Engineering A* Search for the Flip Distance of Plane Triangulations

Philip Mayer
Institute of Computer Science, University of Bonn, Germany

Petra Mutzel
Institute of Computer Science, University of Bonn, Germany

Abstract

The flip distance for two triangulations of a point set is defined as the smallest number of edge flips needed to transform one triangulation into another, where an edge flip is the act of replacing an edge of a triangulation by a different edge such that the result remains a triangulation.

We adapt and engineer a sophisticated A^* search algorithm acting on the so-called flip graph. In particular, we prove that previously proposed lower bounds for the flip distance form consistent heuristics for A^* and show that they can be computed efficiently using dynamic algorithms. As an alternative approach, we present an integer linear program (ILP) for the flip distance problem.

We experimentally evaluate our approaches on a new real-world benchmark data set based on an application in geodesy, namely sea surface reconstruction. Our evaluation reveals that A^* search consistently outperforms our ILP formulation as well as a naive baseline, which is bidirectional breadth-first search. In particular, the runtime of our approach improves upon the baseline by more than two orders of magnitude. Furthermore, our A^* search successfully solves most of the considered sea surface instances with up to 41 points. This is a substantial improvement compared to the baseline, which struggles with subsets of the real-world data of size 25.

Lastly, to allow the consideration of global sea level data, we developed a decomposition-based heuristic for the flip distance. In our experiments it yields optimal flip distance values for most of the considered sea level data and it can be applied to large data sets due to its fast runtime.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Computational Geometry, Triangulations, Flip Distance, A-star Search, Integer Linear Programming

Digital Object Identifier 10.4230/LIPIcs.SEA.2024.23

Supplementary Material *Software (Source Code)*: `https://github.com/PhilipMayer94/AStarFlipDistance`, archived at `swh:1:dir:d36b8698301a94856dab20f55ae7dc794ce747c1`

Funding This work was partially funded by the German Research Foundation (DFG) EXC-2047/1 – 390685813 and FOR-5361 – 459420781.

Acknowledgements We thank the anonymous reviewers for their insightful comments and suggestions.

1 Introduction

A *triangulation* D of a point set $S \subseteq \mathbb{R}^2$ is a maximal set of non-intersecting straight line edges with endpoints in S. Triangulations play an important role in theoretical as well as applied computational geometry. Applications can, for example, be found in computer graphics [15], finite element mesh generation [33], or reconstruction problems in geodesy [28]. Usually, the set $\mathfrak{T}(S)$ of all triangulations of a point set S is exponential in size.

One way to navigate the set $\mathfrak{T}(S)$ is by *edge* or *diagonal flips*. Let e be a *diagonal* of D, i.e., an edge that is not part of the convex hull, and let Q be the quadrilateral given by the two adjacent triangles of e. If Q is convex, then the edge flip of e in D is given by substituting the diagonal e by the other diagonal f in Q; see Figure 1. This procedure of

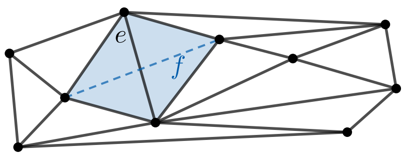


Figure 1 A triangulation with the flip of e indicated by the blue quadrilateral and flipped edge f.

flipping diagonals is often used for iterative local optimization of triangulations with respect to some criteria. The most famous application of this idea is the construction of the *Delaunay triangulation* [8], which uses a min–max angle criterion [20].

A more general question regarding the flip distance is the following: For two triangulations D and D', what is the minimal number of edge flips needed to transform D into D'? This number is called the *flip distance* $\mathrm{d}_F(D, D')$ of D and D'. The set $\mathfrak{T}(S)$ of all triangulations and the flip distance $\mathrm{d}_F(\circ, \circ)$ form a metric space, since the flip distance is symmetric, non-negative, and satisfies the triangle inequality.

Another interesting property of the flip operation is that it induces a graph structure on $\mathfrak{T}(S)$, which is called the *flip graph* $\mathrm{FG}(S)$ of S. The set of vertices of the graph corresponds to $\mathfrak{T}(S)$ and two triangulations $D, D' \in \mathfrak{T}(S)$ are connected by an (unweighted) edge if one can be transformed into the other by exactly one flip. Since every triangulation can be transformed to the Delaunay triangulation in a finite number of flips [20], the flip graph of a point set is connected. Additionally, the flip distance $\mathrm{d}_F(D, D')$ corresponds to the shortest path between D and D' in $\mathrm{FG}(S)$. Thus, standard methods for the calculation of shortest paths can be used to compute the flip distance. It should be noted that the flip graph can have an exponential number of vertices and edges, which implies an exponential runtime for the shortest path algorithms. In fact, Pilz [30] showed that deciding if the flip distance of a pair of triangulations is at most k is NP-complete.

The motivation for our work is a recent paper by Nitzke, Niedermann, Fenoglio-Marc, Kusche and Haunert [28] who used optimal minimum error triangulations to reconstruct the sea surface using tide gauge stations and satellite data. In their work, they compute an optimized triangulation for every month in a period of 22 years and use these optimized triangulations to reconstruct the sea surface at other points in time. Due to seasonal changes in sea level, it is usually more accurate to reconstruct the sea surface for a given date (month and year) from an optimized triangulation for the same month (but a different year). However, due to local shifts and fluctuations, this expectation does not always materialize. Therefore, to gain more insights which triangulations should be used for the reconstruction at a specific point in time, it is of interest to cluster the set of optimized triangulations with respect to some distance measure. One candidate for a distance measure is the flip distance since it is a natural way to define the distance between two triangulations. Additionally, $\mathrm{d}_F(\circ, \circ)$ is a metric on $\mathfrak{T}(S)$, which allows the use of most generic clustering algorithms.

The literature on theoretical results regarding the flip distance of point sets is extensive, but to the best of our knowledge, the practical computation of the flip distance for real-world applications has yet to be discussed in any detail. With our work, we want to initiate this discussion.

Our contribution

- We adapt and engineer an A^* search algorithm for the exact computation of the flip distance. For this, we prove that the lower bound given by Eppstein [9] constitutes a consistent heuristic in the context of the A^* paradigm. For the calculation of the heuristic values during a node extension, we use dynamic algorithms to improve the naive runtime by a factor of $\mathcal{O}(n)$.
- As an alternative approach, we suggest the first integer linear program for the flip distance problem.
- We present a new decomposition-based upper bound for the flip distance problem that can be effectively applied to the real-world data.
- We propose and discuss a new benchmark data set for the flip distance problem. The data set consists of real-world data based on the sea surface reconstruction problem as well as randomly generated data for systematic testing.
- We use our benchmark data to perform an experimental evaluation of our algorithms against bidirectional breadth-first search as a baseline. Our analysis focuses on the runtime and highlights the impact of employing good heuristics on the algorithms' performance.
- Finally, we discuss the performance of our decomposition heuristic on the real-world data.

The remaining paper is structured as follows: We start in Section 2 by providing an overview of related work on the flip distance problem. Then, in Section 3, we present our A^* algorithm. Next, in Section 4, we summarize the known lower bounds and show that they are consistent heuristics for A^*. Additionally, we show how to compute the heuristics efficiently during the A^* search. We present our integer linear program (ILP) for the flip distance problem in Section 5 and our decomposition heuristic in Section 6. In Section 7 we discuss the benchmark data set and conduct our experimental analysis of the algorithms. Lastly, we give our conclusion and propose further research directions in Section 8.

2 Related Works

For a more detailed survey of known results, we refer to the work by Bose and Hurtado [3].

Convex polygons/points in convex position. The setting with n points in convex position has been studied extensively. The main reason for this is that there is a bijection between triangulations of the n-gon and binary trees with $n-2$ inner nodes. Additionally, diagonal flips in triangulations correspond to rotations in these trees. This is, for example, proven by Sleator, Tarjan and Thurston [32]. In the same work, they show that $2n-10$ for $n > 12$ is a tight upper bound on the flip distance in the convex case. Baril [2] and Pallo [29] proposed upper and lower bounds on the flip distance and experimentally evaluated them.

There are multiple fixed-parameter tractable (FPT) algorithms for the convex flip distance where the parameter k is the flip distance [6, 25, 21]. The best known FPT algorithm with runtime $\mathcal{O}(3.82^k)$ was developed by Li and Xia [21].

Interestingly, the complexity of the flip distance problem remains unknown for convex polygons, despite the extensive amount of research invested into the problem.

In his bachelor thesis [22], Lipp used standard shortest path methods, i.e., BFS and A^*, to compute the convex flip distance. To the best of our knowledge, this is the only work that implements and evaluates a method to compute the flip distance to optimality.

Points in arbitrary position. The first published result for the flip distance problem in the general case was the work by Lawson [19], which showed that any two triangulations of a point set can be transformed into each other by a sequence of flips. Additionally, he showed that $\mathcal{O}(n^2)$ flips are sufficient [19]. Hurtardo, Noy and Urrutia [16] showed that this upper bound is tight by providing example triangulations for which $\Omega(n^2)$ flips are needed to transform one triangulation into the other. In a secondary work [20] Lawson also showed that any triangulation can be transformed into the well-known Delaunay triangulation [8].

The work [13] by Hanke, Ottmann and Schuierer shows that for two triangulations the number of (proper) intersections of edges provides an upper bound on the flip distance. Eppstein [9] proved that for point sets that do not contain empty pentagons the flip distance can be computed in $\mathcal{O}(n^2)$ time. As a byproduct of his discussion, he derives a lower bound on the flip distance for point sets that contain empty pentagons, which we discuss in Section 4.

A first FPT algorithm with runtime $\mathcal{O}^*(k \cdot c^k)$, where $c \leq 2 \cdot 14^{11}$, was given by Kanj, Sedgwick, and Xia [17]. Later, Feng, Li, Meng and Wang [10] improved the algorithm, which led to a runtime of $\mathcal{O}^*(k \cdot 32^k)$.

Lubiw and Pathak [24] as well as Pilz [30] independently showed that the flip distance problem is NP-complete. Additionally, Pilz showed the APX-hardness of the problem.

3 Adapting A^* Search for the Flip Graph

In this section, we present an A^* search algorithm adapted for the flip graph $\mathrm{FG}(S)$ of a point set S of size n with c points on its convex hull. To avoid confusion with other graphs, which are defined in Section 4, we call the triangulations *nodes* of $\mathrm{FG}(S)$ instead of vertices.

We start with some basic definitions (contextualized in the triangulation setting). A *heuristic* with respect to a target triangulation D_t is a function $h^{D_t} : \mathfrak{T}(S) \to \mathbb{R}$. We say h^{D_t} is *admissible* if for all triangulations D the heuristic $h^{D_t}(D)$ is a lower bound for $\mathrm{d}_F(D, D_t)$. The heuristic is *consistent* if $h^{D_t}(D_t) = 0$ and for all pairs of adjacent triangulations D and D' in the flip graph $\mathrm{FG}(S)$ we have $h^{D_t}(D) \leq 1 + h^{D_t}(D')$ and vice versa[1]. From now on we assume that the target triangulation is fixed and we write $h(D)$ instead of $h^{D_t}(D)$.

The A^* search algorithm first proposed by Hart, Nilsson and Raphael [14] keeps a priority queue $\mathcal{Q}$ of *open* nodes that need to be processed. It starts with $\mathcal{Q} = \{D_s\}$ and in every iteration the node D from $\mathcal{Q}$ with minimal value $f(D) = \mathrm{d}_F(D_s, D) + h(D)$ is *extended*. During the extension of a node D, it is removed from $\mathcal{Q}$, marked as *closed* and all of its neighbors D' are opened by adding them to $\mathcal{Q}$ with their respective $f(D')$ values. The search is complete when D_t is closed. If h is admissible the resulting path is optimal. Additionally, neighbors that are marked as closed do not need to be re-opened during a node extension if h is consistent. Note that the heuristics we present in Section 4 are consistent.

Next, we discuss the data structures that are needed to traverse $\mathrm{FG}(S)$. Different from usual A^* algorithms, the graph $\mathrm{FG}(S)$ is not known in its entirety and would be far too large to be stored in memory, even for small instances. Thus, we build the graph $\mathrm{FG}(S)$ on the fly and only add a node to the graph when it is first opened. Since every node is associated with a triangulation D, we need to represent D in some way. For the sake of simplicity, we assign an integer value to every triangle that can occur in a triangulation. Note that the number of triangles in a triangulation of n points with c points on the convex hull is given by $z = 2n - c - 2$; see for example [7]. Hence, a triangulation can be represented by a sorted

[1] Our definition differs from the usual definition, but they are equivalent since our graph is unweighted.

Algorithm 1 AStarSearch(D_s, D_t).

```
1: $\mathcal{Q} \leftarrow$ PriorityQueue, OPEN $\leftarrow$ HashMap, CLOSED $\leftarrow$ HashMap
2: $D_s.d = 0$
3: $D_s.f = \text{computeHeuristic}(D_s, D_t)$
4: $\mathcal{Q}.\text{insert}(D_s, D_s.f)$, OPEN.add($D_s$)
5: while $\mathcal{Q} \neq \emptyset$ do
6:     $D = \mathcal{Q}.\text{extractMin}()$
7:     OPEN.remove($D$), CLOSED.add($D$)
8:     if $D = D_t$ then return $D.d$
9:     computeHeuristic($D, D_t$)        ▷ only executed for the computation of $h_E$; see Section 4
10:    for each valid flip $e$ of $D$ do
11:        $D' = \text{doFlip}(D, e)$
12:        if $D' \in$ CLOSED then continue
13:        $h = \text{computeHeuristicNeighbor}(D', D, D_t)$
14:        $D'.d = D.d + 1$
15:        $D'.f = D'.d + h$
16:        if $D' \in$ OPEN then
17:            $\mathcal{Q}.\text{decreaseKey}(D', D'.f)$    ▷ only executed if $D'.f$ is smaller than the known value
18:        else
19:            $\mathcal{Q}.\text{insert}(D', D'.f)$, OPEN.add($D'$)
```

array of z integer values. This representation allows for easy equality tests and we can use it to derive all diagonals and their corresponding quadrilaterals/flips in linear time. We keep track of nodes that have been opened during the search by storing them in a hashmap using the triangulation-representation as keys. The value of a triangulation D in the hashmap is given by the tuple $(h(D), \mathrm{d}_F(D_s, D), i)$ where i is the position in the priority queue.

Algorithm 1 summarizes our A^* algorithm. As stated before, in addition to the usual priority queue $\mathcal{Q}$ we need to keep track of all opened and closed triangulations, which is done in OPEN and CLOSED. Closed nodes can be skipped in line 12, because our used heuristics are consistent. Finally, we note that the calculation of the heuristic value for a triangulation D' in line 13 depends on its parent node D. This is the case because we want to dynamically update the heuristics using the information of the parent node. The dynamic updates for the different heuristics are discussed in Section 4.

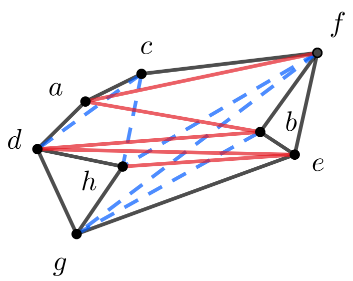
4 Lower Bounds on the Flip Distance

In this section, we present two heuristics given by known lower bounds for the flip distance and show how to compute them efficiently during the A^* search.

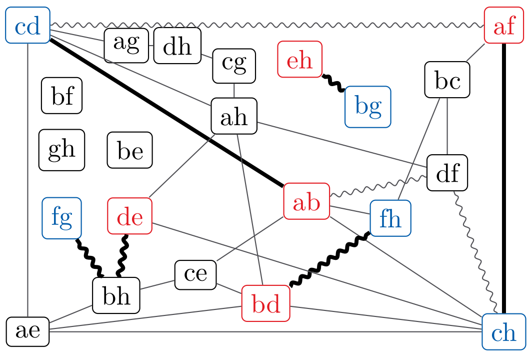
A simple lower bound. Let D and D' be two triangulations in $\mathfrak{T}(S)$. Then we can define $l_S(D, D')$ to be the number of diagonals that are in D and not in D'. Every diagonal in D that does not coincide with a diagonal in D' has to be flipped at some point. Thus, $l_S(D, D')$ is a lower bound on the flip distance. In Figure 2 (a) the lower bound $l_S(D, D')$ is five. Moreover, adjacent triangulations D and $\hat{D}$ in FG(S) only differ by one diagonal, which implies $|l_S(D, D') - l_S(\hat{D}, D')| \leq 1$. Consequently, we get the following simple observation:

▶ **Observation 1.** *The heuristic $h_S(D) = l_S(D, D_t)$ is admissible and consistent.*

A refined lower bound by Eppstein. To describe Eppstein's [9] heuristic, we first need an additional graph, which we call the *quadrilateral graph* QG. The vertex set of QG is given by all $\mathcal{O}(n^2)$ diagonals that may be used in a triangulation of S. Two diagonals $e = \overline{uv}$ and



(a) two superimposed triangulations.



(b) the QG graph.

Figure 2 In (a) two triangulations D (red) and D' (dashed blue) of a point set are given. Diagonals that coincide are given in black. In (b) the QG graph is depicted, the relevant diagonals are indicated in red and blue. A minimal flip sequence is given by $ab \to df$, $af \to cd$, $de \to bh$, $bd \to fh$, $df \to ch$, $eh \to bg$, $bh \to fg$. The paths corresponding to the flip sequence are given as wavy edges and the shortest paths corresponding to a minimum weight perfect matching in $B(D, D')$ are the fat black edges. We have $l_S(D, D') = 5$, $l_E(D, D') = 6$ and $\mathrm{d}_F(D, D') = 7$.

$e' = \overline{u'v'}$ are connected by an edge in QG if e and e' properly intersect and the quadrilateral q given by $\{u, u', v, v'\}$ does not contain any points of S in its interior. Note that q is a convex quadrilateral if e and e' properly intersect. An example of a QG graph is depicted in Figure 2.

▶ **Observation 2** (Eppstein [9]). *Let D and D' be two triangulations of S. Any sequence of flips that transforms D to D' corresponds to a set of walks in QG that connect diagonals of D with diagonals of D'.*

These walks are indicated for a minimal flip sequence in Figure 2. Now we can describe the lower bound $l_E(D, D')$ given by Eppstein: Let $B(D, D')$ be the complete bipartite graph with one part given by the diagonals of D and the other part by the diagonals of D'. We define the cost $c(e, e')$ of an edge between a diagonal of $e \in D$ and a diagonal $e' \in D'$ to be the length of the shortest path between e and e' in QG. Then, $l_E(D, D')$ is given by the value of a minimum weight perfect matching (MWPM) on $B(D, D')$ with respect to the costs c.

▶ **Proposition 3.** *The heuristic $h_E(D) = l_E(D, D_t)$ is admissible and consistent.*

Proof. The proof is given in Appendix A. ◀

The only edges in the bipartite graph $B(D, D_t)$ that have value zero are edges between common diagonals. All other edges in the matching have at least value one, which implies:

▶ **Observation 4.** *For every possible triangulation D of S we have $h_E(D) \geq h_S(D)$.*

Before we discuss how to compute these heuristics efficiently during the A^* search, we investigate the runtime of a node extension, i.e., one execution of the while loop in Algorithm 1. Let $\mathrm{t}[h(D')]$ be the runtime of one heuristic computation and $\mathrm{t}[b(D')]$ be the runtime of the bookkeeping in the priority queue and hashmaps for one neighbor D'. It is well known [7] that every triangulation of a point set with c points on the convex hull consists of $3n - c - 3$ edges. Thus, for one triangulation $\mathcal{O}(n)$ diagonals are candidates for possible flips. After deriving all diagonals and their (valid) flips from the representation of D in linear time, we can perform each flip in constant time. Note that for every neighbor that is inserted into the hashmap a new representation array must be created, which implies that the processing time for a neighbor is at least $\mathcal{O}(n)$. Thus, one node extension needs $\mathcal{O}(n \cdot (n + \mathrm{t}[h(D')] + \mathrm{t}[b(D')]))$ time.

Extending nodes with the simple heuristic. For the initial triangulation D_s, the heuristic value $h_S(D_s)$ has to be calculated from scratch. Note that we can represent the edge sets of D_s and D_t by integer arrays A_s and A_t. After sorting the arrays, the number k of elements that are present in both arrays can be computed using a simultaneous pass through both arrays. Thus, the heuristic, given by $h_S(D_s) = |A_s| - k$, can be computed in $\mathcal{O}(n \log n)$ time.

If we extend a node D, we can query the heuristic value $h_S(D)$ of D. For all neighbors D' we can now compute the heuristic value $h_S(D')$ in constant time. Let e be the diagonal that is flipped to e' in D'. Then we only need to check if $e \in D_t$ or $e' \in D_t$ and we get

$$h_S(D') = \begin{cases} h_S(D) + 1 & \text{if } e \in D_t \text{ and } e' \notin D_t \\ h_S(D) - 1 & \text{if } e \notin D_t \text{ and } e' \in D_t \\ h_S(D) & \text{otherwise.} \end{cases}$$

It follows that the processing time for a single neighbor is dominated by the construction of the representation and we get the runtime $\mathcal{O}(n^2 + n \cdot \mathrm{t}[b(D')])$ for a node extension with the simple heuristic.

Extending nodes with Eppstein's heuristic. To allow for a fast construction of the cost matrix $C_{DD'}$ that corresponds to the bipartite graph $\mathcal{B}(D, D')$, we compute the distance matrix $\mathcal{D}_{\mathrm{QG}}$ for the quadrilateral graph QG in a pre-processing step. To this end we need to solve the all-pairs shortest-path problem on QG. Since the graph QG is unweighted with $\mathcal{O}(n^2)$ vertices and $\mathcal{O}(n^4)$ edges, we can solve the problem in time $\mathcal{O}(n^6)$ using breadth first search. This runtime seems excessive, but for the instances we consider in our experiments, this is fast enough.

It remains to be shown how to compute the minimum weight perfect matching. For this, we can use the Hungarian algorithm which was introduced by Kuhn [18] and later refined by Munkres [27]. Their algorithm works in $\mathcal{O}(n)$ phases that each take $\mathcal{O}(n^2)$ time, which leads to a runtime of $\mathcal{O}(n^3)$. Using this algorithm and the pre-processed matrix $\mathcal{D}_{\mathrm{QG}}$, the heuristic calculation for one neighbor can be done in $\mathcal{O}(n^3)$. Since we have $\mathcal{O}(n)$ neighbors per node extension, the runtime for a node extension with the (static) Hungarian algorithm is $\mathcal{O}(n^4 + n \cdot \mathrm{t}[b(D')])$.

If we consider a neighbor D' during a node extension, only one diagonal e_k changes with respect to the parent node D. This implies that for the other diagonals, all costs remain the same. Thus, we can derive the cost matrix $C_{D'D_t}$ for a neighbor by taking the cost matrix C_{DD_t} of the parent node and changing exactly the row k of costs corresponding to the flipped edge. It seems wasteful to recompute the matching from scratch. In fact, there is a dynamic version of the Hungarian algorithm established by Mills-Tettey, Stent, and Dias [26]. This dynamic algorithm only needs to perform one additional phase of the Hungarian algorithm per changed row. Hence, it has runtime $\mathcal{O}(n^2)$ if only one row is changed.

If we want to use this dynamic algorithm for the neighbors of a node during the node's extension, we need to know the perfect matching of the parent node and the corresponding labeling of the Hungarian algorithm. Storing this information for every opened triangulation would drastically increase the memory consumption. Instead, we perform a "redundant" (static) computation of the matching and labeling for the parent node, which is indicated in line 9 in Algorithm 1. We can then compute the heuristics dynamically for the neighbors using the labels and matching of the parent node. Thus, we get the runtime

$$\mathcal{O}(n^3 + n \cdot n^2 + n \cdot \mathrm{t}[b(D')]) = \mathcal{O}(n^3 + n \cdot \mathrm{t}[b(D')])$$

for the node extension with $\mathcal{O}(n)$ neighbors. Note that at least asymptotically the additional computation of the heuristic does not matter.

5 A Layered Integer Linear Program

For many combinatorial optimization problems integer programming based approaches perform very well. To the best of our knowledge, no ILP for the flip distance has been proposed so far, so we introduce one in this section. Let D and D' be two triangulations of a point set S given by n points. The idea of the ILP is to build a layered formulation where every layer corresponds to a valid triangulation and two consecutive layers differ by exactly one edge. If we fix the first layer to D and the last layer to D' and the number of layers is sufficiently large, then the number of changes between layers is the flip distance. To ensure that the number of layers L is big enough, we choose L to be the upper bound given by Hanke et al. [13]. As mentioned in Section 4, the number of edges of the triangulation D (and D') is $M = 3n - c - 3$, where c is the size of the convex hull of S. Let $\mathcal{E}$ be the set of all edges that can be utilized by a triangulation of the point set S. Additionally, let $E \subseteq \mathcal{E}$ and $E' \subseteq \mathcal{E}$ be the sets of edges for D and D'. For every $e \in \mathcal{E}$ and every $0 \leq i \leq L$ we have variables x_e^i and d_e^i. The x_e^i variables are used to encode valid triangulations on each layer and the d_e^i variables encode the deletion of an edge between layers i and $i+1$. Using these variables, the ILP is given by:

$$\text{minimize} \sum_{k \leq L} \sum_{e \in \mathcal{E}} d_e^k$$

$$x_e^i + x_f^i \leq 1 \quad \forall e, f \in \mathcal{E},\ e \text{ crosses } f,\ i \leq L \tag{T1}$$

$$\sum_{e \in \mathcal{E}} x_e^i = M \quad \forall i \leq L \tag{T2}$$

$$x_e^i - x_e^{i+1} - d_e^i \leq 0 \quad \forall e \in \mathcal{E}, \forall i \leq L-1 \tag{F1}$$

$$\sum_{e \in \mathcal{E}} d_e^i \leq 1 \quad \forall i \leq L-1 \tag{F2}$$

$$\sum_{e \in \mathcal{E}} d_e^{i+1} \leq \sum_{e \in \mathcal{E}} d_e^i \quad \forall i \leq L-1 \tag{F3}$$

$$x_e^0 = 1 \quad \forall e \in E, \quad x_e^0 = 0 \quad \forall e \notin E \tag{I1}$$

$$x_e^L = 1 \quad \forall e \in E', \quad x_e^L = 0 \quad \forall e \notin E' \tag{I2}$$

$$x_e^i \in \{0,1\}, d_e^i \in \{0,1\} \quad \forall e \in \mathcal{E}, \forall i \leq L \tag{B}$$

Since our triangulations have M edges and a triangulation is a set of non-intersecting edges, the constraints (T1) and (T2) guarantee that on every layer the set of positive edge variables corresponds to a valid triangulation. The constraints (F1) encode edge deletions from one layer to the next, i.e., if we have a variables $x_e^i = 1$ and $x_e^{i+1} = 0$, we must also have $d_e^i = 1$. Constraints (F2) ensure that only one edge is deleted from one layer to the next. Note that constraints (F1) do not forbid edge insertions, but (T2) implies that every insertion must correspond to a deletion. Hence, we can only have one edge insertion between layers.

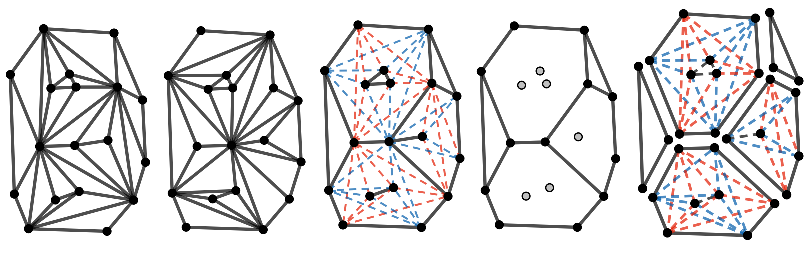


Figure 3 In (a) and (b) the triangulations D_s and D_t are shown. In (c) the common edges are given in black and the dotted red and blue edges indicate the non-common edges. In (d) the 2-connected component C that contains the convex hull is indicated in black and the remaining points are given in gray. The final decomposition is given in (e) where all edges that can be flipped are indicated by dotted lines. Note that some common edges can now be flipped.

Additionally, this added edge must be the other diagonal of the (convex) quadrilateral given by the incident triangles of the deleted edge, since otherwise (T1) would be violated. Hence, every $d_e^i = 1$ variable encodes a valid flip. Constraints (F3) are used to break symmetries. They force the flips to be done as early as possible. This is only relevant if the number of layers is not minimal and they are not needed for the correctness of the formulation. The constraints (I1) and (I2) set the first and last layer to D and D', respectively. Finally, the objective function counts the overall number of edge deletions, which is exactly the number of flips.

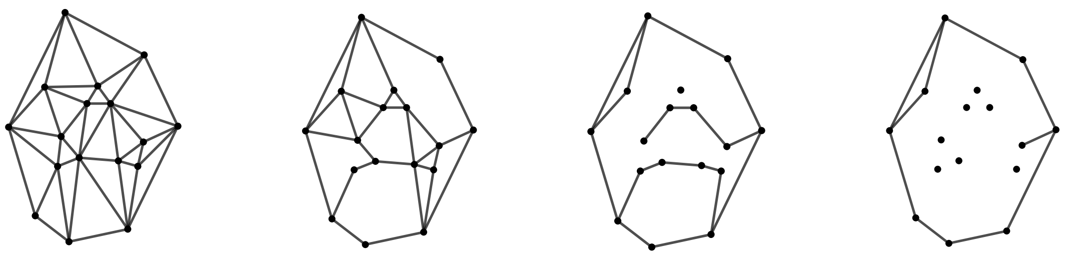
We also considered formulations that use triangles (similar to De Loera et al. [23]) and quadrilaterals as variables, but we do not present them here, since they are very similar to the presented ILP and performed far worse.

6 A Decomposition Approach

In this section, we discuss a decomposition-based approach to heuristically solve the flip distance problem. The basic idea of the approach is that, if two triangulations D_s and D_t already have a common edge, then we assume that it is not flipped in an optimal flip sequence. This assumption is true in the convex case [32] and has been used for multiple FPT-algorithms [4, 6, 25]. For arbitrary point sets the assumption is not always correct (see Pilz [30]), but we get an upper bound for $\mathrm{d}_F(D_s, D_t)$ by only flipping non-common edges. We start this section by discussing our decomposition approach and then we introduce a class of instances (that is relevant in our application) for which the approach is advantageous.

If we fix edges that coincide in both triangulations, there can be paths of fixed edges that split the instance into sub-problems that can be solved individually. Note that the boundaries of the sub-problems are not necessarily convex, but our algorithms also work with non-convex boundaries. Using this simple insight we can present our decomposition scheme (see Figure 3 for an illustration of the different steps):

1. Compute the set F of edges that are used in D_s as well as D_t.
2. Compute the 2-connected components of F and find the component C that contains the convex hull.
3. Compute all faces F_i of the geometric graph given by C.
4. Distribute the remaining points and the sub-triangulations to the faces F_i.
5. Solve the sub-problems individually and combine their solutions.



(a) $k = 0$. **(b)** $k = 2$. **(c)** $k = 5$. **(d)** $k = 17$.

Figure 4 The k-OD fixed edges of a point set for different orders.

The algorithm is designed to only fix a small, necessary set of edges that yield a decomposition. To this end, we only consider the (non-trivial) 2-connected components, since only those decompose our problem into smaller sub-problems. Additionally, we decided to only use the 2-connected component that contains the convex hull. This guarantees that every point from $S \setminus C$ is contained in exactly one sub-problem.

The other known upper bound by Hanke et al. [13], also never flips common edges. Thus, our heuristic is always at least as good as the one by Hanke, as, unlike Hanke's approach, it computes optimal sequences on the sub-problems given by non-coinciding edges. For arbitrary inputs, we cannot expect the decomposition approach to be fast, since we cannot assume that a lot of edges coincide and in particular, they may not form paths that split our instance into smaller sub-problems. We now propose a class of instances for which the decomposition approach can be used effectively.

Higher-order Delaunay triangulations The instances we are interested in are those where D_s and D_t both are higher-order Delaunay triangulations with order k. Higher-order Delaunay triangulations [11, 31] are similar to the Delaunay triangulation while leaving room for optimization. More specifically, for every triangle T of a k-order Delaunay (k-OD) triangulation the circumcircle of T contains at most k points of S. They are often used for interpolation tasks and in particular, they are used in the sea level reconstruction data. It is a well-known fact [11] that only $\mathcal{O}(nk)$ of the $\mathcal{O}(n^2)$ possible edges are used in **all** k-OD triangulations of a point set. For small k this results in a lot of fixed edges that are part of every k-OD triangulation. Figure 4 depicts the fixed edges of a point set for different orders k. For orders $k \leq 7$ the experiments by Arutyunova et al. [1] show that, even for larger data sets, the k-OD fixed edges already yield a good decomposition. Thus, if D_s and D_t both have order $k \leq 7$, their common edges include the k-OD fixed edges and we can expect a useful decomposition of our instance.

7 Experiments

We start this section by presenting the used benchmark instances. Then we give some details on our implementation and evaluate our algorithms.

7.1 Data

We use two types of instances: randomly generated instances and instances given by the sea surface reconstruction task introduced by Nitzke et al. [28].

Table 1 Overview of the relevant data and the number of non-trivial instances with $\Delta b > 0$.

	R15	R20	R25	R30	S41-2	S25-5	S30-5	S35-5	S41-5	S25-25	S30-30	S35-35	S41-41
instances	1900	1900	950	950	630	630	630	630	630	276	276	276	276
nontrivial	682	819	575	751	130	42	244	152	238	187	214	267	271

Random instances. We consider point sets of size $n \in \{15, 20, 25, 30\}$. We generate ten uniformly randomly distributed sets of points for $n \in \{15, 20\}$ and five for $n \in \{25, 30\}$. For each point set, we generate 20 random triangulations (see Appendix B). Then, the flip distance problem instances corresponding to the point set are given by the $\binom{20}{2} = 190$ pairs of triangulations. Thus, for a fixed n we either get 1900 ($n \leq 20$) or 950 instances ($n \geq 25$). We denote these instance sets by `RX` in our experiments where `X` is the size of the instance.

Sea level instances. The sea level triangulations by Nitzke et al. [28] stem from the 41 tide gauge stations in the European North Sea. The triangulations are optimized with respect to monthly generated satellite altimetry. In our experiments, we focus on the data collected from 2009 to 2011, which yields 36 triangulations in our timeframe. Nitzke et al. only consider higher-order Delaunay triangulations. Consequently, they have different sets of optimized triangulations for different orders. In the application [28], distinct orders are not mixed. Consequently, we also treat different orders as separate data sets. We focus on triangulations of order 2 and 5, which yield the most consistent reconstructions according to [28] and [1], respectively. Additionally, we consider unconstrained sea level instances in a smaller time frame of 2009–2010 (an instance is unconstrained if its order equals its size). In addition to the complete North Sea set with 41 points, we consider subsets of the tide gauge set of size $n \in \{25, 30, 35\}$, which still cover the North Sea and therefore allow for optimization. All in all, for a size n, we have 630 instances of order 2 and 5, respectively, and 276 unconstrained instances. We denote these instance sets by `SX-Y` where `X` is the size and `Y` is the order.

Next we want to introduce a measure of hardness for an instance. For this, we define $\Delta b(D, D') = u_{\mathrm{H}}(D, D') - l_{\mathrm{E}}(D, D')$ to be the gap between the upper bound given by our implementation of the algorithm by Hanke et al. [13] and the best known lower bound by Eppstein. If we have $\Delta b(D, D') = 0$, the solution given by the upper bound is already optimal. We expect instances to get harder with increasing $\Delta b(D, D')$. It turns out that the sea level instances with small order are "easier" than unconstrained instances. In particular, all of the instances of sizes 25, 30 and 35 and order 2 can be trivially solved using the bounds. The numbers of non-trivial instances for the other sets are given in Table 1 and their distributions of $\Delta b(D, D')$ are given in Figure 5. Note that for order 5 all of the instances have $\Delta b(D, D') \leq 3$ and most of them have $\Delta b(D, D') = 1$. Additionally, Figure 5 suggests that the unconstrained sea level data sets contain the hardest instances.

7.2 Experimental Evaluation

Before we present our evaluation, we give some details on the used hardware and software. The systems used for the experiments are equipped with an AMD EPYC 7402P CPU, 256 GB of RAM and have Ubuntu 22.04 installed. Our code is written in C++17, compiled with GCC 9.4, and it is available online[2]. In our implementation, we use an array-based binary heap as priority queue. For the hashmap, we utilize the unordered map provided by the C++ standard library, while employing a simple combinatorial hash function.

[2] `https://github.com/PhilipMayer94/AStarFlipDistance`

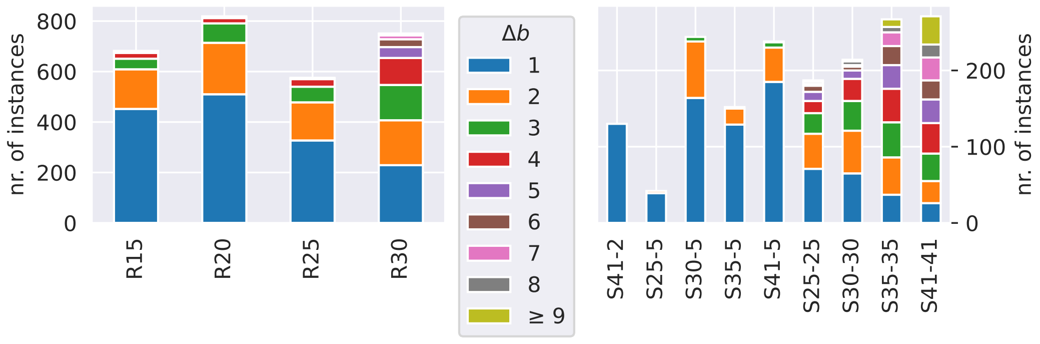


Figure 5 The $\Delta b(D, D')$ distribution of the instances for the different data sets.

Table 2 Overview of the relative number of timeouts (all values are percentages) of the four main algorithms and an additional combined approach A^*_C.

	R15	R20	R25	R30	S41-2	S25-5	S30-5	S35-5	S41-5	S25-25	S30-30	S35-35	S41-41
BFS	0.00	73.26	100.00	100.00	97.69	57.14	99.18	100.00	100.00	98.93	100.00	100.00	100.00
ILP	41.94	85.23	93.91	99.87	4.62	11.90	54.10	23.03	66.81	96.26	99.53	100.00	100.00
A^*_S	0.00	0.73	30.09	89.08	0.00	0.00	0.82	0.66	8.82	23.53	65.42	88.76	99.63
A^*_E	0.00	0.00	8.17	75.63	0.00	0.00	0.82	1.32	17.65	9.09	50.00	83.15	99.63
A^*_C	0.00	0.00	6.61	72.70	0.00	0.00	0.00	0.66	8.40	8.02	42.99	79.03	99.63

For the Hungarian algorithm, we ported the $\mathcal{O}(n^3)$ implementation by Stern[3] to C++ and added the dynamic features. As an ILP-solver Gurobi 9.5.1 [12] using a single thread is employed. We set the number of layers of the ILP to the flip distance d_F, if it is available from other experiments. Note that this information is usually not available to the ILP and it has to use an upper bound for d_F. Our evaluation is going to show that even this idealized version is not competitive with the A^* approaches. As a baseline, we implemented an iterative bi-directional breadth-first search (BFS). We denote the A^* search that uses the simple heuristic h_S by A^*_S and the one using Eppstein's heuristic h_E by A^*_E. With the clustering application in mind, we choose a timeout of five minutes in all experiments.

Comparing all approaches. We start by investigating all presented approaches. Table 2 depicts for each data set the timeout percentages of the presented algorithms (and an additional algorithm we discuss later). The BFS does not run into timeouts for the smallest data set R15 but struggles on all other sets. This can be attributed to the fact that BFS is uninformed and the runtime of BFS exponentially depends on d_F. The ILP performs less consistently than BFS and even has timeouts for R15, but it can at least solve some (up to 95.38%) instances of every data set. This could be attributed to the fact that the ILP works with more information, e.g, a dual bound (which coincides with h_S at the root).

Both of the A^* algorithms perform more consistently than the other two approaches. They both have less than 2% timeouts for instances with size $n \leq 20$ or order $k \leq 5$ (except for S41-5). The simpler algorithm A^*_S has fewer timeouts on the easier instances with $\Delta B < 3$ and A^*_E has fewer timeouts on the harder instances. Overall both approaches still have at least 25% timeouts for the large unconstrained instances and the large random instances.

[3] `https://github.com/KevinStern/software-and-algorithms`

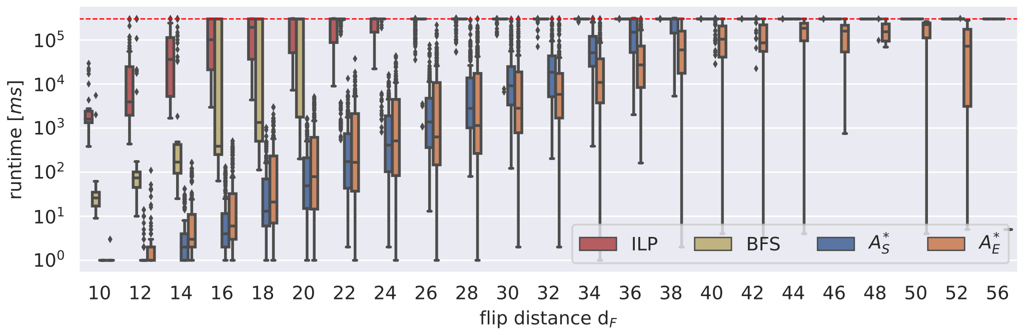


Figure 6 The runtimes (on a log scale) of the different approaches on all non-trivial and solved instances. Note that every odd d_F is grouped with its predecessor and that for $d_F > 27$ the boxes for BFS and ILP degenerate to a line, since they mostly time out.

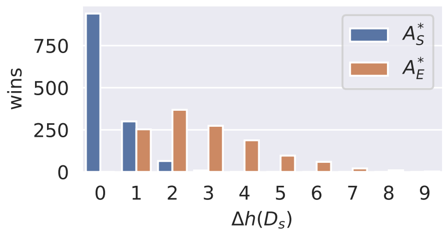


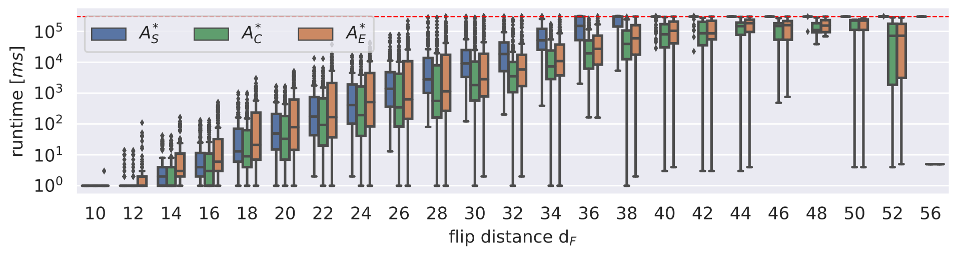
Figure 7 The runtime-wins of A_E^* and A_S^* with respect to $\Delta h(D_s)$.

Next, in Figure 6, we consider the runtimes on all solved instances (an instance is *solved* if at least one approach solved it in the allotted time) depending on the flip distance. Note that, if an algorithms times out for a solved instance, the timeout time is counted as its runtime. We can see that, on average, both A^* algorithms outperform the other approaches by at least two orders of magnitude. On average, BFS outperforms the ILP in runtime, but as mentioned earlier, for the larger flip distances some outliers can be solved faster (or at all) with the ILP. Concerning the A^* algorithms, we can see that A_S^* is on average faster for $d_F < 26$ and for the larger flip distances A_E^* outperforms A_S^* by an order of magnitude. This is a result of the trade-off between the smaller search space of the better heuristic h_E and the faster computation time for h_S.

We now have a brief look at the runtime-wins. Neither the ILP nor BFS are the fastest approach for a single instance. A_S^* has the fastest runtime 1 527 times and A_E^* only 1 218 times. Our previous discussion suggests that these wins depend on the hardness of the instances. We propose an additional measure $\Delta h(D_s) = h_E(D_s) - h_S(D_s)$, which quantifies how much h_E improves upon h_S. In Figure 7 the wins are grouped with respect to $\Delta h(D_s)$. As expected most A_S^* wins happen for $\Delta h(D_s) = 0$ and the rest for $\Delta h(D_s) = 1$ or $\Delta h(D_s) = 2$. When $\Delta h(D_s)$ is two or larger, A_S^* consistently expands over ten times as many nodes as A_E^*. This observation helps to explain why A_E exhibits faster runtimes on such instances. A more in-depth discussion of node extensions can be found in Appendix C.

A combined approach. Note that Δh is apriori knowledge. Thus, we can use it to develop an approach that combines both heuristics. A first naive possibility would be to compute $\Delta h(D_s)$ and then use A_S^* if $\Delta h(D_s) = 0$ and A_E^* otherwise. Since this approach only covers

(a)



(b)

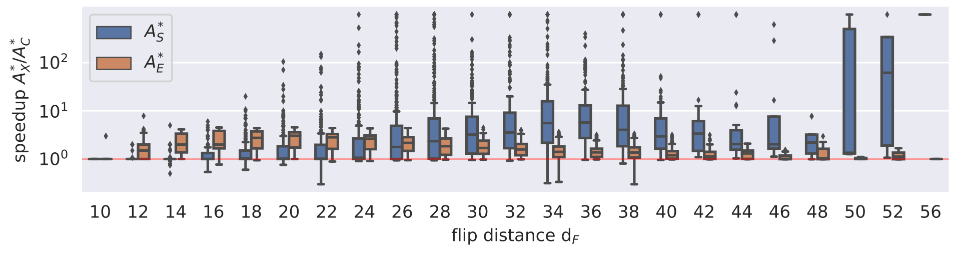


Figure 8 In (a) the runtimes of the A^* approaches on all considered non-trivial and solved random and real-world data are presented. In (b) the speedups of A_C^* with respect to A_S^* and A_E^* are given grouped by flip distance (uneven d_F values are grouped with their even predecessors).

the $\Delta h(D_s) = 0$ wins of A_S^*, we engineered a more sophisticated approach A_C^* in an attempt to use more Δh knowledge. A_C^* is based on the assumption that if we have $h_S(D) = h_E(D)$ for a node D, the heuristics will probably not diverge (too much) for children of D.

Thus, we make a dynamic decision at every node D, by considering $\Delta h(D)$, before extending it. More specifically, before line 9 in Algorithm 1 we compute $\Delta h'(D) = h_{\text{known}}(D) - h_S(D)$, where $h_{\text{known}}(D)$ is the algorithm's best known heuristic value for D. If $\Delta h'(D) = 0$, we do not execute line 9 and in line 13 `computeHeuristicNeighbor`(D', D, D_t) is performed with respect to h_S instead of h_E. If $\Delta h'(D) > 0$, we perform the usual update with respect to h_E. This combined heuristic is valid since the maximum of two admissible and consistent heuristics still satisfies both properties. Note that performing a node extension with respect to h_S implies that $h_{\text{known}}(D') = h_S(D')$ for all children D' of D and, if a child is later extended, it will always be extended using h_S.

As before, we used all of the data to test A_C^*. The timeouts are given in Table 2. For every data set except S41-41, the timeout percentage is smaller than that of A_S^* and A_E^*. In particular, A_C^* can solve all instances for the set S30-5 in the allotted time. The runtimes of all A^* approaches grouped by flip distance are depicted in Figure 8 (a). A_C^* on average computes d_F faster than the other approaches for all distances. It should be noted that the runtime of A_C^* is similar to A_S^* for small flip distances and similar to A_E^* for larger distances. This aligns with our assumption that we primarily utilize h_S for easier instances and mostly rely on h_E for more challenging cases. All of the previous observations can also be verified if we consider the runtime speedups of A_C^* with respect to A_S^* and A_E^*, which are given in Figure 8 (b). Here the speedup with respect to A_E^* decreases with d_F and the speedup with respect to A_S^* increases. We can also deduce from the speedup plot that dynamically choosing the heuristic in A_C^* improves upon A_E^* for hard instances since otherwise, we would expect the speedup for A_E^* to be very close to one for d_F>30.

Next, we consider the runtime-wins grouped by $\Delta h(D_s)$, which are given in Figure 9. Note that if two algorithms have the same (fastest) runtime on an instance, the win is counted for both of them. Algorithm A_C^* has by far the most wins, except for $\Delta h(D_s) = 0$, but on

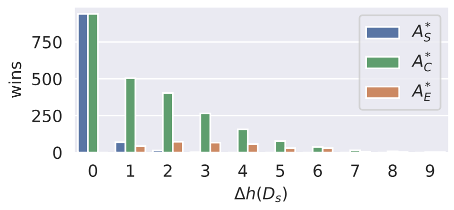


Figure 9 The runtime-wins with respect to $\Delta h(D_s)$ of the different A^* approaches.

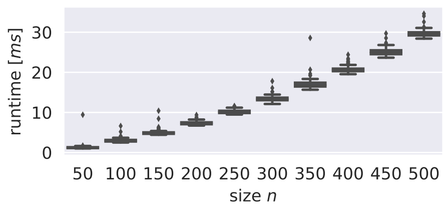


Figure 10 The runtime of the decomposition heuristic for instances of size n.

those instances A_C^* mimics A_S^*. Thus, all of those wins are actually "shared" wins where both approaches have the same runtime. In fact, A_S^* only solves 1.1% and A_E^* only 6.5% of the instances faster than A_C^*. Hence, for 92.4% of instances A_C^* is the (possibly shared) fastest algorithm and for 43.8% instances it even improves upon the basic A^* algorithms. On average our algorithm A_C^* is 20% faster than the virtual best of A_S^* and A_E^* (the virtual best uses a posteriori knowledge to choose the faster of the two approaches for every instance).

Finally, we discuss the peak memory consumption of the algorithm. On the most memory-intensive instance, which consists of 41 points and times out after five minutes with 201 109 543 opened triangulations, the combined algorithm uses 50 GB of memory. Most of this is due to the hashmap that manages the open triangulations. Its memory usage could be improved by changing the representation of the triangulations to a more sophisticated one, such as an adaptation of the planar encodings given by Chuang et al. [5]. Note that maintaining these encodings may come with an increase in runtime. Additionally, the unordered map of the C++ Standard Library has a large memory overhead. Thus, using a more memory-efficient map could also improve memory usage.

A brief evaluation of the decomposition heuristic. We ran our decomposition-based heuristic on all of the application-relevant instances, i.e., the sea level instances with order 2 and 5. On all instances that could be solved by A_C^* in the allotted time, our heuristic value coincided with the optimal flip distance value computed by A_C^*. The average runtime of the heuristic on the considered instances was 0.66 milliseconds. This suggests that, at least for the considered application-oriented instances, finding an optimal solution is usually not difficult, but verifying its optimality takes significant effort. Finally, to get an idea of how the runtime increases for larger instances, we generated 100 random instances of order 5 for $n = 50, 100, ..., 500$ and computed our heuristic on them. The runtimes are given in Figure 10. The increase in runtime on the considered data set is close to linear and even instances of size $n = 500$ can on average be solved heuristically in 30 milliseconds.

8 Conclusion

We have engineered an A^* algorithm that combines two different heuristics and discussed how to dynamically compute them. Our approach significantly outperforms bidirectional breadth-first search as the baseline, as well as our novel integer linear program. Random instances of size up to 25 points can consistently be solved by our algorithm. Our investigation of the instances from the sea surface reconstruction task shows that our approach can solve over 90% of the relevant instances (41 points in the North Sea) with small Delaunay orders in less than five minutes per instance.

On all of our considered application data, our new decomposition-based heuristic rapidly computes a solution that is already optimal, suggesting that in our case (sea surface reconstruction under higher-order Delaunay constraints) possibly only verifying optimality is hard. Overall, with our new optimal algorithm the clustering task in the context of sea surface reconstruction may become feasible for small data sets (e.g., the North Sea), and using our heuristic approach it would also be feasible for the global data sets (with more than 500 points [1]). Employing them here might yield interesting results.

References

1 Anna Arutyunova, Anne Driemel, Jan-Henrik Haunert, Herman Haverkort, Jürgen Kusche, Elmar Langetepe, Philip Mayer, Petra Mutzel, and Heiko Röglin. Minimum-Error Triangulations for Sea Surface Reconstruction. In Xavier Goaoc and Michael Kerber, editors, *38th International Symposium on Computational Geometry (SoCG 2022)*, volume 224 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 7:1–7:18, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SoCG.2022.7.

2 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. doi:10.1016/j.ipl.2006.07.001.

3 Prosenjit Bose and Ferran Hurtado. Flips in planar graphs. *Computational Geometry*, 42(1):60–80, 2009. doi:10.1016/j.comgeo.2008.04.001.

4 Miguel Bosch Calvo and Steven Kelk. An improved kernel for the flip distance problem on simple convex polygons. *Inf. Process. Lett.*, 182:106381, 2023. doi:10.1016/J.IPL.2023.106381.

5 Richie Chih-Nan Chuang, Ashim Garg, Xin He, Ming-Yang Kao, and Hsueh-I Lu. Compact encodings of planar graphs via canonical orderings and multiple parentheses. In Kim Guldstrand Larsen, Sven Skyum, and Glynn Winskel, editors, *Automata, Languages and Programming, 25th International Colloquium, ICALP'98, Aalborg, Denmark, July 13-17, 1998, Proceedings*, volume 1443 of *Lecture Notes in Computer Science*, pages 118–129. Springer, 1998. doi:10.1007/BFb0055046.

6 Sean Cleary and Katherine St. John. Rotation distance is fixed-parameter tractable. *Information Processing Letters*, 109(16):918–922, 2009. doi:10.1016/j.ipl.2009.04.023.

7 Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational geometry: algorithms and applications, 3rd Edition.* Springer, 2008. URL: https://www.worldcat.org/oclc/227584184.

8 Boris Delaunay. Sur la sphère vide. *Bulletin de l'Académie des Sciences de l'URSS, Classe des Sciences Mathématiques et Naturelles*, 6:793–800, 1934.

9 David Eppstein. Happy endings for flip graphs. In *Proceedings of the Twenty-Third Annual Symposium on Computational Geometry*, SCG '07, pages 92–101, New York, NY, USA, 2007. ACM. doi:10.1145/1247069.1247084.

10 Qilong Feng, Shaohua Li, Xiangzhong Meng, and Jianxin Wang. An improved FPT algorithm for the flip distance problem. *Information and Computation*, 281:104708, 2021. doi:10.1016/j.ic.2021.104708.

11 Joachim Gudmundsson, Mikael Hammar, and Marc van Kreveld. Higher order Delaunay triangulations. *Computational Geometry*, 23(1):85–98, 2002. doi:10.1016/S0925-7721(01)00027-X.

12 Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL: https://www.gurobi.com.

13 Sabine Hanke, Thomas Ottmann, and Sven Schuierer. The edge-flipping distance of triangulations. *Journal of Universal Computer Science*, 2, April 1996.

14 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. doi:10.1109/TSSC.1968.300136.

15 John F. Hughes, Andries van Dam, Morgan McGuire, David F. Sklar, James D. Foley, Steven K. Feiner, and Kurt Akeley. *Computer Graphics - Principles and Practice, 3rd Edition*. Addison-Wesley, 2014. URL: `http://vig.pearsoned.com/store/product/1,1207,store-12521_isbn-0321399528,00.html`.

16 F. Hurtado, M. Noy, and J. Urrutia. Flipping edges in triangulations. *Discrete & Computational Geometry*, 22(3):333–346, 1999. `doi:10.1007/PL00009464`.

17 Iyad Kanj, Eric Sedgwick, and Ge Xia. Computing the flip distance between triangulations. *Discrete & Computational Geometry*, 58(2):313–344, 2017. `doi:10.1007/s00454-017-9867-x`.

18 H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955. `doi:10.1002/nav.3800020109`.

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

20 Charles L. Lawson. Software for C^1 surface interpolation. In John R. Rice, editor, *Mathematical Software*, pages 161–194. Academic Press, 1977. `doi:10.1016/B978-0-12-587260-7.50011-X`.

21 Haohong Li and Ge Xia. An $\mathcal{O}(3.82^k)$ Time FPT Algorithm for Convex Flip Distance. In Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté, editors, *40th International Symposium on Theoretical Aspects of Computer Science (STACS 2023)*, volume 254 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 44:1–44:14, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.STACS.2023.44`.

22 Fabian Lipp. Computing the flip distance of triangulations. Bachelor's thesis, Julius-Maximilians-Universität Würzburg, 2012.

23 Jesús A. De Loera, Serkan Hosten, Francisco Santos, and Bernd Sturmfels. The polytope of all triangulations of a point configuration. *Documenta Mathematica*, 1996. URL: `https://api.semanticscholar.org/CorpusID:2943969`.

24 Anna Lubiw and Vinayak Pathak. Flip distance between two triangulations of a point set is NP-complete. *Computational Geometry*, 49:17–23, 2015. 24th Canadian Conference on Computational Geometry (CCCG'12). `doi:10.1016/j.comgeo.2014.11.001`.

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

26 Ayorkor Mills-Tettey, Anthony Stent, and M. Dias. The dynamic Hungarian algorithm for the assignment problem with changing costs, 2007.

27 James Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957. `doi:10.1137/0105003`.

28 Alina Nitzke, Benjamin Niedermann, Luciana Fenoglio-Marc, Jürgen Kusche, and Jan-Henrik Haunert. Reconstructing the dynamic sea surface from tide gauge records using optimal data-dependent triangulations. *Computers & Geosciences*, 157:104920, 2021. `doi:10.1016/j.cageo.2021.104920`.

29 Jean Pallo. An efficient upper bound of the rotation distance of binary trees. *Information Processing Letters*, 73(3):87–92, 2000. `doi:10.1016/S0020-0190(00)00008-9`.

30 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`.

31 Rodrigo I. Silveira and Marc van Kreveld. Optimal higher order Delaunay triangulations of polygons. *Computational Geometry*, 42(8):803–813, 2009. Special Issue on the 23rd European Workshop on Computational Geometry. `doi:10.1016/j.comgeo.2008.02.006`.

32 Daniel Dominic Sleator, Robert Endre Tarjan, and William P. Thurston. Rotation distance, triangulations, and hyperbolic geometry. In Juris Hartmanis, editor, *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA*, pages 122–135. ACM, 1986. `doi:10.1145/12130.12143`.

33 O.C. Zienkiewicz, R.L. Taylor, and J.Z. Zhu. *The Finite Element Method: Its Basis and Fundamentals*. Butterworth-Heinemann, Oxford, seventh edition, 2013. `doi:10.1016/B978-1-85617-633-0.00019-8`.

A Eppstein's Heuristic is Admissible and Consistent

▶ **Proposition 3.** *The heuristic $h_E(D) = l_E(D, D_t)$ is admissible and consistent.*

Proof. We start by showing that the heuristic is *admissible*. This has already been shown by Eppstein [9], but we reiterate the proof for the sake of completeness. By Observation 2 the length of a minimal flip sequence between D and D_t is given by the sum of lengths of the walks in QG. The length of every walk is lower bounded by the length of the shortest path between its endpoints. The shortest paths correspond to a perfect matching in $B(D, D_t)$ and thus, the sum of their lengths is lower bounded by the value of the MWPM in $B(D, D_t)$ which is exactly $h_E(D) = l_E(D, D_t)$; see Figure 2 for an example.

It remains to be shown that h_E is *consistent*. If all diagonals coincide, there is a perfect matching of length zero which implies $h_E(D_t) = l_E(D_t, D_t) = 0$.

Let D and D' be adjacent triangulations in $\mathrm{FG}(S)$. Since they are adjacent they differ by exactly one flip of the diagonal e. Thus, the diagonal e was moved in the QG graph to an adjacent diagonal e'. Consequently, in length, the shortest paths from e' to the diagonals in D_t can differ to the paths from e by at most one.

If we now assume that e and e' correspond to the same node v in $B(D, D_t)$ and we define c to be the costs with respect to D and c' to be the costs with respect to D'. Then, for all $x^t \in D_t$ we have $c(x, x^t) = c'(x, x^t)$ for all $x \neq v$ and $|c(x, x^t) - c'(x, x^t)| \leq 1$ for $x = v$. Now let M' be the MWPM for c' and M for c. We get

$$c'(M') \geq c(M') - 1 \geq c(M) - 1,$$

which implies $h_E(D) \leq h_E(D') + 1$. ◀

B (Our) Random Triangulations for Point Sets

In this section we briefly discuss how to generate random triangulations. To the best of our knowledge, there is no "easy" way to generate triangulations of point sets uniformly at random. We use the following approach:

1. Generate a set S of points uniformly at random in $[0, 1]^2$.
2. For every triangle T that can be used in some triangulation of S, generate a weight w_T uniformly at random in $[0, 1]$.
3. Compute the triangulation D_{rand} that minimizes the sum of its triangle weights, i.e.,

$$D_{\mathrm{rand}} = \operatorname*{argmin}_{D \in \mathfrak{T}(S)} \sum_{T \in D} w_T. \tag{1}$$

4. Repeat Steps 2 and 3 if multiple random triangulations for the same point set are needed.

Note that Step 3 is NP-hard, but for instances with up to 500 points the triangulation can usually be found quickly using an integer linear program. *We want to emphasize, that this approach not necessarily generates triangulations picked uniformly at random, but for our use case it yields an interesting mix of easy and hard instances.*

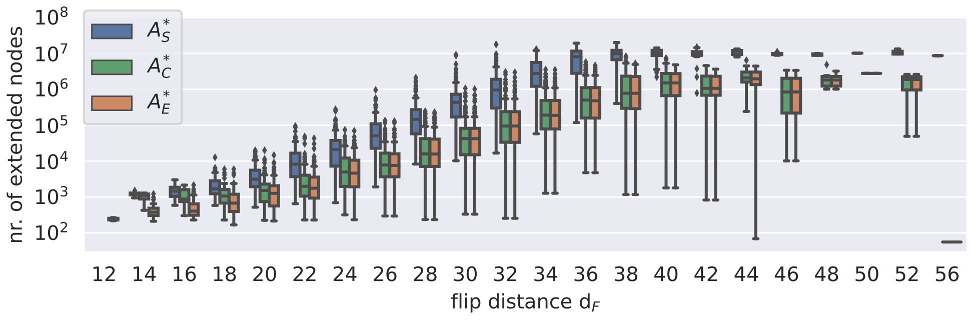
C A Brief Look at Node Extensions

We also considered the number of node extensions during our experiments. Again, we used all non-trivial instances of the datasets. In Figure 11 we present an overview of the node extensions grouped by the flip distance. In (a) the total numbers of node extensions are given and in (b) the ratios of extended nodes A_S^*/A_C^* and A_E^*/A_C^* are depicted. Notice that the numbers of extended nodes for A_S^* are the largest which immediately follows from the fact that Eppstein's heuristic is at least as accurate as the simple heuristic. Starting at flip distance 30 the number of extensions with A_S^* is five to ten times larger than the number of extensions for A_C^* and A_E^*. This can also be seen in the ratio-plot (b).

Up to this point, our discussion suggests that with increasing flip distance the node extension ratio increases. This is of course somewhat true, but there is a second aspect namely the quality difference of the heuristics that needs to be considered. We focus on the heuristic distance Δh at the root/start triangulation D_s. Most instances with large flip distances also have large Δh. A visualization of the ratio of extended nodes A_S^*/A_E^* grouped by Δh is given in Figure 12. As expected, we see that for $\Delta h = 0$ the ratio is close to one. For $\Delta h = 1$ it is between two and five and starting at $\Delta h = 2$, it is larger than ten.

This explains the runtime-wins that A_S^* achieves for $\Delta h < 2$, since for those instances the decreased number of node extensions for A_E^* with respect to A_S^* is not enough to compensate for the increased runtime of the heuristic calculation of h_E.

(a)



(b)

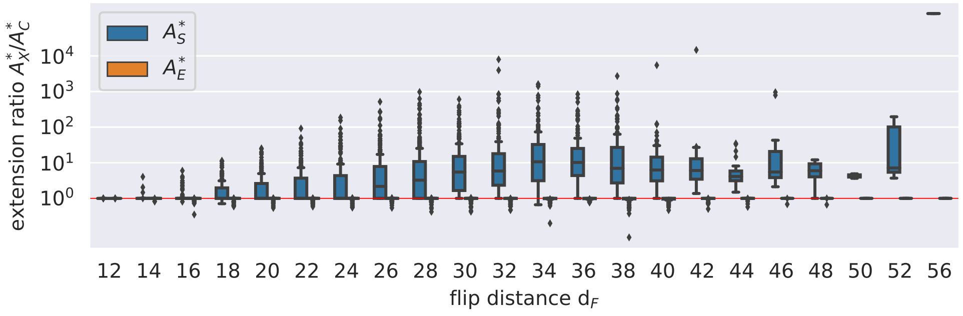


Figure 11 The number of node extensions of the A^* approaches on all considered random and real-world data. Only solvable and non-trivial instances were considered. In (a), the total numbers of node extensions are given, and in (b) the node extension ratios of A_S^* and A_E^* with respect to A_C^* are given grouped by flip distance (uneven d_F values are grouped with their even predecessors).

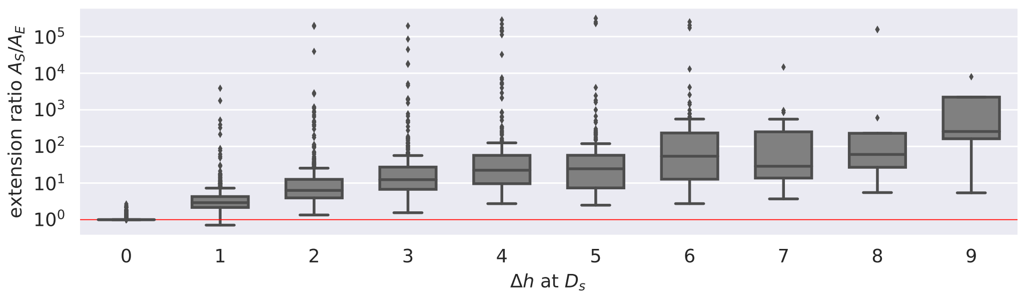


Figure 12 The ratio of extended nodes of A^*_S with respect to A^*_E, grouped by the heuristic distance Δh at the root/start triangulation D_s.