) \cap \text{pred}_A(v)$, which can be done in $O(n)$ time. If $(u, v) \in A$, then we compute the sizes of $\text{pred}_A(u) \setminus \text{pred}_A(v)$ and $\text{succ}_A(v) \setminus \text{succ}_A(u)$, which can also be done in $O(n)$ time. Obviously, inserting or deleting (u, v) can be done in $O(n)$ time as well. □

Finally, we show that the exhaustive application of Rules 1 and 2 leads to a problem kernel containing $O(k^2)$ vertices.

Theorem 5. Transitivity Editing *admits a problem kernel containing at most $k(k+2)$ vertices and it can be computed in $O(n^3)$ time.*

Proof. Assume that there is a digraph $D = (V, A)$ with $|V| > k(k+2)$, that D is reduced with respect to Rules 1 and 2, and that it is possible to make D transitive by applying at most k arc modifications. Let $D' = (V, A')$ denote a transitive digraph obtained by the application of k arc modifications and let $S := A \Delta A'$ denote the corresponding solution set. Consider a two-partition (X, Y) of V, where $Y := \{v \in V \mid \exists u \in V\ (u, v) \in S \vee (v, u) \in S\}$ and $X := V \setminus Y$. Note that all vertices in X are adjacent to at least one vertex in Y because D is reduced with respect to Rule 1. Also note that in order to destroy a P_3 p in D, the solution set S must contain an arc incident to two of the vertices of p, hence for each P_3 p in D at most one of the vertices of p is in X.

Since we assume that D can be made transitive with at most k arc modifications, we know that $|S| \leq k$ and consequently $|Y| \leq 2k$. Clearly, $|V| = |X| + |Y|$, hence $|V| > k(k+2)$ implies $|X| > k^2$. With the above observation, it follows that there are more than k^2 P_3s in D.

For each $(a, b) \in S$, let $Z_{(a,b)} := \{p \mid \text{modifying } (a, b) \text{ destroys the } P_3\ p \text{ in } D\}$. Since there are more than k^2 P_3s in D but $|S| \leq k$, we know that there is a $(u, v) \in S$ with $|Z_{(u,v)}| > k$. We show that Rule 2 applies to (u, v) contradicting the fact that D is reduced.

If (u, v) is an arc insertion, then the set of P_3s destroyed by adding (u, v) is $Z_{(u,v)} := \{(u, w, v) \mid w \in \text{succ}_A(u) \cap \text{pred}_A(v)\}$. Since $|Z_{(u,v)}| > k$, we have $|Z| > k$ for $Z := \text{succ}_A(u) \cap \text{pred}_A(v)$. Hence, Rule 2 applies in this case.

If (u, v) is an arc deletion, then the P_3s destroyed by deleting (u, v) is $Z_{(u,v)} := \{(u, w, v) \mid w \in \text{pred}_A(u) \setminus \text{pred}_A(v) \vee w \in \text{succ}_A(v) \setminus \text{succ}_A(u)\}$. Since $|Z_{(u,v)}| > k$, we have $|Z_u| + |Z_v| > k$ for $Z_u := \text{pred}_A(u) \setminus \text{pred}_A(v)$ and $Z_v := \text{succ}_A(v) \setminus \text{succ}_A(u)$. Hence, Rule 2 applies.

It remains to show the running time. Note that if Rule 2 has been exhaustively applied, then the exhaustive application of Rule 1 does not create any vertex pair (u, v) to which Rule 2 applies. Therefore, exhaustively applying Rule 2 in $O(n^3)$ time and then exhaustively applying Rule 1 in $O(n^3)$ time yields an instance that is reduced with respect to both rules. □

Rule 2 also works for Transitivity Deletion if Case 1 is omitted. The analysis of the kernel size is similar to the proof of Theorem 5.

Corollary 5. Transitivity Deletion *admits a problem kernel containing at most* $k(k + 2)$ *vertices and it can be computed in* $O(n^3)$ *time.*

4.2. Search Tree Algorithm

As mentioned before, a straightforward algorithm that finds an optimal solution set for a given digraph branches on each P_3 (u, v, w) in the digraph, trying to destroy it by either deletion of (u, v), deletion of (v, w), or insertion of (u, w). This directly gives a search tree algorithm solving transitivity Editing on an n-vertex digraph in $O(3^k \cdot n^3)$ time (cf. [3]).

For Transitivity Deletion, the search only needs to branch into two cases, yielding an algorithm running in $O(2^k \cdot n^3)$ time. Using the so-called interleaving technique [25, 24] together with the polynomial-size problem kernel results, however, one actually can achieve running times $O(3^k + n^3)$ and $O(2^k + n^3)$, respectively.

In the following, we shrink the search tree size for Transitivity Editing from 3^k to 2.57^k by applying our combinatorial result on diamond-freeness (Lemma 3).

Theorem 6. Transitivity Editing *and* Transitivity Deletion *can be solved in* $O(2.57^k + n^3)$ *and* $O(2^k + n^3)$ *time, respectively.*

Proof. The modified algorithm employs the following search structure. Upon finding a diamond $(u, \{x, y\}, v)$ in the given digraph $D = (V, A)$, the algorithm recursively asks whether

1. $(V, A \setminus \{(u, x), (u, y)\})$ can be made transitive with $\leq k - 2$ operations,
2. $(V, A \setminus \{(u, x), (y, v)\})$ can be made transitive with $\leq k - 2$ operations,
3. $(V, A \setminus \{(x, v), (u, y)\})$ can be made transitive with $\leq k - 2$ operations,
4. $(V, A \setminus \{(x, v), (y, v)\})$ can be made transitive with $\leq k - 2$ operations, or
5. $(V, A \cup \{(u, v)\})$ can be made transitive with $\leq k - 1$ operations.

Thus, the search branches into five cases and the recurrence for the corresponding search tree size reads as $T_k = O(1) + 4 \cdot T_{k-2} + T_{k-1}$, where $T_0 = T_1 = 1$. Resolving this recurrence yields $O(2.57^k)$ for the search tree size under the assumption that the branching is always performed in this way. Since all other ways of destroying an encountered diamond include all modifications of one of the above cases, this branching strategy is correct. If there are no diamonds in the input graph, then the straightforward search tree for Transitivity Deletion is used to solve the problem, which runs in $O(2^k \cdot n^3)$ time. The correctness of the overall search tree algorithm easily follows.

Applying the interleaving technique [25, 24], and making use of the polynomial-size problem kernels (see Theorem 5) results in algorithms solving Transitivity Editing in $O(2.57^k + n^3)$ time and Transitivity Deletion in $O(2^k + n^3)$ time. □

5. Conclusion

We studied Transitivity Editing and some related problems. While Transitivity Completion is solvable in $O(n^{2.376})$ time [23], we have seen that both Transitivity Editing and Transitivity Deletion are NP-complete, even when restricted to dags or degree-three digraphs. We have shown that both problems admit a problem kernel containing at most $k(k+2)$ vertices and that this kernel can be computed in $O(n^3)$ time. Furthermore, we presented a fixed-parameter algorithm for Transitivity Editing that runs in $O(2.57^k + n^3)$ time, improving on the obvious $O(3^k \cdot n^3)$ time algorithm.

Two immediate challenges arising from our work are to determine whether there is an $O(k)$-vertex problem kernel for Transitivity Editing or Transitivity Deletion in the case of general digraphs (see [10, 15, 7] for corresponding results in the case of undirected graphs, that is, Cluster Editing) or to improve the running time of the kernelization, which so far takes cubic time in the number of vertices (for Cluster Editing, even linear-time kernelization is possible [26]).

Another interesting open question is whether there is a polynomial-time approximation algorithm for Transitivity Editing and/or Transitivity Deletion.

Finally, note that we focused on arc modifications to make a given digraph transitive—it might be of similar interest to start an investigation of the Transitivity Vertex Deletion problem, where the graph shall be made transitive by as few *vertex deletions* as possible (see [17] for corresponding results in the case of undirected graphs, that is, Cluster Vertex Deletion). Finally, from a more general point of view, there seems to be a rich field of studying further modification problems on digraphs. For instance, the concept of quasi-transitivity is of considerable interest in the theory of directed graphs (cf. [1]), hence one might start investigations on problems such as Quasi-Transitivity Editing.

References

[1] J. Bang-Jensen and G. Gutin. *Digraphs: Theory, Algorithms and Applications*. Springer, 2nd edition, 2009.

[2] S. Böcker. A golden ratio parameterized algorithm for cluster editing. In *Proceedings of the 22nd International Workshop on Combinatorial Algorithms (IWOCA '11)*, 2011. To appear.

[3] S. Böcker, S. Briesemeister, and G. W. Klau. On optimal comparability editing with applications to molecular diagnostics. *BMC Bioinformatics*, 10(Suppl 1):S61, 2009.

[4] S. Böcker, S. Briesemeister, and G. W. Klau. Exact algorithms for cluster editing: Evaluation and experiments. *Algorithmica*, 60(2):316–334, 2011.

[5] S. Böcker and P. Damaschke. Even faster parameterized cluster deletion and cluster editing. *Information Processing Letters*, 111(14):717–721, 2011.

[6] H. L. Bodlaender. Kernelization: New upper and lower bound techniques. In *Proceedings of the 4th International Workshop on Parameterized and Exact Computation (IWPEC '09)*, volume 5917 of *LNCS*, pages 17–37. Springer, 2009.

[7] J. Chen and J. Meng. A 2k kernel for the cluster editing problem. *Journal of Computer and System Sciences*, 2011. In press.

[8] B. Dorn, F. Hüffner, D. Krüger, R. Niedermeier, and J. Uhlmann. Exploiting bounded signal flow for graph orientation based on cause-effect pairs. In *Proceedings of the 1st international ICST conference on Theory and practice of algorithms in (computer) systems (TAPAS'11)*, Lecture Notes of ICST, pages 104–115. ICST, Springer, 2011. A long version of this work is to appear in *Algoithms for Molecular Biology*.

[9] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999.

[10] M. R. Fellows, M. A. Langston, F. A. Rosamond, and P. Shaw. Efficient parameterized preprocessing for cluster editing. In *Proceedings of the 16th International Symposium on Fundamentals of Computation Theory (FCT '07)*, volume 4639 of *LNCS*, pages 312–321. Springer, 2007.

[11] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.

[12] S. Fortune, J. Hopcroft, and J. Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10(2):111–121, 1980.

[13] J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier. Automated generation of search tree algorithms for hard graph modification problems. *Algorithmica*, 39(4):321–347, 2004.

[14] J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier. Graph-modeled data clustering: Exact algorithms for clique generation. *Theory of Computing Systems*, 38(4):373–392, 2005.

[15] J. Guo. A more effective linear kernelization for cluster editing. *Theoretical Computer Science*, 410(8-10):718–726, 2009.

[16] J. Guo and R. Niedermeier. Invitation to data reduction and problem kernelization. *ACM SIGACT News*, 38(1):31–45, 2007.

[17] F. Hüffner, C. Komusiewicz, H. Moser, and R. Niedermeier. Fixed-parameter algorithms for cluster vertex deletion. *Theory of Computing Systems*, 47(1):196–217, 2010.

[18] M. Hummel, S. Bentink, H. Berger, W. Klapper, S. Wessendorf, T. F. Barth, H.-W. Bernd, S. B. Cogliatti, J. Dierlamm, A. C. Feller, M.-L. Hansmann, E. Haralambieva, L. Harder, D. Hasenclever, M. Kühn, D. Lenze, P. Lichter, J. I. Martin-Subero, P. Möller, H.-K. Müller-Hermelink, G. Ott, R. M. Parwaresch, C. Pott, A. Rosenwald, M. Rosolowski, C. Schwaenen, B. Stürzenhofecker, M. Szczepanowski, H. Trautmann, H.-H. Wacker, R. Spang, M. Loeffler, L. Trümper, H. Stein, and R. Siebert. A biologic definition of Burkitt's lymphoma from transcriptional and genomic profiling. *New England Journal of Medicine*, 354(23):2419–2430, 2006.

[19] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.

[20] J. Jacob, M. Jentsch, D. Kostka, S. Bentink, and R. Spang. Detecting hierarchical structure in molecular characteristics of disease using transitive approximations of directed graphs. *Bioinformatics*, 24(7):995–1001, 2008.

[21] C. Komusiewicz and J. Uhlmann. Alternative parameterizations for cluster editing. In *Proceedings of the 37th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM '11)*, volume 6543 of *Lecture Notes in Computer Science*, pages 344–355. Springer, 2011.

[22] M. Křivánek and J. Morávek. NP-hard problems in hierarchical-tree clustering. *Acta Informatica*, 23(3):311–323, 1986.

[23] J. I. Munro. Efficient determination of the transitive closure of a directed graph. *Information Processing Letters*, 1(2):56–58, 1971.

[24] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.

[25] R. Niedermeier and P. Rossmanith. A general method to speed up fixed-parameter-tractable algorithms. *Information Processing Letters*, 73(3–4):125–129, 2000.

[26] F. Protti, M. D. da Silva, and J. L. Szwarcfiter. Applying modular decomposition to parameterized cluster editing problems. *Theory of Computing Systems*, 44(1):91–104, 2009.

[27] T. J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing (STOC '78)*, pages 216–226. ACM Press, 1978.

[28] R. Shamir, R. Sharan, and D. Tsur. Cluster graph modification problems. *Discrete Applied Mathematics*, 144(1–2):173–182, 2004.

[29] Y. Shiloach and Y. Perl. Finding two disjoint paths between two pairs of vertices in a graph. *Journal of the ACM*, 25(1):1–9, 1978.