Julius-Maximilians-Universität Würzburg
Institut für Informatik
Lehrstuhl für Informatik I

Bachelor Thesis

Computing the Flip Distance of Triangulations

Fabian Lipp

March 8, 2012

Supervisors:
Prof. Dr. Alexander Wolff
Dr. Joachim Spoerhase

Abstract

Given two binary trees, it is an interesting question how many tree rotations are needed to transform one tree into the other. This problem is equivalent to the flip distance of two triangulations of a convex polygon, that is, the number of diagonal flips needed to transform one triangulation into a given other one. There is no polynomial-time algorithm known that computes the flip distance of two arbitrary triangulations. We use breadth-first search to compute the flip distance in exponential time and afterwards present some improvements to reduce runtime and memory consumption. We develop a polynomial-time algorithm that computes an upper bound for the flip distance.

We present a simple technique to generate uniformly distributed random triangulations. We use this technique to generate huge sets of random triangulations and compare the upper bound computed by our algorithm to upper bounds found in the literature. Especially for triangulations of polygons with a large number of vertices our heuristic seems to be superior to those found in the literature.

Zusammenfassung

Es ist eine interessante Frage, wie viele Rotationen notwendig sind, um einen vorgegebenen Binärbaum in einen anderen gegebenen Binärbaum zu überführen. Dieses Problem ist äquivalent zur Flip-Distanz zweier Triangulierungen eines konvexen Polygons, das heißt der Anzahl an Diagonalen, die geflippt werden müssen, um die eine Triangulierung in die andere zu überführen. Es ist kein Polynomialzeit-Algorithmus bekannt, der die Flip-Distanz zweier beliebiger Triangulierungen berechnet. Wir verwenden eine Breitensuche, um diese Distanz in exponentieller Zeit zu berechnen und präsentieren anschließend einige Verbesserungen, die die Laufzeit und den Speicherbedarf dieses Algorithmus verringern. Wir entwickeln einen Polynomialzeit-Algorithmus, der eine obere Schranke für die Flip-Distanz berechnet.

Wir beschreiben eine einfache Methode, mit derene Hilfe gleichverteilt zufällige Triangulierungen eines konvexen Polygons erzeugt werden können. Wir verwenden diese Methode um eine große Menge zufälliger Triangulierungen zu erzeugen und vergleichen die obere Schranke, die unser Algorithmus liefert, mit oberen Schranken aus anderen Artikeln. Insbesondere für Triangulierungen von Polygonen mit einer großen Anzahl an Ecken scheint unsere Heuristik denen aus der Literatur überlegen zu sein.

Contents

1 Introduction

Binary trees play an important role as a data structure, for instance as binary search trees. To guarantee an efficient upper bound for the runtime it is essential for most applications that this tree is reasonably balanced, that is, its height is limited by $\mathcal{O}(\log n)$, where n is the number of nodes in the tree. After the insertion of nodes into the tree this property possibly is violated, and thus the tree needs to be restructured. Frequently used implementations of binary search trees (like AVL trees or red-black trees) use the tree rotation to achieve this reorganisation [CLRS09, Section 13.2]. It is a natural question to ask how many of these rotations are needed to transform an arbitrary tree containing n internal nodes into another one with the same number of nodes. The minimum number of rotations needed is called the *rotation distance* of the two trees.

One of the most important article in this field of research is that of Sleator, Tarjan and Thurston [STT88]. They use a bijective mapping between binary trees with $n-2$ internal nodes and triangulations of a convex polygon with n vertices. These vertices are labeled with the numbers from 1 to n. A diagonal flip in the triangulation corresponds to a rotation in the tree. They found working on these triangulations more natural. So the rotation distance between two arbitrary trees with $n-2$ internal nodes is equivalent to the *flip distance* between two triangulations of the n-gon, that is, the minimum number of diagonal flips needed to transform one triangulation into the other one. In the following we will only use the system of triangulations and diagonal flips instead of trees and rotations.

Lucas [Luc87] expected as far back as 1987 that her results will lead to a polynomial time algorithm for finding a shortest flip sequence. Up to date, however, neither does there exist a polynomial-time algorithm nor an NP-hardness proof for this problem. In this thesis, we focus on triangulations of *convex* polygons because under flipping they behave like binary trees under rotations. The flip distance, however, is also of interest for triangulations of point sets that are not in convex position. In their textbook on discrete and computational geometry Devadoss and O'Rourke [DO11] state that developing a polynomial-time algorithm for finding a shortest flip sequence in this generalisation is an unsolved problem.

Upper bounds Culik and Wood [CW82] showed an upper bound of $2n-6$ for the flip distance of two arbitrary triangulations of the n-gon. They prove this result using binary trees. In terms of triangulations of polygons their construction works in the following way: If there are diagonals that are not adjacent to the vertex n in the start triangulation, there is at least one of them which has n as an endpoint after it is flipped. On this way while flipping each diagonal at most once, a fan triangulation can be created, that is, a triangulation in that each diagonal is adjacent to a certain vertex of the polygon (here n). The target triangulation is transformed into the fan in the same way. As the triangulation contains $n-3$ diagonals, the constructed flip sequence contains at most $2n-6$ flips.

Sleator et al. [STT88, Lemma 2] improved the upper bound to $2n-10$ if $n > 12$. They showed that there is a vertex x that is adjacent to at least four diagonals (counting diagonals of both triangulations). They used an approach similar to that of Culik and Wood, but build the fan at vertex x instead of vertex n. In this way they save up to four diagonal flips and thus gain a better upper bound. Furthermore they show that this bound is tight for sufficiently large values of n.

Exact computation The exact flip distance between two triangulations can easily be found by a breadth-first search in the flip graph, that is, the undirected graph that contains all triangulations of the n-gon where two triangulations are adjacent if they can be transformed in each other by a diagonal flip. As the size of this graph is exponential in n, breadth-first search needs exponential time in general.

Fixed-parameter tractability Cleary and St. John [CJ09] showed that flip distance is fixed-parameter tractable in the parameter k, the flip distance. They reduce the initial trees to trees that contain at most $5k$ nodes. Lucas [Luc10] improved their approach using triangulations instead of binary trees and reduced the kernel size to $2k$. She states a worst-case runtime of $\mathcal{O}(k^k)$ for her algorithm. Brandes et al. [BFH$^+$11] developed a simpler and faster algorithm that decides whether the rotation distance between two binary trees is at most k in time $\mathcal{O}(n + 4^k/\sqrt{k})$.

Approximation algorithms There are several approximation algorithms. Cleary and St. John [CJ10] presented a simple linear-time 2-approximation based on the upper bound of Culik and Wood.

Li and Zhang [LZ98] used a more involved technique to achieve an approximation ratio that is better than 2. They developed an algorithm with an approximation ratio of

$$2 - \frac{2}{4(d-1)(d+6)+1}$$

for two triangulations in which each vertex of the convex polygon is incident to at most d diagonals. Furthermore they present algorithms with better approximation ratios for triangulations that fulfil certain conditions (for example a 1.97-approximation for triangulations without internal triangles).

Triangulations of restricted forms Lucas [Luc04] presented a polynomial-time algorithm to compute the flip distance if both triangulations are of a restricted form. If the first input is described as a tree, every node has at most one child. The second triangulation is of an even more restricted nature. Wang et al. [WWLZ08] designed linear-time algorithms to compute the flip distance between certain types of triangulations. The first of these types are fan triangulations. The other types are created by flipping one of the outmost diagonals in a fan triangulation.

Computing lower and upper bounds Baril and Pallo [BP06] developed an algorithm that computes a lower and an upper bound for the rotation distance in polynomial time. They did not prove an approximation ratio, but present statistical results that show the efficiency of these bounds. They state that the upper bounds obtained by an earlier algorithm of Pallo [Pal00] are often better than those of the newer one.

Contribution In this thesis, we discuss an algorithm to compute the exact flip distance using a breadth-first search with some optimisations; see Section 3. Afterwards, we show a simple technique to generate uniformly distributed random triangulations. Our main result is an experimental comparison for a number of heuristics for computing the flip distance, including the heuristics of Baril and Pallo [Pal00, BP06]; see Section 4.

2 Preliminaries

Let $\mathcal{T}_n$ denote the set of all triangulations of the labeled, convex n-gon. In the following, we always assume that polygons are convex and labeled.

Definition 1. Let π be a triangulation, and let d be a diagonal in π. Consider the two triangles adjacent to d. Let a and b be the vertices of these triangles that are not adjacent to d. *Flipping* the diagonal d means replacing it with the new diagonal (a, b). The points a and b are called *triangle points* for d.

Definition 2. Let $n \in \mathbb{N}$. The *flip graph* is the undirected graph $\mathcal{F}_n = (\mathcal{T}_n, E)$ with

$$E = \{\{\pi_1, \pi_2\} \in \mathcal{T}_n \times \mathcal{T}_n \mid \pi_1 \text{ can be transformed to } \pi_2 \text{ with exactly one diagonal flip}\}.$$

Let π_1 and π_2 be two triangulations of the n-gon. The *flip distance* $\operatorname{fd}(\pi_1, \pi_2)$ is the minimum number of diagonal flips required to transform π_1 to π_2, or equivalently the distance of π_1 and π_2 in the flip graph.

Definition 3. Let $\pi_1, \pi_2 \in \mathcal{T}_n$.

a) A diagonal d is called a *common diagonal* if it is contained in π_1 and π_2.

b) A diagonal d in π_1 is called a *flip-to-match diagonal* if it can be flipped to make it match a diagonal of π_2 [BP06].

To determine the number of triangulations for a polygon of a given size n we need the sequence of Catalan numbers. This sequence, named after Eugène Charles Catalan (1814–1894), arises in various counting problems.

Definition 4 ([Dow00])**.** The Catalan numbers are defined by the recurrence relation

$$C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i}, \quad \text{for } n \geq 0$$

with $C_0 = 1$.

Lemma 2.1 ([Dow00])**.** *The Catalan numbers can be calculated by the explicit formula*

$$C_n = \frac{1}{n+1}\binom{2n}{n}.$$

Lemma 2.2 ([Dow00])**.** *The asymptotic behaviour of the Catalan series is*

$$C_n \sim \frac{4^n}{\sqrt{\pi n^3}}.$$

Lemma 2.3 ([Dow00])**.** *For any $n \in \mathbb{N}$, the number of distinct triangulations of the n-gon is $|\mathcal{T}_n| = C_{n-2}$.*

Used data structures In the following algorithms, a diagonal is represented by the indices of its endpoints. To simplify flipping we additionally store for each diagonal the indices of its triangle points. A triangulation is stored as the set of its diagonals. We assume that the basic operations on sets (search, insert, delete) can be performed in logarithmic time (for example using a red-black tree) [CLRS09, Chapter 13]. Furthermore we use priority queues that execute the operations Insert, ExtractMin and DecreaseKey in logarithmic time [CLRS09, Section 6.5].

3 Exact Algorithms

In this section we present a simple algorithm that computes a shortest flip sequence between two given triangulations. Because of its exponential runtime and memory consumption, this algorithm is only feasible for triangulations of relatively small polygons ($n \leq 15$). Afterwards we present some improvements to this algorithm that reduce its running time and its memory consumption.

3.1 Simple Breadth-First Search

An intuitive approach to find a shortest path between two triangulations is to perform a breadth-first search (BFS) in the flip graph. The flip graph consists of C_{n-2} vertices. Therefore its size and thus the worst case runtime of BFS is exponential in n.

The pseudocode for the simple BFS is shown in Algorithm 3.1. The queue Q keeps track of the triangulations that need to be visited. The set F contains all triangulations that were found during the search. The triangulations are interpreted as a set of diagonals. For each triangulation x, the distance from s is stored in $x.\mathrm{d}$ and the predecessor on a shortest path from s to x is stored in $x.\pi$. After the termination of the algorithm a shortest path can be obtained by following the π-pointers from t until s is reached. Because of its huge memory consumption, this approach can only be used for polygons with a small number of vertices.

3.2 Improvements to Breadth-First Search

The limit of the algorithm can be raised by decreasing the search space. In the following we present some methods to achieve this. For two of these improvements we use a lemma of Sleator et al.:

Lemma 3.1 ([STT88, Lemma 3])**.** *Let π_1 and π_2 be triangulations of the n-gon.*

a) If π_1 contains a flip-to-match diagonal d, then there is a shortest π_1–π_2 path in $\mathcal{F}_n$ where d is flipped first.

b) If π_1 and π_2 have a diagonal in common, then this diagonal is never flipped on a shortest π_1–π_2 path.

Common diagonal We check for each diagonal in the for-each loop whether it is contained in the target triangulation. In this case the triangulation that is reached by flipping this common diagonal is not added to the queue because this triangulation cannot be on a shortest path from the current triangulation to the target (Lemma 3.1 b).

Algorithm 3.1: Simple breadth-first search

```
Input: start triangulation $s$, target triangulation $t$
Output: flip distance between $s$ and $t$
$Q \leftarrow$ new Queue() // for triangulations
$F \leftarrow$ new Set() // for triangulations
$s$.d $\leftarrow 0$
$s.\pi \leftarrow$ nil
$Q$.Enqueue($s$)
$F$.Add($s$)
while $Q \neq \emptyset$ do
    $cur \leftarrow Q$.Dequeue()
    foreach $diag \in cur$ do
        $next \leftarrow$ triangulation that arises from $cur$ by flipping $diag$
        if $next \notin F$ then
            $next$.d $\leftarrow cur$.d + 1
            $next.\pi \leftarrow cur$
            $Q$.Enqueue($next$)
            $F$.Add($next$)
            if $next == t$ then
                return $next$.d
```

Flip-to-match diagonal The second part of the lemma can be used, too: Whenever a triangulation is taken from the queue, the modified algorithm checks whether one of its diagonals is a flip-to-match diagonal. In this case the triangulation that is created by flipping this flip-to-match diagonal is the only one that is appended to the queue (according to Lemma 3.1 a).

A* search algorithm The number of visited nodes can be reduced further using the A* algorithm [HNR68]. It needs a heuristic that provides a lower bound for the distance to the target for every triangulation. A lower bound, which is easy to compute, is $n - c - 3$, where n is the number of vertices in the polygon and c is the number of diagonals that the triangulation has in common with the target [CJ10, Theorem 1]. The A* algorithm does not visit the nodes in the order in which they are found (as BFS), but chooses the node for which the sum of the distance from the start and the lower bound computed by the heuristic is minimum. Expressed graphically, the search moves in a more directed fashion from the start to the target triangulation.

More compact structure for triangulations An additional way to reduce the amount of allocated memory is a more compact format to store the triangulations. There are a lot of interpretations of the Catalan numbers which are in one-to-one-correspondence to triangulations of a n-gon, for example, the well-formed sequences of parentheses involving

$n-2$ left and $n-2$ right parentheses [Dow00, Dow91]. With this representation, a triangulation of the n-gon can be stored in less than $2n$ bits.

Improved algorithm The improvements that we just presented are combined in Algorithm 3.2. The priority queue Q keeps track of the triangulations that need to be visited. It has the method $Q.\mathsf{Insert}(x, v)$ which inserts the new element x with a value of v. The operation $Q.\mathsf{ExtractMin}()$ removes the element $x \in Q$ with minimum value. The number of common diagonals in the triangulations x and t is given by $|x \cap t|$. The for-each loop in lines 9–17 checks the presence of a flip-to-match diagonal. Line 19 checks whether the current triangulation has a diagonal in common with the target.

Further possible improvements There are several ways how this algorithm could be further improved: A better heuristic for the lower bound would cause the algorithm to visit fewer nodes and thus save memory and runtime. Furthermore the A* algorithm can be generalised to a bidirectional search algorithm and still be guided by a heuristic [SdC77]. Finally, nodes that have already been visited (in the sense that they have been removed from the (priority) queue in BFS) could be dropped to reduce the memory consumption. With this modification it would not be possible to determine a shortest path, but the algorithm would still yield the exact flip distance.

3.3 Runtime

We show an upper bound for the worst-case runtime of the improved algorithm (Algorithm 3.2). The set F contains all nodes of the flip graph $\mathcal{F}_n$ in the worst case, that is, $|F| \leq C_{n-2}$. The procedure AddSuccessor (lines 25–35) needs time $\mathcal{O}(n)$ to compute $|next \cap t|$, $\mathcal{O}(\log |F|)$ for the check in line 26 and $F.\mathsf{Add}$, and $\mathcal{O}(\log |Q|)$ for $Q.\mathsf{Insert}$ or $Q.\mathsf{DecreaseKey}$. Therefore AddSuccessor requires a worst-case runtime of $\mathcal{O}(n + \log C_{n-2}) = \mathcal{O}(n)$ (using Lemma 2.2).

The while loop (line 7) is repeated at most C_{n-2} times. In each of these repetitions the procedure AddSuccessor is called at most once for each diagonal in the triangulation, that is, at most $n-3$ times. This leads to a total worst-case runtime of $\mathcal{O}(n^2\, C_{n-2})$. Using the estimation of Lemma 2.2 we get an exponential runtime of $\mathcal{O}(\sqrt{n}\, 4^n)$.

3.4 Effects of the Improvements

Finally, we present the results of computer experiments that show the effects of the improvements discussed in Section 3.2. We created a set of 100 random pairs of triangulations for each tested value of n and computed the flip distance using BFS with and without the improvements. The algorithms were implemented in Java. The following variants of BFS were tested:

1. Simple breadth-first search (Algorithm 3.1). Triangulations are stored as a TreeSet.

Algorithm 3.2: A* search algorithm

```
   Input: start triangulation $s$, target triangulation $t$ of the $n$-gon
   Output: flip distance between $s$ and $t$
 1 $Q \leftarrow$ new PriorityQueue() // for triangulations
 2 $F \leftarrow$ new Set() // for triangulations
 3 $s$.d $\leftarrow 0$
 4 $s.\pi \leftarrow$ nil
 5 $Q$.Insert($s$, $n - |s \cap t| - 3$)
 6 $F$.Add($s$)
 7 while $Q \neq \emptyset$ do
 8     $cur \leftarrow Q$.ExtractMin()
       // check whether $cur$ has a flip-to-match diagonal
 9     foreach $diag \in cur$ do
10         $diag' \leftarrow$ flipped counterpart of $diag$
11         if $diag' \in t$ then
12             $next \leftarrow$ triangulation that arises from $cur$ by flipping $diag$
13             if $next \notin F$ then
14                 AddSuccessor($cur$, $next$)
15                 if $next == t$ then
16                     return $next$.d
17             continue while
18     foreach $diag \in cur$ do
19         if $diag \notin t$ then
20             $next \leftarrow$ triangulation that arises from $cur$ by flipping $diag$
21             if $next \notin F$ then
22                 AddSuccessor($cur$, $next$)
23                 if $next == t$ then
24                     return $next$.d
25 procedure AddSuccessor($cur$, $next$):
26 if $next \notin F$ then
27     $next$.d $\leftarrow cur$.d $+ 1$
28     $next.\pi \leftarrow cur$
29     $Q$.Insert($next$, $next$.d $+ n - |next \cap t| - 3$)
30     $F$.Add($next$)
31 else
32     if $cur$.d $+ 1 < next$.d then
33         $next$.d $\leftarrow cur$.d $+ 1$
34         $next.\pi \leftarrow cur$
35         $Q$.DecreaseKey($next$, $next$.d $+ n - |next \cap t| - 3$)
```

2. Triangulations are stored in a more compact structure using a `long` for each triangulation.

3. Additionally the check for common diagonals is performed.

4. Additionally the check for flip-to-match diagonals is performed.

5. Use the A* search algorithm including all previous improvements (Algorithm 3.2).

For each algorithm we record the number of nodes that were visited by the search, that is, they have been removed from the (priority) queue. The tests were executed on a 2.8 GHz CPU using a memory limit of 4 GB for the Java Virtual Machine. The results are shown in Figure 3.1.

The graph shows clearly the benefits of the improvements. The simple BFS without improvements is only usable for $n \leq 15$ with the given memory limit. The highest flip distance found in the random sample for $n = 15$ is 18. The A* algorithm can be used for $n \leq 22$, the highest found flip distance is 29. The number of nodes visited by A* is considerably smaller than the number of nodes visited by simple BFS. Note that the y-axis in the graph has a log scale.

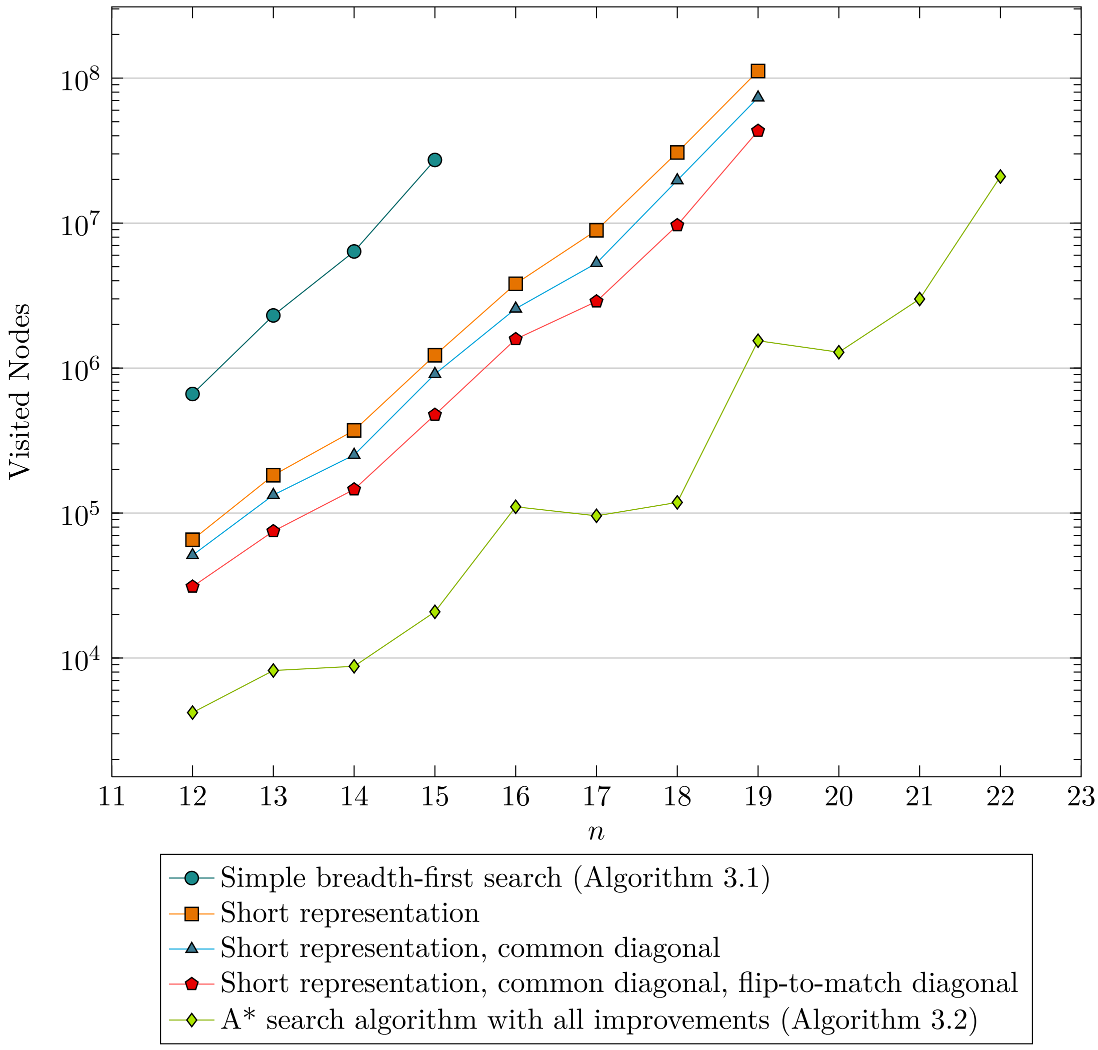


Figure 3.1: The graph shows for each n the number of nodes visited by the search algorithm for 100 randomly chosen pairs of triangulations of the given size. The improvements described in Section 3.2 are added one after the other. Note that the y-axis has a log scale.

4 Heuristics

In this section we first discuss how to create uniformly distributed random triangulations, on which the heuristics can be tested. We present three natural polynomial-time algorithms that compute an upper bound for the flip distance between two triangulations of a convex polygon. These algorithms are quite similar to each other. Afterwards we compare the best one of these to upper bounds found in literature. Furthermore we show an attempt for a straightforward 2-approximation and prove that it does not work out.

4.1 Generating Random Triangulations

As it is not possible to test the implemented heuristics on all pairs of triangulations with a certain amount of vertices in reasonable time, it is necessary to create random triangulations.

A naive approach is the following greedy algorithm: to create a triangulation with n vertices, start with an empty polygon and repeatedly add random diagonals which do not intersect the existing diagonals. The triangulation is complete as soon as $n - 3$ diagonals have been added.

The drawback of this strategy is that the resulting triangulations are not uniformly distributed. This fact is illustrated by the following example. Consider the triangulation π of the hexagon which is depicted in Figure 4.1. There are $6 \cdot 3/2 = 9$ possible diagonals in a hexagon, that is, every diagonal has a probability of $1/9$ to get selected as the first diagonal by a uniformly distributed random generator. Hence there is a chance of $3/9$ that the first selected diagonal is one of the diagonals of π. In the remaining pentagon there are $5 \cdot 2/2 = 5$ possible diagonals and thus a probability of $2/5$ to select one of π. For the last diagonal there are 2 possibilities, that is, a chance of $1/2$ to select the last diagonal of π. So the probability to generate π with the greedy algorithm is $3/9 \cdot 2/5 \cdot 1/2 = 1/15$. There are $C_4 = 14$ triangulations of a hexagon. Therefore the greedy algorithm does not create a uniform distribution because the probability for π is only $1/15$ and thus smaller than $1/C_4$.

We use instead a more difficile technique to generate a uniformly distributed random triangulation. Knott [Kno77] presented algorithms that establish a bijection between the integers in $[1, C_n]$ and the set of n-node binary trees (correlating to the triangulations of a polygon with $n - 2$ vertices). Therefore it is sufficient to pick an integer from $[1, C_n]$ under a uniform random distribution and transform it into the corresponding triangulation.

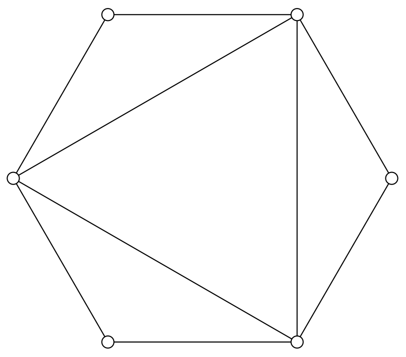


Figure 4.1: A triangulation of the hexagon which shows that the greedy algorithm for generating random triangulations does not create a uniform distribution.

4.2 Simple Heuristics

The heuristics presented in this section are all working in a similar way. The common basis for the heuristics is shown in algorithm 4.1. They begin with the start triangulation. In each step one diagonal specified by the function SelectDiagonal is flipped (line 4). This function differs in the various heuristics. The algorithm finishes when the target triangulation is reached.

Algorithm 4.1: Simple Heuristics

```
   Input: start triangulation $s$, target triangulation $t$ of the $n$-gon
   Output: $s$–$t$ path in $\mathcal{F}_n$
1  $L \leftarrow$ new List()
2  $cur \leftarrow s$
3  while $cur \neq t$ do
4      $diag \leftarrow$ SelectDiagonal($cur$, $t$)
5      flip the diagonal $diag$ in $cur$
6      $L$.Add($cur$)
7  return $L$
```

The auxiliary Function 4.2 is used by the heuristics to determine the number of diagonals in a triangulation that intersect a certain diagonal.

4.2.1 Variants

There are some reasonable methods to select the diagonal to flip. In the following the function SelectDiagonal is shown for each of them.

MostIntersections Select the diagonal that has a maximum number of intersections with the target triangulation (Function 4.3).

Function 4.2: CountIntersections(d, t)

```
Input: diagonal $d$, triangulation $t$
Output: number of diagonals in $t$ that intersect $d$
1 $i \leftarrow 0$
2 foreach $d' \in t$ do
3     if $d$ and $d'$ do intersect then
4         $i++$
5 return $i$
```

FewestIntersectionsAfter Select the diagonal that has a minimum number of intersections after being flipped (Function 4.4).

HighestDifference Compute the difference between the number of intersections before and after flipping for each diagonal. Select the diagonal for which this difference is minimum (Function 4.5).

The HighestDifference heuristic was already studied by Hong and Lee [HL97]. In contrast to this thesis, they tested it only on small polygons ($n < 20$).

Function 4.3: SelectDiagonalWithMostIntersections(cur, t)

```
1 $max \leftarrow 0$
2 foreach $diag \in cur$ do
3     $i \leftarrow$ CountIntersections($diag, t$)
4     if $i > max$ then
5         $max \leftarrow i$
6         $maxDiag \leftarrow diag$
7 return $maxDiag$
```

Function 4.4: SelectDiagonalWithFewestIntersectionsAfter(cur, t)

```
1 $min \leftarrow \infty$
2 foreach $diag \in cur$ do
3     $i \leftarrow$ CountIntersections($diag, t$)
4     $diag' \leftarrow$ flipped counterpart of $diag$
5     $i' \leftarrow$ CountIntersections($diag', t$)
6     if $i' < min$ and $i' < i$ then
7         $min \leftarrow i'$
8         $minDiag \leftarrow diag$
9 return $minDiag$
```

Function 4.5: SelectDiagonalWithHighestDifference(cur, t)

```
1  $max \leftarrow 0$
2  foreach $diag \in cur$ do
3      $i \leftarrow$ CountIntersections($diag, t$)
4      $diag' \leftarrow$ flipped counterpart of $diag$
5      $i' \leftarrow$ CountIntersections($diag', t$)
6      $\Delta \leftarrow i - i'$
7      if $\Delta > max$ then
8          $max \leftarrow \Delta$
9          $maxDiag \leftarrow \Delta$
10 return $maxDiag$
```

4.2.2 Runtime

All of these heuristics have the same asymptotic runtime. We discuss the runtime of the HighestDifference algorithm exemplarily. Let the input be triangulations of the n-gon. To test two diagonals for an intersection only their endpoints have to be considered. Therefore this check can be performed in constant time and the runtime of the function CountIntersections (Function 4.2) is $\mathcal{O}(n)$. Determining the flipped counterpart of a diagonal needs constant time as the triangle points are stored with the diagonal. Therefore the function SelectDiagonalWithHighestDifference has a total runtime of $\mathcal{O}(n^2)$.

Flipping a diagonal requires to replace the diagonal in the set of the according triangulation and updating the triangle points in the neighbouring diagonals. This can be done in time $\mathcal{O}(\log n)$. Hence each iteration of the loop in Algorithm 4.1 needs $\mathcal{O}(n^2)$ time. The while loop is repeated once for each of the flips in the returned triangulation sequence. The algorithm can be aborted if this sequence contains $2n$ flips and instead return the path obtained using the construction for the upper bound of [CW82]. So the loop is repeated $\mathcal{O}(n)$ times leading to a total runtime of $\mathcal{O}(n^3)$.

The runtime for SelectDiagonal can be reduced to $\mathcal{O}(n)$ by storing the difference of the intersection counts before and after the flip for each diagonal. So the function CountIntersections is not called within SelectDiagonal. While flipping a diagonal, this information has to be updated for the neighbouring diagonals. That can be done in time $\mathcal{O}(n)$. This improvement leads to a total runtime of $\mathcal{O}(n^2)$.

4.2.3 Comparison

To compare the variants we first give exhaustive results for small instances, that is, $n \in \{5, 6, \dots, 12\}$. We execute the different algorithms on all pairs $(s, t) \in \mathcal{T}_n \times \mathcal{T}_n$ and check which of the variants leads to the best results for each of them. The best result means naturally the shortest of the computed sequences. The results are given in Table 4.1.

To compare the efficiency of the variants on bigger instances a large set of randomly

n	5	6	7	8	9	10	11	12
MostIntersections	100	100	99.2	97.8	96.0	93.7	91.0	88.1
FewestIntersectionsAfter	100	100	99.1	97.6	95.6	93.1	90.3	87.1
HighestDifference	100	100	99.9	99.7	99.5	99.2	98.7	98.2

Table 4.1: The table lists for all $n \in \{5, 6, \ldots, 12\}$ the percentage of pairs $(s, t) \in \mathcal{T}_n \times \mathcal{T}_n$ $(s \neq t)$ for which the specified heuristic provides the best result (that is, the shortest flip sequence compared to the other two variants).

chosen triangulations is created. The results of these experiments are documented in Figure 4.2. The graphs show clearly that HighestDifference is the best of the three variants.

4.2.4 Non-Optimality

There are triangulations of small polygons for which the HighestDifference heuristic does not yield an optimal result. Consider the triangulations π_1 and π_2 of the octagon shown in Figure 4.3. In the first step the heuristic flips diagonal b because it is the only one that reduces the intersection count by 3. After that, no common diagonal can be reached by the second flip. This means that there are no common diagonals after two flips and therefore at least five more flips are required (one for each diagonal). Hence the sequence constructed by the heuristic needs at least 7 flips.

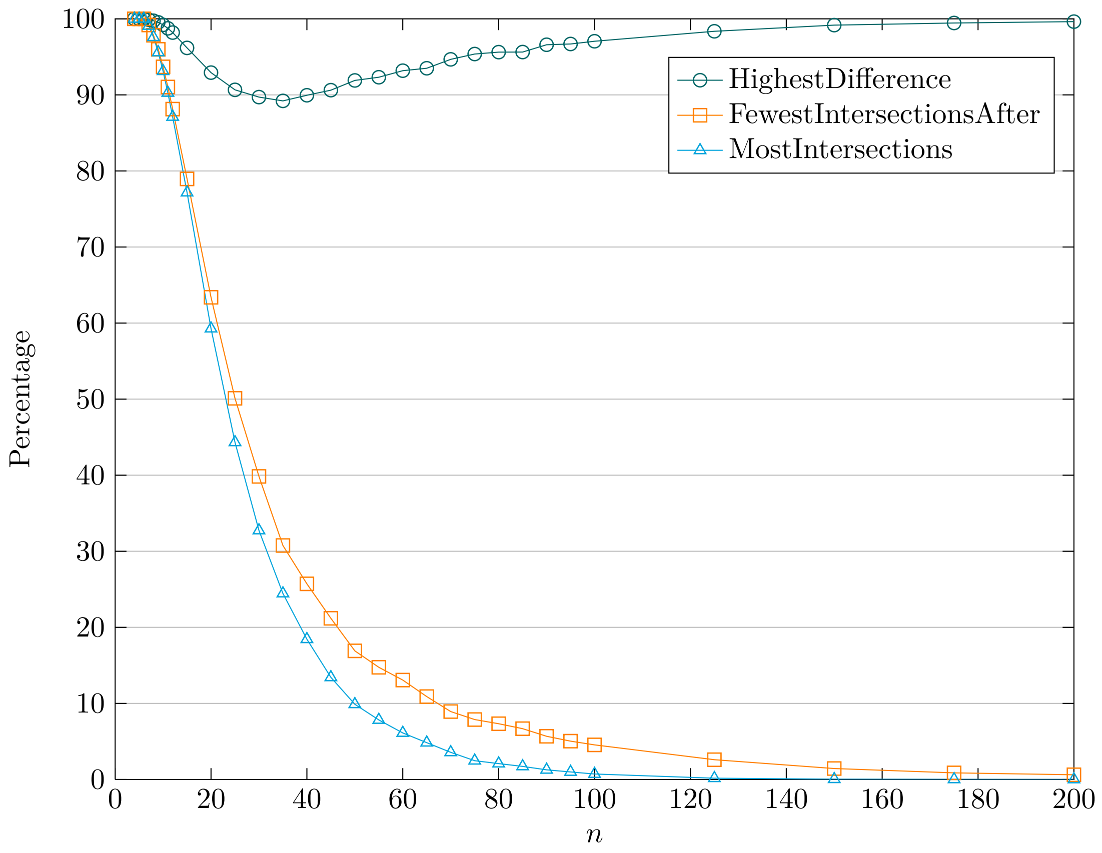
But a shorter flip sequence is possible: If diagonal a had been flipped in the first step, with each of the following flips a common diagonal could have been reached. Hence $\mathrm{fd}(\pi_1, \pi_2) = 6$, and so the sequence constructed by the heuristic is not optimal.

4.2.5 Approaches for Improvements

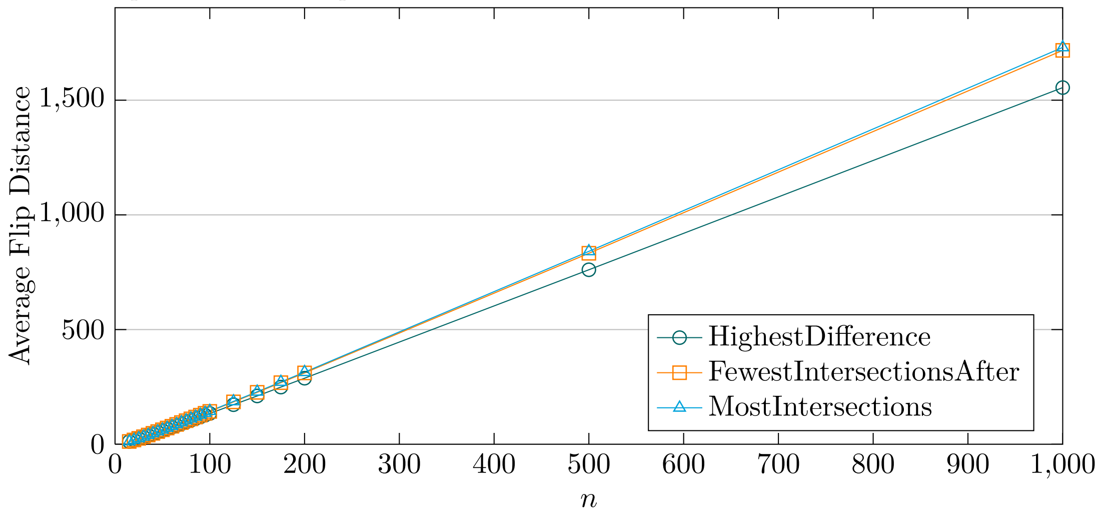
The experimental results showed that HighestDifference is clearly the best variant of the heuristic. This brings up the question if it is possible to improve the heuristic even further. We can first try to compare the intersection difference relative to the count of intersections before flipping the diagonal. To be more precise we calculate $(i - i')/i$ in line 6 of Function 4.5 and return the diagonal for which this fraction is maximum. Experiments show that this variation does not improve the results. Actually the original heuristic is mostly better than the modified one.

Another idea is to consider the length l of the diagonal (that is, the distance between the endpoints on the polygon boundary) and prefer short diagonals if multiple diagonals have the same intersection difference. Another option is to compare the intersection difference relative to the length of the diagonals, that is, selecting the diagonal for which $(i - i')/l$ is maximum. These variants do not improve the heuristic as well.

Additionally, the algorithm could look for flip-to-match diagonals in the target triangulation. But experiments give evidence that this variation does not have a big effect on the result as well.



(a) The graph shows for certain values of n for each algorithm the percentage of the randomly chosen pairs for which it provides the best results.



(b) The graph shows the average flip distance between the randomly chosen pairs.

Figure 4.2: For $n \in [15, 30]$ we tested 1 000 000 randomly chosen pairs, for $n \in [35, 200]$ we tested 10 000 pairs, for $n \in [500, 1000]$ we tested 1 000 pairs.

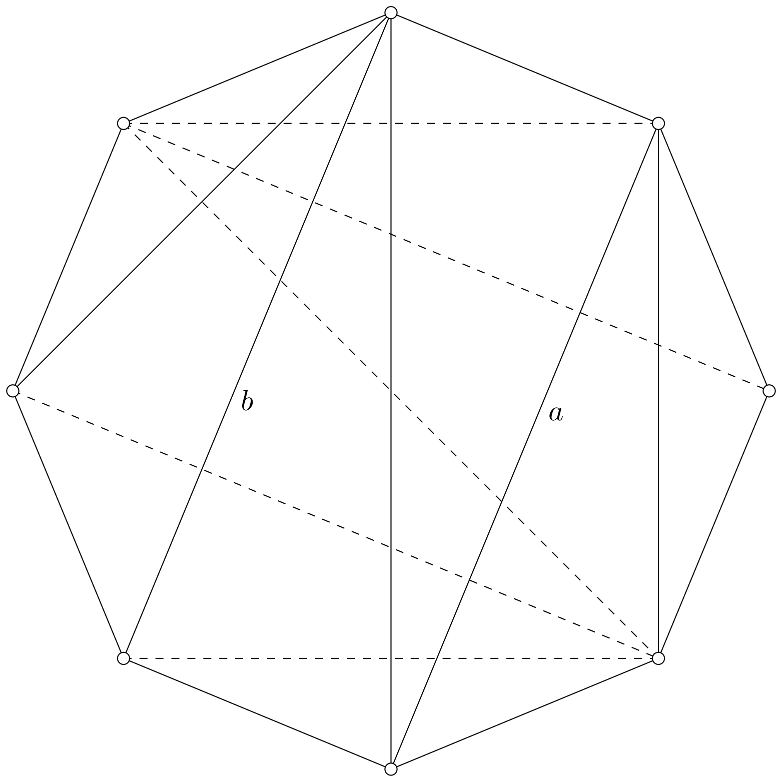


Figure 4.3: Two triangulations of the octagon showing that the HighestDifference heuristic does not always lead to optimal results. The diagonals of π_1 are drawn solid, those of π_2 are dashed.

4.3 Comparison to Other Heuristics

We are going to compare the upper bound for the flip distance provided by the HighestDifference heuristic to those proposed by Pallo and Baril. Pallo's first heuristic [Pal00] operates on weight sequences representing binary trees [Pal86]. Based on a restricted tree rotation he constructs a lattice containing these trees (and thus equivalent weight sequences). To obtain an upper bound of the rotation distance between to trees, Pallo's algorithm computes the lengths of paths in this lattice. Additionally he uses the flexion of the trees, that is, rotating the corresponding triangulation by one vertex. Rotating both the start and the target triangulation does not change the flip distance between them, but the algorithm can possibly compute a better upper bound on the pair of rotated triangulations. Pallo states a worst-case runtime of $\mathcal{O}(n^3)$ for his algorithm. We refer to this heuristic as PalloUp in the following.

The second heuristic by Baril and Pallo [BP06] does not only provide an upper bound but also a lower bound. In the algorithm they transform the given triangulations into T' and S' to create an additional common edge. They provide an estimation of $\mathrm{fd}(T, S)$ in relation to $\mathrm{fd}(T', S')$. Using this technique recursively for T' and S' the algorithm results finally in a lower and upper bound for $\mathrm{fd}(T, S)$. They give a worst-case runtime

n	4	6	7	8	9	10	11	12
HighestDifference	100.0	100.0	99.4	98.3	96.8	94.9	92.8	90.6
BarilPalloUp	100.0	100.0	99.5	98.7	97.5	95.8	93.7	91.3
PalloUp	100.0	100.0	100.0	100.0	99.9	99.5	98.7	97.1

Table 4.2: The table lists for all $n \in \{5, 6, \ldots, 12\}$ the percentage of pairs $(s, t) \in \mathcal{T}_n \times \mathcal{T}_n$ $(s \neq t)$ for which the specified heuristic provides the best result (that is, the shortest sequence compared to the other heuristics).

of $\mathcal{O}(n^3)$. We call this upper bound heuristic BarilPalloUp.

To compare these algorithms to the HighestDifference heuristic developed in the previous sections, we use experiments similar to those in Section 4.2.3. The results are documented in Table 4.2 and Figure 4.4. These experiments show clearly that for triangulations of large polygons ($n \geq 20$) the HighestDifference heuristic is superior to those of Baril and Pallo.

Figure 4.5 shows detailed results for a set of randomly chosen pairs of triangulations. For each pair the results of the three heuristics are plotted. Additionally a simple upper and lower bound is given for each pair. The lower bound is the number of diagonals that are not common between the start and the target triangulation. The upper bound is computed by doubling the value of the lower bound [CJ10, Theorem 1]. Furthermore the lower bound computed by the algorithm of Baril and Pallo [BP06] is given.

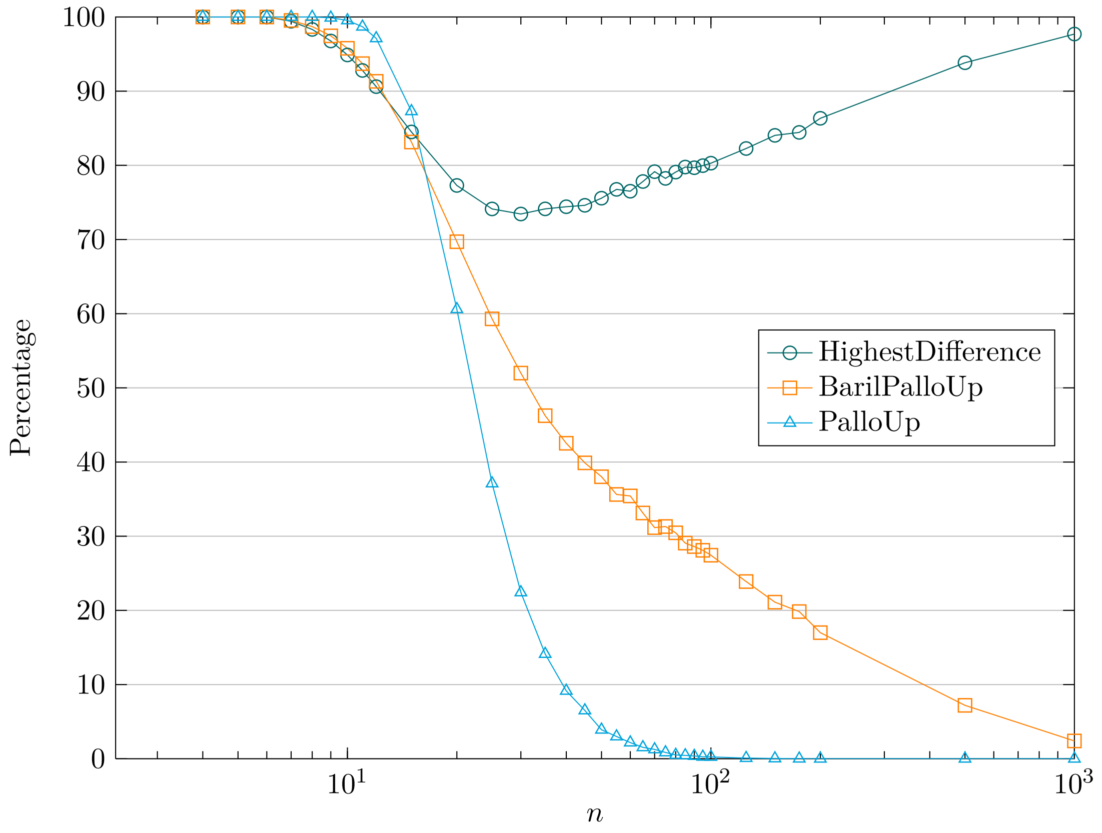
4.4 Attempt for a 2-Approximation

The following conjecture would lead to a straightforward 2-approximation for the flip distance between two triangulations:

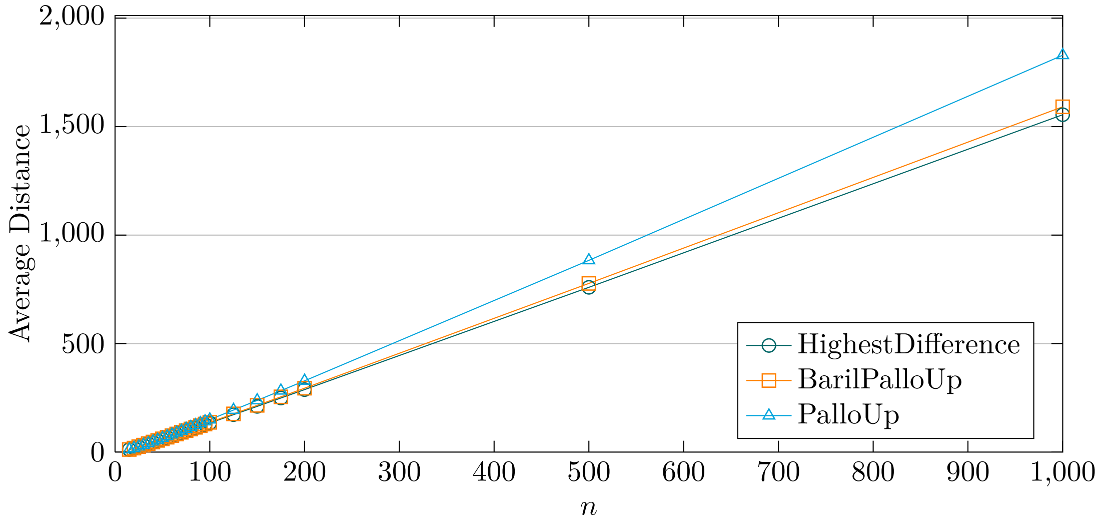
Conjecture. *It is possible to create an additional common diagonal in every pair of triangulations by performing at most two flips. These two flips can either be applied consecutively in one of the triangulations or one flip in each of the triangulations.*

This conjecture is proven wrong by the triangulations π_1 and π_2 shown in Figure 4.6. This example is symmetric since rotating the diagonals of π_1 by two vertices results in π_2 and rotating them by four vertices leads to π_1 itself. There is no common diagonal and each diagonal has at least three intersections with diagonals of the other triangulation.

We assume without loss of generality that the first flip is performed in π_1. Observe that after this step every diagonal still has at least two intersections with diagonals of the other triangulation and therefore the triangulations do not have a common diagonal. We now look at an arbitrary diagonal in π_2, which is intersected by at least two diagonals of π_1. By flipping one diagonal in π_1 we can only eliminate one of these intersections and thus cannot produce a common diagonal. By an analogous argument we see that it is not possible to generate a common diagonal via performing a flip in π_2. Therefore the triangulations do not share a common edge after two arbitrary flips.



(a) The graph shows for certain values of n for each algorithm the percentage of the randomly chosen pairs for which it provides the best results. Note that the x-axis has a log scale.



(b) The graph shows the average flip distance between the randomly chosen pairs.

Figure 4.4: For $n \in [15, 30]$ we tested 1 000 000 randomly chosen pairs, for $n \in [35, 500]$ we tested 10 000 pairs, for $n = 1000$ we tested 1 000 pairs.

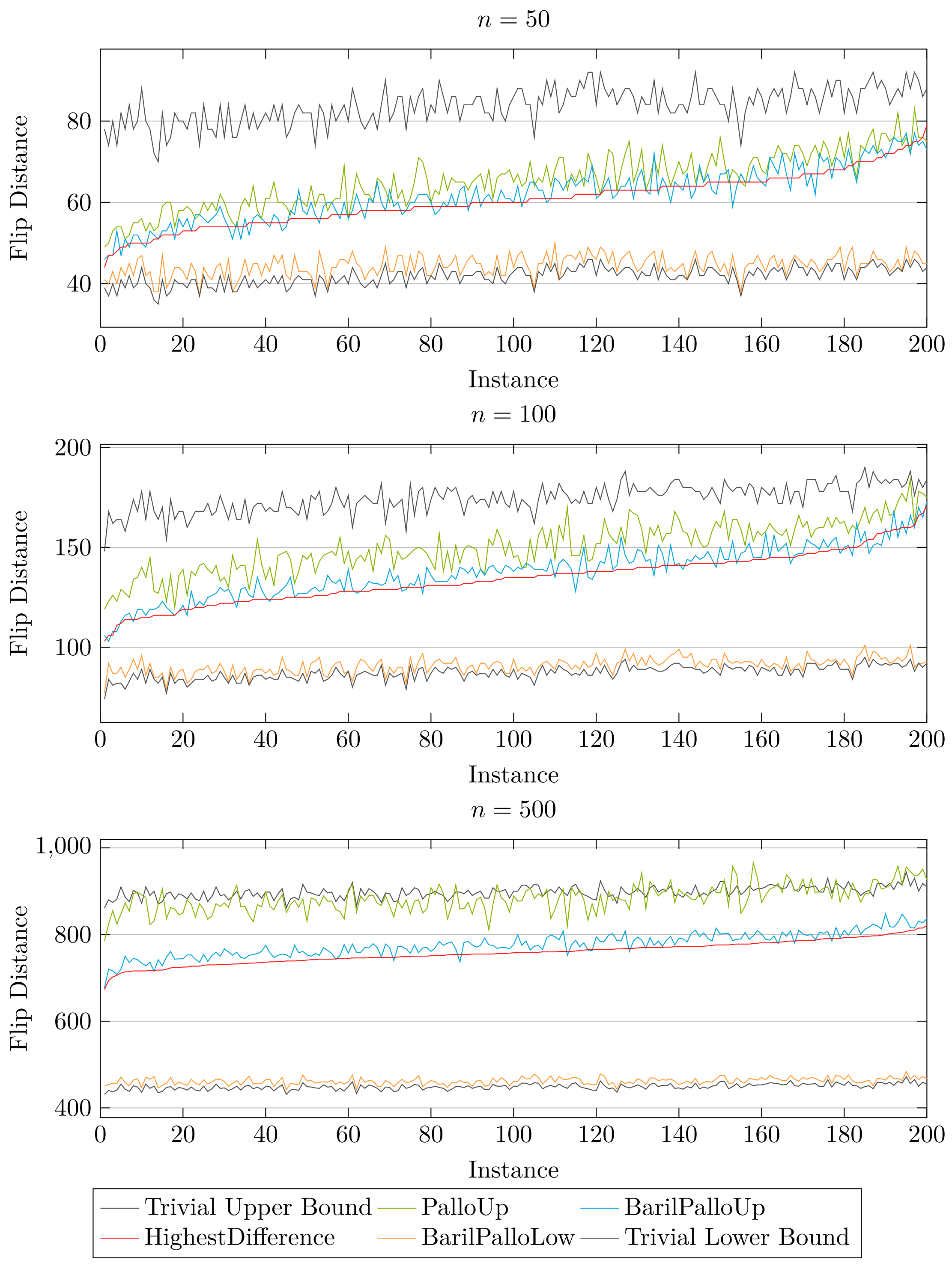


Figure 4.5: The graphs show for $n \in \{50, 100, 500\}$ the results of the heuristics for 200 randomly chosen pairs of triangulations in each case. These pairs are sorted by the flip distance computed by the HighestDifference heuristic.

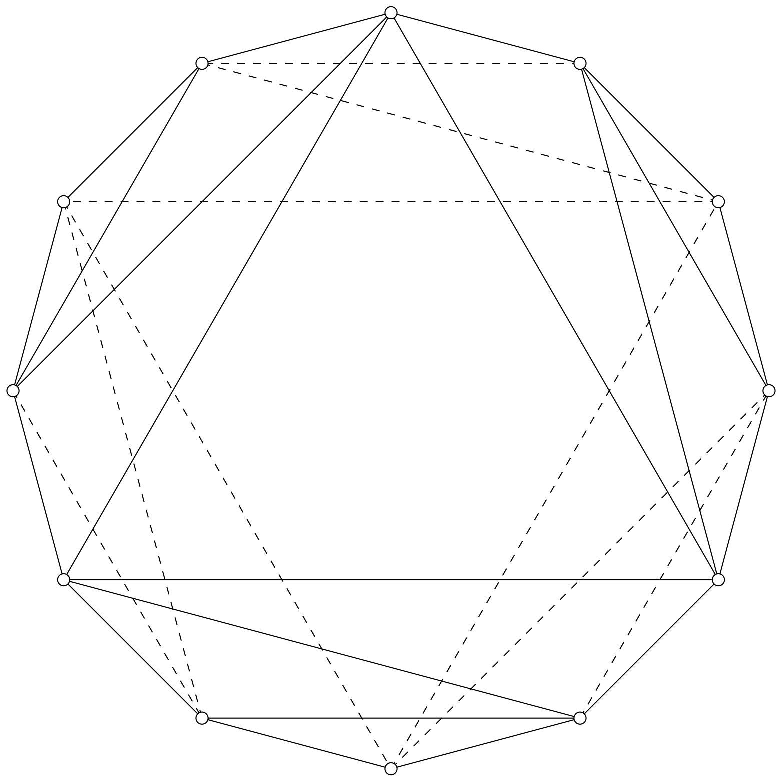


Figure 4.6: The triangulations of the 12-gon used to falsify the conjecture in Section 4.4. The diagonals of π_1 are drawn solid, those of π_2 are dashed.

5 Conclusion

We have presented an algorithm that computes the exact flip distance between two triangulations of a polygon of reasonable size. This algorithm is based on breadth-first search. Hence its worst-case runtime is exponential in the number of vertices in the polygon. The runtime and memory consumption of the algorithm can be reduced by decreasing the number of nodes that are visited by the search. We achieved this using a heuristic to guide the search (A* search algorithm) and checking for common and flip-to-match diagonals. The number of expanded nodes could be further reduced using a better lower bound for the A* algorithm.

Following this, we have developed a polynomial-time heuristic that computes a path between two given triangulations in the flip graph. Computer experiments show that this algorithm yields better results (that is, shorter paths) than heuristics found in the literature for triangulations of large polygons ($n > 20$). We discussed some ideas that did not improve the heuristic.

Additionally, we have presented a simple technique to generate a set of uniformly distributed random triangulations. We used this technique throughout the thesis to demonstrate the exponential-time algorithm and to test the efficiency of the heuristic.

The main problem, a polynomial-time algorithm that computes the flip distance of two arbitrary triangulations, remains unsolved. Our heuristic seems to be effective especially for triangulations of large polygons, but we cannot guarantee a certain approximation ratio. The approximation factor of the best algorithm known is only slightly below 2. It would be interesting if there are algorithms that guarantee a better approximation. Moreover, triangulations with certain properties could be considered. Exact algorithms are known for triangulations that are rather similar to fan triangulations. Perhaps this algorithms could be enhanced to work on a more general class of triangulations.

It could be useful to study the properties of the flip graph $\mathcal{F}_n$. Sleator et al. showed that the diameter of this graph is $2n - 10$ for sufficiently large values of n. It is also known that this graph is hamiltonian. A better understanding of the structure of the flip graph could lead to better approximations or even to a faster exact algorithm.

Another interesting result would be the classification of the problem in terms of complexity theory. The problem is in NP obviously, but there is no proof for NP-hardness yet. It may be possible to show that flip distance is a member of other well-known complexity classes.

Bibliography

[BFH$^+$11] Ulrik Brandes, Rudolf Fleischer, Seok Hee Hong, Tamara Mchedlidze, Ignaz Rutter, and Alexander Wolff: On the rotation distance of rooted binary trees. In Camil Demetrescu, Michael Kaufmann, Stephen Kobourov, and Petra Mutzel (editors): *Graph Drawing with Algorithm Engineering Methods*, volume 1 of *Dagstuhl Reports*, pages 57–59. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2011.

[BP06] Jean Luc Baril and Jean Marcel Pallo: Efficient lower and upper bounds of the diagonal-flip distance between triangulations. *Information Processing Letters*, 100(4):131–136, 2006.

[CJ09] Sean Cleary and Katherine St. John: Rotation distance is fixed-parameter tractable. *Information Processing Letters*, 109(16):918–922, 2009.

[CJ10] Sean Cleary and Katherine St. John: A linear-time approximation for rotation distance. *Journal of Graph Algorithms and Applications*, 14(2):385–390, 2010.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein: *Introduction to algorithms.* The MIT Press, third edition, 2009.

[CW82] Karel Culik II and Derick Wood: A note on some tree similarity measures. *Information Processing Letters*, 15(1):39–42, 1982.

[DO11] Satyan L. Devadoss and Joseph O'Rourke: *Discrete and Computational Geometry.* Princeton University Press, 2011.

[Dow91] Thomas A. Dowling: Catalan numbers. In John G. Michaels and Kenneth H. Rosen (editors): *Applications of Discrete Mathematics*, chapter 7. McGraw-Hill, 1991.

[Dow00] Thomas A. Dowling: Special sequences. In Kenneth H. Rosen (editor): *Handbook of Discrete and Combinatorial Mathematics*, chapter 3.1. CRC Press, 2000.

[HL97] Seok Hee Hong and Sang Ho Lee: The flip distance problem of triangulations. In *Proceedings of the Japan-Korea Joint Workshop on Algorithms and Computation*, pages 9–15, 1997.

[HNR68] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael: A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[Kno77] Gary D. Knott: A numbering system for binary trees. *Communications of the ACM*, 20(2):113–115, 1977.

[Luc87] Joan M. Lucas: The rotation graph of binary trees is Hamiltonian. *Journal of Algorithms*, 8(4):503–535, 1987.

[Luc04] Joan M. Lucas: Untangling binary trees via rotations. *The Computer Journal*, 47(2):259–269, 2004.

[Luc10] Joan M. Lucas: An improved kernel size for rotation distance in binary trees. *Information Processing Letters*, 110(12–13):481–484, 2010.

[LZ98] Ming Li and Louxin Zhang: Better approximation of diagonal-flip transformation and rotation transformation. In Wen Lian Hsu and Ming Yang Kao (editors): *Proceedings of the 4th Annual International Conference on Computing and Combinatorics*, volume 1449 of *Lecture Notes in Computer Science*, pages 85–94. Springer Berlin / Heidelberg, 1998.

[Pal86] J. M. Pallo: Enumerating, ranking and unranking binary trees. *The Computer Journal*, 29(2):171–175, 1986.

[Pal00] Jean Pallo: An efficient upper bound of the rotation distance of binary trees. *Information Processing Letters*, 73(3–4):87–92, 2000.

[SdC77] Lenie Sint and Dennis de Champeaux: An improved bidirectional heuristic search algorithm. *Journal of the Association for Computing Machinery*, 24(2):177–191, 1977.

[STT88] Daniel D. Sleator, Robert E. Tarjan, and William P. Thurston: Rotation distance, triangulations, and hyperbolic geometry. *Journal of the American Mathematical Society*, 1(3):647–681, 1988.

[WWLZ08] Deqiang Wang, Xian Wang, Shaoxi Li, and Shaofang Zhang: Diagonal-flip distance algorithms of three type triangulations. In *Proceedings of the First International Conference on Computer Science and Software Engineering (CSSE'08)*, volume 2, pages 980–983. IEEE Computer Society, 2008.

Erklärung

Hiermit versichere ich die vorliegende Abschlussarbeit selbstständig verfasst zu haben, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben, und die Arbeit bisher oder gleichzeitig keiner anderen Prüfungsbehörde unter Erlangung eines akademischen Grades vorgelegt zu haben.

Würzburg, den 8. März 2012

. . . .
Fabian Lipp