An Improved FPT Algorithm for the Flip Distance Problem[*]

Shaohua Li[1], Qilong Feng[2], Xiangzhong Meng[3], and Jianxin Wang[4]

1 **School of Information Science and Engineering, Central South University, Changsha, Hunan, P. R. China**
2 **School of Information Science and Engineering, Central South University, Changsha, Hunan, P. R. China**
3 **School of Information Science and Engineering, Central South University, Changsha, Hunan, P. R. China**
4 **School of Information Science and Engineering, Central South University, Changsha, Hunan, P. R. China**
`jxwang@mail.csu.edu.cn`

Abstract

Given a set $\mathcal{P}$ of points in the Euclidean plane and two triangulations of $\mathcal{P}$, the flip distance between these two triangulations is the minimum number of flips required to transform one triangulation into the other. The Parameterized Flip Distance problem is to decide if the flip distance between two given triangulations is equal to a given integer k. The previous best FPT algorithm runs in time $O^*(k \cdot c^k)$ ($c \leq 2 \times 14^{11}$), where each step has fourteen possible choices, and the length of the action sequence is bounded by $11k$. By applying the backtracking strategy and analyzing the underlying property of the flip sequence, each step of our algorithm has only five possible choices. Based on an auxiliary graph G, we prove that the length of the action sequence for our algorithm is bounded by $2|G|$. As a result, we present an FPT algorithm running in time $O^*(k \cdot 32^k)$.

1998 ACM Subject Classification F.2.2 Geometrical Problems and Computations, G.2.1 Combinatorial Algorithms

Keywords and phrases triangulation, flip distance, FPT algorithm

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.65

1 Introduction

Given a set $\mathcal{P}$ of n points in the Euclidean plane, a *triangulation* of $\mathcal{P}$ is a maximal planar subdivision whose vertex set is $\mathcal{P}$ [10]. A *flip* operation to one diagonal e of a convex quadrilateral in a triangulation is to remove e and insert the other diagonal into this quadrilateral. Note that if the quadrilateral associated with e is not convex, the flip operation is not allowed. The *flip distance* between two triangulations is the minimum number of flips required to transform one triangulation into the other.

Triangulations play an important role in computational geometry, which are applied in areas such as computer-aided geometric design and numerical analysis [11, 13, 21].

Given a point set $\mathcal{P}$ in the Euclidean plane, we can construct a graph $G_T(\mathcal{P})$ in which every triangulation of $\mathcal{P}$ is represented by a vertex, and two vertices are adjacent if their

[*] This work is supported in part by the National Natural Science Foundation of China under Grants (61420106009, 61232001, 61472449, 61672536, 61572414).

corresponding triangulations can be transformed into each other through one flip operation. $G_T(\mathcal{P})$ is called the *triangulations graph* of $\mathcal{P}$. Properties of the triangulations graph are studied in the literature. Aichholzer et al. [1] showed that the lower bound of the number of vertices of $G_T(\mathcal{P})$ is $\Omega(2.33^n)$. Lawson and Charles [17] showed that the diameter of $G_T(\mathcal{P})$ is $O(n^2)$. Hurtado et al. [14] proved that the bound is tight. Since $G_T(\mathcal{P})$ is connected [17], any two triangulations of $\mathcal{P}$ can be transformed into each other through a certain number of flips.

The Flip Distance problem consists in computing the flip distance between two triangulations of $\mathcal{P}$, which was proved to be NP-complete by Lubiw and Pathak [18]. Pilz [20] showed that the Flip Distance problem is APX-hard. Aichholzer et al. [2] proved that the Flip Distance problem is NP-complete on triangulations of simple polygons. However, the complexity of the Flip Distance problem on triangulations of convex polygons has been open for many years, which is equivalent to the problem of computing the rotation distance between two rooted binary trees [23].

The Parameterized Flip Distance problem is: given two triangulations of a set of points in the plane and an integer k, deciding if the flip distance between these two triangulations is equal to k. For the Parameterized Flip Distance problem on triangulations of a convex polygon, Lucas [19] gave a kernel of size $2k$ and an $O^*(k^k)$-time algorithm. Kanj and Xia [15] studied the Parameterized Flip Distance problem on triangulations of a set of points in the plane, and presented an $O^*(k \cdot c^k)$-time algorithm ($c \leq 2 \cdot 14^{11}$), which applies to triangulations of general polygonal regions (even with holes or points inside it).

In this paper, we exploit the structure of the Parameterized Flip Distance problem further. At first, we give a nondeterministic construction process to illustrate our idea. The nondeterministic construction process contains only two types of actions, which are the *moving action* as well as the *flipping and backing action*. There are 4 choices for the moving action and one choice for the flipping and backing action. Given two triangulations and a parameter k, we prove that either there exists a sequence of actions of length at most $2k$, following which we can transform one triangulation into the other, or we can reject this instance. Thus we get an improved $O^*(k \cdot 32^k)$-time FPT algorithm, which also applies to triangulations of general polygonal regions (even with holes or points inside it).

2 Preliminaries

In a triangulation T, a flip operation f to an edge e that is the diagonal of a convex quadrilateral $\mathcal{Q}$ is to delete e and insert the other diagonal e' into $\mathcal{Q}$. We define e as the *underlying edge* of f, denoted by $\varepsilon(f)$, and e' as the *resulting edge* of f, denoted by $\varphi(f)$. (For consistency and clarity, we continue to use some symbols and definitions from [15]). Note that if e is not a diagonal of any convex quadrilateral in the triangulation, flipping e is not allowed. Suppose that we perform a flip operation f on a triangulation T_1 and get a new triangulation T_2. We say f transforms T_1 into T_2. T_1 is called an *underlying triangulation* of f, and T_2 is called a *resulting triangulation* of f. Given a set $\mathcal{P}$ of n points in the Euclidean plane, let T_{start} and T_{end} be two triangulations of $\mathcal{P}$, in which T_{start} is the initial triangulation and T_{end} is the objective triangulation. Let $F = \langle f_1, f_2, ..., f_r \rangle$ be a sequence of flips, and $\langle T_0, T_1, ..., T_r \rangle$ be a sequence of triangulations of $\mathcal{P}$ in which $T_0 = T_{start}$ and $T_r = T_{end}$. If T_{i-1} is an underlying triangulation of f_i, and T_i is a resulting triangulation of f_i for each $i = 1, 2, ..., r$, we say F transforms T_{start} into T_{end}, or F is a *valid sequence*, denoted by $T_{start} \xrightarrow{F} T_{end}$. The flip distance between T_{start} and T_{end} is the length of a shortest valid flip sequence.

Now we give the formal definition of the Parameterized Flip Distance problem.

Parameterized Flip Distance
Input: Two triangulations T_{start} and T_{end} of $\mathcal{P}$ and an integer k.
Question: Decide if the flip distance between T_{start} and T_{end} is equal to k.

The triangulation on which we are performing a flip operation is called the *current triangulation*. An edge e which belongs to the current triangulation but does not belong to T_{end} is called a *necessary edge* in the current triangulation. It is easy to see that for any necessary edge e, there must exist a flip operation f in a valid sequence such that $e = \varepsilon(f)$. Otherwise, we cannot get the objective triangulation T_{end}.

For a directed graph D, a maximal connected component of its underlying graph is called a *weakly connected component* of D. We define the size of an undirected tree as the number of its vertices.

A *parameterized problem* is a decision problem for which every instance is of the form (x, k), where x is the input instance and $k \in \mathbb{N}$ is the parameter. A parameterized problem is *fixed-parameter tractable* (*FPT*) if it can be solved by an algorithm (*FPT algorithm*) in $O(f(k)|x|^{O(1)})$ time, where $f(k)$ is a computable function of k. In addition to computational geometry, parameterized problems in other areas such as graph theory [8, 9, 12], computational biology [3, 22] and MAX-SAT [6] are also studied extensively. For a further introduction to parameterized algorithms, readers could refer to [4, 7].

3 The Improved Algorithm for the Parameterized Flip Distance Problem

Given T_{start} and T_{end}, let $F = \langle f_1, f_2, ..., f_r\rangle$ be a valid sequence, that is, $T_{start} \xrightarrow{F} T_{end}$. Definition 1 defines the adjacency of two flips in F.

▶ **Definition 1.** [15] Let f_i and f_j be two flips in F $(1 \le i < j \le r)$. We define that flip f_j is adjacent to flip f_i, denoted by $f_i \to f_j$, if the following two conditions are satisfied:
(1) either $\varphi(f_i) = \varepsilon(f_j)$, or $\varphi(f_i)$ and $\varepsilon(f_j)$ share a triangle in triangulation T_{j-1};
(2) $\varphi(f_i)$ is not flipped between f_i and f_j, that is, there does not exist a flip f_p in F, where $i < p < j$, such that $\varphi(f_i) = \varepsilon(f_p)$.

By Definition 1, we can construct a directed acyclic graph (DAG), denoted by D_F. Every node in D_F represents a flip operation of F, and there is an arc from f_i to f_j if f_j is adjacent to f_i. For convenience, we label the nodes in D_F using labels of the corresponding flip operations. In other words, we can see a node in D_F as a flip operation and vice versa.

The following lemma shows that any topological sorting of D_F is a valid sequence.

▶ **Lemma 2.** *[15] Let T_0 and T_r be two triangulations and $F = \langle f_1, f_2, ..., f_r\rangle$ be a sequence of flips such that $T_0 \xrightarrow{F} T_r$. Let $\pi(F)$ be a permutation of the flips in F such that $\pi(F)$ is a topological sorting of D_F. Then $\pi(F)$ is a valid sequence of flips such that $T_0 \xrightarrow{\pi(F)} T_r$.*

Lemma 2 ensures that if we repeatedly remove a source node from D_F and flip the underlying edge of this node until D_F becomes empty, we can get a valid sequence and the objective triangulation T_{end}.

Here we introduce the definition of a walk.

▶ **Definition 3.** [16] A *walk* in a triangulation T (starting from an edge $e \in T$) is a sequence of edges of T beginning with e in which any two consecutive edges share a triangle in T.

According to Lemma 2, if there is a valid sequence F for the input instance, any topological sorting of D_F is also a valid sequence for the given instance. The difficulty is that F is unknown. In order to find the topological sorting of D_F, the algorithm in [15] takes a nondeterministic walk to find an edge e which is the underlying edge of a source node, flips this edge (removing the corresponding node from D_F), nondeterministically walks to an edge which shares a triangle with e and recursively searches for an edge corresponding to a source node. They deal with weakly connected components of D_F one after one (refer to Corollary 4 in [15]), that is, their algorithm tries to find a solution F in which all flips belonging to the same weakly connected component of D_F appear consecutively. In order to keep searching procedure within the current weakly connected component, they use a stack to preserve the nodes (defined as *connecting point* in [15]) whose removal separates the current weakly connected component into small weakly connected components. When removing all nodes of a small component, their algorithm jumps to the connecting point at the top of the stack and moves to deal with another small component.

We observe that it is not necessary to remove all nodes of a weakly connected component before dealing with other weakly connected components, that is, our algorithm may find a solution F in which the nodes belonging to the same weakly connected components appear dispersedly. Thus we do not need a stack to preserve connecting points. Instead of five types of actions in [15], we only need two types, that is, moving action as well as flipping and backtracking action (see Section 3.2). Moreover, every time we find a source node, we remove the node, flip the underlying edge and backtrack instead of searching for the next node in four directions, thus reducing the number of choices for the actions. Another contribution of this paper is that we construct an auxiliary graph G and prove that G is a forest. Since there is a bijection between nondeterministic actions and nodes as well as edges of G, we prove that there exists a sequence of actions of length at most $2|D_F|$, which is smaller than $11|D_F|$ in [15]. In addition, we make some optimization on the strategy of finding the objective sequence. As a result, we improve the running time of the algorithm from $O^*(k \cdot c^k)$ where $c \leq 2 \cdot 14^{11}$ to $O^*(k \cdot 32^k)$.

3.1 Nondeterministic construction process

Now we give a description of our nondeterministic construction process **NDTRV** (see Fig. 1). The construction is nondeterministic as it always guesses the optimal choice correctly when running. The actual deterministic algorithm enumerates all possible choices to simulate the nondeterministic actions (see Fig. 3). Readers could refer to [5] as an example of nondeterministic algorithm. We present this construction process in order to depict the idea behind our deterministic algorithm clearly and vividly.

Let T_{start} be the initial triangulation, and T_{end} be the objective triangulation. Suppose that F is a shortest valid sequence, that is, F has the shortest length among all valid sequences. Let D_F be the DAG constructed after F according to Definition 1. **NDTRV** traverses D_F, removes the vertices of D_F in a topologically-sorted order and transforms T_{start} into T_{end}. Although D_F is unknown, for further analysis, we assume that **NDTRV** can remove and copy nodes in D_F so that it can construct an auxiliary undirected graph G and a list L based on D_F during the traversal. In later analysis we show that G is a forest, and there is a bijection between actions of **NDTRV** and nodes as well as edges of G. Obviously G and L are unknown as well. We just show that if a shortest valid sequence F exists, then D_F exists. So do G, L and Q. We can see D_F and G as conceptual or dummy graphs. We construct G instead of analysing a subgraph of D_F because one move action (see Section 3.2) of **NDTRV** may correspond to one or two edges in D_F (see Fig. 2), while there is a one-to-one correspondence between move actions and edges in G .

At the beginning of an iteration, **NDTRV** picks a necessary edge $e = \varepsilon(f_h)$ arbitrarily and nondeterministically guesses a walk W to find the underlying edge of a source node f_s. Lemma 5 shows that there exists such a walk W whose length is bounded by the length of a directed path B from f_s to f_h, and every edge e' in W is the underlying edge of some flip f' on B. **NDTRV** uses L to preserve a sequence of nodes $\Gamma = \langle f_s = v_1, ..., f_h = v_\ell \rangle$ on B, whose underlying edges are in W. Simultaneously **NDTRV** constructs a path S by copying all nodes in Γ as well as adding an undirected edge between the copy of v_i and v_{i+1} for $i = 1, ..., \ell$. S is defined as a *searching path*. The node f_h is called a *starting node*. If a starting node is precisely a source node in D_F, the searching path consists only of the copy of this starting node. When finding $\varepsilon(f_s)$, **NDTRV** removes f_s from D_F, flips $\varepsilon(f_s)$ and moves back to the previous edge $\varepsilon(v_2)$ of $\varepsilon(f_s)$ in W. If v_2 becomes a source node of D_F, **NDTRV** removes v_2 from D_F, flips $\varepsilon(v_2)$ and moves back to the previous edge $\varepsilon(v_3)$. **NDTRV** repeats the above operations until finding a node v_i in Γ which is not a source node in D_F. Then **NDTRV** uses v_i as a new starting node, and recursively guesses a walk nondeterministically from $\varepsilon(v_i)$ to find another edge which is the underlying edge of a source node as above. **NDTRV** performs these operations until the initial starting node f_h becomes a source node in D_F. Finally **NDTRV** removes f_h and flips $\varepsilon(f_h)$, terminating this iteration. **NDTRV** repeats the above iteration until T_{start} is transformed into T_{end}. We give the formal presentation of **NDTRV** in Fig. 1 and an example in Fig. 2.

3.2 Actions of the construction

Our construction process contains two types of actions operating on triangulations. The edge which the algorithm is operating on is called *the current edge*. The current triangulation is denoted by $T_{current}$.

(i) Move to one edge that shares a triangle with the current edge in $T_{current}$. We formalize it as $(move, e_1 \mapsto e_2)$, where e_1 is the current edge and e_2 shares a triangle with e_1.

(ii) Flip the current edge and move back to the previous edge of the current edge in W. We formalize it as $(f, e_4 \mapsto e_3)$, where f is the flip performed on the current edge, e_4 equals $\varphi(f)$ and e_3 is the previous edge of $\varepsilon(f)$ in the current walk W.

Since there are four edges that share a triangle with the current edge, there are at most four directions for an action of type (i). However, there is only one choice for an action of type (ii).

3.3 The sequence of actions

The following theorem is the main theorem for the deterministic algorithm **FLIPDT**, which bounds the length of the sequence of actions by $2|V(D_F)|$.

▶ **Theorem 4.** *There exists a sequence of actions of length at most $2|V(D_F)|$ following which we can perform a sequence of flips F' of length $|V(D_F)|$, starting from a necessary edge in T_{start}, such that F' is a topological sorting of D_F.*

In order to prove theorem 4, we need to introduce some lemmas. We will give the proof for Theorem 4 at the end of Section 3.3.

Lemma 5 shows the existence of a length-bounded walk to find an edge which is the underlying edge of a source node.

▶ **Lemma 5.** *[16] Suppose that a sequence of flips F^- is performed such that every time we flip an edge, we delete the corresponding source node in the DAG resulting from preceding*

```
NDTRV($T_{start}$, $T_{end}$; $D_F$)
  Input: the initial triangulation $T_{start}$ and objective triangulation $T_{end}$. Assuming $F$ is a
         shortest sequence, $D_F$ is the corresponding DAG according to Definition 1.
  /*$G$ is an auxiliary undirected graph */
  /*$L$ is a list keeping track of searching paths for backtracking */
  /*$Q$ is a list preserving the sequence of nondeterministic actions */
  /* $T_{current}$ is the current triangulation*/

   a. Let $V(G)$ and $E(G)$ be empty sets, $L$ and $Q$ be empty lists;
   b. $T_{current} = T_{start}$;
   c. While $T_{current} \neq T_{end}$ do
   c.1.   Pick a necessary edge $e = \varepsilon(f_h)$ in $T_{current}$ arbitrarily;
   c.2.   Add a copy of $f_h$ into $G$;
   c.3.   Add $f_h$ into $L$;
   c.4.   TrackTree($T_{current}$, $e$, $D_F$, $G$, $L$, $Q$);

 TrackTree($T_{current}$, $\varepsilon(f_h)$, $D_F$, $G$, $L$, $Q$) /*construct searching paths starting from $\varepsilon(f_h)$*/
   1. Nondeterministically guess a walk in $T_{current}$ from $\varepsilon(f_h)$ to find $\varepsilon(f_s)$ according to
      Lemma 5, let $\Gamma = \langle f_s = v_1, ..., f_h = v_\ell \rangle$, where $f_s$ is a source node in $D_F$, be a sequence
      of nodes on the backbone $B$ whose underlying edges are in the walk $W$ such that $\varepsilon(v_i)$
      and $\varepsilon(v_{i+1})$ are consecutive in $W$ for $i = 1, ..., \ell - 1$;
   2. Add a copy of $v_1$,...,$v_{\ell-1}$ into $G$ respectively;
   3. Connect the copies of $v_1$,...,$v_\ell$ in $G$ into a path;
   4. Append $v_{\ell-1}$,...,$v_1$ to $L$;                        /*record current searching path*/
   5. Append $(move, \varepsilon(v_\ell) \mapsto \varepsilon(v_{\ell-1}))$,...,$(move, \varepsilon(v_2) \mapsto \varepsilon(v_1))$ to $Q$;   /*record actions*/
   6. Remove $f_s = v_1$ from $L$;
   7. Remove $f_s = v_1$ from $D_F$;
   8. Flip $\varepsilon(f_s)$ in $T_{current}$ and move back to $\varepsilon(v_2)$;
   9. Append $(f_s, \varphi(v_1) \mapsto \varepsilon(v_2))$ to $Q$;       /*record actions*/
  10. For $i = 2$ to $\ell$ do
  10.1    Nondeterministically guess if $v_i$ is a source node in $D_F$;
  10.2    If $v_i$ is a source node of $D_F$ then   /*flip and move back*/
  10.2.1      Remove $v_i$ from $L$;
  10.2.2      Remove $v_i$ from $D_F$;
  10.2.3      Flip $\varepsilon(v_i)$ in $T_{current}$ and move back to $\varepsilon(v_{i+1})$;
  10.2.4      Append $(v_i, \varphi(v_i) \mapsto \varepsilon(v_{i+1}))$ to $Q$;
  10.3    Else                  /*construct searching paths from $v_i$*/
  10.3.1      TrackTree($T_{current}$, $\varepsilon(v_i)$, $D_F$, $G$, $L$, $Q$);
```

Figure 1 Nondeterministic construction **NDTRV**

deleting operations. Let f_h be a node in the remaining DAG such that $\varepsilon(f_h)$ is an edge in the triangulation T resulting from performing the sequence of flips F^-. There is a source node f_s in the remaining DAG satisfying:

(1) *There is a walk W in T from $\varepsilon(f_h)$ to $\varepsilon(f_s)$.*

(2) *There is a directed path B from f_s to f_h in the remaining DAG that we refer to as the backbone of the DAG.*

(3) *The length of W is at most that of B.*

(4) *Any edge in W is the underlying edge of a flip in B, that is, $W = \langle \varepsilon(v_1), ..., \varepsilon(v_\ell) \rangle$, where $v_1 = f_s, ..., v_\ell = f_h$ are nodes in B and there is a directed path B_i from v_i to v_{i+1} for $i = 1, ..., \ell - 1$ such that $B_i \subset B$. A searching path is an undirected path constructed by copying $v_1, ..., v_\ell$ and connecting the copies of $v_1, ..., v_\ell$ into a path.*

▶ **Lemma 6.** ***NDTRV** transforms T_{start} into T_{end} with the minimum number of flips and stops in polynomial time if it correctly guesses every moving and flipping action.*

Proof. Suppose that F is a shortest valid sequence. According to Lemma 5, every edge

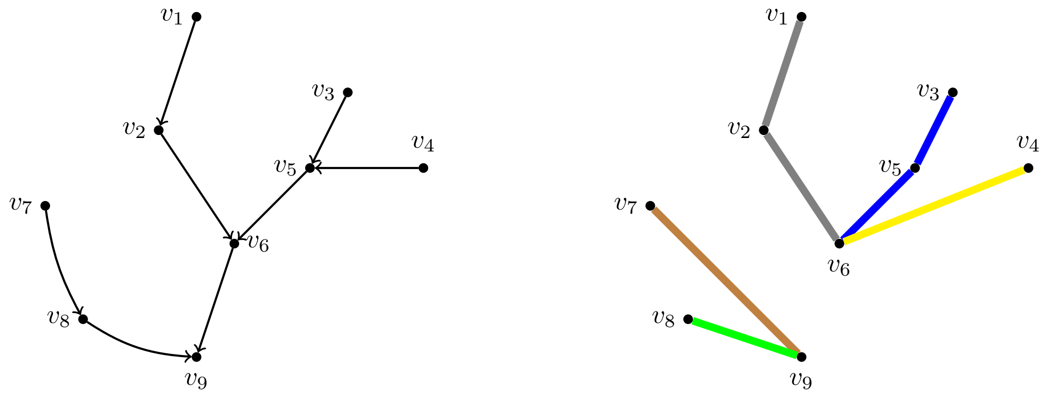

Figure 2 An example for constructing G. The graph on the left is the DAG. The graph on the right is the auxiliary graph G constructed by **NDTRV**. $\varepsilon(v_6)$ is the first chosen necessary edge, and $\varepsilon(v_9)$ is the second one. Edges with the same color belong to the same searching path. Searching paths in the same connected component are constructed in the same iteration.

flipped in **NDTRV** is the underlying edge of a source node in the remaining graph of D_F, and every node removed from the remaining graph of D_F in **NDTRV** is a source node. If $T_{current}$ is equal to T_{end} but D_F is not empty, then there exists a valid sequence F' which is shorter than F, contradicting that F is a shortest valid sequence. Thus **NDTRV** traverses D_F, removes all nodes of D_F in a topologically-sorted order and transforms T_{start} into T_{end} with the minimum number of flips by Lemma 2. Since the diameter of a transformations graph $G_T(\mathcal{P})$ is $O(n^2)$ [17], **NDTRV** stops in polynomial time. ◀

NDTRV constructs G, Q and L during its execution. Lemma 7, Lemma 8 and Lemma 9 show some properties of G, Q, L and D_F.

▶ **Lemma 7.** *At the end of* ***NDTRV****, the graph G consists of all searching paths constructed during the execution of* ***NDTRV****. Moreover, the list Q contains a sequence of nondeterministic actions starting from a necessary edge in T_{start} following which we can perform a sequence of flips F' of length $|V(D_F)|$ such that F' is a topological sorting of D_F.*

Proof. From the construction of G, we see that one node is added into G when it is in some searching path. Moreover, we add an edge between two nodes in G if and only if they are adjacent in a searching path. It follows that G consists of all searching paths constructed during the execution of **NDTRV**.

By the proof of Lemma 6, **NDTRV** removes the nodes of D_F in a topologically-sorted order. Since Q records every action of **NDTRV**, if we perform the actions in Q, we get a valid sequence F'. This completes the proof. ◀

▶ **Lemma 8.** *The following statements are true:*

(1) *At the end of any iteration of* ***NDTRV****, all nodes whose copies are in the searching paths constructed in this iteration are removed from D_F, and L becomes empty.*

(2) *During every step of* ***NDTRV****, there exists a directed path in the remaining graph of D_F passing through every node in L from the last node of L to the first node of L.*

(3) *All searching paths in G are edge-disjoint. Any two searching paths which belong to two different iterations respectively are node-disjoint. Searching paths constructed in each iteration form a tree respectively, namely a track tree. Moreover, G is a forest.*

Proof. Suppose we are in the first iteration. We name the searching paths in this iteration $S_1, ..., S_m$ by the order they are constructed. Suppose now we have constructed searching paths $S_1, ..., S_i$ and $i < m$. We prove that statements (2) and (3) hold for the first iteration by induction on i. When $i = 1$, the statement is true because there is only one searching path S_1 forming a tree, and there is a directed path in the remaining graph of D_F passing through every node in L from the last to the first according to Lemma 5. Assume that the statements are true for any $i < m$, that is, $S_1, ..., S_i$ are edge-disjoint, and they form a tree. Moreover, there exists a directed path P in the remaining graph of D_F passing through every node in L from the last to the first (inductive hypothesis). The next searching path we will construct is S_{i+1}. Note that before constructing S_{i+1}, we may remove the last node of L repeatedly from the remaining graph of D_F and L if it is a source node in the remaining graph of D_F. Since every node we removed was the beginning node of the directed path P, there was still a directed path passing through every node in L from the last to the first. Let $f_{start,i+1}$ be the starting node whose copy is in S_{i+1}, which means that $f_{start,i+1}$ is the last node in L, and $f_{start,i+1}$ is not a source node in the remaining graph of D_F. By the inductive hypothesis and the analysis above, there is a directed path P_1 from $f_{start,i+1}$ to f_h which is the first node of L. Let f'_s be a source node in the remaining graph of D_F such that the copy of f'_s is in S_{i+1}. By Lemma 5, there is a directed path P_2 in the remaining graph of D_F passing through the nodes whose copies are in S_{i+1} from f'_s to $f_{start,i+1}$. We argue that P_1 has only one common node with P_2, which is exactly $f_{start,i+1}$. Otherwise, there is a directed cycle induced by some of the nodes on P_1 and P_2 in the remaining graph of D_F, contradicting the acyclicity of D_F. It follows that there is a directed path in the remaining graph of D_F passing through every node in L from the last node of L to the first node of L. Since S_{i+1} cannot contain any copies of the nodes that have been removed from the remaining graph of D_F and L, S_{i+1} has only one common node with the searching paths S_1,...,S_i, namely the copy of $f_{start,i+1}$. It is implied that a node of D_F cannot be added to L more than once for the same reason. As a result, S_1,...,S_i and S_{i+1} are edge-disjoint, and S_1,...,S_{i+1} form a tree. The tree formed by S_1,...,S_m is a track tree. This completes the inductive proof.

We prove statement (1) in the first iteration by contradiction. Suppose that there exists one or several nodes left in L at the end of an iteration, and w is the last one in L. Suppose w was added to L when searching path S_p was constructed and **TrackTree**(v_p) was called. Let u be the node appended to L on its heel. Such a node u existed since w was not a source node then. Otherwise w was removed, resulting in a contradiction as one node of D_F cannot be added to L more than once (according to the proof in the above paragraph). Since w is the last one in L at the end of this iteration, u was removed, and **NDTRV** moved back to $\varepsilon(w)$. According to **NDTRV**, w has never become a source node in D_F. Otherwise, w has been removed from L, leading to a contradiction since one node of D_F cannot be added to L more than once. Thus **TrackTree**(w) was called. The terminal condition of **TrackTree**(w) is that w becomes a source node in D_F, and w is removed from L and D_F. This implies that this iteration does not end, a contradiction. It follows that L becomes empty at the end of the first iteration. According to **NDTRV**, all nodes whose copies are in the searching paths constructed in the first iteration are removed from D_F.

We prove that statements (1), (2) and (3) hold for the whole procedure by induction. We have proved that they are true for the first iteration. Suppose that there are t iterations in the procedure, and statements (1), (2) and (3) hold for the first j iterations for any $1 \leq j < t$. By the inductive hypothesis, at the end of the j-th iteration, L is empty. A node x whose copy is in the searching paths constructed in the first j iterations can never appear in L in the

$(j+1)$-th iteration because it has been removed from D_F. It follows that the searching paths constructed in the $(j+1)$-th iteration are node-disjoint with the searching paths constructed in the first j iterations. Thus they are edge-disjoint. Since L is empty, it is not difficult to see that the proof for the first iteration also holds for the $(j+1)$-th iteration. It follows that the searching paths constructed in the $(j+1)$-th iteration form a tree that is node disjoint with other j trees belonging to the first j iterations respectively, that is, these trees form a forest all together. By Lemma 7, G consists of all searching paths constructed during the execution of **NDTRV**. It follows that G is a forest. This completes the inductive proof for the whole procedure. ◀

▶ **Lemma 9.** *$|V(G)|$ is equal to $|V(D_F)|$.*

Proof. According to **NDTRV**, a node v is added to L if and only if its copy is added to G, and v is removed from L if and only if v is removed from D_F. By the proof in Lemma 8, no node in D_F can be added to L more than once. By the proof of Lemma 6, all nodes of D_F are removed by **NDTRV**. It follows that all nodes in D_F have exactly one copy in G. Thus $|V(G)| = |V(D_F)|$. ◀

We give the proof of Theorem 4 below.

Proof. (Theorem 4) In the procedure of constructing G in Fig. 1, we construct a list Q which consists of actions of type (i) and (ii). We claim that Q is exactly the sequence satisfying the requirement of this theorem. By Lemma 7, the number of actions of type (ii) in Q is exactly $|V(D_F)| = |V(G)|$. **NDTRV** adds an action of type (i) to Q if and only if it adds an an edge to $E(G)$. Moreover, Q and $E(G)$ are both empty at the beginning of **NDTRV**. It follows that there is a one-to-one correspondence between actions of type (i) in Q and edges in G. By Lemma 8, G is a forest. As a result, $|E(G)| \leq |V(G)|$, and the length of Q is bounded by $|E(G)| + |V(G)| \leq 2|V(G)| = 2|V(D_F)|$. ◀

3.4 The deterministic algorithm

Now we are ready to give the deterministic algorithm **FLIPDT** for the Parameterized Flip Distance problem. The specific algorithm is presented in Fig. 3. As mentioned above, we assume that **NDTRV** is always able to guess the optimal choice correctly. In fact, **FLIPDT** achieves this by trying all possible sequences of actions and partitions of k. At the top level, **FLIPDT** branches into all partitions of k, namely $(k_1, ..., k_t)$ satisfying $k_1 + ... + k_t = k$ and $k_1, ..., k_t \geq 1$, in which k_i $(i = 1, ..., t)$ equals the size of the track tree A_i constructed during the i-th iteration.

Suppose that **FLIPDT** is under some partition $(k_1, ..., k_t)$. Let $T^0_{iteration} = T_{start}$. **FLIPDT** permutates all necessary edges in T_{start} in the lexicographical order, and the ordering is denoted by O_{lex}. Here we number the given points of $\mathcal{P}$ in the Euclidean plane from 1 to n arbitrarily and label one edge by a tuple consisting of two numbers of its endpoints (the smaller number is ahead of the other one). Thus we can order the edges lexicographically. **FLIPDT** performs t iterations. At the beginning of the i-th iteration, $i = 1, ..., t$, we denote the current triangulation by $T^{i-1}_{iteration}$. For $i = 1, ..., t$, $T^i_{iteration}$ is also the triangulation resulting from the execution of the i-th iteration. At the beginning of the i-th iteration $(i = 1, ..., t)$, **FLIPDT** repeatedly picks the next edge in O_{lex} until finding a necessary edge e belonging to $T^{i-1}_{iteration}$ (just pick the first edge in O_{lex} in the first iteration). Note that one edge in O_{lex} may not be a necessary edge anymore with respect to $T^{i-1}_{iteration}$. Moreover, if **FLIPDT** reaches the end of O_{lex} but does not find a necessary edge belonging

```
FLIPDT($T_{start}$, $T_{end}$, $k$)
  Input: two triangulations $T_{start}$ and $T_{end}$ of a point set $\mathcal{P}$ in the Euclidean plane and an
         integer $k$.
  Output: return YES if there exists a sequence of flips of length $k$ that transforms $T_{start}$
          into $T_{end}$; otherwise return NO.

  1. For each partition $(k_1, ..., k_t)$ of $k$ satisfying $k_1 + k_2 + ... + k_t = k$ and $k_1, ..., k_t \geq 1$ do
  1.1    Order all necessary edges in $T_{start}$ lexicographically and denote this ordering by $O_{lex}$;
  1.2    FDSearch($T_{start}$,1,$(k_1, ..., k_t)$); /*iteration 1 distributed with $k_1$*/
  2. Return NO;

FDSearch($T$,$i$,$(k_1, ..., k_t)$)   /*the concrete branching procedure*/
  Input: a triangulation $T$, an integer $i$ denoting that the algorithm is at the $i$-th iteration and
         a partition $(k_1, ..., k_t)$ of $k$.
  Output: return YES if the instance is accepted.

  1. Repeatedly pick the next edge in $O_{lex}$ until finding a necessary edge $e$ with respect to $T$
     and $T_{end}$;
  2. If it reaches the end of $O_{lex}$ but finds no necessary edge in $T$ then
  2.1    Update $O_{lex}$ by permutating all necessary edges in $T^{i-1}_{iteration}$ in lexicographical order,
         and pick the first edge $e$ in $O_{lex}$;
  3. For each possible sequence of actions $seq_i$ of length $2k_i - 1$ do
  3.1    $T'$ = Transform($T$,$seq_i$,$e$);
  3.2    If $i < t$ then   /*continue to the next iteration distributed with $k_{i+1}$*/
  3.2.1      FDSearch($T'$,$i+1$,$(k_1, ..., k_t)$);
  3.3    Else if $i = t$ and $T' = T_{end}$ then   /*compare $T'$ with $T_{end}$*/
  3.3.1      Return YES;

Transform($T$,$s$,$e$)   /*subprocess for transforming triangulations*/
  Input: a triangulation $T$, a sequence of actions $s$ and a starting edges $e$.
  Output: a new triangulation $T'$.

  1. Perform a sequence of actions $s$ starting from $e$ in $T$, getting a new triangulation $T'$;
  2. Return $T'$;
```

Figure 3 The deterministic algorithm for the Flip Distance problem

to $T^{i-1}_{iteration}$, it needs to update O_{lex} by clearing O_{lex} and permutating all necessary edges in $T^{i-1}_{iteration}$ lexicographically, and choose the first edge in the updated ordering O_{lex}. Then **FLIPDT** branches into every possible sequence of actions seq_i of length $2k_i - 1$. For each enumeration of seq_i, **FLIPDT** performs the actions of seq_i on $T^{i-1}_{iteration}$ starting from e and gets a new triangulation $T^{i}_{iteration}$. For every triangulation $T^{i}_{iteration}$ resulting from seq_i, **FLIPDT** performs the $(i+1)$-th iteration on $T^{i}_{iteration}$. **FLIPDT** proceeds as above from the first iteration to the last iteration. When **FLIPDT** finishes the last iteration, it judges if the resulting triangulation $T^{t}_{iteration}$ is equal to T_{end}. If they are equal, the input instance is a yes-instance. Otherwise, **FLIPDT** rejects this case and proceeds.

Now we analyse how to enumerate all possible sequences of length $2k_i - 1$. By the proof of Lemma 8 and Theorem 4, the searching paths constructed during each iteration form a track tree in which a node corresponds to an action of type (ii) while an edge corresponds to an action of type (i). It follows that the number of actions of type (ii) is k_i, and the number of actions of type (i) is $k_i - 1$ since the number of nodes equals the number of edges plus one in a tree. According to **NDTRV**, the last action γ in seq_i must be of type (ii), and in any prefix of $seq_i - \gamma$ the number of actions of type (i) must not be less than that of type (ii). Thus **FLIPDT** only needs to enumerate all sequences of length $2k_i - 1$ satisfying the above constraints.

The following theorem proves the correctness of the algorithm **FLIPDT**.

▶ **Theorem 10.** *Let (T_{start}, T_{end}, k) be an input instance.* ***FLIPDT*** *is correct and runs in time $O^*(k \cdot 32^k)$.*

Proof. Suppose that (T_{start}, T_{end}, k) is a yes-instance. There must exist a sequence of flips F of length k such that $T_{start} \xrightarrow{F} T_{end}$. Thus D_F exists according to Definition 1. By **NDTRV**, Lemma 7 and Lemma 8, there exists an undirected graph G consisting of a set of node-disjoint track trees $A_1, ..., A_t$. Moreover, Theorem 4 shows that there exists a sequence of actions Q following which we can perform all flips of D_F in a topologically-sorted order. Due to **NDTRV**, Q consists of several subsequences $seq_1, ..., seq_t$, in which seq_i is constructed in the i-th iteration and corresponds to the track tree A_i for $i = 1, ..., t$. Supposing the size of A_i is λ_i for $i = 1, ..., t$ satisfying $\lambda_1 + ... + \lambda_t = k$, seq_i contains λ_i actions of type (ii) corresponding to the nodes of A_i as well as $\lambda_i - 1$ actions of type (i) corresponding to the edges of A_i. **FLIPDT** guesses the size of every track tree by enumerating all possible partitions of k into $(k_1, ..., k_t)$ such that $k_1 + ... + k_t = k$ and $k_1, ..., k_t \geq 1$. We say that k_i is distributed to the i-th iteration or the distribution for the i-th iteration is k_i for $i = 1, ..., t$.

We claim that **FLIPDT** is able to perform a sequence Σ of actions which correctly guesses every subsequence $seq_1, ..., seq_t$ of the objective sequence Q, that is, Σ is a concatenation of $seq_1, ..., seq_t$. Suppose that **FLIPDT** has completed i iterations. We prove this claim by induction on i. At the first iteration, **FLIPDT** starts by picking the first necessary edge e_1 in list O_{lex}. In the first iteration of constructing Q, **NDTRV** starts by picking an arbitrary necessary edge. Without loss of generality, it chooses e_1 and construct seq_1 starting from e_1. The length of seq_1 is $2\lambda_1 - 1$. Since **FLIPDT** tries every distribution in $\{1, ..., k\}$ for the first iteration and $1 \leq \lambda_1 \leq k$, there is a correct guess of the distribution equal to λ_1 for this iteration. Under this correct guess, **FLIPDT** tries all possible sequences of actions of length $2\lambda_1 - 1$ starting from e_1. It follows that **FLIPDT** is able to perform a sequence that is equals to seq_1 in the first iteration resulting in a triangulation T_1.

Suppose that the claim is true for any first i iterations $(1 \leq i < t)$. That is, under some guess for the partition of k, $\lambda_1, ..., \lambda_i$ are distributed to the first i iterations respectively. Moreover, **FLIPDT** has completed i iterations and performed a sequence of actions $seq_{concat,i}$, which is equal to the concatenation of $seq_1, ..., seq_i$, resulting in a triangulation T_i. Based on T_i and $seq_{concat,i}$, **FLIPDT** is ready to perform the $(i + 1)$-th iteration. Suppose that **FLIPDT** picks e_{i+1} from O_{lex}. Let us see the construction of Q in **NDTRV**. Suppose **NDTRV** has constructed the first i track trees $A_1, ..., A_i$, and it is ready to begin a new iteration by arbitrarily picking a necessary edge in the current triangulation. Since **FLIPDT** correctly guessed and performed the first i subsequences of Q, T_i is exactly equal to the current triangulation in **NDTRV**. Thus e_{i+1} is a candidate edge belonging to the set of all selectable necessary edges for **NDTRV** in this iteration. Without loss of generality, it chooses e_{i+1} and constructs seq_{i+1} of length $2\lambda_{i+1} - 1$ starting from e_{i+1}. Since the sizes of $A_1, ..., A_i$ are $\lambda_1, ..., \lambda_i$ respectively, we get that $1 \leq \lambda_{i+1} \leq k - (\lambda_1 + ... + \lambda_i)$. We argue that **FLIPDT** is able to perform a sequence that is equal to the concatenation of $seq_1, ..., seq_{i+1}$. Since the edges in O_{lex} are ordered lexicographically and **FLIPDT** chooses necessary edges in a fixed manner, **FLIPDT** is sure to choose e_{i+1} to begin the $(i + 1)$-th iteration for every guessed sequence in which the first i subsequences are equal to seq_1,...,seq_i respectively. Thus **FLIPDT** actually tries every distribution in $\{1, ..., k - (\lambda_1 + ... + \lambda_i\})$ for the $(i + 1)$-th iteration starting from e_{i+1} based on T_i and $seq_{concat,i}$. It follows that there is a correct guess of distribution for the $(i + 1)$-th iteration which is equal to λ_{i+1}. Under this correct guess of distribution, **FLIPDT** tries all possible sequences of length $2\lambda_{i+1} - 1$ starting from e_{i+1} on T_i based on $seq_{concat,i}$, ensuring that one of them is equal to seq_{i+1}. It follows that the claim is true for the first $i + 1$ iterations. This completes the inductive proof for the claim.

If (T_{start}, T_{end}, k) is a yes-instance, the action sequence Q of length at most $2k$ exists and the deterministic algorithm can find such a sequence. Otherwise, there is no valid sequence F of length k. Thus there is no such action sequence Q. As a result, **FLIPDT** returns NO. It is proved that **FLIPDT** decides the given instance (T_{start}, T_{end}, k) correctly.

Finding and ordering all necessary edges in T_{start} takes $O(n+k\log k)$ time, and **FLIPDT** may update the ordering O_{lex} at the beginning of each iteration. The number of partitions of k is known as the composition number of k, which is 2^{k-1}. Under each partition $(k_1, ..., k_t)$ of k and for each k_i, $i = 1, ..., t$, we enumerate all possible subsequences of actions in which there are k_i actions of type (ii) and $k_i - 1$ actions of type (i). It follows that the number of all possible subsequences is bounded by $\binom{2(k_i-1)}{k_i-1} \times 4^{k_i-1} = O^*(16^{k_i})$ since there are four choices for action (i) and one choice for action (ii). Here we use Stirling's approximation $n! \approx \sqrt{2\pi n}(n/e)^n$ and get that $\binom{2(k_i-1)}{k_i-1} = O^*(4^{k_i})$. It follows that there are $O^*(16^{k_1}) \times O^*(16^{k_2}) \times ... \times O^*(16^{k_t}) = O^*(16^k)$ cases under each partition. Since for each case we can perform the sequence of actions in $O(k)$ time, and the resulting triangulation can be compared to T_{end} in $O(k)$ time, the running time of the whole algorithm is bounded by $O^*(k \cdot 2^{k-1} \cdot (n + k\log k) + k \cdot 2^{k-1} \cdot 16^k) = O^*(k \cdot 32^k)$.

According to the definition of the Flip Distance problem, we need to decide if we can find a shorter valid flip sequence for the given triangulations T_{start} and T_{end}. This is achieved by calling **FLIPDT** on each instance (T_{start}, T_{end}, k') for $k' = 0, ..., k$. The running time is bounded by $\sum_{k'=0}^{k} O^*(k' \cdot 32^{k'}) = O^*(k \cdot 32^k)$. ◀

4 Conclusion

In this paper we presented an FPT algorithm running in time $O^*(k \cdot 32^k)$ for the Parameterized Flip Distance problem, improving the previous $O^*(k \cdot c^k)$-time ($c \leq 2 \times 14^{11}$) FPT algorithm by Kanj and Xia [15]. An important related problem is computing the flip distance between triangulations of a convex polygon, whose traditional complexity is still unknown. Although our algorithm can be applied to the case of convex polygon, it seems that an $O(c^k)$ algorithm with smaller c for this case probably exists due to its more restrictive geometric property. In addition, whether there exists a polynomial kernel for the Parameterized Flip Distance problem is also an attractive problem.

References

1 Oswin Aichholzer, Ferran Hurtado, and Marc Noy. A lower bound on the number of triangulations of planar point sets. *Computational Geometry*, 29(2):135–145, 2004. `doi:10.1016/j.comgeo.2004.02.003`.

2 Oswin Aichholzer, Wolfgang Mulzer, and Alexander Pilz. Flip distance between triangulations of a simple polygon is NP-complete. *Discrete & Computational Geometry*, 54(2):368–389, 2015. `doi:10.1007/s00454-015-9709-7`.

3 Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, Michael T. Hallett, and Harold T. Wareham. Parameterized complexity analysis in computational biology. *Computer Applications in the Biosciences*, 11(1):49–57, 1995. `doi:10.1093/bioinformatics/11.1.49`.

4 Jianer Chen. Parameterized computation and complexity: A new approach dealing with np-hardness. *J. Comput. Sci. Technol.*, 20(1):18–37, 2005. `doi:10.1007/s11390-005-0003-7`.

5 Jianer Chen, Donald K. Friesen, Weijia Jia, and Iyad A. Kanj. Using nondeterminism to design efficient deterministic algorithms. *Algorithmica*, 40(2):83–97, 2004. `doi:10.1007/s00453-004-1096-z`.

6 Jianer Chen, Chao Xu, and Jianxin Wang. Dealing with 4-variables by resolution: An improved maxsat algorithm. *Theor. Comput. Sci.*, 670:33–44, 2017. `doi:10.1016/j.tcs.2017.01.020`.

7 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. `doi:10.1007/978-3-319-21275-3`.

8 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science, FOCS, Palm Springs, USA*, pages 150–159, 2011. `doi:10.1109/FOCS.2011.23`.

9 Marek Cygan, Marcin Pilipczuk, Michal Pilipczuk, and Jakub Onufry Wojtaszczyk. Subset feedback vertex set is fixed-parameter tractable. *SIAM J. Discrete Math.*, 27(1):290–309, 2013. `doi:10.1137/110843071`.

10 Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational geometry: algorithms and applications, 3rd Edition*. Springer, 2008.

11 Gerald E. Farin. *Curves and surfaces for computer-aided geometric design - a practical guide (4. ed.)*. Computer science and scientific computing. Academic Press, 1997.

12 Fedor V. Fomin, Daniel Lokshtanov, and Saket Saurabh. Efficient computation of representative sets with applications in parameterized and exact algorithms. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA, Portland, USA*, pages 142–151, 2014. `doi:10.1137/1.9781611973402.10`.

13 Bernd Hamann. Modeling contours of trivariatc data. *Mathematical Modeling and Numerical Analysis*, 26:51–75, 1992.

14 Ferran Hurtado, Marc Noy, and Jorge Urrutia. Flipping edges in triangulations. *Discrete & Computational Geometry*, 22(3):333–346, 1999. `doi:10.1007/PL00009464`.

15 Iyad A. Kanj and Ge Xia. Flip Distance is in FPT time $O(n + k \cdot c^k)$. In *proceedings of the 32nd International Symposium on Theoretical Aspects of Computer Science, STACS, Garching, Germany*, pages 500–512, 2015. `doi:10.4230/LIPIcs.STACS.2015.500`.

16 Iyad A. Kanj and Ge Xia. Computing the flip distance between triangulations. *to appear in Discrete & Computational Geometry*, 2017.

17 Charles L. Lawson. Transforming triangulations. *Discrete Mathematics*, 3(4):365–372, 1972. `doi:10.1016/0012-365X(72)90093-3`.

18 Anna Lubiw and Vinayak Pathak. Flip distance between two triangulations of a point set is NP-complete. *Computational Geometry*, 49:17–23, 2015. `doi:10.1016/j.comgeo.2014.11.001`.

19 Joan M. Lucas. An improved kernel size for rotation distance in binary trees. *Information Processing Letters*, 110(12-13):481–484, 2010. `doi:10.1016/j.ipl.2010.04.022`.

20 Alexander Pilz. Flip distance between triangulations of a planar point set is APX-hard. *Computational Geometry*, 47(5):589–604, 2014. `doi:10.1016/j.comgeo.2014.01.001`.

21 Larry L. Schumaker. Triangulations in CAGD. *IEEE Computer Graphics and Applications*, 13(1):47–52, 1993. `doi:10.1109/38.180117`.

22 Feng Shi, Jianxin Wang, Yufei Yang, Qilong Feng, Weilong Li, and Jianer Chen. A fixed-parameter algorithm for the maximum agreement forest problem on multifurcating trees. *SCIENCE CHINA Information Sciences*, 59(1):1–14, 2016. `doi:10.1007/s11432-015-5355-1`.

23 Daniel D. Sleator, Robert E. Tarjan, and William P. Thurston. Rotation distance, triangulations, and hyperbolic geometry. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, STOC, Berkeley, USA*, pages 122–135, 1986. `doi:10.1145/12130.12143`.