Improving Curve Arrangements Through Local Changes

John Krijger
Master's Thesis
Game and Media Technology
Utrecht University
Supervised by Maarten Löffler and Frank Staals
ICA-5847001

July 2022

A curve arrangement can contain popular faces if a face is bounded the same curve multiple times. Popular faces and closed loops can make generated curved nonogram puzzles complex. To remove these features we make changes to small areas in the curve arrangement. We identify possible local changes in a curve arrangement and test several method that try to make the biggest improvement with the fewest changes.

1 Introduction

A curve arrangement is a subdivision of a 2D shape defined by a set of curves. These curves are contained within a border polygon and either form closed loops or start and end at the borders. These curves and the border create a planar graph: the vertices are the points where curves intersect each other or where they meet the border, the edges are segments of curves or the border between the vertices and the faces are the areas enclosed by curves and the border. Figure 1 is an example of a curve arrangement that contains four curves. One curves forms a closed loop and the other three start and end at the border. The planar graph of this curve arrangement contains 12 faces (within the border), 25 edges (including border edges) and 14 vertices (including those on the border).

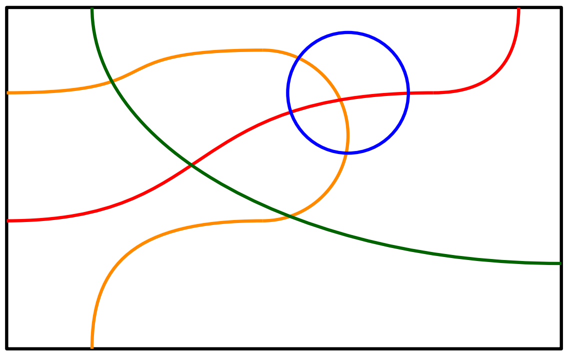

Figure 1: A simple curve arrangement consisting of four curves.

We will improve a curve arrangement by removing and reducing bad features, such as closed loops and popular faces. Popular faces are faces in the curve arrangement of which two or more edges incident to the face belong to the same curve [1], we will call a pair of such edges a popular edge pair. Figure 2 contains a popular face: edges ℓ_0 and ℓ_4 both belong to curve ℓ and are bordering face F, edges ℓ_0 and ell_4 form a popular edge pair over face F.

We will make these improvements to the curve arrangement with local changes. A local change is one or more crossings and uncrossings of curves within

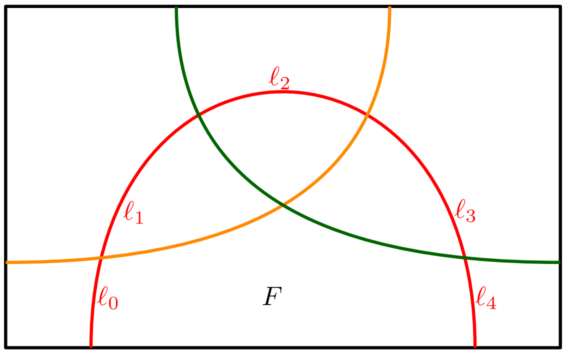

Figure 2: A simple curve arrangement with a popular edge pair. Edges ℓ_0 and ℓ_4 are both segments of curve ℓ and border face F.

a circle with a predefined local radius. If (un)crossings cannot be contained to the same circle, we need multiple local changes to achieve these (un)crossings. Figure 3 shows a curve arrangement with all bad features and a corresponding desirable result, the resulting curve arrangement has no bad features after four local changes. We will investigate an exact and two stochastic methods to try to find the optimal set of local changes to minimize closed loops and popular faces.

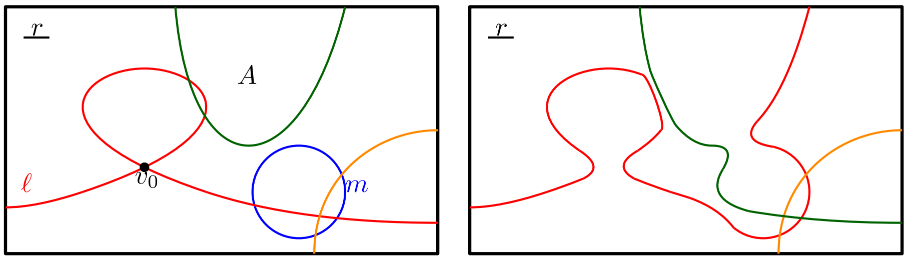

Figure 3: (left) An example of a curve arrangement with all forms of bad features. Curve ℓ is a looped curve, due to this loop, all faces adjacent to v_0 are popular faces. Face A is another popular face, independent from the looped curve. Lastly, curve m is a closed loop. (right) A solution of the same arrangement where four local changes have been applied, this resulting arrangement has no closed loops or popular faces.

2 Related Work

The problem of popular faces in curve arrangements has previously been encountered in research on curved nonograms.
Nonograms, also known as Japanese puzzles or paint-by-number puzzles, are a

type of puzzle where the puzzler has to color cells on a grid. Each row and column of the grid contains a hint: a set of numbers that indicate the sequences of cells that should be colored in that row or column. Van de Kerkhof et al. [2] introduced a variant named curved nonograms, where the puzzle uses a curve arrangement instead of a grid. Hints are given per side of a curve that originates from the border instead of per row or column. If the curved nonogram contains popular faces, these faces could be referenced multiple times by the same hint. Therefore De Nooijer et al. [1] categorize nonograms without popular faces as *basic* nonograms and nonograms with popular faces as *advanced* or *expert* nonograms and if popular faces can be removed from a curve arrangement, a advanced or expert nonogram can be converted into basic nonogram.
The method De Nooijer et al. used to improve curve arrangements is to insert extra curves. They have proven that finding one curve that removes all popular faces is NP-complete, by reducing the problem to finding a non-intersecting Eulerian cycle in a planar graph, which has been proven by Bent and Manber [3] to be NP-complete.
Constructs similar to curve arrangements are pseudoline arrangements and knots, though research in these fields focuses to much on their restrictions. However, we can make use of Reidemeister moves [4] to change curves without reconnecting them.

3 Curves

A curve can be defined by the trace left by a continuous function that maps a single parameter to two-dimensional space. A curve is similar to a line segment except that is allowed to bend.
The curves we allow can consist of one or more connected segments. The segment types we consider here are:

- Line segments
- Quadratic, cubic or higher degree Bézier curves
- Circular or (rotated) elliptical arcs

Figure 4 shows examples of a quadratic Bézier curve, a cubic Bézier curve and an elliptical arc. We only consider parametric curves because they can be trivially changed locally by cutting one segments into multiple segments and replacing one of those segments.

As we will see later, we will need to find intersections and near misses between curve segments to find locations where local changes have the most impact on a curve arrangement.
We can find exact intersections and distances between points, line segments and circular arcs. Therefore, we can find exact solutions for curve arrangements that only consist of line and circular arc segments. We must use approximations to find solutions for curve arrangements that include other segment types. One

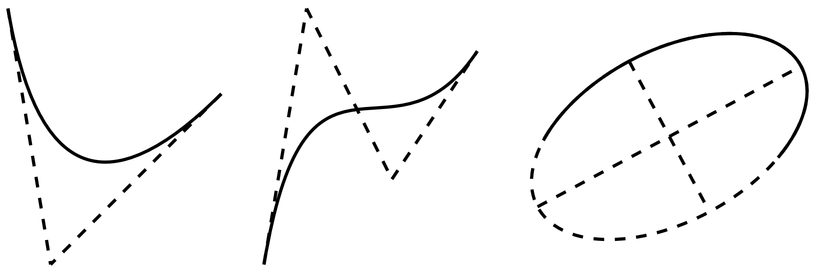

Figure 4: Three different segment types. For the Bézier curves, the dashed lines connect successive support points. For the elliptical arc the dashed lines show the supporting ellipse and its major and minor axes.

approach is to split all curves into xy-monotone curves without inflection points and create triangles that bound these segments to find intersections and points within twice the local radius between themselves. The vertices of the bounding triangle of an xy-monotone curve segment are start point of the segment, the end point of the segment and the intersection between the tangents to the curve segment at the two previous points.

3.1 Nontrivial Intersections between Curve Segments

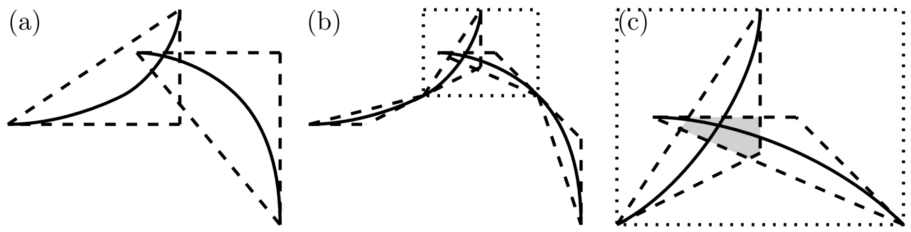

Figure 5: (a) The bounding triangles of the curve segments overlap, but an intersection is not guaranteed. (b) We split each segments in two subsegments and find their bounding triangles. (c) The bounding triangles of two subsegments overlap and this overlap splits the bounding triangles into four polygons that each contain one segment start or end. There is an intersection within the overlap.

To find intersections between curve sections, we check if their bounding triangles overlap. If the bounding triangles do not overlap, there is no intersection between their corresponding curve segments. If the boxes do overlap and an intersection is still uncertain or the point of intersection needs to be more precise, we split both curve segments into smaller segments and try to find intersections between the smaller segments. We can know for certain that there is an intersection if the overlap between the the bounding triangles separate both triangles into parts that contain only the start or end of the curve segment. Figure 5 shows the steps to find an intersection.

3.2 Nontrivial Near Misses between Curve Segments

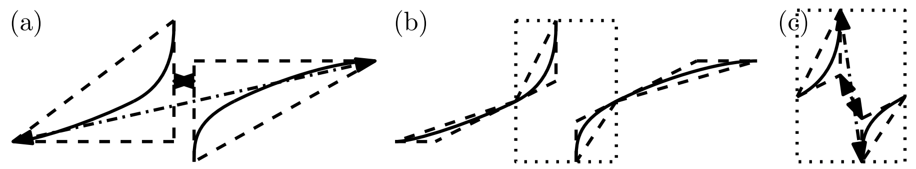

Figure 6: (a) The difference between the minimum and maximum distances between the bounding triangles of the segments is too large to be certain of a near miss within twice the local radius. (b) We split each segment into two subsegments and find their bounding triangles. (c) The maximum distance between the bounding triangles of these subsegments is now smaller than the maximum distance for a near miss, there is for certain a near miss between these subsegments.

To find near misses, we need to find curve segments that have a distance between each other of at most twice the local radius. To approximate the distance between two curve segments, we check the minimum and maximum distance between their bounding triangles. If the minimum distance between the bounding triangles is greater than the given twice the local radius, the distance between the curve segments is also greater than twice the local radius. If the maximum distance between the bounding triangles is lesser than twice the local radius, we know for certain that both segments are within twice the local radius to each other. Otherwise, if twice the local radius is between the minimum and maximum distance between the bounding triangles, we split the curve segments into smaller segments and try to find whether these smaller segments are within twice the local radius between each other. Figure 6 shows the steps to determine if two curve segments are within twice the local radius to each other.

4 Abstract Representations

We abstract the curve arrangement into a global arrangement and local arrangements. The global arrangement is a static planar subdivision wherein vertices represent local areas containing two or more curves and edges represent the parts of the curves that connect these local areas. Local arrangements are dynamic planar subdivisions wherein vertices represent intersections between curves or between a curve and the boundary of the local area. Figure 8 shows the global

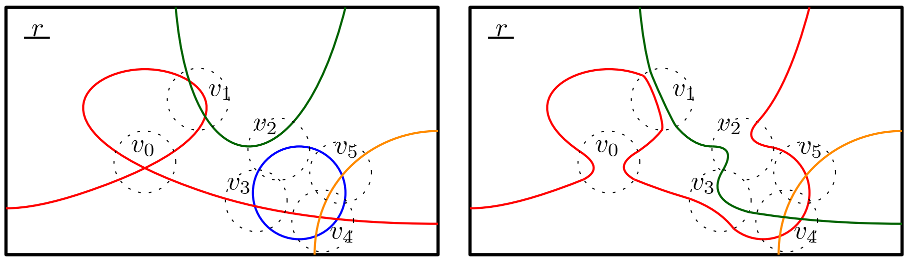

Figure 7: (left) An example of an input curve arrangement and the local areas where we allow changes. (right) A desired solution to the same arrangement where changes have been made in four local areas, this resulting arrangement has no closed loops or popular faces.

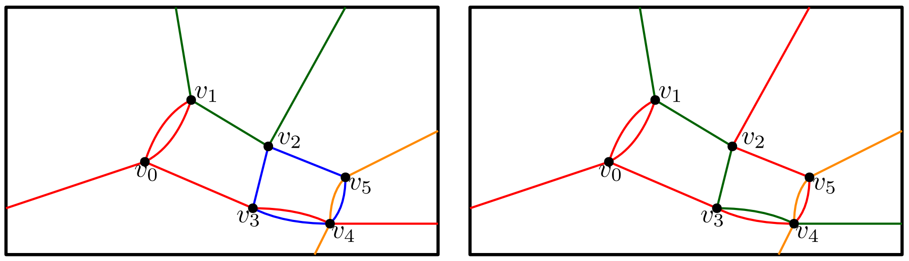

Figure 8: The global arrangements corresponding to the curve arrangements in Figure 7. The edges are colored according to their corresponding curve. (left) The input arrangement. (right) A desired solution. Only the curves that the edges belong to has changed in the global arrangement.

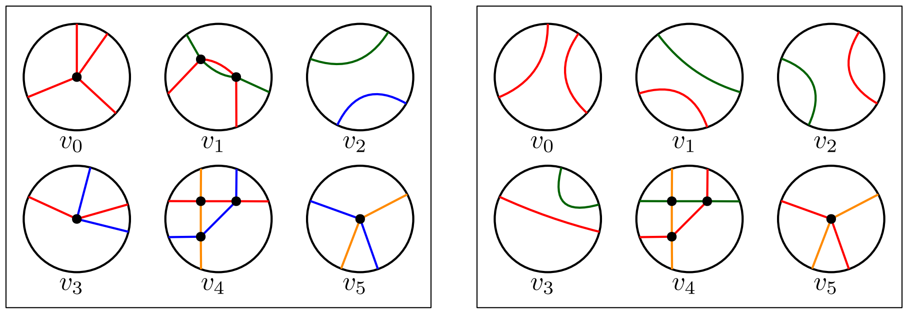

Figure 9: The local arrangements corresponding to the vertices in Figure 8. (left) The input arrangement. (right) A desired solution. Local changes have been made to vertices v_0, v_1, v_2 and v_3.

arrangements corresponding to the curve arrangements in Figure 7. Figure 9 shows the local arrangements corresponding to the vertices in Figure 8.

4.1 Global Arrangement

The global arrangement needs to store how local areas are connected to each other and to the boundary. Local areas can be stored as vertices in the global arrangement and their connections can be stored as edges. Local areas can have multiple connections between each other and their orientations are important; most importantly, faces need to be preserved. The global arrangement needs to be constructed only once thus reading this arrangement is more important than changing it.

4.2 Finding Intersections in the Curve Arrangement

Intersections are important candidates for local changes: changing an intersection to a near miss might improve or worsen the curcve arrangement. To find all intersections in the curve arrangement, we need to find all intersections between or within curve segments. Several papers [5, 6, 7, 8, 9] describe methods to create a curve arrangement from a set of curves, most of which use the Bentley-Ottmann algorithm [10]. If no exact solution for an intersection exist, we need can use the approximation method described before.

4.3 Finding Near Misses in the Curve Arrangement

Intersections are not the only locations where local changes can improve curve arrangements, two or more sections of curves that pass each other within twice the local radius but do not share an intersection are also local areas to consider. These near misses could be changed to intersections or near misses in perpendicular direction. We add these near misses as vertices in our global arrangement and create local arrangements for them.
To find pair-wise near misses, we consider only potential edge pairs from our current global arrangement, which are pairs of edges that share a face but not a vertex in the global arrangement. If a potential pair has multiple near misses, we only consider the nearest miss.
If three edges adjacent to the same face have distances a, b and c between each other, and the radius of a circle circumscribing a triangle with side lengths a, b and c is lesser than the local radius, we try to find a circle that includes parts of or is tangential to all three edges. If all three edges consist of only line segments, circular arcs and bend points, the problem of finding a tangential circle is one of the special cases of the Problem of Apollonius, which could be solved using inversion [11].

5 Local Changes

Local changes consist of one or more moves applied to part of the same local area. We define five different moves: uncrossing, crossing, R1, R2 and R3. Crossing and uncrossing, shown in Figure 10, are necessary to construct any local change, they turn two adjacent curves in to an intersection or vice versa respectively.

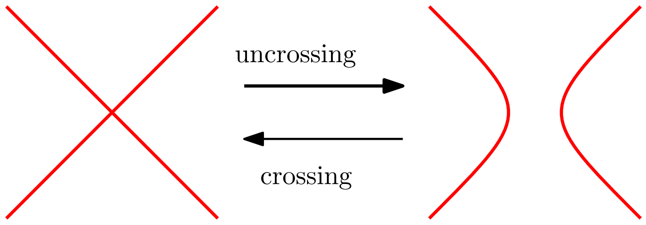

Figure 10: Crossing and uncrossing.

R1, R2 and R3 are the Reidemeister moves, these can be constructed using crossings and/or uncrossings, though we choose to allow Reidemeister moves as single moves, because these keep curves consistent and thus only affect the curve arrangement locally. Figures 11, 12 and 13 show the Reidemeister moves.

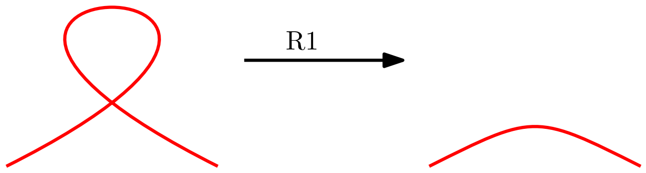

Figure 11: The first Reidemeister move, a loop is undone.

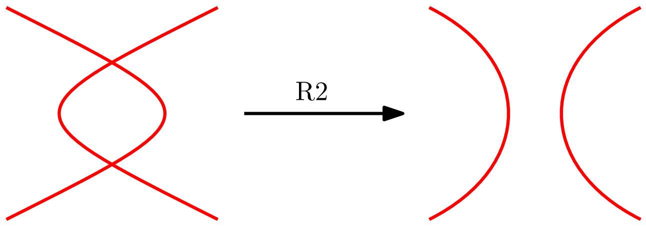

Figure 12: The second Reidemeister move, two overlapping curves are separated.

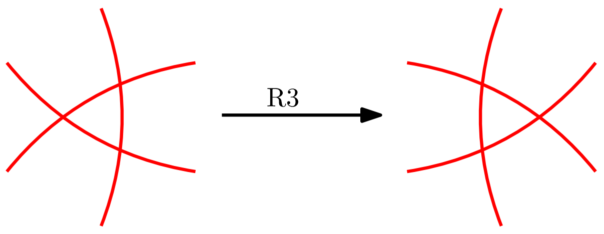

Figure 13: The third Reidemeister move, a curve is moved over the intersection of two other curves.

6 Improving the Curve Arrangement

We use three different methods to find improvements to the curve arrangements. These method try to reduce the badness of an arrangement, a linear combination of the number of popular edge pairs and number of closed loops, with changes to as few local areas as possible. The methods we use are Breadth First Search, Simulated Annealing and our own Heuristic Search.

6.1 Breadth First Search

Breadth First Search checks all variations on the curve arrangement with an increasing number of changed local areas. However, an arrangement with n local areas that have (at least) m possible local arrangements each has (at least) $\binom{n}{d} \times (m-1)^d$ variations that have d local areas changed. (We choose d out of n local areas to change, these areas have, at least, $m-1$ variations.) This means that for curve arrangements with many local areas, the number of variations grows almost exponentially with a low number of changes. Therefore, BFS requires many iterations to check all variations with more than a few changes in a large curve arrangement. For those large arrangements, we can use BFS mainly to find the optimal badness for variants with a limited number of changes.

6.2 Simulated Annealing

Simulated Annealing evaluates a single random change to the curve arrangement every iteration. By chance, dependant on a decreasing temperature parameter, we accept the change without evaluating the difference in badness this change makes. Otherwise we do evaluate the difference in badness and accept the change only if it decreases the badness or keeps the same badness and reverses a change. The process can be restarted after a given number of iterations from the best solution found previously. Simulated Annealing can find solutions that require many changes to the curve arrangement without evaluating the difference in badness of many of these changes, thus iterations can be fast. This method prioritizes minimizing the badness of an arrangement and its solutions may have more changed local areas than necessary. We can use this method manly to find a low upper bound to the optimal badness of changes to a curve arrangement.

6.3 Heuristic Search

Heuristic Search, like Breadth First Search, checks all variations at most once. Though the order in which variations are checked are dependant on a priority queue that is sorted by a heuristic function that takes the badness of a variation and the number of changed local areas from the initial curve arrangement. Initially, the only element in the queue is the original state of the curve arrangement. In each iteration we take and remove the first element of the priority queue. For each change that can be made to this state that we have not processed before, we add the resulting variation to the priority queue. Afterwards, we add this state to a set of all states we have processed. After a given number of iterations, this method returns the state with the best badness (and fewest changed local areas) that we have processed. The heuristic function must weigh reducing the badness up against keeping the number of changes low.

6.4 Applying Changes to the Curve Arrangement

For the resulting curve arrangement, we only have to change curves within changed local areas. We can cut these curves on the borders of the local areas and reconnect them based on how these local areas have changed. For local areas that contained only a crossing, we replace this crossing with two quadratic Bézier curves with endpoints connecting to the original curve and a support point at the original intersection.

7 Implementation

The program has been implemented in `C++` using CGAL [12] for geometry functionality and TinyXML-2 [13] to read and write arrangements from and to IPE [14] files. We limit our implementation to a local radius of zero, this means that we do not search for near misses in the input curve arrangement and each local area initially contains one intersection. If an intersection is changed to a near miss, the distance of the near miss is zero, but does not count as an intersection. Instead, when applying the changes we found we treat these changes as if the local radius is a small value greater than zero.

7.1 Algorithm Input

The program receives a curve arrangement and several parameters. The curve arrangement is read from an IPE file containing only lines, quadratic and cubic Bézier curves, ellipses and elliptical arcs. Figure 14 is an example of an input arrangement These segments can be color coded to restrict manipulation on it or its endpoints:

- Blue segments are the borders of the curve arrangement, its endpoints cannot be manipulated and these do not count towards popular edge pairs.

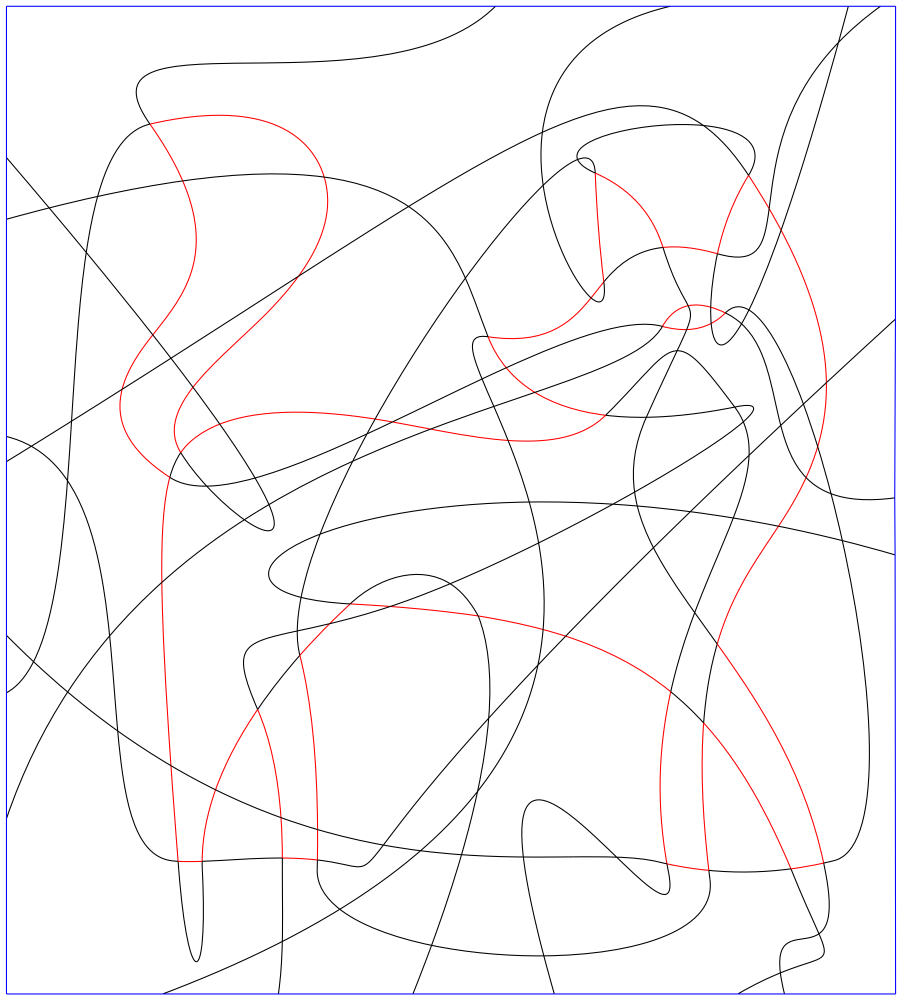
Figure 14: Arrangement *fox*, the border is formed by blue edges and the image is formed by red fixed edges. These fixed edges therefore have to stay connected.

- Red segments are fixed edges and are part of the original image, connected fixed segments in the input must remain consecutive edges along a face in the output arrangement.
- Purple segments are both part of the border of the curve arrangement and part of the original image, these are treated the same as border segments by the program.
- Black segments are free edges in the arrangement, a vertex connected to only free edges can be freely manipulated.

The arrangement and three other parameters form a scenario, these other parameters are:

- IFE, whether to ignore fixed edges: if this value is true, all fixed edges are treated as free edges.
- Open Area Ratio: a number between 0 and 1 (both inclusive) that indicates the chance that a local area will start as an open state instead of an intersection.
- Closed Loop Penalty: this value determines how much a closed loop counts towards the badness of a state compared to a popular edge pair.

Lastly, we use three parameters for the Heuristic Search method:

- Heuristic ratio: this value determines the heuristic function we use. If this value is a number r_h between (and including) zero and 1, we use the function

  $$H_r(\mathit{badness}, \#\mathit{changes}) = (1 - r_h) \times \mathit{badness} + r_h \times \#\mathit{changes}$$

  as heuristic function. A heuristic ratio of 0 or 1 means that the heuristic only consists of the badness or number of changes respectively. A heuristic ratio of 0.5 means that one change of the state counts equally to the heuristic as one popular edge pair. If this parameter is $-q$, we use the function

  $$H_q(\mathit{badness}, \#\mathit{changes}) = \frac{\mathit{badness} - \mathit{initial\ badness}}{\#\mathit{changes}}$$

  as heuristic function and call this the quotient heuristic. The quotient heuristic tries to optimize the decrease in badness per change.
- Maximum number of changes: this parameter simply limits the number of changes the algorithm is allowed to make to the initial state of the arrangement.
- Maximum number of iterations: this parameter limits the number of states the algorithm will check.

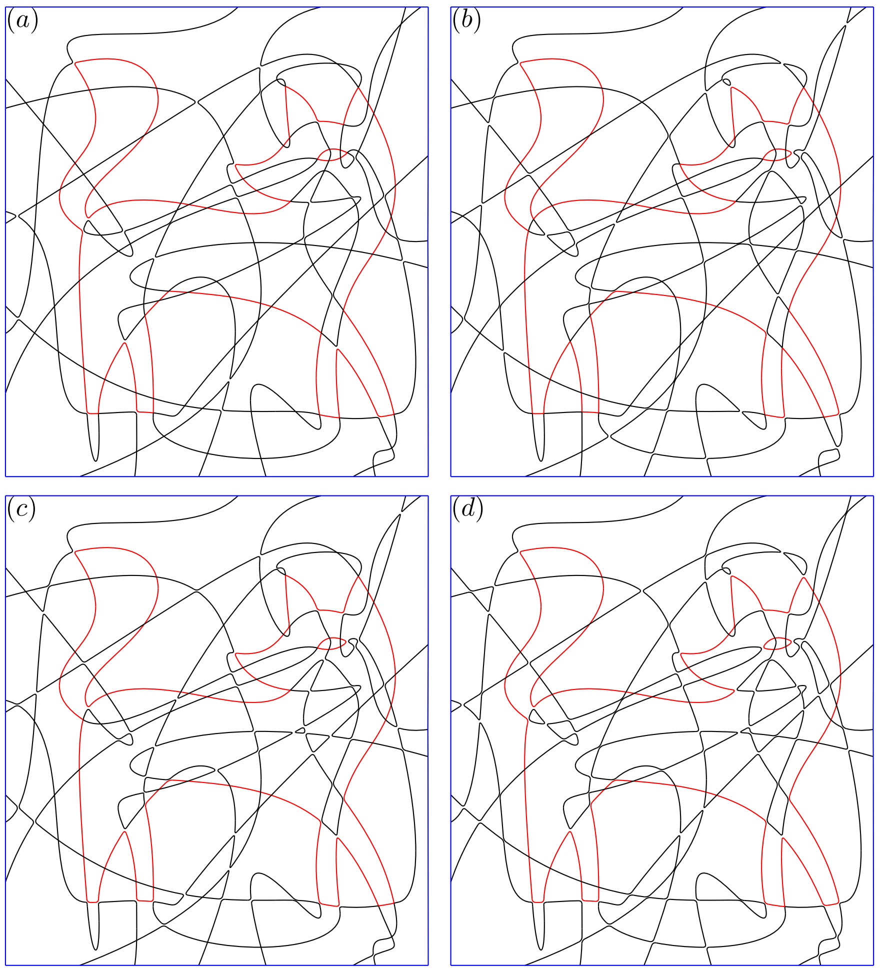

Figure 15: Four variants on arrangement *fox* with different open area ratios. The open area ratios are 0.25, 0.5, 0.75 and 1.0 for variants (a), (b), (c) and (d) respectively.

7.2 Algorithm Output

The program outputs an IPE file containing the output curve arrangement and a file containing data about set of subsolutions found during the run of the Heuristic Search. All subsolutions in this have a lower badness, fewer changes and/or were found sooner than other subsolutions. The output curve arrangement corresponds to the first state found with the lowest number of changes among the states with the lowest badness found. In addition to the badness, number of changes and the iteration at which the state was found, the data file contains the elapsed time between the start of the search algorithm and when the state was found, the number of each type of change made to the arrangement and the number of closed loops in the resulting arrangements.

7.3 Program Steps

The program can be divided in four phases: Reading the input, constructing the global and local arrangements and finding solutions and writing output.

7.3.1 Reading Input

During this face, we construct an arrangement in the form of a doubly connected edge list containing cubic Bézier curves from the input arrangement file. We restrict our program to a local radius of 0, therefore we only look for intersections in the input and ignore near misses. We use TinyXML-2 to read all segments from the input IPE file. CGAL uses the Bentley-Ottmann sweep line algorithm [10] to find all intersections between the line segments and Bézier curves. This algorithm takes time $\mathcal{O}((n + k) \log n)$ for arrangements whose curves can be split in n x-monotone curves and that have k intersection. For the arrangement, CGAL constructs a DCEL (doubly connected edge list) which we can use for the second phase of the program. We split ellipses and elliptical arcs into xy-monotone arcs and then approximate them by xy-monotone cubic Bézier curves, because this was easier than getting CGAL to allow Bézier curves and elliptical arcs in the same arrangement.

7.3.2 Constructing Global and Local Arrangements

We use the DCEL constructed in the first phase to construct the global arrangement. The global arrangement is a planar graph where its vertices represent local areas, areas at which we can change the local arrangement, and its edges represent global edges, collections of curve segments that connect local areas. In this implementation, local areas contain one intersection or near miss or are marked as a border area. Local areas not part of the border can have at most three possible local arrangements: crossed or either of the uncrossed configurations, these three local arrangements are depicted in Figure 16. Fixed edges can restrict local areas to fewer local arrangements. Local areas and global edges can be part of the border of the arrangement, border areas cannot change their

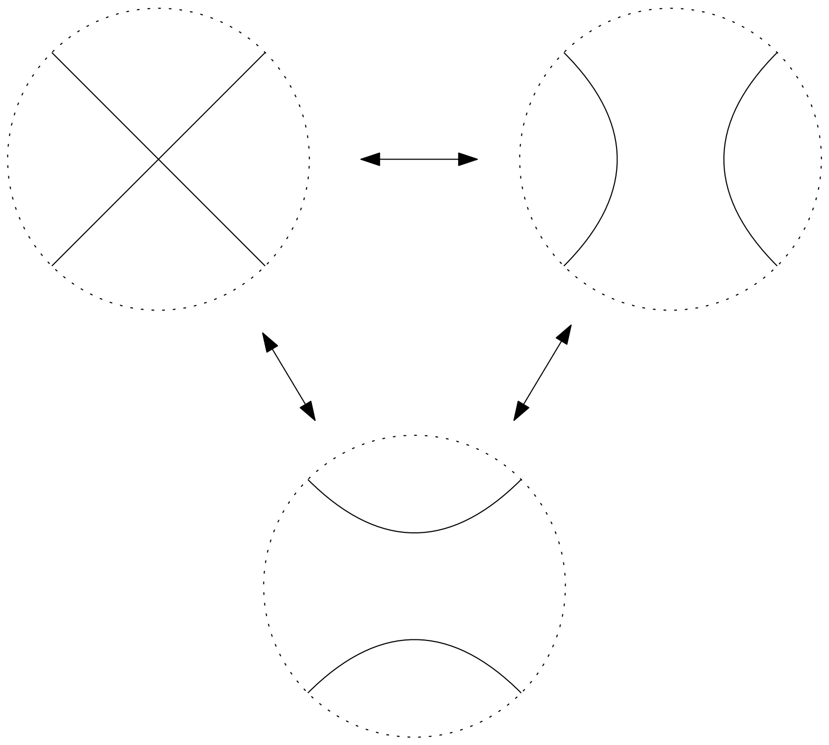
Figure 16: The three local arrangements we allow in our implementation.

local arrangement and border edges do not count towards the number of popular edge pairs and the number of closed loops in the global arrangement.
We also construct a state object that contains the local arrangement of each local area, which global edges belong to the same curve, the number of popular edge pairs in the global arrangement and the number of closed loops in the global arrangement. Each state can provide its badness, given a closed loop penalty, the number of differences between it and another state, a random neighbor or all its neighbors, though calculating this takes time $\mathcal{O}(n \log n)$ with n as the number of edges in the global arrangement.
A neighbor of a state is another state where one local area has a different local arrangement.

7.3.3 Finding Solutions

We use the three different methods described earlier to find solutions. We restart Simulated Annealing 100 times and iterate 100,000 times per restart. The chance of taking any change decreases linearly from one to zero per iteration. BFS is limited to two million iterations. For the Heuristic Search we us a priority queue implemented using a multimap that maps a heuristic value to states that have that value. The multimap keeps itself sorted on the heuristic, which allows us to efficiently get a state with the lowest heuristic to process and it allows us to remove states with the highest heuristic if the size of the queue exceeds the number of iterations the search has left.

7.3.4 Writing Output

Lastly, we apply the changes in the solutions to the curve arrangement by cutting the edges a small distance from changed intersections and replacing them with two quadratic Bézier curves. We then write the resulting to a new IPE file. We write the gathered data to an csv file.

8 Experiments

We will preform two sets of experiments: one where we vary many parameters on few arrangements and one where we vary few parameters on many arrangements. The tested arrangements and parameters per set of experiments are detailed in Table 1. We test all combinations of parameters, except when an arrangement has no fixed edges, in which case we only test cases where we ignore fixed edges.

Parameter	Set 1	Set 2
Arrangement	fox, snowman	beaver, butterfly, elephant, fox, snowman, tree, drop, eye, fish, lens, para, ring, wave, test0, test1, test2, test3, test4, test6
Ignore Fixed Edges (IFE)	true, false	true, false
Random Open Area Ratio	0, 0.25, 0.5, 0.75, 1	0
Re-rolls Per Ratio	2	1
Closed Loop Penalty	0, 1, 3, 5	3
Max Number Of Changes	5, 20, ∞	∞
Max Number Of Iterations	2000	40000
Heuristic Ratio	0.001, 0.2, 0.5, 0.8, 0.999, -q	0.001, 0.2, 0.5, 0.8, 0.999, -q
Number Of Trials Per Set Of Parameters	5	3

Table 1: The possible values for each parameter per experiment set.

Table 2 shows the size and source of each curve arrangement used. These arrangements were provided by my thesis supervisor and have been used for the research by Van de Kerkhof et al. [2] and De Nooijer et al. [1] or have been created for Master's Theses that are still in progress. The curve arrangements can be found in Appendix A.

Arrangement	#Edges	#Border Edges	#Fixed Edges	Size Category
fox	278	20	63	Large
tree	175	22	30	Large
elephant	135	22	40	Large
snowman	133	14	29	Large
butterfly	116	16	35	Large
beaver	53	10	14	Medium
ring	42	12	12	Medium
eye	32	8	8	Medium
test6	30	12	0	Medium
test4	29	10	0	Medium
fish	28	8	9	Medium
test1	19	6	4	Small
lens	17	6	4	Small
test2	17	6	5	Small
test0	15	6	2	Small
test3	15	6	0	Small
drop	12	4	3	Small
para	11	6	2	Small
wave	8	4	2	Small

Table 2: The size and source of each curve arrangement used. These arrangements were provided by my thesis supervisor and have been used for the research by Van de Kerkhof et al. [2] and De Nooijer et al. [1] or have been created for Master's Theses that are still in progress.

A scenario is the combination of an arrangement, whether to ignore fixed edges, an open area re-roll and a closed loop penalty. For each scenario, we record the best solution BFS found per number of changes and the solution found by Simulated Annealing. Furthermore, we record the solution found by each trial of the Heuristic Search method. The best badness for a scenario is the lowest badness found for that scenario among all trials. The best number of changes for a scenario is the lowest number of changes that resulted in the best badness found among all trials.
Furthermore, we store additional data for the Heuristic Search method: if an iteration finds a solution with either a better badness or an equal badness but with fewer changes, we store the parameters for that trial, the badness, number of changes and iteration number of that solution.
All experiments are run using an AMD Ryzen 7 3700X processor.

9 Results

9.1 Best results per method

Tables 3 and 4 show the best results found by each method for each scenario in experiment set 1 and 2 respectively.

Scenario			Initial Badness	BFS		Simulated Annealing		Heuristic Search		Best	
Arrangement	Ignore Fixed Edges	Closed Loop Penalty		Badness	#Changes	Badness	#Changes	Badness	#Changes	Badness	#Changes
fox	True	0	59	31	3	**0**	51	**0**	**16**	0	16
		1	59	32	3	**0**	43	**0**	**18**	0	18
		3	59	32	3	**0**	40	**0**	**18**	0	18
		5	59	32	3	**0**	43	**0**	**18**	0	18
	False	0	59	25	4	**1**	**36**	4	16	1	36
		1	59	26	4	**1**	**32**	4	16	1	32
		3	59	27	4	**1**	27	**1**	22	1	19
		5	59	27	4	**1**	**25**	4	16	1	25
snowman	True	0	32	4	5	**0**	18	**0**	**11**	0	11
		1	32	5	5	**0**	16	**0**	**11**	0	11
		3	32	7	5	**0**	16	**0**	**11**	0	11
		5	32	8	5	**0**	17	**0**	**11**	0	11
	False	0	32	4	5	**0**	14	**0**	**10**	0	10
		1	32	5	5	**0**	**17**	1	10	0	17
		3	32	7	5	**0**	**17**	1	15	0	17
		5	32	9	5	**0**	**18**	1	15	0	18

Table 3: The best solution found by each method for each scenario with an open area ratio of 0.0 in experiment set 1. The best solution is the best solution for a scenario by these methods in both experiment sets. Values matching the best solution are made bold. If no solution in this table matches the best solution, the best solution is found by experiment set 2.

For experiment set 1, BFS was unable to find the optimal solution for any scenario in its given iterations. BFS was able to make up to 3 to 5 changes which was insufficient for finding the optimal badness. Simulated Annealing found the best badness for all scenarios in experiment set 1 whereas Heuristic Search found best solutions for scenarios without fixed edges and for some with fixed edges. Simulated Annealing only found the best number of changes for scenarios where Heuristic Search did not find the best badness and for scenarios where Heuristic Search did find the best badness, it did so with significantly

Scenario			Initial Badness	BFS		Simulated Annealing		Heuristic Search		Best	
Arran-gement	Ignore Fixed Edges	Closed Loop Penalty		Badness	#Changes	Badness	#Changes	Badness	#Changes	Badness	#Changes
fox	True	3	59	32	3	**0**	40	**0**	**18**	0	18
	False	3	59	27	4	**1**	27	**1**	**19**	1	19
tree	True	3	7	**0**	**4**	**0**	**4**	**0**	**4**	0	4
	False	3	7	2	2	**0**	5	**0**	**4**	0	4
ele-	True	3	31	13	4	**0**	**11**	**0**	**11**	0	11
phant	False	3	31	9	6	**2**	**11**	**2**	**11**	2	11
snow-	True	3	32	7	5	**0**	16	**0**	**11**	0	11
man	False	3	32	7	5	**0**	**17**	**0**	**17**	0	17
butter-	True	3	30	17	4	**0**	**13**	**0**	**13**	0	13
fly	False	3	30	25	3	**23**	**10**	**23**	12	23	10
bea-	True	3	30	1	6	**0**	**7**	**0**	**7**	0	7
ver	False	3	30	**4**	**5**	-	-	**4**	**5**	4	5
ring	Either	3	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0	0
eye	True	3	4	**0**	**1**	**0**	**1**	**0**	**1**	0	1
	False	3	4	**3**	**2**	-	-	**3**	**2**	3	2
test6	True	3	2	**0**	**1**	-	-	**0**	**1**	0	1
test4	True	3	6	**0**	**3**	**0**	**3**	**0**	**3**	0	3
fish	True	3	4	**0**	**1**	**0**	**1**	**0**	**1**	0	1
	False	3	4	**3**	**2**	**-**	**-**	**3**	**2**	3	2
test1	Either	3	1	**0**	**1**	**0**	**1**	**0**	**1**	0	1
lens	Either	3	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0	0
test2	True	3	**3**	**0**	**1**	**0**	**1**	**0**	**1**	0	1
	False	3	3	**3**	**0**	**-**	**-**	**3**	**0**	3	0
test0	True	3	2	**0**	**1**	**0**	**1**	**0**	**1**	0	1
	False	3	2	**2**	**0**	**-**	**-**	**2**	**0**	2	0
test3	True	3	2	**0**	**1**	**0**	**1**	**0**	**1**	0	1
drop	True	3	5	**0**	**2**	**0**	**2**	**0**	**2**	0	2
	False	3	**5**	**5**	**0**	**-**	**-**	**5**	**0**	5	0
para	Either	3	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0	0
wave	Either	3	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0	0

Table 4: The best solution found by each method for each scenario in experiment set 2. Scenarios where fixed edges did not impact the best solutions found are combined. If BFS for certain found the optimal solution (due to exhausting all states), Simulated Annealing was sometimes skipped. The best solution is the best solution for a scenario by these methods in both experiment sets. Values matching the best solution are made bold.

fewer changes.

For experiment set 2, BFS was able to find the optimal solution for scenarios that were small enough to be exhausted or for a scenario where a solution with a badness of 0 could be found within a few changes. For all scenarios where BFS found the optimal solution, Simulated Annealing (if not skipped) and Heuristic Search did so as well. For the ten scenarios where BFS did not find the optimal solution, both Simulated Annealing and Heuristic Search found the best badness. Out of these ten scenarios, Simulated Annealing found the best number of changes for six scenarios and Heuristic Search found the best number of changes in nine scenarios.

9.2 Subsolutions in Heuristic Search

A subsolution is any state visited by our search methods that has a lower badness, fewer changes and/or was found in an earlier iteration than all other subsolutions. Figure 17 shows the subsolutions found by BFS and the first trial of Heuristic Search per Heuristic Ratio in experiment set 2 on one scenario. The scenario used is arrangement *fox* with fixed edges, no initial open areas and a closed loop penalty of 3. The subsolutions of Heuristic Search are labelled with the iteration at which they were found.

The trials with heuristic ratios 0.001 and 0.2 find the best badness after around 6000 and 10000 iterations respectively. Both trials find mostly the same subsolutions with suboptimal badness but neither finds the optimal badness for four changes. The trial with heuristic ratio 0.001 has enough iterations left after finding the best badness to find the best number of changes after around 10000 iterations. The trial with heuristic ratio 0.5 finds many subsolutions with a badness around 16 and 4, but no solution with a lower badness than 4 and no subsolutions after 844 iterations. This trial also does not find the optimal badness for four changes. The trial with heuristic ratio 0.8 finds no subsolutions with a badness lower than 12, but finds subsolutions with better or equal badness than trials with a lower heuristic ratio with ten or fewer changes. The trial with a heuristic ratio of 0.999 finds the same subsolutions as BFS, though no subsolution with more than three changes. The trial with a quotient heuristic finds subsolutions with up to eight changes of which all subsolutions of BFS are matched. This trial does not find more than one subsolution per number of changes, meaning that this trial cannot find better subsolutions than the first subsolution found for a number of changes. Overall, it seems that subsolutions with lower badness than previous subsolutions require an exponentially growing number of iterations, especially for trials with a high heuristic ratio or quotient heuristic. The badness of subsolutions seem to decrease exponentially with the number of changes made.

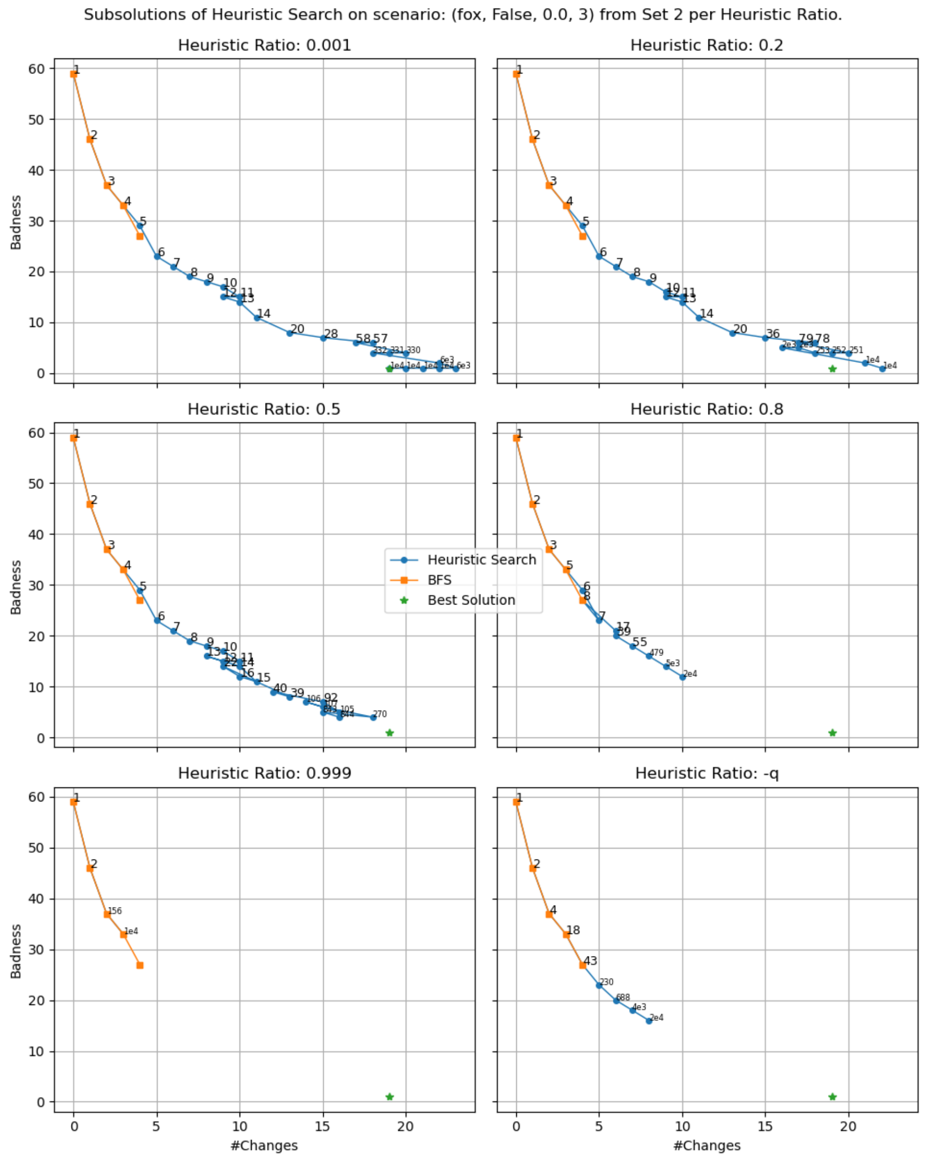

Figure 17: The badness and number of changes of subsolutions of Heuristic Search trials with differing Heuristic Ratios on the same scenario are plotted in blue and labelled with the iteration at which the subsolution was found. The optimal badness per number of changes for this scenario found by BFS is plotted in orange. The best solution found by all methods for this scenario is plotted with a green star.

9.3 Trial Times of Heuristic Search per Arrangement

Figure 18 summarized the time required per trial of Heuristic Search using box plots per arrangement per experiment set.

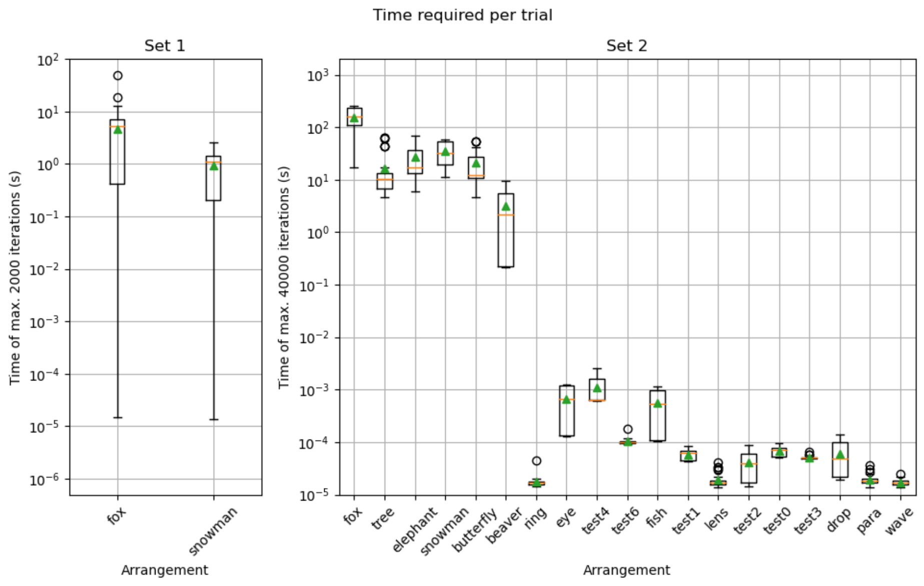

Figure 18: Box plots representing the time required per trial for each arrangement per experiment set, ordered by the number of edges of the arrangement in decreasing order. The means are represented by a green triangle. The vertical axis are in logarithmic scale and the plots for experiment set 1 and 2 are scaled to accommodate the difference in maximum number of iterations. If trials in different sets would require their maximum number of iterations and require the same time per iteration, their vertical position would be the same in both plots.

The arrangements of experiment set 1 have more variation in the times per trial than those of experiment set 2, most likely due to a greater number of and variety in trials per arrangement. The minimum time required for the trials on the arrangements of set 1 and some arrangements of set 2 is about 1.4×10^{-5} seconds. Generally, larger arrangements require more time per trial, requiring both more time per iteration and most often more iterations. Interestingly, the times required for the arrangement *beaver* in experiment set 2 matches the times required for the arrangements categorized as large than the other arrangements in the medium size category, this might be due to its high number of popular edge pairs in the initial arrangement (19 compared to 6, the number of initial edge pairs of *test4*, the medium sized arrangement with the second highest number of initial popular edge pairs).

9.4 Results of Heuristic Search

Appendix B contains tables summarizing the results of the Heuristic Search for experiment set 1 and 2. Figures 19 and 20 show the percentage of Heuristic Search trials that found the best badness and/or best number of changes within a number of elapsed iterations for set 1 and 2 respectively. The Heuristic Search found relatively more solutions for experiment set 1 than for experiment set 2. This is not surprising, since set 1 only contains some of the largest arrangements from set 2 and trials in set 2 were allowed twenty times as many iterations than those in set 1. The curves for best badness and for best badness and best number of changes generally follow the same shape, though not requiring the best number of changes leads to more solutions for most elapsed iterations.

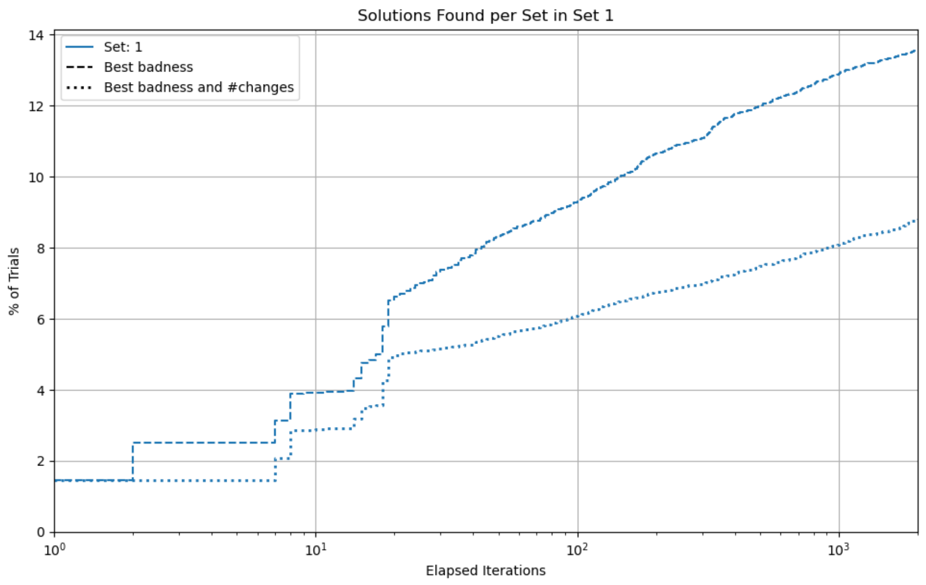

Figure 19: The percentage of trials in experiment set 1 that found the best badness and/or best number of changes within a number of elapsed iterations.

In the following subsections, we will vary one parameter and observe how its values affect the results.

9.4.1 Influence of Arrangement Size

For experiment set 1, we used two different arrangements, these are *fox* and *snowman* with 278 and 133 edges in their global arrangement respectively. Figure 21 shows the percentage of trials for each arrangement that found the best badness and best number of changes within a number of iterations in experiment set 1.

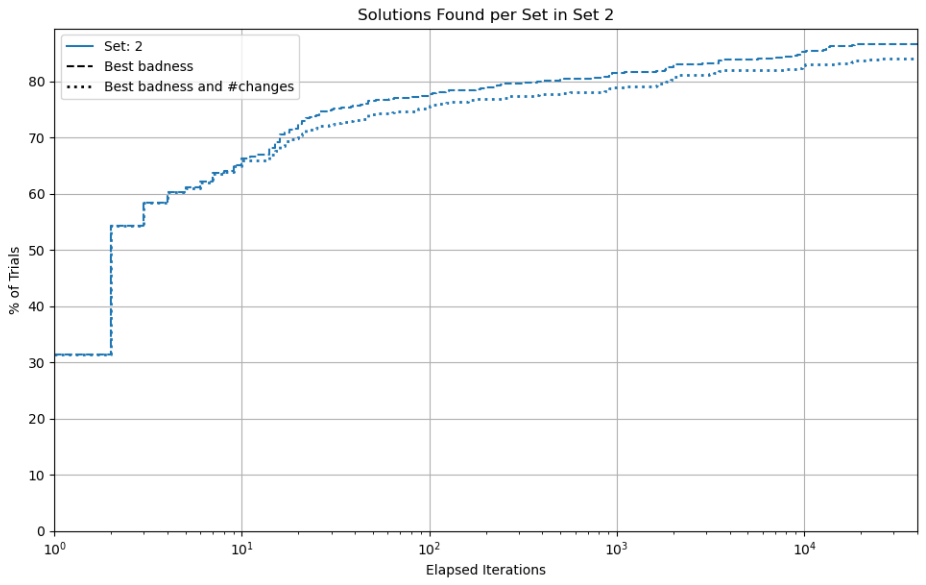

Figure 20: The percentage of trials in experiment set 2 that found the best badness and/or best number of changes within a number of elapsed iterations.

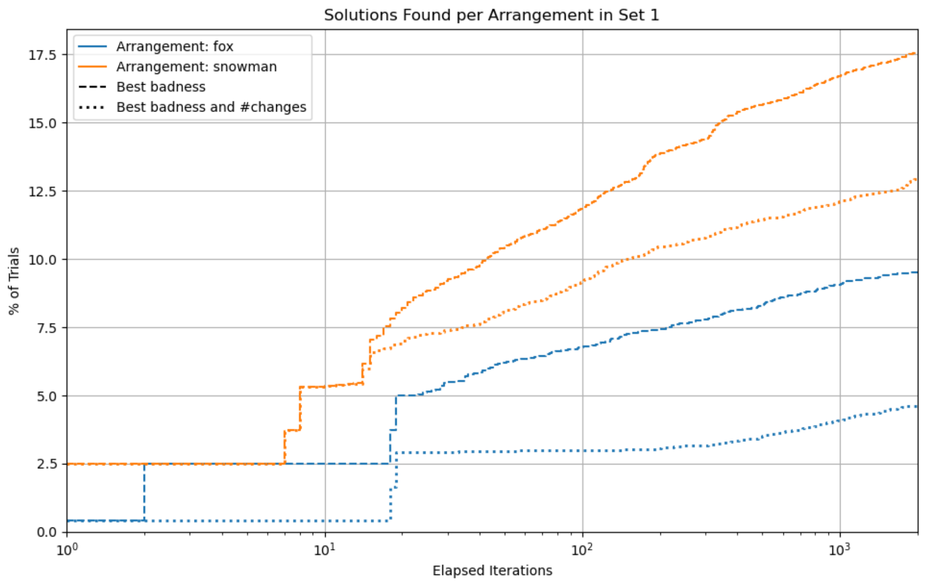

Figure 21: The percentage of trials per arrangement in experiment set 1 that found the best badness and/or best number of changes within a number of elapsed iterations.

For experiment set 2, we used 19 different arrangements, we pooled these into three different size categories: large, medium and small. Large arrangements contain at least 100 edges, the arrangement in this category are: *butterfly*, *elephant*, *fox*, *snowman* and *tree*. Medium arrangements contain between 25 and 100 edges, the arrangements in this category are: *beaver*, *eye*, *ring*, *test4* and *test6*. Small arrangements contain fewer than 25 edges, the arrangements in this category are: *drop*, *lens*, *para*, *test0*, *test1*, *test2*, *test3* and *wave*. Figure 22 shows the percentage of trials for each size category that found the best badness and best number of changes within a number of iterations in experiment set 2.

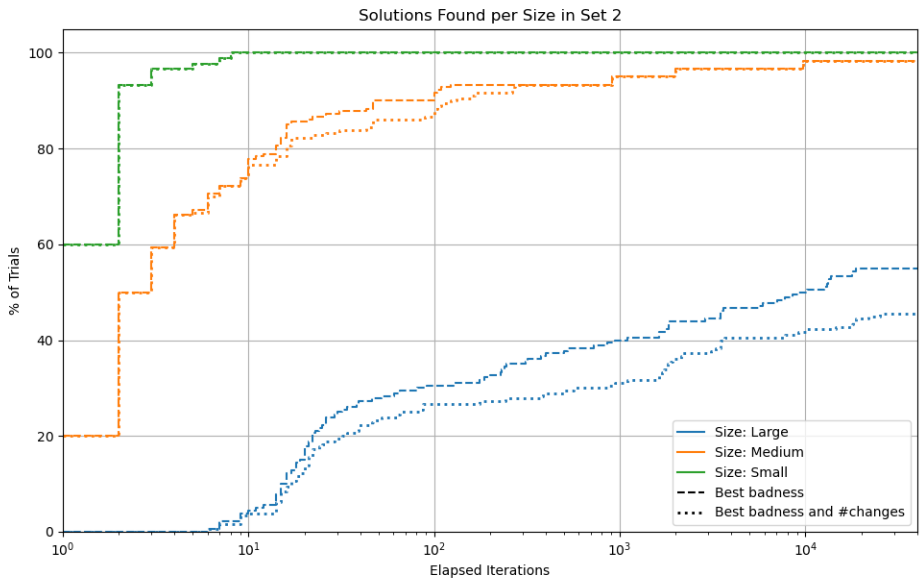

Figure 22: The percentage of trials per arrangement in experiment set 1 that found the best badness and/or best number of changes within a number of elapsed iterations.

The Heuristic Search finds more solutions per maximum number of elapsed iterations for smaller arrangements (for set 1) or size categories (for set 2) than larger ones. This was the expected result, because smaller arrangements have exponentially fewer states to iterate through. And with fewer states, a solution is generally encountered sooner.
The graphs for number of solutions that found both the best badness and best number of changes have similar shapes to those that only found the best badness.

9.4.2 Influence of Fixed Edges

Figures 23 and 24 show the percentage of trials for which fixed edges are either ignores or enforced that found the best badness and best number of changes

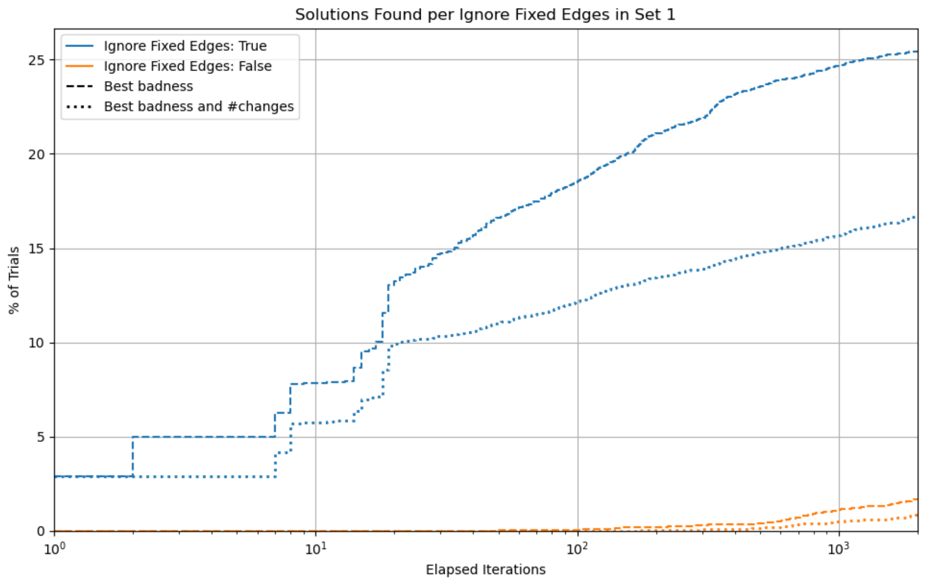

Figure 23: The percentage of trials in experiment set 1 that found the best badness and/or best number of changes within a number of elapsed iterations for scenarios where fixed edges are either ignored or enforced.

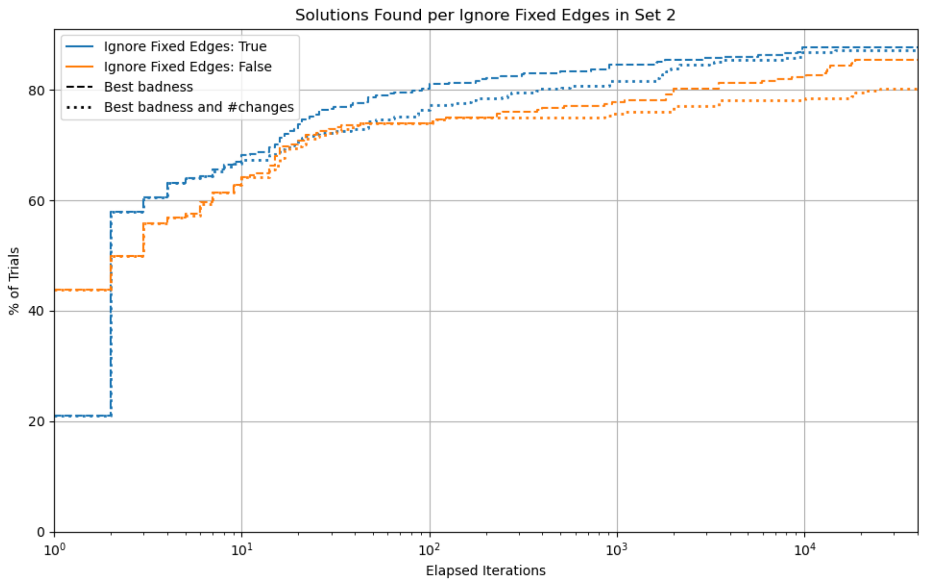

Figure 24: The percentage of trials in experiment set 2 that found the best badness and/or best number of changes within a number of elapsed iterations for scenarios where fixed edges are either ignored or enforced.

within a number of iterations in experiment set 1 and 2 respectively.

In experiment set 1, the Heuristic Search found significantly more solutions when fixed edges could be ignored than when they were enforced. In fact, with fixed edges, only 1.69% of trials in set 1 found the best badness. However in experiment set 2, this difference is drastically smaller. There may be at least two reasons for this. The first reason may be that set 1 has scenarios that are significantly easier without fixed edges, such as when the open area ratio is high (there are few intersections, areas with fixed edges may remain intersections) and the closed loop penalty is 0 (only intersections can cause badness). The second may be that ignoring fixed edges allow more possible states of the arrangement, which may have a bigger influence on the smaller arrangements in set 2.
The graphs for number of solutions that found both the best badness and best number of changes have similar shapes to those that only found the best badness.

9.4.3 Influence of Open Areas in the Initial Arrangement

The open area ratio was only varied in experiment set 1, it created variants where each intersection has a chance to start as near miss. Figure 25 shows the percentage of trials for each open area ratio that found the best badness and best number of changes within a number of iterations in experiment set 1.

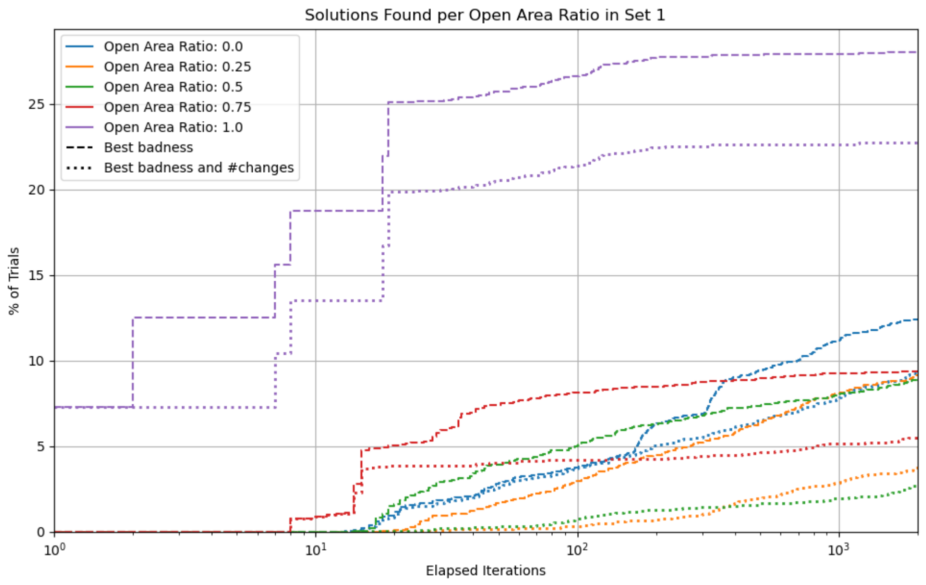

Figure 25: The percentage of trials per open area ratio in experiment set 1 that found the best badness and/or best number of changes within a number of elapsed iterations.

Scenarios with an open area ratio of 1.0 have significantly more solutions than those with fewer open areas. These scenarios can only have initial intersections if there are fixed edges, meaning that the initial badness due to popular edge pairs will be low. A low initial badness allows the Heuristic Search to skip many states that have an equal or higher badness, probably resulting in fewer required iterations per scenario. Similarly, the Heuristic Search finds the second most solutions with the best badness in scenarios with an open area ratio of 0.75, though only up to around 370 iterations, after which the benefit on an initially lower badness drop off and scenarios with an open area ratio of 0.0 gain more solutions. The same thing happens to scenarios with an open area ratio of 0.5, though these have fewer solutions than those with a ratio of 0.75 and it takes fewer iterations before scenarios with an open area ratio of 0.0 gains more solutions. Scenarios with an open area ratio on 0.25 have the fewest solutions with the best badness, though these have found about as many solutions as those with ratios of 0.5 and 0.75 after 2000 iterations.
The graphs for number of solutions that found both the best badness and best number of changes have similar shapes to those that only found the best badness.

9.4.4 Influence or Closed Loop Penalties

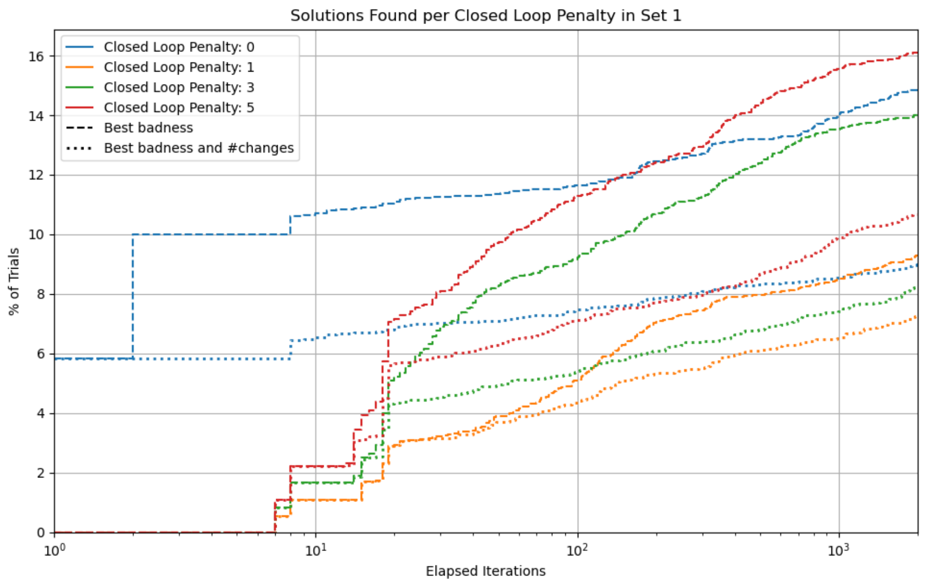

Figure 26: The percentage of trials per closed loop penalty in experiment set 1 that found the best badness and/or best number of changes within a number of elapsed iterations.

The closed loop penalty was only varied in experiment set 1, it determines how much a closed loop counts towards the badness compared to a popular edge pair. Figure 26 shows the percentage of trials for each closed loop penalty that found the best badness and best number of changes within a number of iterations in experiment set 1.

Scenarios with a closed loop penalty of 0 have the most solutions for about the first 135 iterations, though scenarios with a closed loop penalty of 5 gain more solutions after about 270 iterations. Scenarios with no or few initial intersections have solutions that require few iterations if closed loops are not penalized. Scenarios with a closed loop penalty of 0 might have a lower best badness than those with a higher closed loop penalty, because arrangements with the optimal badness may contain closed loops. This lower best badness may be harder to find in scenarios with many initial intersections. Scenarios with a closed loop penalty of 1 have the fewest solutions for all iterations, the Heuristic Search might have difficulty in prioritizing removing closed loops that need to be removed over removing popular edge faces.
The graphs for number of solutions that found both the best badness and best number of changes have similar shapes to those that only found the best badness.

9.4.5 Influence of Limits on the Number of Changes

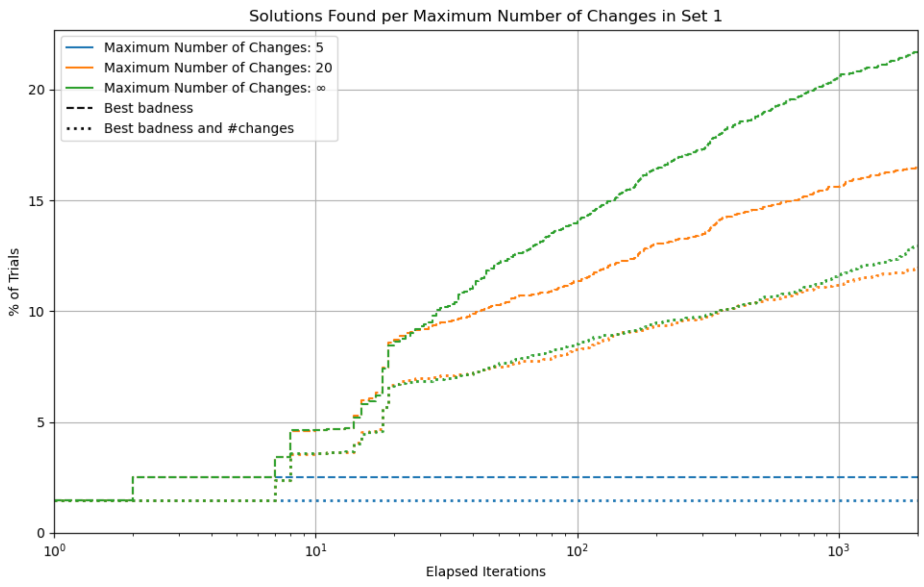

Figure 27: The percentage of trials per maximum number of changes in experiment set 1 that found the best badness and/or best number of changes within a number of elapsed iterations.

The maximum number of changes was only varied in experiment set 1. Figure 27 shows the percentage of trials for each closed loop penalty that found the best badness and best number of changes within a number of iterations in experiment set 1.

Trials that are only allowed fewer than 5 changes from the initial state match the number of solutions found of the other trials for the first seven iterations, however, these do not find any more solutions after the second iteration. Most solutions require at least 5 changes, which cannot be found with a maximum number of changes of 5. Trials that are only allowed fewer than 20 changes from the initial state find about the same number of solutions as trials without a limit on the number of changes for more iterations, though trials without a maximum number of changes gain more solutions with the best badness than those with a maximum of 20 after around 24 iterations. Similarly, trials without a maximum number of changes gain more solutions with the best badness and number of changes than those with a maximum of 20 after around 500 iterations.

9.4.6 Influence of Heuristic Ratio

We varied the heuristic ratio on both experiment sets. Figures 28 and 29 show the percentage of trials for each heuristic ratio that found the best badness and best number of changes within a number of iterations in experiment set 1 and 2 respectively.

Trials with a heuristic ratio of 0.999 find the fewest solutions for all elapsed iterations for both experiment sets. The Heuristic Search with this ratio strongly prioritizes keeping the number of changes low over lowering the badness, thus processing states by an increasing number of changes, similar to BFS. However, this means that trials with a heuristic ratio of 0.999 require many iterations to find solutions for large scenarios and scenarios that require many changes to achieve the best badness. Trials with a heuristic ratio of 0.8 find the second fewest solutions for all iterations for experiment set 2 and for most iterations for experiment set 1, only finding more or as many solutions as trials with a quotient heuristic in about the first 115 iterations. Interestingly all trials with a quotient heuristic in experiment set 1 that find the best badness also find the best number of changes in the same iteration, this also holds for most iterations in experiment set 2. Trials with heuristic ratios of 0.001, 0.2 or 0.5 find the most solutions for any number of elapsed iterations for both experiment sets. In experiment set 1, trials with heuristic ratios of 0.001 or 0.2 find about as many solutions with the best badness as each other and more than those with a heuristic ratio of 0.5 for each iteration. This is also the case for solutions with both the best badness and best number of changes for about the first 150 iterations, after which trials with a heuristic ratio of 0.5 gain more solutions with the best number of changes than trials with a lower heuristic ratio. In experiment set 2, trials with heuristic ratios of 0.001 or 0.2 also find about as many solutions with the best badness as each other and more than those with

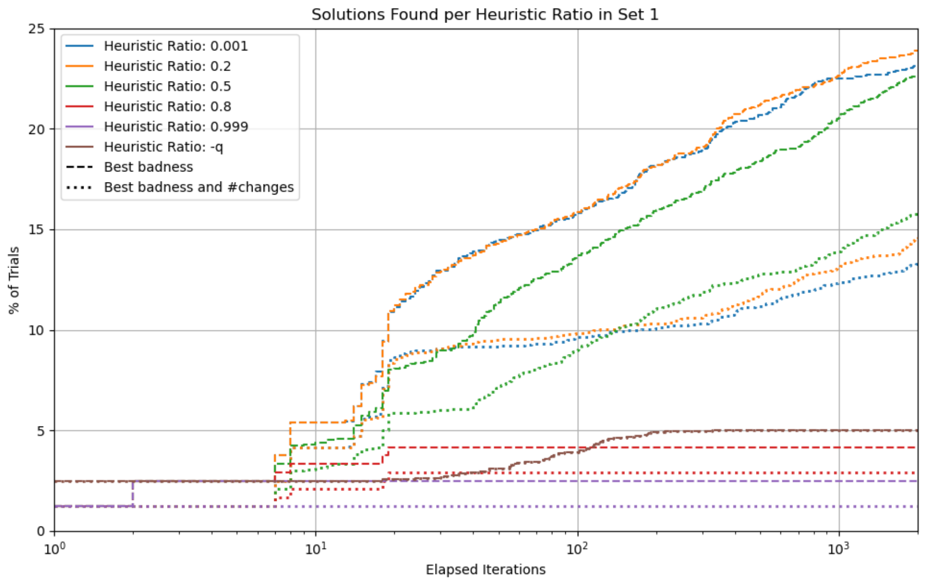

Figure 28: The percentage of trials per heuristic ratio in experiment set 1 that found the best badness and/or best number of changes within a number of elapsed iterations.

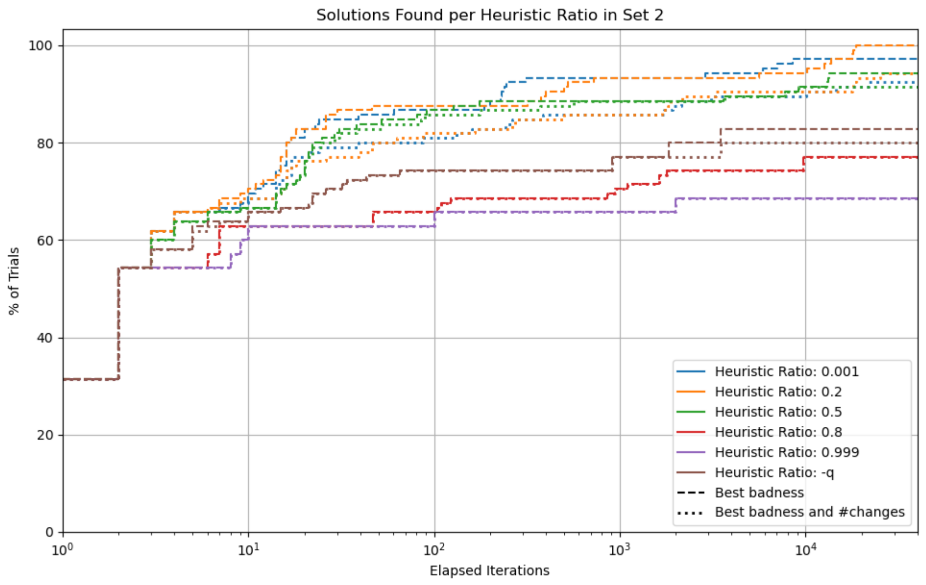

Figure 29: The percentage of trials per heuristic ratio in experiment set 2 that found the best badness and/or best number of changes within a number of elapsed iterations.

a heuristic ratio of 0.5 for each iteration. However, trials with a heuristic ratio of 0.001, 0.2 and 0.5 found about as many solutions with the best badness and best number of changes as each other.

10 Conclusions

Most large curve arrangements require too many iterations for BFS to find the optimal solution. Simulated Annealing finds solutions with a lower or equal badness than Heuristic Search in their given iterations, though Heuristic Search generally finds solutions with fewer changes from the original arrangement than Simulated Annealing when it does find a solution with the best badness.

Heuristic Search finds many subsolutions, trials with a high heuristic ratio or quotient ratio find subsolutions with a lower badness for few changes, though trials with a lower heuristic ratio find subsolutions with lower badness and often fewer changes for low badness subsolutions.

Trials on larger arrangements often require more time and the minimum time for all trials was around 1.4×10^{-5} seconds.

Larger arrangements, fixed edges, fewer but not no open areas and a lower closed loop penalty greater than 0 make scenarios harder for the Heuristic Search method to solve. Restricting Heuristic Search to a maximum number of changes decreases the number of solutions found. Heuristic Search with a heuristic ratio of 0.001 or 0.2 seems to find the most solutions with the best badness, whereas Heuristic Search with a heuristic ratio of 0.5 seems to find the most solutions with the best badness and the best number of changes.

11 Future work

There were some problems we encountered we were unable to solve and some parts of our algorithms we were unable to test. We will discuss these in the following subsections.

11.1 Research

The first problem is finding the minimum closed loop penalty that guarantees no closed loops in an optimal badness solution. In our current implementation, we use a closed loop penalty of at most 5, though this does not guarantee that the solution with the minimum badness has no closed loops. Finding the minimum closed loop penalty (for an arrangement) that does guarantee this may aid in finding fast deterministic methods.
The second problem is finding the minimum badness for an arrangement. This is currently done by the slow BFS method, or a stochastic method if BFS requires too many iterations. Finding a faster exact method that analyses a curve

arrangement to find its minimum badness aids the Heuristic Search method that finds such a solution or may lead to finding a fast exact method to find such a solution.
The last problem is finding the lower bound of the time complexity of a deterministic method that finds the optimal badness (with the minimal number of changes). We currently use BFS to find exact solutions, however, the asymptotic runtime of this method is exponential in the size of the curve arrangement. There may be algorithms with a lower time complexity and the problem might be NP-hard.

11.2 Program

Our first possible improvement to the implementation would be to find a method that finds the difference in badness for a change without evaluating the entirety of both states. A change affects on the changed global curves and the faces surrounding the changed area. Only reevaluating affected parts of the arrangement might drastically reduce the running time of larger arrangements.
Our second possible improvement is exact support for rational curves, including elliptical arcs. Currently, we approximate elliptical arcs, which means that these curves in the output never exactly the input. Our third possible improvement is for the program to find near misses within given radius. Our current implementation only finds intersections in the input and behaves as if the local radius is 0. We can currently only simulate arrangements with open areas by changing some local areas in the initial state. However, the program should be able to find near misses in the arrangement and be able to change these.
Our final possible improvement is to support more complex local areas. Currently, local areas contain two curves, but local areas with a larger radius may contain more than two curves. The program should be able to handle these by describing a local area by more than three states.

References

[1] Phoebe de Nooijer, Soeren Nickel, Alexandra Weinberger, Zuzana Masárová, Tamara Mchedlidze, Maarten Löffler, and Günter Rote. Removing popular faces in curve arrangements. *arXiv preprint arXiv:2202.12175*, 2022.

[2] Mees van de Kerkhof, Tim de Jong, Raphael Parment, Maarten Löffler, Amir Vaxman, and Marc van Kreveld. Design and automated generation of japanese picture puzzles. *Computer Graphics Forum*, 38(2):343–353, 2019.

[3] Samuel W Bent and Udi Manber. On non-intersecting eulerian circuits. *Discrete Applied Mathematics*, 18(1):87–94, 1987.

[4] Kurt Reidemeister. Elementare begründung der knotentheorie. In *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, volume 5, pages 24–32. Springer, 1927.

[5] Nancy M Amato, Michael T Goodrich, and Edgar A Ramos. Computing the arrangement of curve segments: Divide-and-conquer algorithms via sampling. In *SODA*, pages 705–706. Citeseer, 2000.

[6] Arno Eigenwillig and Michael Kerber. Exact and efficient 2d-arrangements of arbitrary algebraic curves. In *SoDA*, volume 8, pages 122–131, 2008.

[7] Arno Eigenwillig, Lutz Kettner, Elmar Schömer, and Nicola Wolpert. Exact, efficient, and complete arrangement computation for cubic curves. *Computational Geometry*, 35(1-2):36–73, 2006.

[8] Victor Milenkovic. Calculating approximate curve arrangements using rounded arithmetic. In *Proceedings of the fifth annual symposium on Computational geometry*, pages 197–207, 1989.

[9] Victor Milenkovic and Elisha Sacks. An approximate arrangement algorithm for semi-algebraic curves. *International journal of computational geometry & applications*, 17(02):175–198, 2007.

[10] Jon Louis Bentley and Thomas A Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Transactions on computers*, 28(09):643–647, 1979.

[11] A Bruen, JC Fisher, and JB Wilker. Apollonius by inversion. *Mathematics Magazine*, 56(2):97–103, 1983.

[12] The computational geometry algorithms library. https://www.cgal.org/. Accessed: 2022-07-01.

[13] Tinyxml-2. https://www.grinninglizard.com/tinyxml2/. Accessed: 2022-07-01.

[14] The ipe extensible drawing editor. https://ipe.otfried.org. Accessed: 2022-07-01.

A Curve arrangements used for Experiments

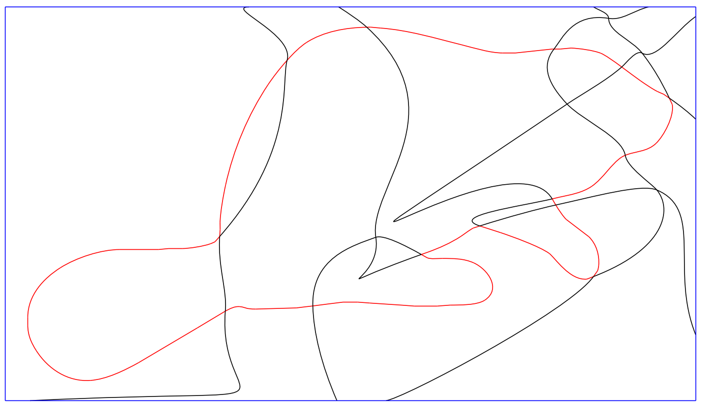
Figure 30: Arrangement *beaver*.

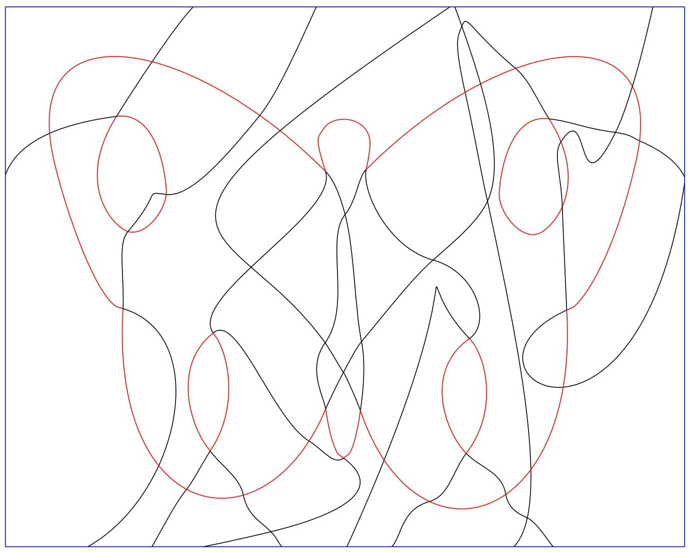
Figure 31: Arrangement *butterfly*.

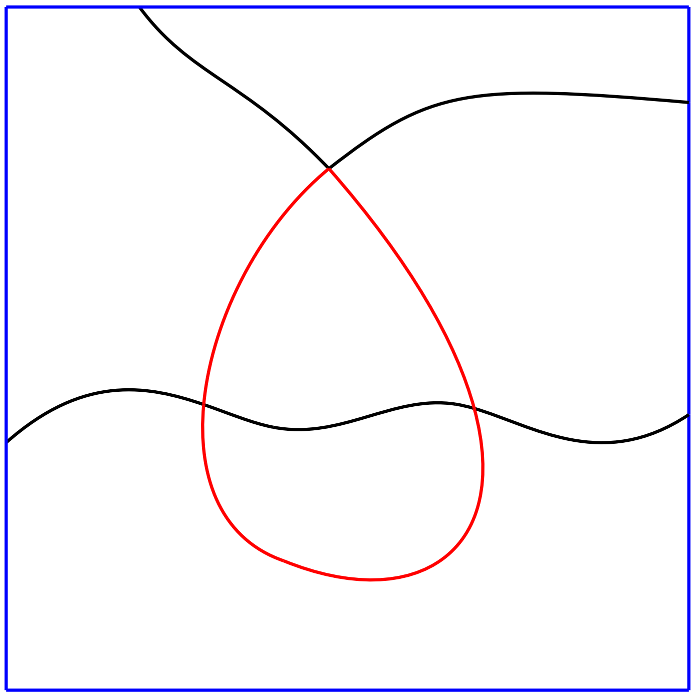
Figure 32: Arrangement *drop*.

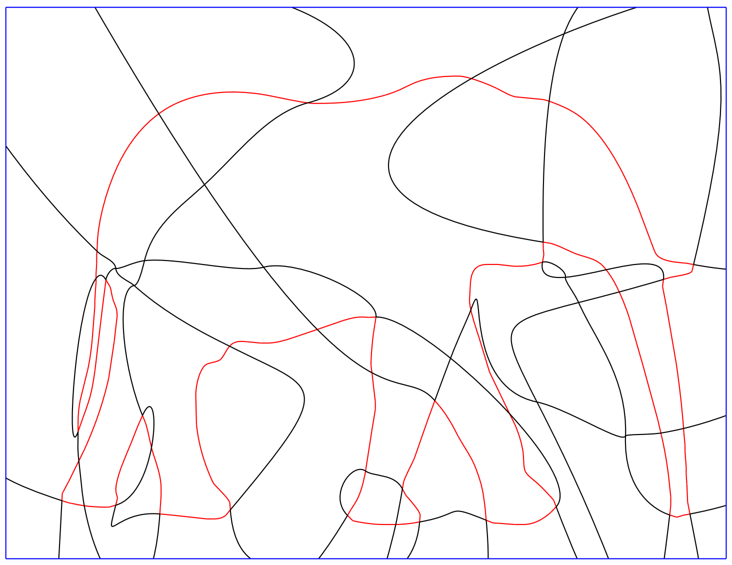
Figure 33: Arrangement *elephant*.

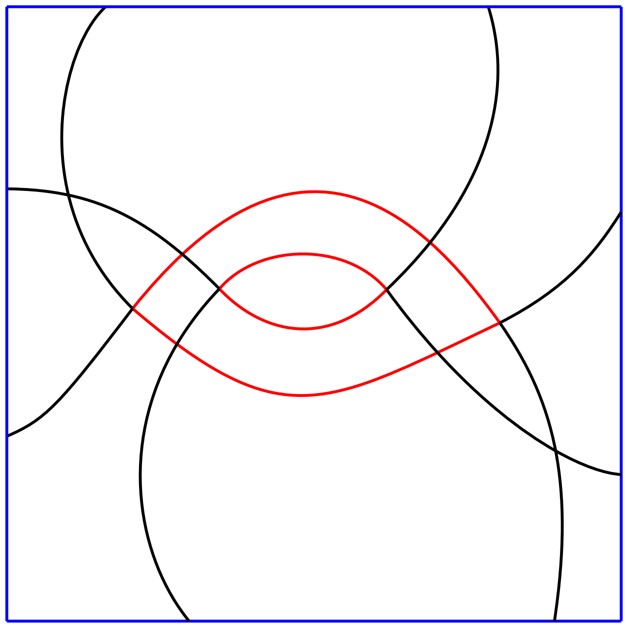

Figure 34: Arrangement *eye*.

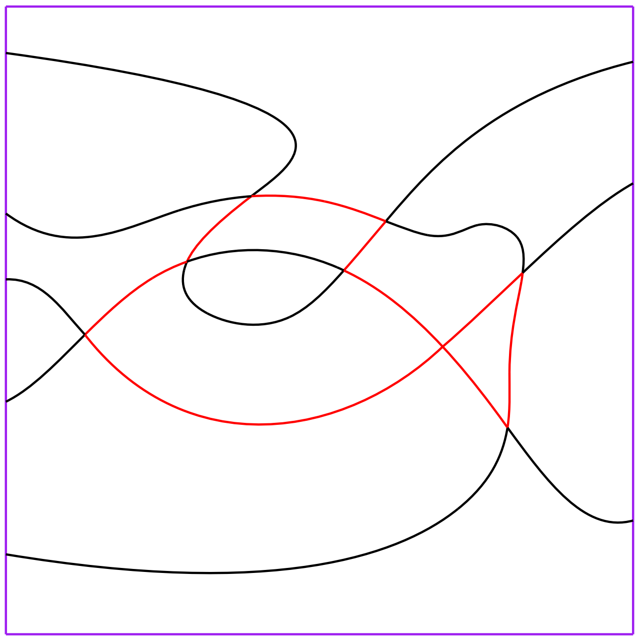

Figure 35: Arrangement *fish*.

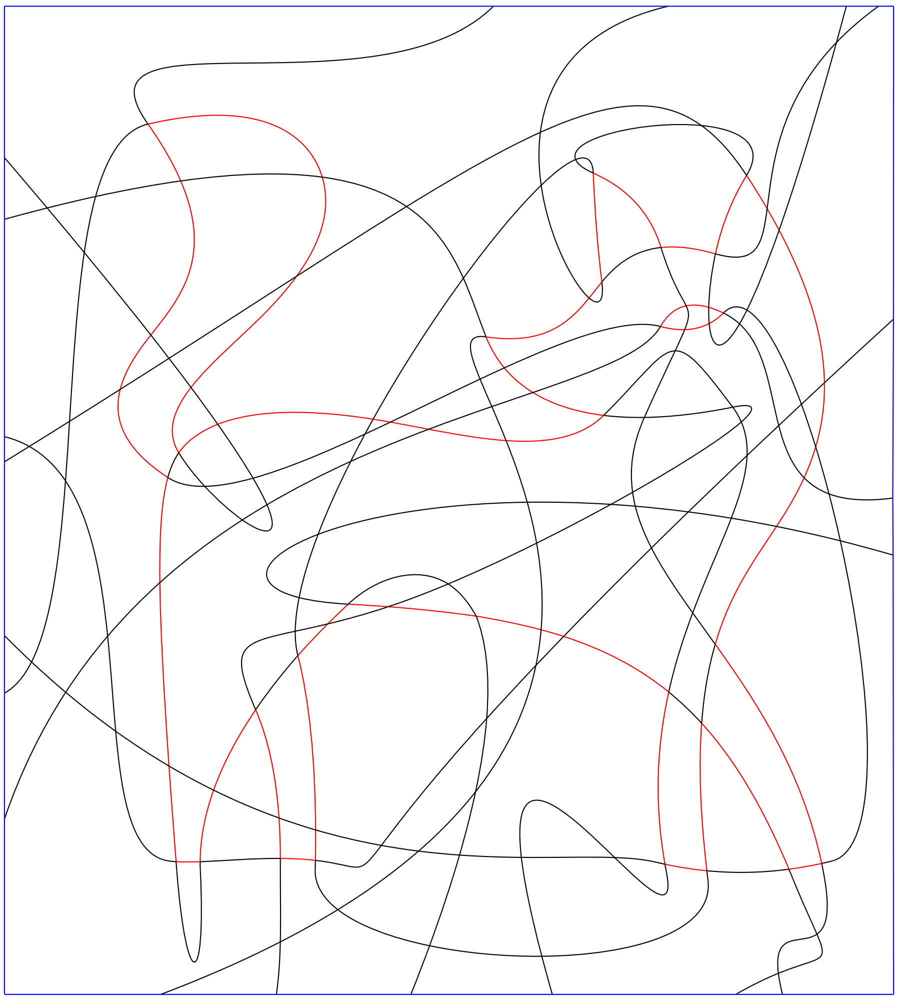

Figure 36: Arrangement *fox*.

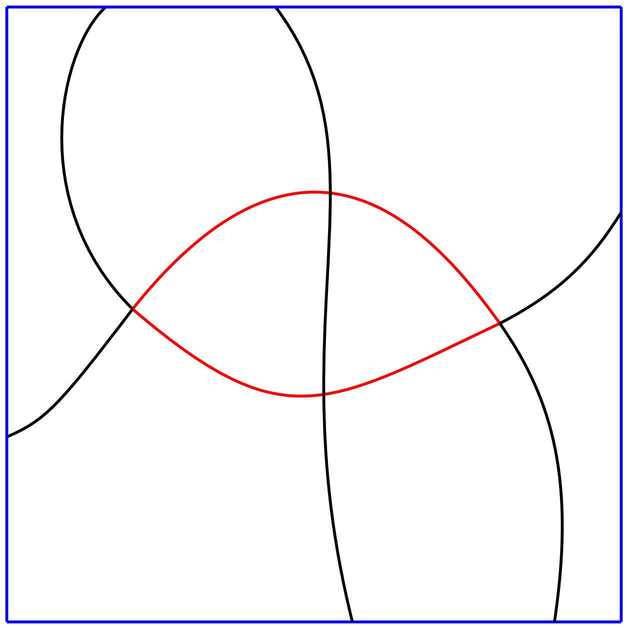

Figure 37: Arrangement *lens*.

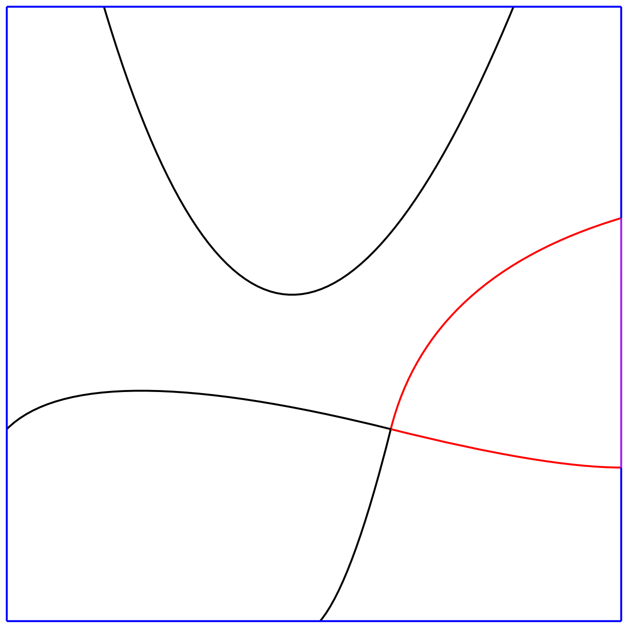

Figure 38: Arrangement *para*.

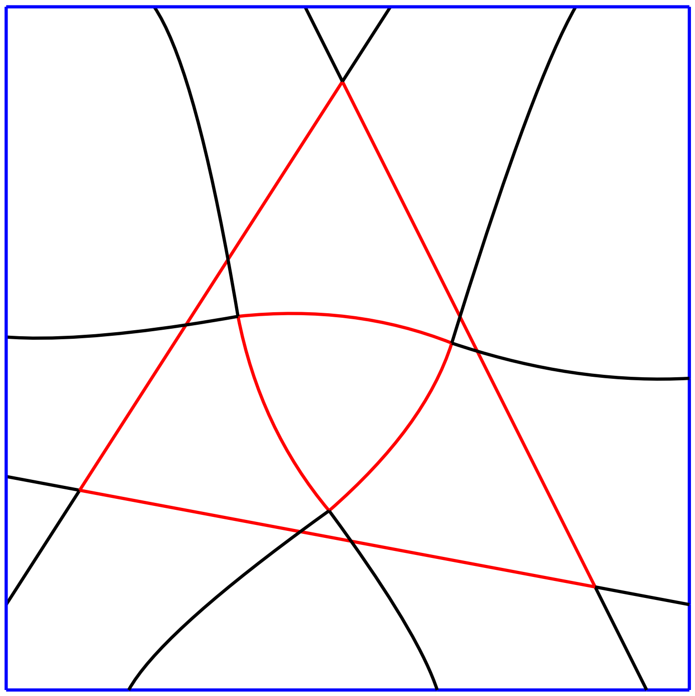
Figure 39: Arrangement *ring*.

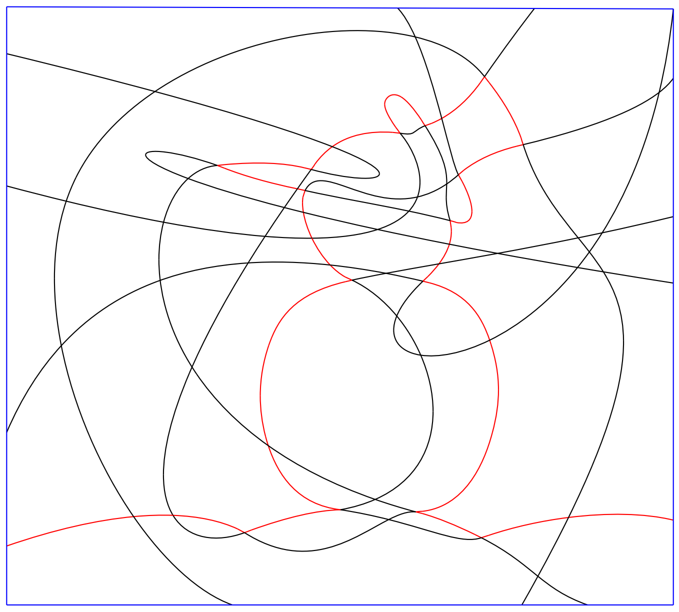
Figure 40: Arrangement *snowman*.

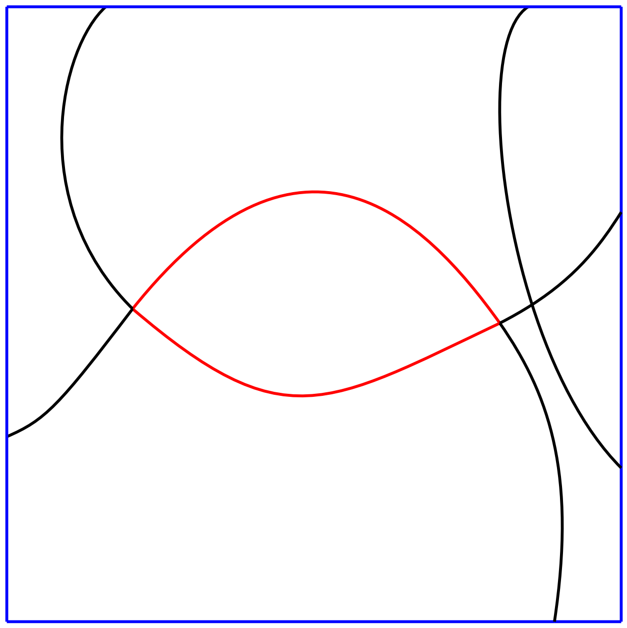
Figure 41: Arrangement *test0*.

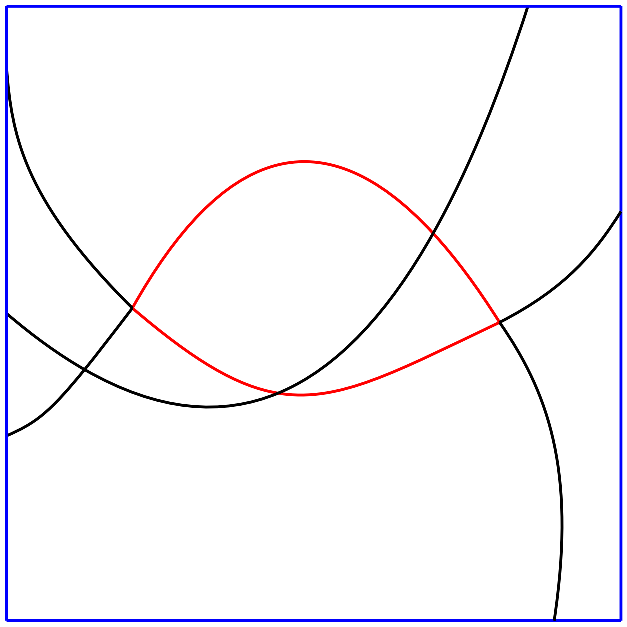
Figure 42: Arrangement *test1*.

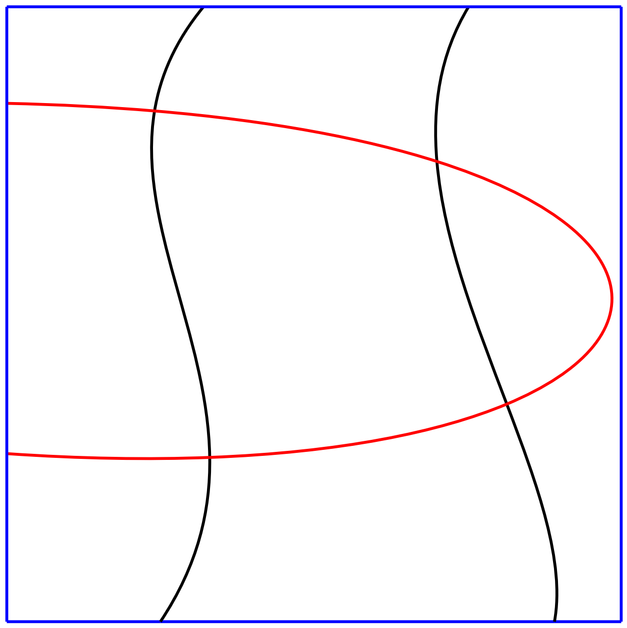

Figure 43: Arrangement *test2*.

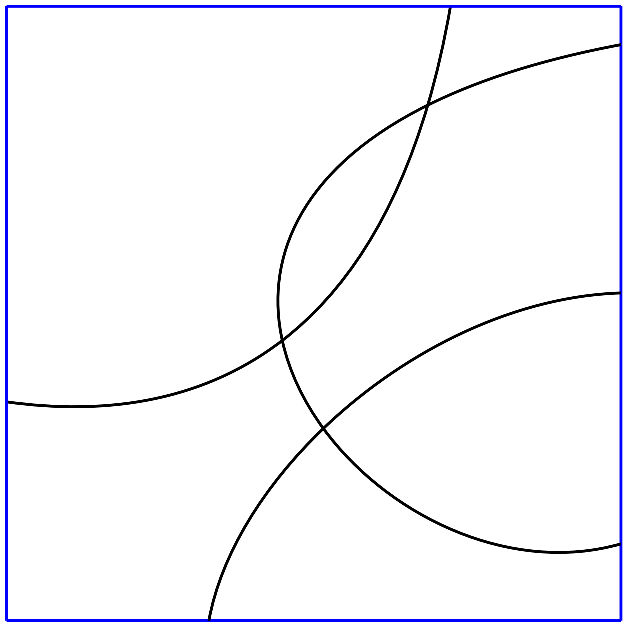

Figure 44: Arrangement *test3*.

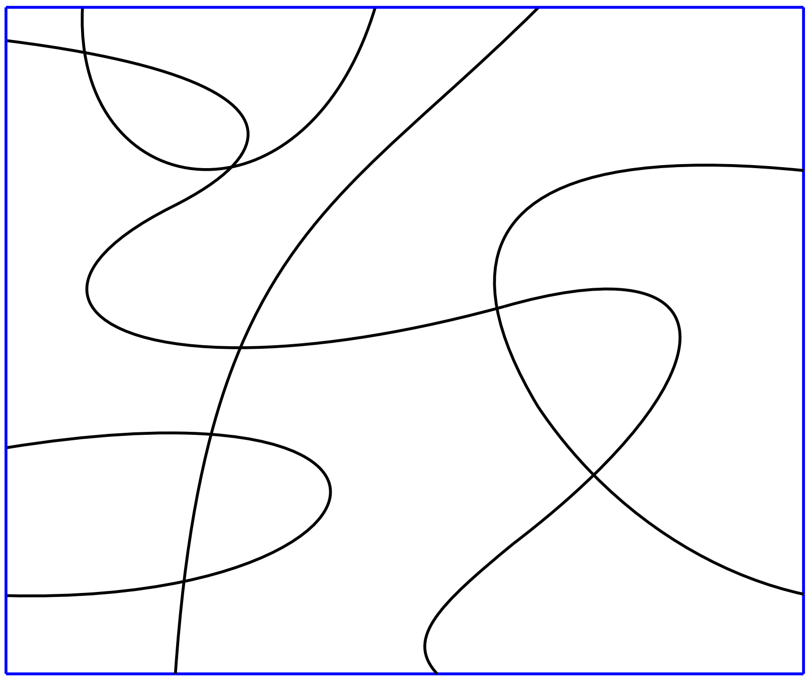
Figure 45: Arrangement *test4*.

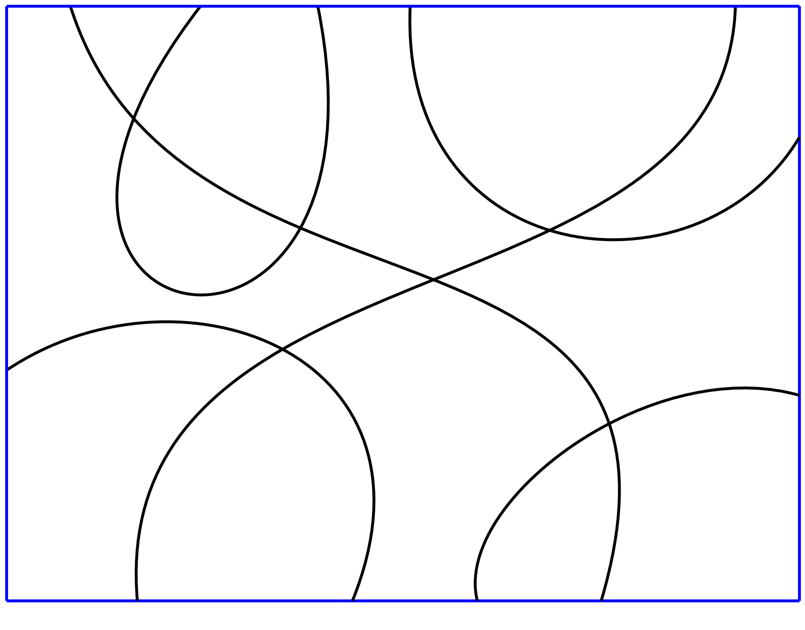
Figure 46: Arrangement *test6*.

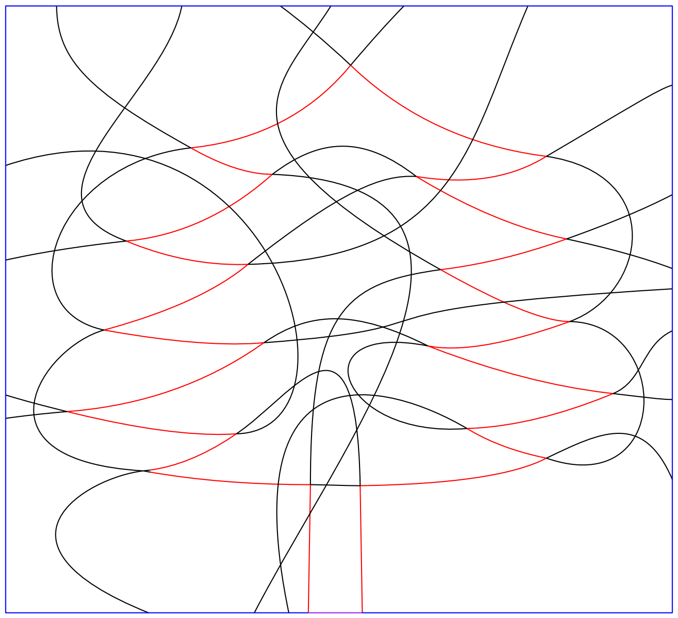

Figure 47: Arrangement *tree*.

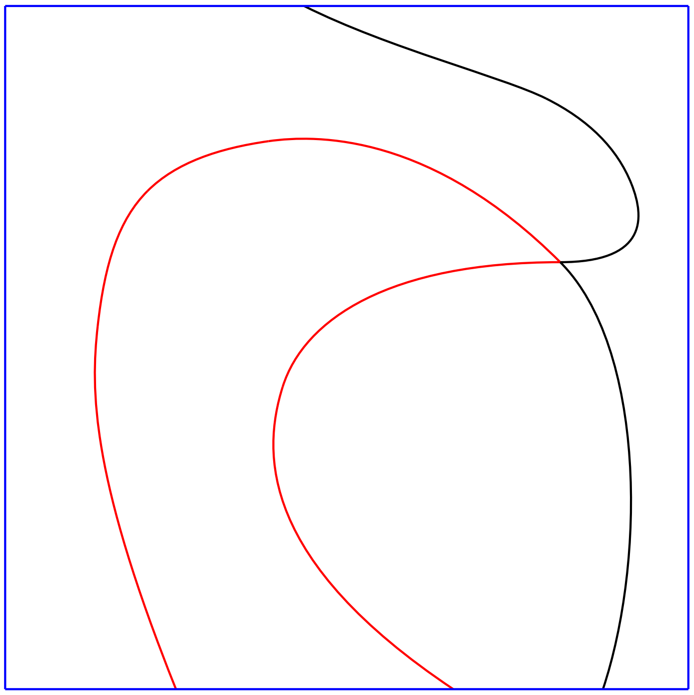

Figure 48: Arrangement *wave*.

B Summarized Results of Heuristic Search

Param.	Value	#Trials	Best Badness: Solutions Found	Best Badness: Avg. of Best 5% of Trials	Best Badness: Max. of 95% of Solutions	Best #Changes: Solutions Found	Best #Changes: Avg. of Best 5% of Trials	Best #Changes: Max. of 95% of Solutions
All of Set 2		630	86.7%	1.00	1829	84.0%	1.00	1801
Arrangement Size	Large	180	55.0%	8.33	13349	45.6%	9.44	17858
	Medium	180	98.3%	1.00	911	98.3%	1.00	911
	Small	270	100%	1.00	3	100%	1.00	3
IFE	True	342	87.7%	1.00	726	87.1%	1.00	1801
	False	288	85.4%	1.00	3487	80.2%	1.00	1998
Heuristic Ratio	0.001	105	97.1%	1.00	245	92.4%	1.00	2152
	0.200	105	100%	1.00	10180	94.3%	1.00	3294
	0.500	105	94.3%	1.00	7775	91.4%	1.00	370
	0.800	105	77.1%	1.00	1637	77.1%	1.00	1637
	0.999	0.5	68.6%	1.00	100	68.6%	1.00	100
	-q	105	82.9%	1.00	1829	80.0%	1.00	911

Table 5: Summarized results for experiment set 2. Each row represents a subset of trials, it contains the size of the subset, the percentage of trials that found the best badness, the average number of elapsed iterations of the 5% of trials of the subset that required the fewest number of iterations to find the best badness (if less than 5% found the best badness, this is the average over the trials that did) and the maximum number of iterations required to find 95% of the solutions that found the best badness. All three statistics regarding the trials that found the best badness are repeated for trials that found both the best badness and the best number of changes. These statistics are given for the entire set and divided by each value for each parameter.

Param.	Value	#Trials	Best Badness: Solutions Found	Best Badness: Avg. of Best 5% of Trials	Best Badness: Max. of 95% of Solutions	Best #Changes: Solutions Found	Best #Changes: Avg. of Best 5% of Trials	Best #Changes: Max. of 95% of Solutions
All of Set 1		14400	13.5%	6.07	983	8.81%	9.77	1289
Arrangement	*fox*	7200	9.51%	10.2	945	4.63%	299	1440
	snowm.	7200	17.6%	4.25	996	13.0%	4.25	1229
IFE	True	7200	25.4%	1.42	725	16.7%	3.67	1106
	False	7200	1.69%	880	1902	0.89%	1020	1919
Open Area Ratio	0.00	2880	12.4%	63.4	1326	9.34%	68.3	1757
	0.25	2880	9.13%	96.7	1402	3.78%	646	1883
	0.50	2880	8.85%	35.9	1499	2.67%	607	1831
	0.75	2880	9.38%	13.4	481	5.49%	91.3	1370
	1.00	2880	28.0%	1.00	96	22.7%	1.00	114
Closed coop Penalty	0	3600	14.8%	1.00	1052	9.03%	1.00	998
	1	3600	9.28%	31.1	1318	7.31%	38.8	1464
	3	3600	14.0%	14.0	783	8.22%	16.5	1560
	5	3600	16.1%	11.9	875	10.7%	12.7	1165
Max. #Changes	5	4800	2.50%	1.42	2	1.46%	1.00	1
	20	4800	16.5%	4.94	1007	12.0%	78.6	1140
	∞	4800	21.7%	4.93	1012	13.0%	7.89	1560
Heuristic Ratio	0.001	2400	23.2%	4.50	763	13.3%	7.04	1250
	0.200	2400	23.9%	4.50	1012	14.6%	7.05	1735
	0.500	2400	22.6%	5.25	1267	15.8%	9.02	1434
	0.800	2400	4.17%	6.10	19	2.92%	7.86	19
	0.999	2400	2.50%	1.50	2	1.25%	1.00	1
	-q	2400	5.00%	47.7	182	5.00%	47.7	182

Table 6: Summarized results for experiment set 1. Each row represents a subset of trials, it contains the size of the subset, the percentage of trials that found the best badness, the average number of elapsed iterations of the 5% of trials of the subset that required the fewest number of iterations to find the best badness (if less than 5% found the best badness, this is the average over the trials that did) and the maximum number of iterations required to find 95% of the solutions that found the best badness. All three statistics regarding the trials that found the best badness are repeated for trials that found both the best badness and the best number of changes. These statistics are given for the entire set and divided by each value for each parameter.