Lines: 407

Shadoks Approach to Parallel Reconfiguration of Triangulations

Guilherme D. da Fonseca
LIS, Aix-Marseille Université

Fabien Feschet
LIMOS, Université Clermont Auvergne

Yan Gerard
LIMOS, Université Clermont Auvergne

Abstract

We describe the methods used by Team Shadoks to win the CG:SHOP 2026 Challenge on parallel reconfiguration of planar triangulations. An instance is a collection of triangulations of a common point set. We must select a center triangulation and find short parallel-flip paths from each input triangulation to the center, minimizing the sum of path lengths. Our approach combines exact methods based on SAT with several greedy heuristics, and also makes use of SAT and MaxSAT for solution improvement. We present a SAT encoding for bounded-length paths and a global formulation for fixed path-length vectors. We discuss how these components interact in practice and summarize the performance of our solvers on the benchmark instances.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Heuristic, exact algorithm, SAT solver, MaxSAT solver, computational geometry

Category CG Challenge

Related Version arxiv.org/abs/

Supplementary Material *(Source Code)*: `https://github.com/gfonsecabr/shadoks-CGSHOP2026`

Acknowledgements We would like to thank the Challenge organizers and other competitors for their time, feedback, and making this whole event possible. We would like to thank Hélène Toussaint, Raphaël Amato, Boris Lonjon, and William Guyot-Lénat from LIMOS, as well as the Qarma and TALEP teams and Manuel Bertrand from LIS, who continue to make the computational resources of the LIMOS and LIS clusters available to our research. We would also like to thank Aldo Gonzalez-Lorenzo for the very useful discussion on SAT models.

1 Introduction

The CG:SHOP Challenge is an annual competition in geometric optimization. In its eighth edition in 2026, the challenge focuses on a reconfiguration problem between planar triangulations. Our team, called *Shadoks*, won first place with the best solution (among the 28 participating teams) to 249 instances out of 250 instances and provably optimal solutions to 189 instances.

In this paper, we outline the exact methods and heuristics that we employed. We start with some definitions that allow us to describe the problem. Throughout, we consider triangulations of a common point set $S \subset \mathbb{R}^2$. Given a triangulation T, a *unit flip* is the operation that removes an edge $e \in T$ and adds an edge e', obtaining a new triangulation $T' = T \setminus \{e\} \cup \{e'\}$. Notice that the edge e must cross e'. Similarly, a *parallel flip* removes a set of edges $E \subset T$ and adds a set of edges E', in a way that $T' = T \setminus E \cup E'$ is a triangulation, with the condition that no two edges of E are in the same triangle in T. A *path* of *length* ℓ is a sequence of triangulations $T_0, \ldots, T_\ell$ such that for all i, the triangulation T_{i+1} is obtained from T_i by performing a parallel flip.

An *instance* is a set $S \subset \mathbb{R}^2$ of n points and a set $\mathcal{T} = \mathbf{T}_1, \ldots, \mathbf{T}_{|\mathcal{T}|}$ of triangulations of S, called *input triangulations*. A *solution* is a set of paths $P_1, \ldots, P_{|\mathcal{T}|}$ such that P_i starts at $\mathbf{T}_i$ for all i and all paths end in a common triangulation called *center*. The goal is to find a solution that minimizes the *objective value* defined as the sum of the lengths of its paths.

During the competition, the organizers provided a total of 250 instances, with n ranging from 15 to 12,500 points and $|\mathcal{T}|$ ranging from 2 to 200 triangulations. The 250 instances are divided into three classes: 100 `random` instances, 101 `woc` instances, and 49 `rirs` instances. The former two instances have up to 320 points and 2 to 20 input triangulations (hence, we call them `small` instances), while the latter have 500 to 12500 points and 20 to 200 input triangulations. The centers of some of our best solutions are presented in Figure 1. Additional details about the challenge can be found in the organizers' survey paper [5].

Our best solvers heavily rely on the SAT solver `CaDiCal` [3] and the MaxSAT solver `EvalMaxSAT` [2]. Nevertheless, we also developed heuristics that do not rely on any external solver, which are important to find initial solutions to some large instances, which are then improved by roughly 10% using SAT and MaxSAT solvers. Furthermore, we managed to solve 189 of the 201 `small` instances exactly by repeatedly using the SAT solver as well as some lower bounds.

Mention the other teams strategy here...

We describe our exact algorithms in Section 2, the heuristics in Section 3, and discuss the results we obtained in Section 4. Concluding remarks and open problems are presented in Section 5.

2 Exact Algorithms

This section describes all elements of our exact solver, many of which are also used in the heuristic solvers. We first show how to use a SAT solver to compute shortest paths between two triangulations (Section 2.1). We then show how to extend this result to test if a solution with a list of path lengths exists (Section 2.2). We show how to obtain lower bounds in Section 2.3 and put the previous elements together to describe our exact solver in Section 2.4.

2.1 Path SAT Formulation

We now describe a SAT formulation for the following decision problem. The input is a set S of n points, an integer ℓ and two triangulations T_0, T_ℓ. The output is whether there exists a

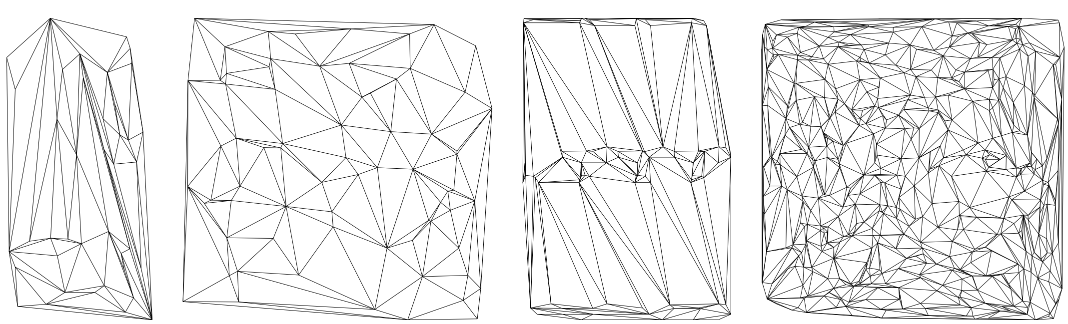

Figure 1 Our best centers to instances `random_78_40_10`, `woc-70-random-9a7d18d3`, `woc-90-tsplib`, and `rirs-500-50-23d00ec5`, respectively.

path $T_0, \ldots, T_\ell$ of length ℓ.

We define two types of variables. For $i = 0, \ldots, \ell$ and for $u \neq v \in S$, we define an *edge variable* $e(u, v, i)$. The variable $e(u, v, i)$ represents that the edge uv is in the triangulation T_i. There are $O(n^2\ell)$ edge variables. It would be possible to define a SAT formulation using only such variables. However, a SAT formulation that performed much better in our experiments uses a second type of variable.

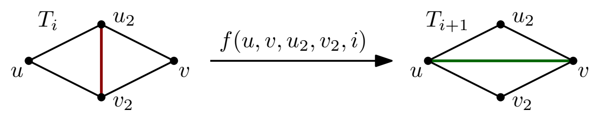

Figure 2 Illustration of a flip variable $f(u, v, u_2, v_2, i)$.

We say that a convex quadrilateral is *empty* if it contains no point of S except for its vertices. For $i = 0, \ldots, \ell$ and for $u \neq v \neq u_2 \neq v_2 \in S$ such that u, u_2, v, v_2 form an empty convex quadrilateral, we define a *flip variable* $f(u, v, u_2, v_2, i)$. The variable $f(u, v, u_2, v_2, i)$ represents a unit flip such that the edge uv is in triangulation T_i and u_2v_2 is in triangulation T_{i+1}, as shown in Figure 2. Notice that if the points are uniformly distributed, then the number of empty convex quadrilaterals is $\Theta(n^2)$ [4], which means that for uniformly distributed points, the number of flip variables is also $O(n^2\ell)$. However, the number of flip variables is $\Theta(n^4\ell)$ if the points are in convex position (which is not the case for the challenge instances). Next, we describe the different types of clauses.

Start. For every edge variable $e(u, v, 0)$, we have the clause $e(u, v, 0)$ if $uv \in T_0$ and $\neg e(u, v, 0)$ if $uv \notin T_0$.

Target. For every edge variable $e(u, v, \ell)$, we have the clause $e(u, v, \ell)$ if $uv \in T_\ell$ and $\neg e(u, v, \ell)$ if $uv \notin T_\ell$.

Flips need edges. For every flip variable $f(u, v, u_2, v_2, i)$, we have the clause

$$f(u, v, u_2, v_2, i) \implies e(u, v, i) \wedge e(u, v_2, i) \wedge e(u, u_2, i) \wedge e(v, v_2, i) \wedge e(v, u_2, i),$$

which easily translates to 5 binary CNF clauses.

Flips keep edges. For every flip variable $f(u, v, u_2, v_2, i)$, we have the clause

$$f(u, v, u_2, v_2, i) \implies e(u_2, v_2, i{+}1) \wedge e(u, v_2, i{+}1) \wedge e(u, u_2, i{+}1) \wedge e(v, v_2, i{+}1) \wedge e(v, u_2, i{+}1),$$

which easily translates to 5 binary CNF clauses.

Flips flip edges. For every flip variable $f(u, v, u_2, v_2, i)$, we have the two clauses

$$f(u, v, u_2, v_2, i) \implies e(u_2, v_2, i+1) \text{ and } f(u, v, u_2, v_2, i) \implies \neg e(u, v, i+1).$$

Edge changes require flips. The last type of clause is the only one that has more than 2 variables in CNF form. It states that if the edge variable changes from triangulation i to $i + 1$, then there must be a flip. The $\bigvee$ below considers all values that produce valid flips. We have two such clauses for every edge variable:

$$e(u, v, i) \wedge \neg e(u, v, i + 1) \implies \bigvee_{u_2, v_2} f(u, v, u_2, v_2, i) \text{ and}$$

$$\neg e(u_2, v_2, i) \wedge e(u_2, v_2, i+1) \implies \bigvee_{u,v} f(u, v, u_2, v_2, i).$$

The number of variables and clauses grows very fast, even though the number of clauses is linear in the number of variables. Next, we show how to eliminate many variables from the model. All eliminated variables are assumed to be *false* in every clause, and the clauses that become tautologies are also eliminated. If a CNF clause becomes empty, then the problem is unsatisfiable.

Notice that we can easily eliminate $e(u, v, 0)$ for $uv \notin T_0$ and $e(u, v, \ell)$ for $uv \notin T_\ell$. This is, however, only a special case of a more general rule. The following theorem is easy to prove and implies that $\Omega(\log n)$ parallel flips are sometimes necessary to reconfigure two triangulations of n points, even when the points are in convex position. We say that two segments *cross* if they intersect at a point that is not an endpoint of either segment.

▶ **Theorem 1.** *Consider two triangulations T, T' of S such that a parallel flip transforms T into T' and a segment s with endpoints in S. Let χ, χ' respectively denote the number of edges of T, T' crossed by s. We then have $\chi' \geq \lfloor \chi/2 \rfloor$.*

Proof. Consider the sequence L of edges of T crossed by s in the order they cross the segment s. A parallel flip cannot remove two consecutive edges of L because they share a triangle, hence the theorem follows. ◀

Consequently, we only define the variable $e(u, v, i)$ when uv crosses strictly less than 2^i edges of T_0 and strictly less than $2^{\ell-i}$ target edges. We only define flip variables when a certain set of edge variables is defined. Namely, $f(u, v, u_2, v_2, i)$ is only defined when $uv, uv_2, uu_2, vv_2, vu_2$, are all defined at i and $u_2v_2, uv_2, uu_2, vv_2, vu_2$, are all defined at $i+1$.

2.2 Solution SAT Formulation

Next, we describe a SAT formulation for the following decision problem. Recall that an instance is a set S of n points and a list $\mathcal{T}$ of input triangulations $\mathbf{T}_1, \ldots, \mathbf{T}_{|\mathcal{T}|}$. The input of the decision problem is an instance and $|\mathcal{T}|$ integers $\ell_1, \ldots, \ell_{|\mathcal{T}|}$. The output is whether there exists a solution $P_1, \ldots, P_{|\mathcal{T}|}$ such that path P_i has length ℓ_i for all i.

We model the $|\mathcal{T}|$ paths $P_1, \ldots, P_{|\mathcal{T}|}$ independently as before, starting path P_i at the input triangulation $\mathbf{T}_i$. The final triangulation of each path is unknown, but the same edge variables are used for the final triangulation of every path, since a valid solution requires that all paths end in the same triangulation. It is easy to see that the SAT formulation is satisfiable if and only if there exists a solution with the given lengths.

2.3 Lower Bound

In order to obtain an exact solution to an instance $\mathcal{I}$, we start by computing a lower bound to its objective value. We say that the *distance* between two triangulations T, T' is the length of the shortest path from T to T'. We create a complete directed graph $G(\mathcal{I})$ with edge weights as follows. The vertices are the triangulations $\mathcal{T}$ and the weight of the edges are the distances between the corresponding triangulations. A *cycle packing* of G is a collection of vertex-disjoint directed cycles, i.e. a subset of edges such that each vertex has at most one outgoing and at most one incoming edge in the subset. The graph is directed to allow for cycles with only 2 edges. The *length of a cycle* is the sum of the lengths of its edges, and the *length of a cycle packing* is the sum of the lengths of its cycles. We have the following theorem.

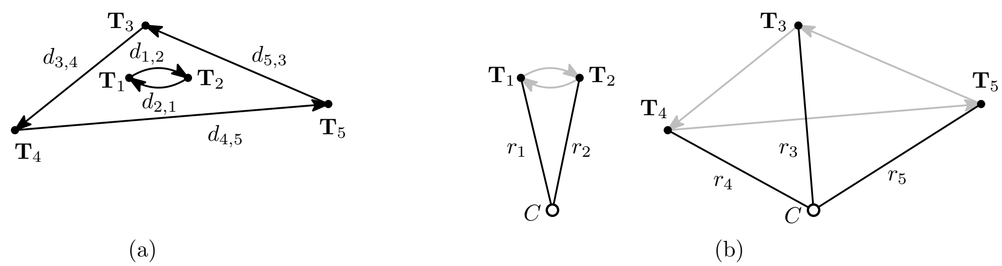

Figure 3 (a) A cycle packing. (b) Illustration of the proof. In this example, $d_{1,2} \leq r_1 + r_2$, $d_{2,1} \leq r_2 + r_1$, $d_{3,4} \leq r_3 + r_4$, $d_{4,5} \leq r_4 + r_5$, and $d_{5,3} \leq r_5 + r_3$ by triangle inequality.

▶ **Theorem 2.** *Given an instance $\mathcal{I}$, the objective value of a solution is at least the length of any cycle packing of $G(\mathcal{I})$ divided by 2.*

Proof. Let $d_{i,j}$ denote the distance between the input triangulations $\mathbf{T}_i, \mathbf{T}_j$. Let C be a potential center and let r_i be the distance from C to $\mathbf{T}_i$. Consider a cycle $\mathbf{T}_1, \ldots, \mathbf{T}_k$ in the cycle packing. By triangle inequality $d_{i,i+1} \leq r_i + r_{i+1}$ with indices taken modulo k (see Figure 3 (b)). Summing over the inequalities for i from 1 to k, we have that the length of the cycle is at most $2 \sum_i r_i$. Applying the same argument to every cycle, the theorem follows. ◀

2.4 The Exact Solver

First, we use the exact path formulation from Section 2.1 to calculate the distance between all $\binom{\mathcal{T}}{2}$ pairs of input triangulations using a SAT solver (in our case, `CaDiCal`). It is easy to formulate the problem of finding a maximum length cycle packing as a weighted MaxSAT problem, which provides a lower bound b to the objective value (see Section 2.3). We solve this problem using a weighted MaxSAT solver (in our case `EvalMaxSAT`). We then use backtracking to list all integer solutions to $\ell_0, \ldots, \ell_{|\mathcal{T}|} = b$ that satisfy $\ell_i + \ell_j \geq \text{distance}(T_i, T_j)$. We use the SAT formulation from Section 2.2 to test the existence of a solution with the given lengths $\ell_0, \ldots, \ell_{|\mathcal{T}|}$, again using the `CaDiCal` SAT solver. If a solution is found, then it is optimal. Otherwise, we increment b and repeat. Notice that b is always a lower bound to the objective value. Hence, if a solution obtained by a heuristic attains this lower bound, then it is optimal.

3 Heuristics

In this section, we describe different approaches that can be used when we do not need to guarantee the optimality of the solution. In Section 3.1, we present a conjecture that allows us to significantly increase the performance of the SAT solver. In Section 3.2, we show how to further increase the performance of the SAT solver using a heuristic coupled with some instance-independent preprocessed data. In Section 3.3, we show how to use the SAT solver to improve existing solutions. In Sections 3.4 and 3.5, we respectively show how to compute short paths and good solutions without a SAT solver.

3.1 Happy Edges Conjecture

The happy edges conjecture [1] is a general conjecture that is *false* for some reconfiguration problems and *true* for others.

▶ **Conjecture 3.** *For any pair of configurations T, T', there always exist a shortest path between T, T' where the edges that are common to both T and T' appear in every intermediate configuration.*

The conjecture is *false* for triangulations under unit flips and arbitrary points [10] but *true* when the points are in convex position [11]. Our experiments lead us to believe that the conjecture is *true* for parallel flips and arbitrary point sets.

Enforcing that an edge never disappears and later reappears along a path actually makes the SAT formulation harder to solve. However, there are some implications of the conjecture that are very useful to make the SAT formulation shorter and easier to solve.

When computing a path of length ℓ from T_0 to T_ℓ using SAT, for every edge uv that appears in both T_0 and T_ℓ, we add clauses $e(u, v, i)$ that force the variable to be true for all i. More importantly, we then eliminate every edge variables corresponding to edges that cross uv. The same idea can be applied to the SAT formulation that finds a solution, but then only the edges that appear in all input triangulations are forced to be true for all i, and again the edges that cross them are eliminated.

Furthermore, when computing a path, for every edge $uv \in T_\ell$, we eliminate flip variables that remove uv, i.e. $f(u, v, u_2, v_2, i)$ for all u_2, v_2, i. Similarly, for every edge $u_2v_2 \in T_0$, we eliminate flip variables that insert u_2v_2, that is $f(u, v, u_2, v_2, i)$ for all u, v, i.

3.2 Crossing Lower Bound

Let $b(uv, T_0)$ denote the number of parallel flips needed to obtain an edge uv starting at a triangulation T_0. Clearly, when creating the SAT formulation for a path $T_0, T_1, \ldots$ we only need to define the edge variable $e(u, v, i)$ for $i \geq b(T_0, uv)$. Theorem 1 implies that if an edge uv crosses $\chi(uv, T_0)$ segments of T_0, then $b(uv, T_0) \geq \lceil \log_2(\chi(uv, T_0) + 1) \lceil$. This bound is tight when all the edges of T_0 that cross uv share a common endpoint and the endpoints are in convex position (Figure 4). However, there are different ways in which the edges of T_0 may cross uv that may imply higher values of $b(uv, T_0)$.

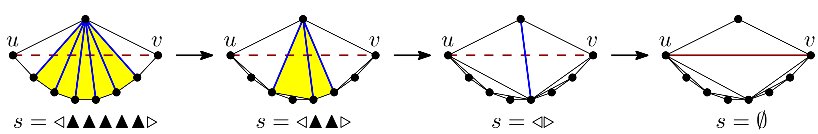

Figure 4 A path of length 3 to insert an edge that had 6 crossings.

We consider estimations of $b(uv, T_0)$ based on the sequence of triangles that contain an upper or a lower edge (Figure 5). An edge uv that crosses $\chi(uv, T_0)$ segments of T_0 is translated into a *string* $s = s(uv, T_0)$ of $\chi(uv, T_0) + 1$ symbols in the alphabet $\{\blacktriangledown, \blacktriangle, \triangleleft, \triangleright\}$, according to which sides of uv contain the edges that are not crossed by uv in each triangle.

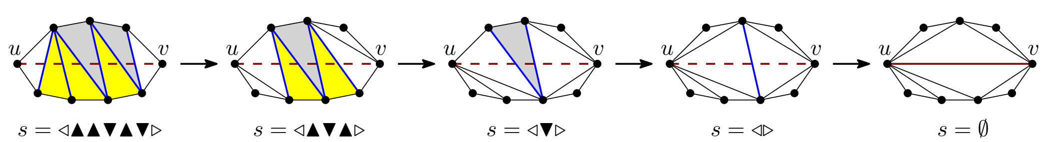

Figure 5 A path of length 4 to insert an edge that had 6 crossings. Each triangle is labeled and colored as containing an upper or a lower edge.

The unit flips on T_0 have equivalent *productions* on s that replace the substring on the *left-hand side* by the substring on the *right-hand side*. We define a *substring* as a contiguous subsequence. Flipping the two extreme triangles translates to the *extreme productions* ◁▲ $\to$ ◁, ◁▼ $\to$ ◁, ▲▷ $\to$ ▷, and ▼▷ $\to$ ▷. Flipping intermediate triangles with the same orientation translates to ▲▲ $\to$ ▲ and ▼▼ $\to$ ▼, while flipping intermediate triangles of different orientations translates to ▼▲ $\to$ ▲▼ and ▲▼ $\to$ ▼▲. The last flip to insert the edge uv is ◁▷ $\to \emptyset$. There are other productions that may increase the string length such as ▲ $\to$ ▲▲, which consist of exchanging the left-hand and right-hand side of some aforementioned productions, but we show that we can always obtain shortest paths without using them.

A parallel flip means that we may apply several productions simultaneously as long as their left-hand side corresponds to disjoint substrings. We call the application of several productions a *rewriting*. A *rewriting sequence* of length ℓ is a sequence of $\ell + 1$ strings connected by rewriting operations and ending at the empty string $\emptyset$. We want to find shortest rewriting sequences. Let $b(s)$ be the length of the shortest rewriting sequence of the string s.

Notice that if the points are not in convex position, then some productions may correspond to invalid flips. Hence, assuming convex position provides a lower bound $b(s(uv, T_0)) \leq b(uv, T_0)$. Furthermore, the productions assume that there is an unbounded number of points, so that new triangles may be created freely.

We use a heuristic together with a depth first search exact solution to estimate $b(s)$. The heuristic is very involved and sometimes gives incorrect bounds (50182 wrong values for 33554432 tested strings, which is roughly 1/600 wrong bounds, the smallest wrong bound being for ◁▼▲▲▼▲▲▲▲▷). To fix some wrong bounds, we use an instance-independent preprocessing to compute tight values of $b(s)$ for small enough strings s and store only the values where the heuristic incorrectly computes $b(s)$ in a file. This file is loaded by our solver and stored in a hash table.

3.3 Improving a Solution

Given a solution $P_1, \ldots, P_{|\mathcal{T}|}$, we may improve it as follows. We choose a random path P_i and use the SAT formulation to find a solution where the length of P_i is decremented and all other lengths remain the same. We repeat this as often as necessary. Notice that the method may converge to a locally optimal minimal list of lengths that is not globally optimal.

If the number of clauses is too large, there are two different approaches that we can take (and they may be combined). We may force the new solution to be close to the original one by only creating edge variables that cross few edges in the corresponding triangulation of the previous solution.

We may also trim the solution to a certain radius r, by only rebuilding the portion of the solution that is within r steps from the center. In this case, it is helpful to use MaxSAT first, in order to reduce the number of unit-flips performed in the last steps as follows. Given a path $T_0, \ldots, T\ell$, we first use a MaxSAT solver to find the path of length ℓ that minimizes the number of unit flips performed from $T_{\ell-1}$ to T_ℓ. The MaxSAT formulation is equal to the SAT formulation with soft clauses $\neg f(u, v, u_2, v_2, \ell)$ for each last step flip variable. We then find the path of length $\ell - 1$ from T_0 to the $T_{\ell-1}$ of the previosu path that minimizes the number of unit flips performed from $T_{\ell-2}$ to $T_{\ell-1}$. We continue this way for r steps.

3.4 Path Heuristic

The SAT formulation finds the shortest path connecting two triangulations reasonably fast, but it may be too slow for some usages. We also designed a heuristic that produces reasonably short paths quickly. We are given two triangulations T_0, T' and the goal is to find a short path from T_0 to T'.

We use a greedy approach that iteratively obtains a triangulation T_{i+1} from a triangulation T_i as follows. Let F denote the set of possible unit flips in T_i. More precisely, the elements of F are pairs e, e' of edges such that a unit flip from T_i removes e and inserts e'. We then build a graph $G(F)$ with vertex set F and edges between two unit flips that share a triangle. Notice that the possible parallel flips correspond to independent sets in $G(F)$. We assign a weight to the vertices as follows. The weight of a vertex e, e' is the number of edges in T' crossed by e' minus the number of edges in T' crossed by e. Vertices of zero or negative weight are eliminated.

We then proceed to greedily find an independent set I in $G(F)$. We iteratively add to I the *unmarked* vertex of maximum weight, breaking ties by minimum degree. We then *mark* all the vertices in the closed neighborhood of I. We repeat until all vertices are marked.

This iterative approach is repeated until we reach T', which will happen in $O(n^2)$ flips because of the following Theorem from [7].

▶ **Theorem 4.** *Let T, T' be two triangulations of the same point set. If $T \neq T'$, then there exists a unit flip that replaces an edge $e \in T$ by an edge e' such that e crosses more edges of T' than e' does.*

We further improve the heuristic using the squeaky wheel paradigm [8]. Initially, we assign weight 1 to every edge. We modify the definition of the weight of a flip as follows. The weight of a flip e, e' is the sum of the weights of the edges in T' crossed by e' minus the sum of the weights of the edges in T' crossed by e. When we reach triangulation $T_{i+1} = T'$ from a triangulation T_i we increment the weight of all edges in T' that are not in T_{i+1}. We repeat the greedy algorithm with the new weights, keeping the best solution found. This procedure is repeated multiple times.

3.5 Solution Heuristic

We use the following strategy to obtain reasonably good initial solutions that we may use as a basis to improve with the aforementioned methods. We start by adding the Delaunay triangulation as a candidate center. We then build other candidate centers by performing flips starting from the Delaunay triangulation. Given an edge e and a triangulation T, let $\chi(e, T)$ denote the number of segments of T crossed by e. We repeatedly perform unit flips that remove an edge e and add an edge e' maximizing

$$\sum_{T \in \mathcal{T}} \chi(e)^p - \chi(e')^p,$$

as long as the value of the sum is positive. We add the triangulation we obtained to the set of candidate center and repeat the process for a different value of p.

We calculate the paths from the candidate centers to each solution, building a solution pool. For the next step it will be useful to have a small number of unit-flips in the last flip of the path from the input triangulation to the center. To do that, we either use the greedy heuristic or a MaxSAT formulation.

To improve the solution pool, we pick a solution from the pool and look at the triangulations that are one flip away from the center in each path. For each such triangulation, we compute the distance to the input triangulations and add the solution to the pool as before.

4 Results

In this section, we present the computational results that we obtained with our implementation of the aforementioned algorithms and heuristics. In Section 4.1, we present the results on computing short paths between two given triangulations. In Section 4.2, we present our exact solver, with and without the happy edges conjecture. In Section 4.3, we present the heuristics used to find solutions to the instances that we could not solve exactly.

The solvers were coded in C++ and compiled with GCC and run a single thread. During the competition, they were executed on several Linux computers, either using GNU Parallel [12] for local executions or Slurm [13] for cluster executions. It was very useful to have access to machines with 128GB or more RAM to be able to solve large SAT formulations with `CaDiCal` [3], which has been able to solve SAT instances with more than 50 million variables and 500 million clauses. The time measurements on this paper have all been taken on an AMD Ryzen 9 9900X CPU and ASUS TUF GAMING B650M motherboard with 128GB of RAM running Fedora Core 43.

4.1 Path Calculation

Computing short paths between two given triangulations is a key component to obtain good solutions. Typically, these paths are computed with an input triangulation as one extreme, and a triangulation that makes a reasonably good center as the other extreme. In this section, we use the Delaunay triangulation as one extreme, because it is a well defined triangulation that makes a reasonably good (but not very good) center. Table 1 shows the length of the path and the running time of different heuristics and SAT solutions with and without the happy edges conjecture. The paths are calculated from the Delaunay triangulation to the first input triangulation of several instances. The SAT paths are obtained by first running the squeaky wheel heuristic in both directions, and then iteratively decreasing the path length. Notice that the running time of the heuristics is much smaller, while the result is rarely more than 1 unit away from the optimal distance, especially if we take the minimum of both directions. We run the squeaky wheel heuristic for at most 16 iterations, but stops earlier if the length increases from one iteration to the next.

4.2 Exact Solutions

This section presents the exact solver that we used to solve most instances exactly. Figure 6 shows the number of exact solutions found as a function of the running time, both without and with the happy edges conjecture (the value of the solutions found in both cases is the same, hence the conjecture stands). Notice that the impact of the happy edges conjecture is less significant than when computing only paths, as there are few common edges on all input triangulations. We remark that during the competition, we managed to find exact solutions to 189 of the 201 `small` instances without assuming the happy edges conjecture.

4.3 Heuristic Solutions

The typical process to solve instances that we did not solve exactly consists of several steps. First, we use the heuristic from Section 3.5 to obtain a reasonably good center. Notice that no SAT solver is used in this part. The evolution of the objective value for this step is shown in Figure 7. The name of the `rirs` instances is composed of two values. The first is the number of points and the second is the number of input triangulations.

	Greedy forward		Greedy backward		Squeaky forward		Squeaky backward		SAT happy		SAT exact	
n	ℓ	t	ℓ	t	ℓ	t	ℓ	t	ℓ	t	ℓ	t
500	13	4.2	**12**	3.7	13	45	**12**	18	**12**	1512	**12**	10191
1000	12	7.8	12	6.8	12	25	12	38	**11**	6806	**11**	78264
1500	13	13	12	12	12	52	12	23	**11**	19302	**11**	201353
2000	15	21	**13**	18	**13**	65	**13**	104	**13**	16937	**13**	485177
3000	15	36	15	33	**14**	193	15	433	**14**	140729	·	·
4000	18	57	16	51	16	577	**15**	655	**15**	101402	·	·
5000	18	70	17	70	**16**	238	17	278	**16**	1037840	·	·
6000	17	93	**16**	83	17	1071	**16**	644	**16**	271081	·	·
7000	17	94	17	96	17	180	17	407	**16**	471389	·	·

Table 1 Length and computation time (in milliseconds) from the Delaunay triangulation to the first input triangulation of the `rirs-`n`--20` instance for different values of the number of points n. The columns respectively correspond to the greedy heuristic forward and backward, the squeaky wheel heuristic forward and backward, the SAT solution with the happy edges conjecture, and the SAT solution without the happy edges conjecture, unless it takes too long. The best lengths found are shown in bold.

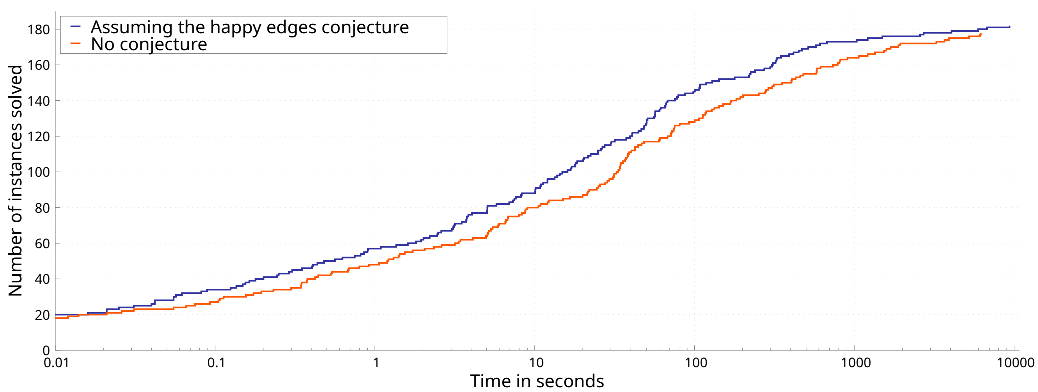

Figure 6 Number of exact solutions found as a function of the running time over 3 hours of execution with and without assuming the happy edges conjecture.

Second, we build a new solution that keeps the same center but we recalculate the paths using the SAT formulation from 2.1, assuming the happy edges conjecture (Section 3.1) and inexact lower bounds (Section 3.2). This will typically reduce the objective value by 1 to 4 percent. The calculation of 50 paths in each solution is shown in Figure 8. Notice that the running time increases rapidly with the number of points and that finding a shorter path (satisfiable SAT problem) is typically slower than when no shorter path exists (unsatisfiable SAT problems).

Third, we improve the solution using the SAT formulation from Section 2.2, assuming the happy edges conjecture (Section 3.1) and inexact lower bounds (Section 3.2). This part requires adjusting a large number of parameters in order to obtain SAT problems that are not too hard. The parameters include the distance to the previous center, the distance to the previous path, and whether the solution will be trimmed. Optionally but recommended when the problem is trimmed, we use MaxSAT to reduce the number of unit flips close to the center. The improvement of some solutions over time is shown in Figure 9. Notice that

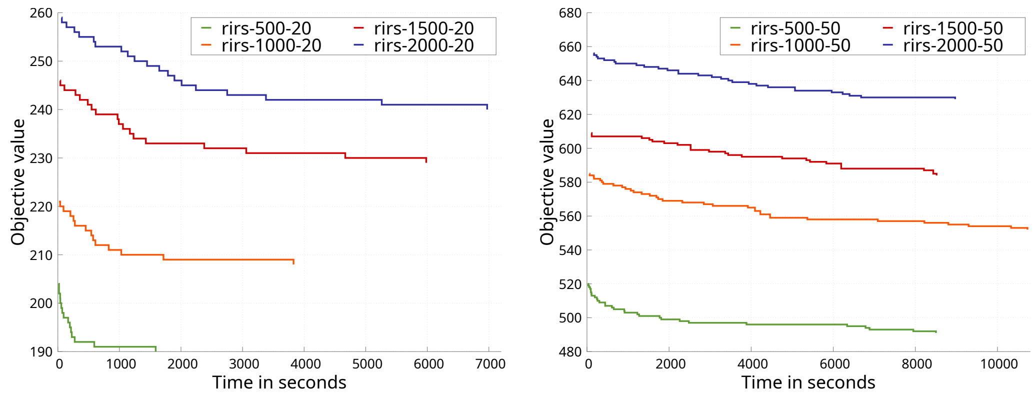

Figure 7 Evolution of the best solution found by the heuristic solver without any SAT solver for different instances.

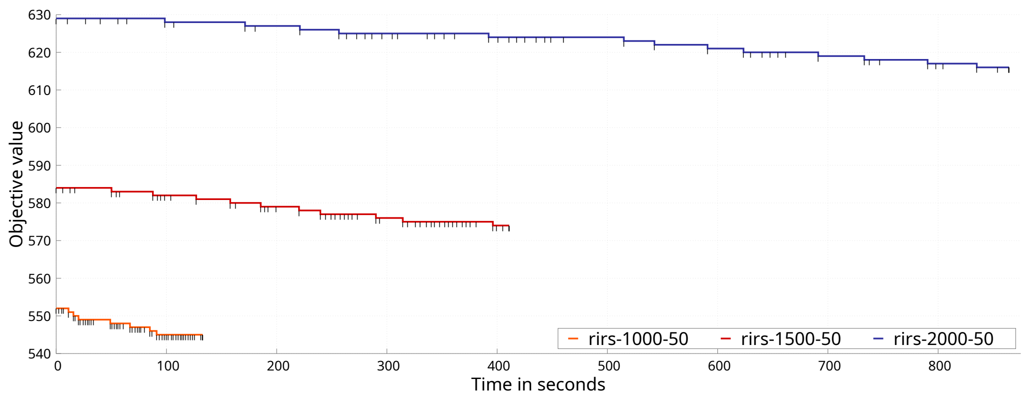

Figure 8 Path by path improvement of the heuristic solutions using a SAT solver, where each path is recalculated but the center is kept unchanged. Each black vertical bar represents the beginning of the computation of a path.

there is a significant preprocessing time to calculate edge variables with a limited number of crossings. This preprocessing is applied initially and after every improvement. Also notice that unsatisfiable SAT problems, which mean no improvement in the solution, are solved much faster than satisfiable ones.

5 Conclusion and Open Problems

We were surprised that we managed to solve so many instances exactly and how well an heuristic approximates the exact distance. We believe the following factors help explain the strong practical performance:

- The short path length that allows a somewhat small number of variables.
- A SAT model where most clauses have size 2.
- The ability to eliminate many variables through several arguments.
- The fact that the happy edges conjecture is either true or at least holds in most practical cases.

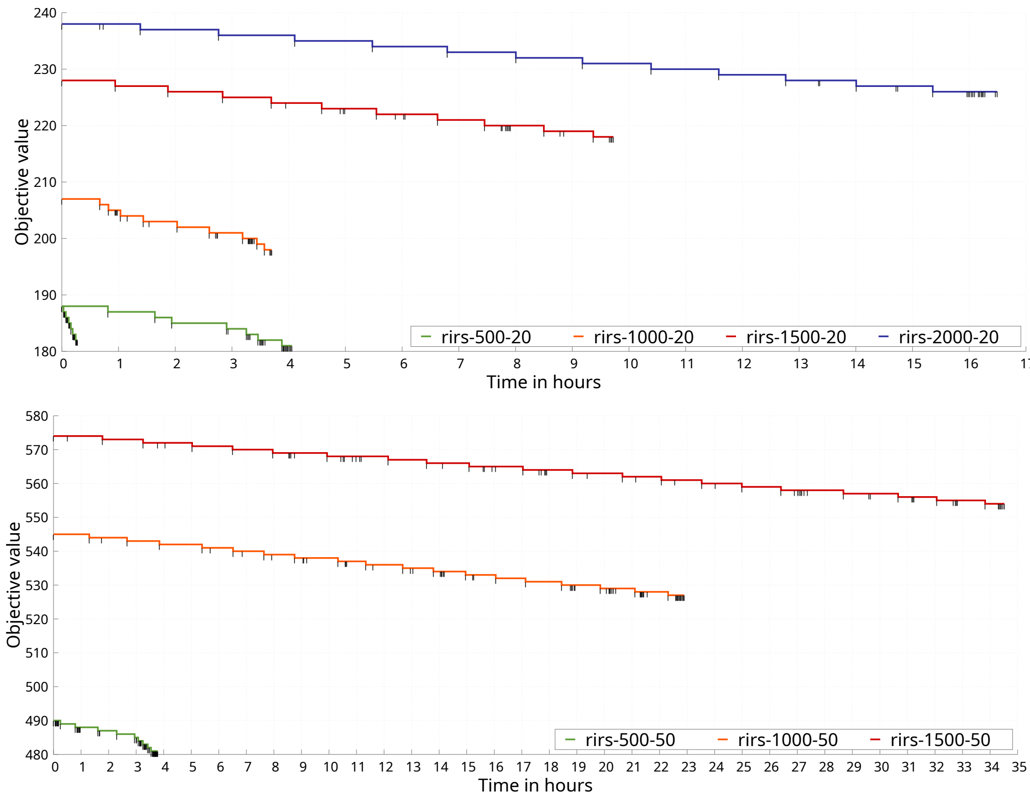

Figure 9 Improvement of the whole solution using a SAT solver and reducing the length of a random path by one unit at a time. We constrain the edges in the new solution to cross at most 3 edges of the corresponding triangulation in the solution (except for the instance `rirs-500-20`, where the slower but more effective execution with the parameter set to at most 7 edges is also pictured) but do not use trimming. Each black vertical bar represents a new path length that we try to decrement.

The short path lengths are somewhat surprising, given that an $\Omega(n)$ lower bound is presented in [6], in contrast to the $\Theta(\log n)$ bound for combinatorial triangulations [9], where we are allowed to flip non-convex quadrilaterals. In the challenge instances, we observed path lengths that are roughly $O(\log n)$.

Still, there are `random` instances with only 160 points and 20 input triangulations and `woc` instances with only 185 points and 6 triangulations that we could not solve exactly. Also, we still managed to improve solutions to instances with as few as 320 points and 20 input triangulations months after the beginning of the challenge.

The challenge instances did not have many points in convex position. Surprisingly, the case where all points are in convex position is the hardest for our SAT formulation, as there are $\Theta(n^4)$ empty convex quadrilaterals. We wonder if a different model works better when all and also when most points are in convex position.

We believe that the same problem with unit flips is significantly harder because the long path lengths make the SAT formulation much more complex and removing a happy edge can reduce a path length from $\Theta(n^2)$ to $\Theta(n)$.

Many theoretical open problems remain, such as proving Conjecture 3. We also wonder if parallel flip distance problem is NP-hard in general and in convex position, as well as the unit flip distance problem in the convex case (the problem is NP-hard for general point sets [10]). We also do not know if the problem of calculating $b(s)$ (see Section 3.2) can be solved in polynomial time, possibly using dynamic programming.

References

1 Oswin Aichholzer, Brad Ballinger, Therese Biedl, Mirela Damian, Erik D Demaine, Matias Korman, Anna Lubiw, Jayson Lynch, Josef Tkadlec, and Yushi Uno. Reconfiguration of non-crossing spanning trees, 2022. `arXiv:2206.03879`.

2 Florent Avellaneda. A short description of the solver EvalMaxSAT. *MaxSAT Evaluation*, 8:364, 2020.

3 Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froleyks, and Florian Pollitt. CaDiCaL 2.0. In *Computer Aided Verification - 36th International Conference, CAV*, volume 14681 of *LNCS*, pages 133–152, 2024. `doi:10.1007/978-3-031-65627-9\_7`.

4 Ruy Fabila-Monroy, Clemens Huemer, and Dieter Mitsche. The number of empty four-gons in random point sets. *Electronic Notes in Discrete Mathematics*, 46:161–168, 2014. `doi:10.1016/j.endm.2014.08.022`.

5 Sándor P. Fekete, Phillip Keldenich, Dominik Krupke, and Stefan Schirra. Minimum coverage by convex polygons: The CG:SHOP challenge 2023, 2023. `arXiv:2303.07007`.

6 Jerôme Galtier, Ferran Hurtado, Marc Noy, Stéphane Pérennes, and Jorge Urrutia. Simultaneous edge flipping in triangulations. *International Journal of Computational Geometry & Applications*, 13(02):113–133, 2003.

7 Sabine Hanke, Thomas Ottmann, and Sven Schuierer. The edge-flipping distance of triangulations. *Journal of Universal Computer Science*, 2(8):570–579, 1996.

8 David E. Joslin and David P. Clements. Squeaky wheel optimization. *Journal of Artificial Intelligence Research*, 10:353–373, 1999.

9 Tanvir Kaykobad. An interlaced algorithm for transforming plane triangulations using simultaneous flips. *Journal of Computational Geometry*, 16(1):517–550, 2025.

10 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`.

11 Daniel D. Sleator and William P. Tarjan, Robert E.and Thurston. Rotation distance, triangulations, and hyperbolic geometry. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 122–135, 1986.

12 O. Tange. Gnu parallel - the command-line power tool. *;login: The USENIX Magazine*, 36(1):42–47, Feb 2011. URL: `http://www.gnu.org/s/parallel`.

13 Andy B Yoo, Morris A Jette, and Mark Grondona. Slurm: Simple linux utility for resource management. In *Workshop on job scheduling strategies for parallel processing*, pages 44–60, 2003.