Proc. 11th WADS, 2009

On Making Directed Graphs Transitive

Mathias Weller*, Christian Komusiewicz**, Rolf Niedermeier, and Johannes Uhlmann***

Institut für Informatik, Friedrich-Schiller-Universität Jena
Ernst-Abbe-Platz 2, D-07743 Jena, Germany
{mathias.weller, c.komus, rolf.niedermeier, johannes.uhlmann}@uni-jena.de

Abstract. We present the first thorough theoretical analysis of the Transitivity Editing problem on digraphs. Herein, the task is to perform a minimum number of arc insertions or deletions in order to make a given digraph transitive. This problem has recently been identified as important for the detection of hierarchical structure in molecular characteristics of disease. Mixing up Transitivity Editing with the companion problems on undirected graphs, it has been erroneously claimed to be NP-hard. We correct this error by presenting a first proof of NP-hardness, which also extends to the restricted cases where the input digraph is acyclic or has maximum degree four. 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. In particular, providing an $O(k^2)$-vertex problem kernel, we positively 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. We also demonstrate that if the input digraph contains no "diamond structure", then one can always find an optimal solution that exclusively performs arc deletions. Most of our results (including NP-hardness) can be transferred to the Transitivity Deletion problem, where only arc deletions are allowed.

1 Introduction

To make a directed graph (digraph for short) transitive by a minimum number of arc modifications has recently been identified to have important applications in detecting hierarchical structure in molecular characteristics of disease [11,3]. A digraph $D = (V, A)$ is called *transitive* if $(u, v) \in A$ and $(v, w) \in A$ implies $(u, w) \in A$ (also cf. [1, Section 4.3]). Thus, the central problem Transitivity Editing studied here asks, given a digraph and an integer $k \geq 0$, to find a set of at most k arcs to insert or delete in order to make the resulting digraph transitive. 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 [11,3]. We also study

* Partially supported by the DFG, research project DARE, GU 1023/1.
** Supported by a PhD fellowship of the Carl-Zeiss-Stiftung.
*** Supported by the DFG, research project PABI, NI 369/7.

the special case when only arc deletions (Transitivity Deletion) are allowed and restricted classes of digraphs (acyclic and bounded-degree). Note that the corresponding problem 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 clearly solvable in polynomial time [13].

Previous work. Transitivity Editing can be seen as the "directed counterpart" of the so far much better studied problem Cluster Editing on undirected graphs (see [2,5,7,8,15]). Indeed, both problems are also referred to as Transitive Approximation problem on directed and undirected graphs, respectively. Unfortunately, this is perhaps a reason why Transitivity Editing has erroneously been claimed to be NP-hard [11,3] 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] to transfer their previous results for Cluster Editing [2] 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. [11].

Our contributions. We eventually prove the so far only claimed NP-hardness[1] 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 with maximum vertex degree four (more precisely, indegree two and outdegree two). To this end, we also make the helpful combinatorial observation that if a digraph does not contain a so-called "diamond structure", then there is an optimal solution for Transitivity Editing that only deletes arcs. This observation is also useful for developing more efficient fixed-parameter algorithms than the ones presented in previous work. First, 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 can actually prove an $O(k \cdot d)$-vertex kernel. Finally, exploiting the aforementioned observation on diamond-freeness, 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 has become additive instead of multiplicative due to our kernelization result). Finally, we mention that Transitivity Deletion can be solved in $O(2^k + n^3)$ time. To conclude, note that Gutin and Yeo [9] asked in their recent survey about parameterized problems on digraphs for extending the so far small list of fixed-parameter tractability results for NP-hard problems on *digraphs*—we hope that our work makes a useful addition to this list. Due to the lack of space, several details are deferred to a full version of this article.

[1] Indeed, all corresponding decision problems are NP-complete

2 Preliminaries and a Structural Result

Our algorithmic results are in the context of fixed-parameter algorithms. Parameterized complexity is a two-dimensional framework for studying the computational complexity of problems [4,6,14]. One dimension is the input size n (as in classical complexity theory), and the other one is the *parameter* k (usually a positive integer). 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 iff (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*.

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*. Throughout this work, let $n := |V|$. If $V' \subseteq V$, then $D[V'] := (V', A \cap (V' \times V'))$ 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 sets of arcs 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, while $\text{succ}_A(u) := \{v \in V \mid (u, v) \in A\}$ denotes its *successors*. The vertices in $\text{pred}_A(u) \cup \text{succ}_A(u)$ are said to be *adjacent to* u.

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

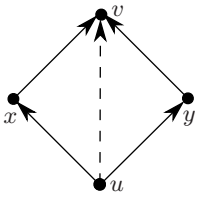
$$\forall_{u,v,w \in V} \ ((u, v) \in A \wedge (v, w) \in A) \Rightarrow (u, w) \in A.$$

In other words, D is transitive if A is a transitive relation on $(V \times V)$. The central problem of this work (formulated as decision problem, but our algorithms can also solve the corresponding minimization problem) is defined as follows.

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

Analogously, Transitivity Deletion is defined via only allowing arc deletions.

To derive our results, we make use of the fact that transitive digraphs can be characterized by "forbidden P_3s". Slightly abusing notation, 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 $(u, w) \notin A$. 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 iff it does not contain a P_3.



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

Fig. 1. The diamond structure and its adjacency matrix. In order to meet the definition, 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, for each vertex, the endpoints of its outgoing arcs are determined by its row. Stars represent wildcards, that is, these entries do not matter for the definition.

A central tool for our combinatorial studies is based on the consideration of "diamonds". The absence of diamonds in a given digraph simplifies the Transitivity Editing problem. This helps us in proving NP-hardness and in 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 Fig. 1).[2] If D does not contain a diamond, then it is said to be *diamond-free*.

A set $S \subseteq V \times V$ is called 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_{DEL} 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 1. *Let $D = (V, A)$ be a diamond-free digraph and let S be a solution set for D. Then $D_{DEL} := (V, A \Delta S_{DEL})$ is diamond-free.*

The following important result shows that in order to solve Transitivity Editing on diamond-free digraphs, it is optimal to only perform arc deletions.

Lemma 2. *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, $S = S_{DEL}$.*

Proof. Let S' be any optimal solution set for D. By Lemma 1, 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 now 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 vertex pair (a, c), there is at most one b meeting the criteria $(a, b) \in A$ and $(b, c) \in A$. Hence, for each inserted arc (a, c) in S', there is at most one arc (a, b) in S and hence $|S| \leq |S'|$.

[2] Note that this is not a common definition and should not be mixed-up for instance with diamonds in undirected graphs.

Let $D' := (V, A')$ with $A' := A \setminus S$. We now 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 $S \subseteq A$ (that is, S contains only arc deletions), 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-four digraphs and in acyclic digraphs. Both results are derived by a reduction from Positive-Not-all-equal-3SAT, which is an NP-complete variant of 3SAT [12].

> Positive-Not-all-equal-3SAT (PNAE-3SAT):
> **Input**: A Boolean formula φ in n variables $x_0, \ldots, x_{n-1}$ which is a conjunction of m clauses C_i, $0 \leq i < m$, each consisting of three positive literals.
> **Question**: Is there a truth assignment to 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?

First, we show that PNAE-3SAT can be reduced to Transitivity Editing in degree-four digraphs. To this end, we construct an input instance of Transitivity Editing from a given input instance of PNAE-3SAT in polynomial time as follows. For each of the n Boolean variables, we construct a *variable cycle*, that is, a directed cycle of length $8m$, with m being the number of clauses in the given formula φ. More specifically, for each variable x_i, the corresponding variable cycle consists of the vertices $V_i := \{i_0, \ldots, i_{8m-1}\}$. The vertices in V_i are connected into a cycle by adding the arcs $A_i := \{\{i_p, i_{p+1}\} \mid 0 \leq p \leq 8m-1\}$ (for the ease of presentation, let $i_{8m} = i_0$). The collection of all variable cycles is then referred to by (V, A) with $V := \bigcup_{i=0}^{n-1} V_i$ and $A := \bigcup_{i=0}^{n-1} A_i$. In the following, we refer to the arcs $(i_0, i_1), (i_2, i_3), \ldots, (i_{8m-2}, i_{8m-1})$ as *even arcs* and to all other arcs in the variable cycle as *odd arcs*.

Moreover, for each clause $C_j = \{x_p, x_q, x_r\}$ in φ with $0 \leq j < m$, we construct a *clause cycle*, that is, a directed length-three cycle between the variable cycles of its three variables consisting of the arcs $A'_j := \{(p_{8j}, q_{8j}), (q_{8j}, r_{8j}), (r_{8j}, p_{8j})\}$. See Fig. 2(a) for an illustration. This completes the construction.

The set of all arcs in the clause cycles is denoted by $A' := \bigcup_{j=0}^{m-1} A'_j$. Note that two vertices of a variable cycle contained in different clause cycles have distance

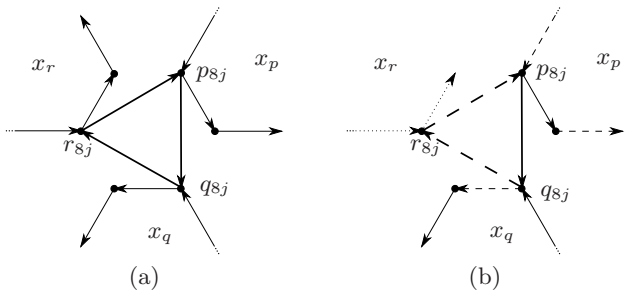


Fig. 2. (a): The clause cycle of clause $C_i = \{x_p, x_q, x_r\}$ connecting the corresponding variable cycles. Bold arcs are in A'. (b): All P_3s containing an arc of the clause cycle can be destroyed by deleting two arcs if in the variable cycle of x_p all odd arcs are deleted and in the variable cycle of x_q all even arcs are deleted. Dashed lines indicate deleted arcs and exactly one of the two dotted arcs incident to r_{8j} is deleted.

at least 8, which makes it easy to see that the constructed digraph is diamond-free. Finally, let $D := (V, A \cup A')$ denote the resulting digraph and $k := 2m+4mn$. Observe that D is diamond-free and has maximum degree four.

Theorem 1. Transitivity Editing *is NP-complete, even if the maximum degree is bounded by four (indegree two and outdegree two).*

Proof. Obviously, one can verify in polynomial time whether a digraph is transitive. This implies that Transitivity Editing is in NP. We now show that it is NP-hard by reducing from PNAE-3SAT. Let $D = (V, A \cup A')$ be a digraph constructed as described above from a given instance φ of PNAE-3SAT. We show that (D, k) with $k := 2m + 4mn$ is a yes-instance for Transitivity Editing iff there is a satisfying assignment to the variables of φ.

"$\Leftarrow$": Suppose that there is a satisfying assignment β to the variables of a PNAE-3SAT input instance φ. Then, we can construct a transitive digraph by modifying D in the following way: First, for each variable x_i, we remove all odd arcs of its variable cycle if $\beta(x_i) =$ true, and all even arcs if $\beta(x_i) =$ false. All in all, we remove $4m$ arcs for each of the n variable cycles, which is a total of $4mn$ arc deletions. Note that all remaining P_3s contain at least one arc of a clause cycle.

To destroy these P_3s, for each clause $C_j = \{x_p, x_q, x_r\}$, $0 \leq j < m$, the clause cycle is modified in the following way: Since β is a satisfying assignment for the PNAE-3SAT instance, we can assume without loss of generality that $\beta(x_p) =$ true and $\beta(x_q) =$ false. Hence, we have deleted all odd arcs in the variable cycle of x_p and all even arcs in the variable cycle of x_q, that is, the arcs (p_{8j-1}, p_{8j}) and (q_{8j}, q_{8j+1}) are deleted and the arcs (p_{8j}, p_{8j+1}) and (q_{8j-1}, q_{8j}) are not deleted. Moreover, observe that deleting the arcs (q_{8j}, r_{8j}) and (r_{8j}, p_{8j}) of the clause cycle makes p_{8j} a source and q_{8j} a sink. Hence, all P_3s containing an arc of the clause cycle of C_j are destroyed. See Fig. 2(b) for an illustration. For

all clauses, this requires $2m$ arc deletions in total. In summary, it is possible to make D transitive with $2m + 4mn$ arc deletions.

"⇒": Suppose that $(D, 2m + 4mn)$ is a yes-instance of Transitivity Editing. Hence, a solution set S for D exists such that $|S| \leq 2m + 4mn$. Since D is diamond-free we can assume, by Lemma 2, that $S \subseteq A \cup A'$. Let $\hat{A} := (A \cup A') \setminus S$ and $\hat{D} := (V, \hat{A})$.

Next, we show that S contains exactly two arcs from each clause cycle and $4m$ arcs from each variable cycle. First, note that one needs at least two arc deletions to make a directed cycle of length three transitive. Hence, turning all m clause cycles transitive requires at least $2m$ arc deletions. Second, note that making a variable cycle (which has length $8m$) transitive requires at least $4m$ arc deletions since it contains $4m$ arc-disjoint P_3s. This implies that S contains exactly two arcs from each clause cycle and $4m$ arcs from each variable cycle (note that the variable and clause cycles are arc-disjoint). Moreover, observe that, to make a variable cycle transitive by deleting $4m$ arcs, either all $4m$ even or all $4m$ odd arcs must be deleted (since it is clearly optimal to delete every second arc).

Consider a clause $C_j = \{x_p, x_q, x_r\}$, $0 \leq j < m$. We show that for one of the three corresponding variable cycles all even arcs and for another all odd arcs are deleted, and, as a consequence, the assignment β with $\beta(x_i) :=$ true if all odd arcs of the corresponding variable cycle are deleted and $\beta(x_i) :=$ false, otherwise, is satisfying. Assume towards a contradiction that there exists a clause $C_j = \{x_p, x_q, x_r\}$, $0 \leq j < m$, such that for all three variables x_p, x_q, and x_r all even (odd) arcs of the variable cycles are deleted. Recall that for each clause cycle all but one arc are deleted. Without loss of generality, let (p_{8j}, q_{8j}) be this arc, that is, $(p_{8j}, q_{8j}) \in \hat{A}$. If all even arcs are deleted in the variable cycles, then the odd arc $(p_{8j-1}, p_{8_j}) \in \hat{A}$ and $(p_{8j-1}, p_{8_j}, q_{8j})$ is a P_3 in $\hat{D}$. Otherwise, if all odd arcs are deleted, then the even arc $(q_{8j}, q_{8j+1}) \in \hat{A}$ and $(p_{8j}, q_{8j}, q_{8j+1})$ is a P_3 in $\hat{D}$. Both cases contradict the fact that S is a solution. □

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

Corollary 1. Transitivity Deletion *is NP-complete, even if the maximum degree is bounded by four (indegree two and outdegree two).*

The undirected "sister" problem Cluster Editing becomes polynomial-time solvable when the input is a tree, that is, acyclic. 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 [16] which is NP-hard in general but polynomial-time solvable on acyclic digraphs. However, we have to give up the bounded degree constraint.

To show the NP-hardness, we reduce again from PNAE-3SAT. The technical effort, however, significantly increases. The trickiness of the proof lies in incorporating an "information feedback" between the variable gadgets while using only acyclic variable and clause gadgets.

Theorem 2. Transitivity Editing *and* Transitivity Deletion *are NP-complete, even when restricted to acyclic digraphs.*

4 Fixed-Parameter Tractability Results

In this section, we complement the NP-hardness results of the previous section with encouraging algorithmic results. Note that Böcker et al. [3] observed that "most graphs derived from real-world applications are almost transitive". Consequently, as Böcker et al., 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. Thus, in what follows, we first develop kernelization results, and then we present an improved search tree strategy, altogether yielding the so far fastest fixed-parameter algorithms for Transitivity Editing.

First, observe that Transitivity Editing is fixed-parameter tractable with respect to the parameter k: The task is simply to destroy all P_3s in a given digraph. Clearly, there are exactly three possibilities to destroy a P_3, either by deleting one of the two arcs or by inserting the "missing" one. This yields a search tree of size $O(3^k)$ (cf. [3]), which indeed can be used to enumerate *all* solutions of size at most k because it exhaustively tries all possibilities to destroy P_3s.

Kernelization. In the following, we describe a kernelization for Transitivity Editing. We show a kernel consisting of $O(k^2)$ vertices for the general problem and a kernel of $O(k)$ vertices for 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 remove u and all arcs that are incident to it.*

Lemma 3. *Rule 1 is correct and can be exhaustively applied in $O(n^3)$ time.*

Proof. To prove the correctness, we construct a sequence of arc modifications that form an optimal solution set. Then, we will prove that, if at some point in this sequence a vertex u does not take part in any P_3, then u does not take part in any P_3 at any later point in the sequence. Thus, removing u never changes the set of P_3s to be destroyed.

Let (D, k) with $D = (V, A)$ denote the given input instance and let S denote an optimal solution set for D with $s := |S| \leq k$ and $D' := (V, A \Delta S)$. Let $\mathcal{Q}$ be the straightforward search-tree algorithm that searches a P_3 in the digraph and destroys it by branching into all three possibilities of inserting or deleting an arc. Clearly, $\mathcal{Q}$ returns a shortest sequence of digraphs $(D = D_0, D_1, \ldots, D_s = D')$ with $D_i := (V, A_i)$ and a sequence of arc modifications $F_1, \ldots, F_s$ with $F_i := A_{i-1} \Delta A_i$ for each $1 \leq i \leq s$. We prove the following: For each $i \geq 1$, if a vertex $u \in V$ does not take part in any P_3 in D_{i-1}, then it does not take part

in any P_3 in D_i. Hence, by induction, if u does not take part in any P_3 in D_0, then there is no $j > 0$ such that u takes part in a P_3 in D_j. Thus, D and $D - u$ yield the same sequence of arc modifications $F_1, \ldots, F_s$ and thus $(D, k) \in$ Transitivity Editing $\Leftrightarrow (D - u, k) \in$ Transitivity Editing.

In the following, we show the contraposition of the claim: For each $i \geq 1$, if a vertex $u \in V$ takes part in a P_3 p in D_i, then it takes part in a P_3 q in D_{i-1}. Let $F_i = \{(a, b)\}$. Since $\mathcal{Q}$ only inserts or deletes (a, b) to destroy a P_3, we know that there is a P_3 r in D_{i-1} that contains both a and b. Hence, if $u = a$ or $u = b$, then $q = r$ and thus u takes part in q. Otherwise, we consider the following cases.

Case 1: (a, b) is inserted.
Clearly, there is a vertex $v \in V$ such that (a, v, b) is a P_3 in D_{i-1}; hence, if $u = v$, then $q = (a, u, b)$. Furthermore, if $p \neq (a, b, u)$ and $p \neq (u, a, b)$, then $q = p$. Otherwise, without loss of generality, assume that $p = (a, b, u)$. Obviously, $(a, u) \notin A_i$. Since $(a, u) \notin F_i$, we know that $(a, u) \notin A_{i-1}$. If $(v, u) \in A_{i-1}$, then $q = (a, v, u)$, otherwise $q = (v, b, u)$.

Case 2: (a, b) is deleted.
Clearly, there is a vertex $v \in V$ such that either (a, b, v) or (v, a, b) is a P_3 in D_{i-1}; hence, if $u = v$, then $q = (a, b, u)$ or $q = (u, a, b)$. If $u \neq v$, without loss of generality assume that (a, b, v) is a P_3 in D_{i-1}. Furthermore, if $p \neq (a, u, b)$, then $q = p$. If $p = (a, u, b)$, then if $(u, v) \in A_{i-1}$, then $q = (a, u, v)$; otherwise, $q = (u, b, v)$.

Finally, the running time can be seen as follows. We enumerate all P_3s in $O(n^3)$ time, thereby labeling all vertices that are part of a P_3. Afterwards, we remove all unlabeled vertices. □

Surprisingly, this data reduction rule is sufficient to show a linear-size problem kernel if the maximum degree of the given digraph is constant.

Theorem 3. Transitivity Editing *restricted to digraphs with maximum degree d admits a problem kernel containing at most $2k \cdot (d + 1)$ vertices.*

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 $X := V \backslash Y$. 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 $|V| = |X| + |Y| \leq 2k + d2k = 2k(d + 1)$. □

The above data reduction also works for Transitivity Deletion:

Corollary 2. Transitivity Deletion *restricted to digraphs with maximum degree d admits a problem kernel containing at most $2k \cdot (d + 1)$ vertices.*

Next, we prove an $O(k^2)$-vertex kernel for general digraphs. The following data reduction rule roughly follows an idea for Cluster Editing [7]: If there is some vertex pair (a, b) such that not modifying (a, b) results in a solution size of at least $k + 1$, then every solution of size at most k must contain (a, b).

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

1. *Let $(u, v) \in (V \times V) \backslash A$ and $Z := \mathrm{succ}_A(u) \cap \mathrm{pred}_A(v)$. If $|Z| > k$, then insert (u, v) into A and decrease k by one.*
2. *Let $(u, v) \in A$, $Z_u := \mathrm{pred}_A(u) \setminus \mathrm{pred}_A(v)$ and $Z_v := \mathrm{succ}_A(v) \setminus \mathrm{succ}_A(u)$. If $|Z_u| + |Z_v| > k$, then delete (u, v) from A and decrease k by one.*

Lemma 4. *Let (D, k) be an input instance of* Transitivity Editing. *Then, Rule 2 causes an arc modification iff it destroys more than k P_3s in D.*

Lemma 4 is decisive for proving the correctness of Rule 2.

Lemma 5. *Rule 2 is correct and can be exhaustively applied in $O(n^3)$ time.*

We now show that the exhaustive application of both rules leads to a problem kernel of $O(k^2)$ vertices.

Theorem 4. Transitivity Editing *admits a problem kernel containing at most $k(k + 2)$ vertices.*

Proof. Assume that there is a digraph $D = (V, A)$ with $|V| > k(k + 2)$, D is reduced with respect to Rules 1 and 2, and 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 $X := V \backslash Y$. 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 the assumption that $|V| > k(k + 2)$ 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 an $(a, b) \in S$ with $|Z_{(a,b)}| > k$, a contradiction to Lemma 4. □

The above data reduction works also for Transitivity Deletion:

Corollary 3. Transitivity Deletion *admits a problem kernel containing at most $k(k + 2)$ vertices.*

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]). Note that, to solve Transitivity Deletion, the search only needs to branch into two cases,

yielding an algorithm running in $O(2^k \cdot n^3)$ time. Indeed, using more clever data structures, these running times can be improved to $O(3^k \cdot n \log n + n^3)$ and $O(2^k \cdot n \log n + n^3)$, respectively. Using the so-called interleaving technique [14] 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.

Theorem 5. Transitivity Editing *and* Transitivity Deletion *can be solved in* $O(2.57^k + n^3)$ *and* $O(2^k + n^3)$ *time, respectively.*

Proof. Recall from Lemma 2 that in diamond-free digraphs we only need to consider arc deletions. This helps us to improve the branching strategy. 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. $(V, A \backslash \{(u, x), (u, y)\})$ can be made transitive with $\leq k - 2$ operations,
2. $(V, A \backslash \{(u, x), (y, v)\})$ can be made transitive with $\leq k - 2$ operations,
3. $(V, A \backslash \{(x, v), (u, y)\})$ can be made transitive with $\leq k - 2$ operations,
4. $(V, A \backslash \{(x, v), (y, v)\})$ 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 $T_k = 1 + 4 \cdot T_{k-2} + T_{k-1}$, 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. The correctness of this branching is easy to check. 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 [14], and making use of the polynomial-size problem kernels from Theorem 3 results in the running times $O(2.57^k + n^3)$ for Transitivity Editing and $O(2^k + n^3)$ for Transitivity Deletion. □

5 Conclusion

Two immediate theoretical challenges (of significant practical relevance) arising from our work are to find out whether there is an $O(k)$-vertex problem kernel for Transitivity Editing in the case of general digraphs (see [5,8] for corresponding results in the case of undirected graphs, that is, Cluster Editing) or to investigate whether *linear-time* polynomial size kernelization (so far the kernelization takes cubic time in the number of vertices) is possible (see [15] for corresponding results in case of Cluster Editing). 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 [10] 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, 2002.
2. S. Böcker, S. Briesemeister, and G. W. Klau. Exact algorithms for cluster editing: Evaluation and experiments. *Algorithmica*, 2009. 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. *Proc. 7th APBC*.
4. R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999.
5. M. R. Fellows, M. A. Langston, F. A. Rosamond, and P. Shaw. Efficient parameterized preprocessing for Cluster Editing. In *Proc. 16th FCT*, volume 4639 of *LNCS*, pages 312–321. Springer, 2007.
6. J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.
7. J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier. Graph-modeled data clustering: Exact algorithms for clique generation. *Theory Comput. Syst.*, 38(4):373–392, 2005.
8. J. Guo. A more effective linear kernelization for cluster editing. *Theor. Comput. Sci.*, 410(8-10):718–726, 2009.
9. G. Gutin and A. Yeo. Some parameterized problems on digraphs. *Comput. J.*, 51(3):363–371, 2008.
10. F. Hüffner, C. Komusiewicz, H. Moser, and R. Niedermeier. Fixed-parameter algorithms for cluster vertex deletion. *Theory Comput. Syst.*, 2009. To appear.
11. 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.
12. J. Kratochvíl and Z. Tuza. On the complexity of bicoloring clique hypergraphs of graphs. *J. Algorithms*, 45(1):40–54, 2002.
13. J. I. Munro. Efficient determination of the transitive closure of a directed graph. *Inf. Process. Lett.*, 1(2):56–58, 1971.
14. R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.
15. F. Protti, M. D. da Silva, and J. L. Szwarcfiter. Applying modular decomposition to parameterized cluster editing problems. *Theory Comput. Syst.*, 44(1):91–104, 2009.
16. B. Yang, S. Q. Zheng, and E. Lu. Finding two disjoint paths in a network with minsum-minmin objective function. In *Proc. International Conference on Foundations of Computer Science (FCS2007)*, pages 356–361. CSREA Press, 2007.