

# Controlled Perturbation of Sets of Line Segments in $\mathbb{R}^2$ with Smart Processing Order\*

Eli Packer, State University of New York at Stony Brook

## Abstract

Controlled Perturbation is a framework for perturbing geometric input to make it more robust for fixed-precision manipulation. We present a Controlled Perturbation scheme for sets of line segments in  $\mathbb{R}^2$  (CPLS, for short). CPLS perturbs the endpoints of the line segments to eliminate degeneracies that may cause errors when using fixed-precision arithmetic. We implemented CPLS and provide experimental results.

In the core of this work, we present novel techniques to decrease the perturbation magnitude by determining a smart order for processing the endpoints. We designed and implemented several methods for that purpose. Our experiments show that these methods are effective in decreasing the perturbation magnitude.

## 1 Introduction

The use of fixed-precision datatypes in geometric algorithms is problematic. It often results in either wrong output or program crashes due to round-off errors. Much effort has been devoted in recent years to deal with these problems [15].

By *robustness* we refer to the general goal of making geometric implementations reliable enough to produce correct results. Closely related to robustness is the idea of *degeneracy*, as fixed-precision arithmetic errors are often caused by degenerate input data. We use the term *potential degeneracy* to refer to situations where standard fixed-precision arithmetic computations cannot decide whether the configurations are degenerate. On the other hand, we use the term *definite degeneracy* to refer to degenerate cases that are guaranteed to be precise thanks to topological information about the input.

Let  $E_1(\zeta)$  and  $E_2(\zeta)$  be the evaluations of a predicate  $\zeta$  using fixed-precision arithmetic and exact arithmetic,

respectively.  $E_1(\zeta)$  induces potential degeneracy when there is no guarantee that the sign of  $E_1(\zeta)$  and  $E_2(\zeta)$  will be the same (positive, negative or zero). This leads to the following definition. A *resolution bound* (with respect to predicate  $\zeta$ ) is the minimum separation of features that is required to guarantee that the sign of the evaluation of  $\zeta$  can be safely computed with fixed-precision arithmetic. Note that there are different resolution bounds for different kinds of predicates.

*Finite-Precision Approximation* is a framework for converting input sets into fixed-precision arithmetic, in a way that makes them more robust for fixed-precision manipulation. The idea is that applications that use fixed-precision arithmetic will produce reliable output, while working on the converted input. Usually, one expects the corresponding conversion algorithm not to change both the geometry and the topology of the input much.

*Controlled Perturbation* (CP for short) is a Finite-Precision approximation approach whose main idea is to perturb the input slightly in order to increase the robustness. It is carried out in an incremental fashion where features of the input are processed one at a time. In this process, some of the features are perturbed to eliminate potential degeneracies. CP schemes define the potential degeneracies that they handle and set corresponding resolution bounds that affect the perturbation process.

We present a perturbation scheme for sets of line segments in  $\mathbb{R}^2$  that follows the ideas of CP. In the next section we describe the main ideas of our scheme and thus also illustrate the general concepts of CP.

The main drawback of any CP scheme is the possible large deviation from the original input. It is evident that in complicated and congested regions that naturally contain many potential degeneracies, the perturbation tends to increase. In such cases, the output may not be acceptable for further use. Thus, the idea of developing techniques to constrain the perturbation magnitude should be highly encouraged in CP schemes.

---

\*Work on this paper has been partially supported by the National Science Foundation (CCR-0098172, CCF-0431030).

We focus in a novel technique to decrease the perturbation magnitude, based on determining the processing order of the input features intelligently. We use this technique in our scheme and present experimental results showing its effectiveness.

**Related Work.** Earlier perturbation schemes can be found at [1, 3, 9, 13]. Halperin and Shelton [11] introduced the concept of Controlled Perturbation. They worked on arrangements of spheres in  $\mathbb{R}^3$  that supports geometric queries on molecular models. Raab [14] followed by proposing Controlled Perturbation of polyhedral surfaces in  $\mathbb{R}^3$  to eliminate degeneracies in swept volume applications. Halperin and Leiserovich [10] described a framework for circles in  $\mathbb{R}^2$ . They were the first to actually compute the resolution bounds instead of setting them as parameters. Funke et al. [7] used Controlled Perturbation in randomized incremental constructions, and designed specific schemes for planar Delaunay triangulations and convex hulls in arbitrary dimensions. Mehlhorn et al. [12] extended this work and developed a general methodology for deriving quantitative relations between the amount of perturbation and the precision of the approximate arithmetic.

The rest of the paper is organized as follows. In the next section we discuss the main ideas of our scheme. In Section 3 we describe the algorithm. In Section 4 we present the different potential degeneracy cases of our model. In Section 5 we discuss our new methods for decreasing the perturbation magnitudes. In Section 6 we present experiments performed with our implementation. We conclude and present ideas for future research in Section 7.

## 2 Main Ideas

In this section we describe the main ideas of our perturbation scheme. The input is an Euclidean graph  $G = (V, E)$  where  $V$  is the set of the endpoints and  $E$  is the set of edges. We perturb  $G$  to eliminate potential degeneracies. Our perturbation scheme preserves the edges (two edges that are incident to an endpoint will remain incident to it after perturbation), thus inducing a specific degeneracy. However, this degeneracy is definite and given in the input, thus should be easily identified in subsequent algorithms as edges that share the same endpoint (note that we eliminate other kinds of degeneracies). Preserving incidences assists in maintaining the structure of many geometric settings, like

triangulations and Voronoi diagrams. In comparison, previous CP schemes either did not allow definite degeneracies or allowed them partially. Note that, like other CP schemes, we do not claim to preserve the topology of the input, thus vertices may cross edges during the perturbation.

We perturb the endpoints of  $V$ . Let  $v \in V$  be an endpoint. We subdivide the plane into two regions (not necessarily connected). The first,  $F(v)$ , contains all the points that are forbidden to  $v$  because if  $v$  lands on them, potential degeneracies are induced. We call  $F(v)$  the *forbidden loci* of  $v$ . The second region,  $\bar{F}(v) = \mathbb{R}^2 - F(v)$ , contains valid locations for  $v$  to land. We say that these locations are *degeneracy-free* with respect to  $v$ . The task of CP is to find a placement for  $v$  within  $\bar{F}(v)$ .

We process the endpoints of  $V$  one at a time, constructing the output  $V'$  which consists of the possibly perturbed endpoints. An edge is considered processed only after both of its endpoints have been processed. Let  $v \in V$  be any endpoint and  $\Upsilon(v)$  be the adjacent edges of  $v$  whose other endpoint have already been processed before  $v$ . We say that  $v$  *induces* potential degeneracies if  $v$  or any edge of  $\Upsilon(v)$  are involved in potential degeneracies. In each step we test if  $v$  induces potential degeneracies with the features that have been processed before  $v$  (endpoints and edges). In other words, we test if  $v$  is initially positioned in a forbidden locus. If this is not the case,  $v$  is simply inserted into  $V'$ . Otherwise,  $v$  is perturbed to a degeneracy-free placement. Thus, by processing  $v$  we eliminate potential degeneracies that involve both  $v$  and  $\Upsilon(v)$ . After the processing of  $v$  is complete, its placement is fixed and it will not be perturbed again. Note that by a simple induction argument, no potential degeneracies will be induced after processing all the endpoints of  $V$ .

Each endpoint  $v$  and a parameter  $\delta$  defines the *perturbation disc*  $C(v, \delta)$ .  $C(v, \delta)$  is centered at  $v$  and has a radius  $\delta$ , called the *perturbation radius*. In order to resolve potential degeneracies,  $v$  is perturbed randomly inside  $C(v, \delta)$  once or more, if necessary, until a degeneracy-free placement is found. In order to guarantee successful completion of the perturbation process, and guarantee reasonable chances to find a degeneracy-free placement for  $v$ ,  $\delta$  should be large enough so that  $C(v, \delta)$  will necessarily contain relatively large degeneracy-free loci for  $v$ . See Figure 1 for an illustration of a perturbation step.

Let  $F$  be the theoretical maximum area that can be captured by forbidden loci and let  $\varphi = \frac{|C(v, \delta)| - F}{|C(v, \delta)|}$ .

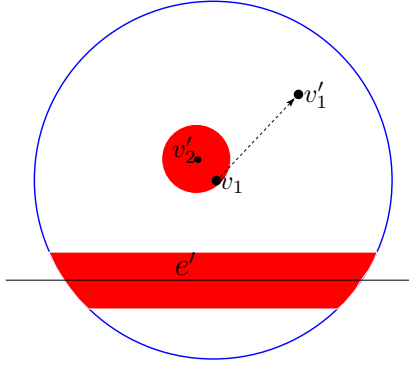


Figure 1: Perturbing an endpoint  $v_1$  to eliminate potential degeneracies. The portion of forbidden loci inside the perturbation disc of  $v_1$  are drawn in red and corresponds to an endpoint  $v'_2$  and an edge  $e'$ , both already processed. Since the location of  $v_1$  is inside a forbidden locus, it has to be perturbed to a degeneracy-free locus ( $v'_1$  in this example).

Then  $\varphi$  is a lower bound on the probability that  $v$  will be placed within a degeneracy-free placement, if perturbed randomly inside  $C(v, \delta)$ . (Note that we require that  $|C(v, \delta)| > F$ .) Since the random perturbations are independent,  $\frac{1}{\varphi}$  trials on average are required to find a potential degeneracy-free placement for  $v$ . Since  $\varphi$  depends on  $\delta$ , the selection of  $\delta$  is a tradeoff between the size of the perturbation and the efficiency of the computation.

Following previous CP schemes, we handle potential degeneracy types that involve separation of geometrically close features. In our case we have two kinds of degeneracies: a degeneracy between a vertex and a non-incident edge and a degeneracy between two vertices. In section 4 we present all the ways that the potential degeneracies can be created and show their corresponding forbidden loci.

### 3 Algorithm

Recall that  $G(V, E)$  is the input for our schemes where  $V = \{v_1, v_2, \dots, v_n\}$  is the set of the endpoints and  $E$  is the set of the edges. Our algorithm perturbs some of the endpoints in  $V$  and maintains the connectivity described by  $E$ . The algorithm starts by computing  $\delta$  and the order of the endpoint processing (see Section 5). Then the endpoints are processed one by one. Let  $v \in V$  be any endpoint. If  $v$  induces potential degeneracies it is randomly perturbed, until all potential degeneracies it induces are resolved. We denote

by  $V'(v)$  the temporary output obtained just before processing  $v$ . After processing  $v$ , it is inserted in the output and its placement will be fixed. In the processing of  $v$ , features that consist of the endpoints of  $V'(v)$  and edges whose both endpoints belong to  $V'(v)$  are tested for potential degeneracies with  $v$  and  $\Upsilon(v)$ .

We set  $\delta$  such that  $\varphi = \frac{1}{2}$ . Thus, if an endpoint is located within a forbidden locus, we need at most two trials on average to find a degeneracy-free placement for  $v$  and the probability for larger number of trials decreases exponentially.

Let  $\mathcal{D}(v)$  be a predicate that is true if an endpoint  $v$  induces potential degeneracies. We conclude this section with a high level pseudocode of CPLS.

#### CONTROLLED PERTURBATION SETS OF LINE SEGMENTS IN $\mathbb{R}^2$

**Input:** An Euclidean graph  $G = (V, E)$  where  $V$  are the endpoints and  $E$  are the edges

**Output:** A set  $V'$  of the perturbed endpoints

1. Compute the perturbation radius  $\delta$  (see Appendix A)
2. Compute a processing order  $\Pi(V)$  (see Section 5)
3. **foreach** endpoint  $v \in V$  in the order of  $\Pi(V)$
4.     **while**  $\mathcal{D}(v)$
5.         Perturb  $v$  randomly inside  $C(v, \delta)$ .
6.     **end while**
9.     insert  $v$  to  $V'$
10. **end foreach**

## 4 Inventory of Potential Degeneracies and Forbidden Loci

In this section we present the potential degeneracies that we handle with their corresponding forbidden loci. Similar to previous CP schemes, we solve potential degeneracies that correspond to close proximity. Following possible queries of our scheme, close proximities can be induced by either two vertices or a vertex and a non-incident edge. We make each pair of this kind well separated (see Theorem 4.1).

In our scheme we encounter several scenarios in which close proximity degeneracies occur. All are described below. We hold five distinct resolution bounds ( $\varepsilon_1 - \varepsilon_5$ ). Their relationship and necessity for computing  $\delta$  will become clear in Appendix A.

We refer to  $v$  as the endpoint that is currently pro-

cessed. We denote by  $v'$  its output. Next we present the forbidden loci of our scheme.

**Forbidden Loci induced by Endpoints.** In order to guarantee a separation between  $v$  and another endpoint that has been processed (denoted by  $v_j$ ), we define the forbidden locus of  $v_j$  to be a disc centered at  $v_j$ . Let  $\varepsilon_1$  be its corresponding resolution bound (thus the radius of the disc). It follows that the area of the forbidden locus is  $\pi\varepsilon_1^2$ . We denote this forbidden loci by  $\mathcal{F}_1$ . In the entire process, we guarantee a separation of  $\varepsilon_1$  between two endpoints. We refer to this as an invariant of the process and denote it by  $\mathcal{I}_1$ .

**Forbidden Loci Induced by  $v$  and a Non-incident Edge.** Let  $e \in E$  be an edge that has already been processed. The forbidden locus of  $e$  is the Minkowski sum of  $e$  and a disc  $D$  centered at the origin. Let  $\varepsilon_2$  be the corresponding resolution bound, thus the radius of  $D$ .  $\varepsilon_2$  should be large enough to guarantee sufficient separation between  $v'$  and both  $e$  and any vertex on  $e$ . It is easy to show that the maximum area which the forbidden locus can cut from  $P(v)$  is when  $e$  passes through  $v$  and intersects  $P(v)$  twice (see Figure 2(a)). This area is bounded by a rectangle whose area is  $2\varepsilon_2 \times 2\delta$ . We denote this forbidden loci by  $\mathcal{F}_2$ . Similar to invariant  $I_1$ , we define  $I_2$  here. It refers to a lower bound ( $\varepsilon_2$ ) on the minimum separation between endpoints and non-incident edges.

**Forbidden Loci Induced by Intersections of Edges and Edges incident to  $v$ .** We separate the intersection of two edges, non-incident to  $v$ , and an edge  $e$ , which is both incident to  $v$  and whose other endpoint has already been processed. Let  $u$  be the above intersection. Let  $\varepsilon_3$  be the corresponding resolution bound. It should be large enough to guarantee sufficient separation between  $u$  and both  $e$  and any vertex on  $e$ . Thus,  $e$  must not penetrate the disc centered at  $u$  with radius  $\varepsilon_3$  (denoted by  $C$ ). This is illustrated in Figure 2(b), where  $e$  is the thick line whose first endpoint,  $w$ , has already been processed. In order to prevent  $e'$  (the output of  $e$ ) from penetrating  $D$ ,  $v'$  must not be located inside the wedge  $dwc$ . It defines a quadrilateral which bounds the part of the forbidden locus inside  $P(v)$  (quadrilateral  $abcd$  in Figure 2(b)). Quadrilateral  $abcd$  has maximum area both when  $u$  is located infinitesimally close the intersection of  $e$  and the disc with radius  $\varepsilon_2$  around  $w$  (note that this disc

is empty of edges thanks to invariant  $I_2$ ) and when the length of  $e$  is maximum. This case is illustrated in Figure 2(b). We denote this forbidden loci by  $\mathcal{F}_3$ .

**Forbidden Loci Induced by Endpoints and Edges incident to  $v$ .** This case is illustrated in Figure 2(c). It is similar to  $\mathcal{F}_3$ . To satisfy  $I_2$ , we define a forbidden disc around the endpoint  $u$  with radius  $\varepsilon_2$ . The goal is to separate  $e'$  (the output of  $e$ ) and any vertex on it from any non-incident endpoint by at least  $\varepsilon_2$  units. Note that the distance between  $u$  and  $w$  is at least  $\varepsilon_1$  (based on  $I_1$ ). We denote this forbidden loci by  $\mathcal{F}_4$ .

**Forbidden Loci Induced by Endpoints for Robust Distance Computation.** One of the tests that we perform involves the computation of the distance between a point  $v$  and an edge  $e$ . It involves the projection of  $v$  onto the line that contains  $e$  and a decision whether the projection is on  $e$ . Thus, the computation needs to determine whether the projected point is on  $e$ . Since we need this computation to be robust, we perturb  $v$  such that its projection is sufficiently far from the endpoints of  $e$ . It follows that the forbidden loci for  $v$  are the two strips that are illustrated in Figure 2(d). We denote by  $\varepsilon_5$  the corresponding resolution parameter which is the width of the strips and by  $\mathcal{F}_5$  this forbidden loci.

We note that there are more potential degeneracy cases that are eliminated in our scheme, once we make sure that the other potential degeneracy cases (whose forbidden loci are described above) are eliminated. Thus, we do not make explicit tests for their existence, but only take them into account when computing the various resolution bounds. The first is the close proximity between an intersection point  $f$  and a non-incident edge that have been inserted before  $f$ . We note that there is sufficient separation between the two when  $f$  is created. This case is illustrated in Figure 2(e), which is the situation after an edge  $e_i$  (with an output  $e'_i$ ) was processed and created the intersection point  $f$  with  $e'_j$ ; by that time the edges  $e'_j$  and  $e'_k$  have already been processed. A potential degeneracy occurs if  $f$  is too close to  $e'_k$ . We show in Appendix A that if all other potential degeneracy cases are eliminated, then there is a lower bound on the separation between  $f$  and  $e'_k$ . We make sure that this lower bound is large enough to eliminate the corresponding potential degeneracy. We denote this case by  $\mathcal{C}_1$ .

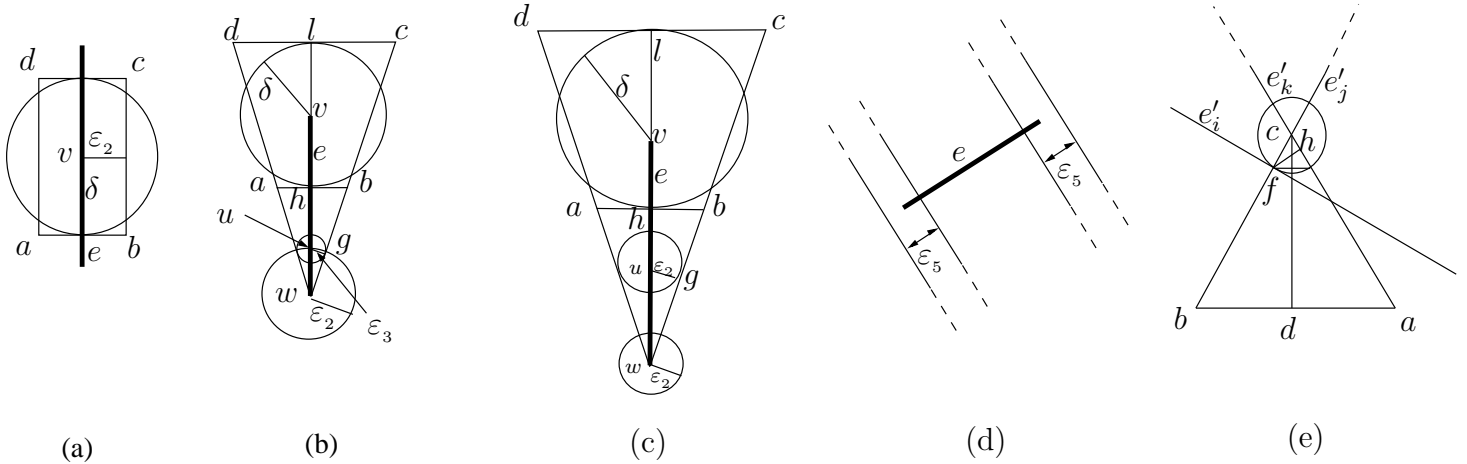


Figure 2: Forbidden loci defined to resolve different scenarios of potential degeneracies

The second potential degeneracy case consists of a close proximity between  $f$  and another intersection point along one of the adjacent edges to  $f$ . Note that for the same reasons we describe above, there is a lower bound on the distance between  $f$  and another intersection point on adjacent edges, because we showed a lower bound on the distance from non-incident edges to  $f$ . We make sure that this bound is large enough to eliminate the corresponding potential degeneracy. We denote this case by  $\mathcal{C}_2$ . We denote the minimum separation necessary to eliminate both potential degeneracies described in cases  $\mathcal{C}_1$  and  $\mathcal{C}_2$  by  $\varepsilon_4$ .

In Appendix A we analyze the relationship among  $\varepsilon_1 - \varepsilon_5$  and compute the value of  $\delta$ .

**Theorem 4.1** *All potential degeneracies of the kinds vertex-non incident edge and vertex-vertex are eliminated in our scheme.*

**Proof:** We show that since we eliminate the potential degeneracies that correspond to the forbidden loci that are described above, no potential degeneracy of close proximity exists in any step of the algorithm. We consider the process of some endpoint  $v$ . Recall (Section 2) that when processing  $v$ , we need to eliminate potential degeneracies that  $v$  and  $\Upsilon(v)$  involve, in order to eliminate all potential degeneracies. We next demonstrate that these degeneracies are indeed eliminated.

After processing an endpoint  $v$ , it will be separated from other endpoints (thanks to forbidden loci of type  $\mathcal{F}_1$ ), and non-incident edges and other vertices (thanks to the forbidden loci of type  $\mathcal{F}_2$ ). Thus  $v$  will not be involved in potential degeneracies. After processing an edge  $e$ ,  $e'$  (the output of  $e$ ) and any vertex on it

will be separated from all vertices that do not lie on  $e$  (thanks to forbidden loci of types  $\mathcal{F}_3$  and  $\mathcal{F}_4$ ). Also, any intersection point on  $e'$  will be separated from any non-incident edge (based on case  $\mathcal{C}_1$ ) and any two intersection points on  $e'$  will be separated (based on case  $\mathcal{C}_2$ ). Moreover, any intersection point on  $e'$  will be separated from each endpoint of  $e'$  (thanks to forbidden loci  $\mathcal{F}_2$  and  $\mathcal{F}_4$ ). We note that the above is a complete description of cases in which the edges of  $\Upsilon(v)$  may involve in potential degeneracies. Thus, these potential degeneracies will be eliminated after processing  $v$ . By a simple induction on the endpoint processing, the proof follows.  $\square$

## 5 Ordering the Endpoint Processing

The main drawback of the Controlled Perturbation scheme is the possibility of large input deviation. In this section we describe a novel technique to decrease the perturbation magnitude. Recall that we incrementally process endpoints. Let  $\Pi(V)$  be any endpoint processing order. For each endpoint  $v \in V$ , let  $\Pi(v)$  denote its place in  $\Pi(V)$ . Let  $v_1$  and  $v_2$  be two endpoints such that both  $\Pi(v_1) < \Pi(v_2)$  and  $[v_1, v_2] \in E$ . It follows that the test for a degeneracy-free placement for  $v_2$  involves testing potential degeneracies for the segment  $e = [v_1, v_2]$  (and possibly other tests as well). On the other hand, since  $v_2$  is processed only after  $v_1$ , no tests that involve  $e$  have to be performed while processing  $v_1$ . This situation would be symmetric and involve different potential degeneracy tests if  $\Pi(v_1) > \Pi(v_2)$ ,

in which case potential degeneracies for  $e$  are tested only while processing  $v_1$ . Since different forbidden loci (with potentially different areas) are involved in the above two different cases, each may result, on average, in different perturbation magnitudes.

For each endpoint  $v \in V$ , let  $E(v) = \{[v, v'] \mid [v, v'] \in E, \Pi(v') < \Pi(v)\}$ . Then the processing of  $v$  involves eliminating potential degeneracies induced by the edges of  $E(v)$  (and possibly other potential degeneracies as well).

Consider Figure 3 which illustrates an arrangement of four line segments (in solid lines). Suppose  $a$  and  $b$  are processed after the other six endpoints. We need to separate  $e = [a, b]$  from all non-incident vertices. Vertices that are sufficiently close to  $e$  induce potential degeneracies (the vertices surrounded by their forbidden discs in the figure). Hence, the forbidden loci associated with  $a$  and  $b$  are the wedges originating from  $b$  and  $a$  respectively; These are tangent to the union of the three forbidden discs (the wedges are drawn with dashed rays in the figure). Let  $\gamma(a)$  and  $\gamma(b)$  be the wedges originating from  $a$  and  $b$  respectively. If  $b$  is processed before  $a$ , the processing of  $a$  involves the insertion of  $e$  and  $\gamma(b)$  is used as a forbidden locus. If their order is reversed, we use  $\gamma(a)$  instead. Since  $b$  is much closer to the forbidden discs than  $a$ ,  $\gamma(b)$  cuts a larger area from  $P(a)$  than  $\gamma(a)$  cuts from  $P(b)$ . Hence, it would be a better choice to process  $a$  before  $b$  in order to handle smaller forbidden locus that results in potentially smaller perturbations.

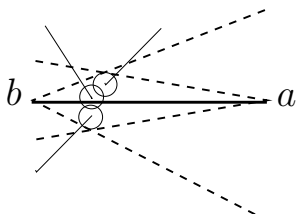


Figure 3: The forbidden wedges (in dashed lines) that affect the final placement of edge  $ab$ .

Based on this idea, we develop and use efficient algorithms to improve the quality of the output. The idea is to analyze the two potential forbidden loci that affect the placement of each edge—one for each possible order of its endpoints. We use two measurements to evaluate the efficiency of the processing order:

- MIN-SUM: Minimize the perturbation sum.
- MIN-MAX: Minimize the maximum perturbation.

We work on the geometric graph induced by the input. We define two different directed graphs in this section, one for the MIN-SUM problem and the other for the MIN-MAX. Information that correspond to the forbidden locus areas that the edges are involved with are embedded in their weights. These forbidden loci are induced by the vertices that are close enough to the edges to induce potential degeneracies.

In the rest of this section, we define the optimization problems for MIN-SUM and MIN-MAX and describe their corresponding algorithms.

## 5.1 Minimizing the Sum

The goal here is to minimize the sum of the perturbation magnitudes. Given  $v \in V$  and an adjacent edge  $e$ , let  $F_e(v)$  be the forbidden locus areas induced inside  $P(v)$  as the result of the forbidden loci that  $e$  involves<sup>1</sup>. These areas include forbidden loci induced by endpoints and intersections. The directed graph  $G$  is defined as follows<sup>2</sup>. Let  $v_1$  and  $v_2$  denote a pair of adjacent endpoints in  $V$ . We define a directed edge  $e \in E$  in  $G(V, E)$  that corresponds to the edge that connects  $v_1$  and  $v_2$  as follows. If  $F_e(v_2) > F_e(v_1)$ , then it would likely be beneficial (ignoring forbidden loci that do not involve  $e$ ) to order  $v_1$  before  $v_2$ . The reason is that in this way we create smaller forbidden loci. Thus, the direction of  $e$  is from  $v_1$  to  $v_2$  and its weight is  $w(e) = F_e(v_2) - F_e(v_1)$ , reflecting the benefit of placing  $v_1$  before  $v_2$  (note that  $w(e)$  is positive). If  $F_e(v_2) \leq F_e(v_1)$  the direction of  $e$  and its weight will be analogous. In this computation, we consider all endpoints and intersections in the initial arrangement of  $S$ .

Given a permutation  $\Pi(V)$  on the endpoints, an edge  $e$  is a *forward edge* if it complies with the direction of  $e$  in  $G$  (namely its source appears before its destination in  $\Pi(V)$ ). Similarly, an edge is a *backward edge* (or *backedge*) if its direction does not comply with  $\Pi(V)$ . Note that if an edge is a backedge, its corresponding forbidden loci is larger than the forbidden loci it would have if it was a forward edge. Thus, we would be interested in avoiding as many backedges in  $\Pi(V)$  as possible.

Let  $E_\Pi = \{e \in E \mid \text{The source of } e \text{ appears after}$

<sup>1</sup>In practice, to simplify the computation, we use approximating polygons in a similar way to our computation in Appendix A.3.

<sup>2</sup>By a slight abuse of notation, we use the notations we used to describe the input here too.

its target in  $\Pi(V)$ . We define MIN-SUM formally as follows.

**MIN-SUM:** Given a directed graph  $G = (V, E)$  with positive weights on  $E$ , find a permutation  $\Pi(V)$  that minimizes  $\sum_{e \in E_\Pi} w(e)$ .

It turns out that this problem is NP-complete by observing that it is equivalent to the *Feedback Arc Set* problem (FAS) [8]. In the FAS problem, we are given a directed graph  $G = (V, E)$  with positive weights on  $E$ . The goal is to find a subset  $E' \subseteq E$  with minimal weights such that  $G' = (V, E \setminus E')$  is acyclic. Since  $G'$  is a directly acyclic graph, it defines a topological order on the vertices of  $V$ . Edges which comply with this order are included in  $E \setminus E'$  as the goal is to maximize the total weight of this set. Others are included in  $E'$ . This is identical to our problem in which the backedges, whose sum we want to minimize, are exactly the set  $E'$  in the FAS problem.

There are several heuristics in the literature to approximate FAS. We applied three [6, 5, 2] and adapt them to our MIN-SUM version. We refer to these algorithms by MIN-SUM-1, MIN-SUM-2 and MIN-SUM-3, respectively.

In order to improve the results, we implemented and used the following procedure before applying the FAS approximation algorithms. It is performed after determining the direction of the edges. Let  $S(G) = (S_1, S_2, \dots, S_k)$  be the strongly connected components (SCC) of  $G$ , topologically ordered. Let  $G_i = (S_i, E_i)$  be the subgraph induced by taking the nodes that belong to  $S_i$ ,  $1 \leq i \leq k$ , and the edges that connect two nodes of  $S_i$  (denoted by  $E_i$ ). Let  $\Pi_i$  be an optimal permutation of  $S_i$  for the MIN-SUM problem of  $G_i$ . The next theorem claims that an optimal solution for  $G$  is in the form  $(\Pi_1, \Pi_2, \dots, \Pi_{k-1}, \Pi_k)$ . The proof is postponed to Appendix B.

**Theorem 5.1** *Let  $S(G) = (S_1, S_2, \dots, S_k)$  be the strongly connected components of  $G(V, E)$ , in their topological order. Let  $G_i$  be the subgraph that includes both the vertices of  $S_i$  and the edges connecting them. An optimal permutation for the MIN-SUM problem is in the form  $(\Pi_1, \Pi_2, \dots, \Pi_{k-1}, \Pi_k)$  where  $\Pi_i$  is an optimal permutation of  $G_i$ .*

Following this theorem, we divide the graph into strongly connected components and guarantee that no edge that connects two different strongly connected

components will be a backedge. Then we apply the approximation on each subgraph and merge the results.

## 5.2 Minimizing the Maximum

We consider the problem of minimizing the maximum perturbation. The definition of  $G(V, E)$  in this case is somewhat different from the definition in Section 5.1. Given an edge  $e(v_1, v_2)$ , the definitions of  $F_e(v_1)$  and  $F_e(v_2)$  are the same as in the MIN-SUM problem. We define two weighted halfedges in  $G$ :  $h(v_1, v_2)$  with the weight  $F_e(v_2)$  and  $h(v_2, v_1)$  with the weight  $F_e(v_1)$ . Let  $w(v', v)$  be the weight of the halfedge  $h(v', v)$ . Let  $\Pi(v)$ ,  $v \in V$ , be the place of  $v$  in  $\Pi(V)$ . We use the following formula which sums up the weights of some of the incoming halfedges to  $v$ . The halfedges that are used here are the ones which comply with the permutation  $\Pi(V)$ .

$$w_\Pi(v) = \sum_{\{v' | [v, v'] \in E, \Pi(v') < \Pi(v)\}} w(v', v)$$

It follows that  $w_\Pi(v)$  measures the actual forbidden locus area induced by the edges that are directed to  $v$ .<sup>3</sup> Thus, our objective is to minimize  $\max\{w_\Pi(v) | v \in V\}$ . In other words, considering only halfedges that comply with  $\Pi(V)$ , our goal is to minimize the maximum sum of weights of incoming halfedges to a vertex.

It turns out that this problem is polynomially solvable. For each  $v \in V$ , let  $w(v) = \sum_{v' \neq v} h(v', v)$ . The following is a pseudocode for optimally solving the MIN-MAX problem using a greedy approach in polynomial time.

---

<sup>3</sup>For simplicity, we ignore intersections between corresponding forbidden loci that decrease the actual forbidden locus area, and simply sum up the forbidden locus areas.

## MINIMIZE MAXIMUM PERTURBATION

**Input:** a directed graph  $G = (V, E)$  with positive weights on the edges.

**Output:** a permutation  $\Pi(V)$  that minimizes  $\max\{w_{\Pi}(v) | v \in V\}$ .

1. **foreach**  $v \in V$ , compute  $w(v)$
2. sort  $V$  with respect to  $w(v)$
3. **while**  $V$  is not empty
4.      $v = \text{extractMin}(V)$  /\* Extracting the vertex with the minimum  $w(v)$  \*/
5.     **foreach**  $v' \in V$
6.          $w(v') = w(v') - w(v, v')$
7.     **remove**  $v$  from  $V$  and insert it to the front of  $\Pi$
8. **end while**

We postpone the proof that the algorithm produces an optimal permutation to Appendix C.

The two loops that start in lines **3** and **5** imply that the running time of the algorithm is  $O(n^2)$ , where  $n$  is the number of vertices. There is no need to use auxiliary memory besides  $V$ , thus the space is linear.

Since we use only one algorithm to solve the MIN-MAX problem, we denote it by MIN-MAX too. We implemented all of the above algorithms (MIN-SUM[1-3] and MIN-MAX) and analyze their performance experimentally in Section 6.

**Remark.** Since the smart ordering algorithms are performed prior to the perturbation process, it does not comply with a Controlled Perturbation version of a randomized incremental algorithm because the input is not known in advance. However, there are certain input sets that can still use the ideas of our smart ordering algorithms with a randomized incremental algorithm. One important example is a set of separate line segments. In this case, for each segment we would use our ideas to decide the processing order of its vertices on-line.

We conclude the algorithmic part of this paper with a theorem that summarizes our work.

**Theorem 5.2** *Given an arrangement of  $n$  line segments in  $\mathbb{R}^2$  with  $n$  endpoints, CPLS produces a perturbed arrangement in  $O(n^3 + \Psi(n))$  expected time and  $O(n^2 + \Phi(n))$  working storage where  $\Psi$  and  $\Phi$  are the time and the space the smart ordering algorithm consumes. The output is linear in the size of the input.*

**Proof:** The complexities are derived immediately by analyzing the pseudocode presented in Section 3.  $\square$

In comparison, constructing an arrangement of line segments in the plane takes  $O(n^2)$  time and space. We note that we implemented an optimization technique to decrease the actual time taken by our software significantly (see the beginning of the following section for details).

We finally note that the smart ordering algorithms that we used did not increase the overall time and space complexity.

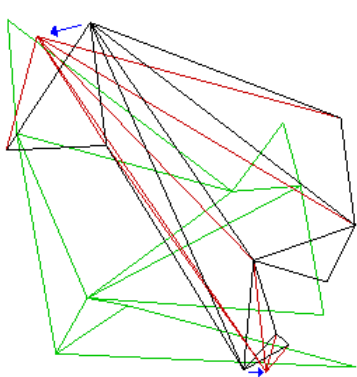
## 6 Experiments

We have implemented our scheme on a PC with Microsoft Visual C++ .NET (version 7.1). We used the OpenGL and CGAL libraries [4]. The experiments were performed on a Microsoft Windows XP workstation with an Intel Pentium 4 3.2 GHz CPU and 2GB of RAM.

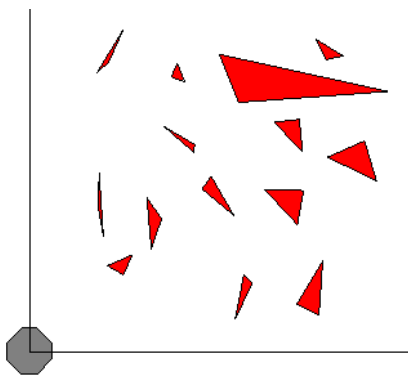
We note that we implemented and used two optimizations to improve the quality and the performance of our software. Both use similar ideas from previous Controlled Perturbation work. The first technique uses a grid and associate features to the cells that occupy them. Then, degeneracy tests consist of only features that lie only within cells that are close enough to induce degeneracies. This technique usually reduces the theoretic cubic time complexity which relates to rare cases in which many endpoints are involved in  $O(n^2)$  potential degeneracies. The goal of the second technique is to decrease the perturbation. Its idea is to start with a small perturbation radius which is much smaller than  $\delta$  (which in practice is usually a crude upper bound), and gradually increase it if a valid location is not obtained.

We performed many experiments to evaluate our heuristics. In these experiments we fit the resolution bounds to make the results interesting. They were big enough to create potential degeneracies, but small enough to obtain reasonable perturbations. This fit can be viewed as scaling up the input. Figure 4 depicts seven different input sets before and after perturbation.

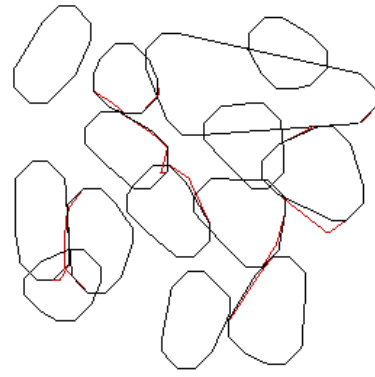
We implemented and experimented with the four algorithms we present in Section 5 (MIN-MAX, MIN-SUM-1, MIN-SUM-2 and MIN-SUM-3). We denote by RAN our scheme when random permutation is used. On a whole, the deviation magnitude obtained with all four smart ordering heuristics were very similar in all experiments. It turned out that for minimizing the



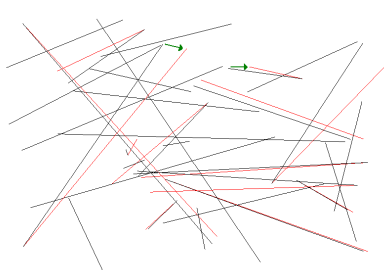
(a) Two triangulation overlay



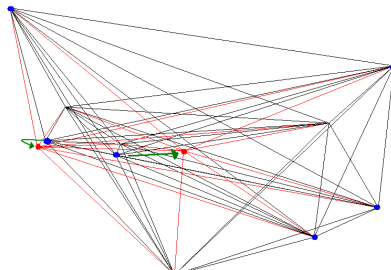
(b) A robot  $R$  (in grey) and a set of obstacles  $Q$



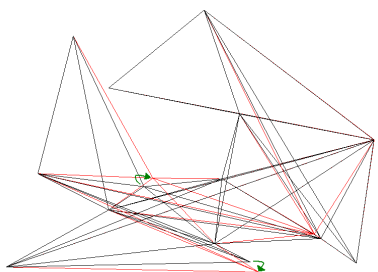
(c)  $R \oplus Q$



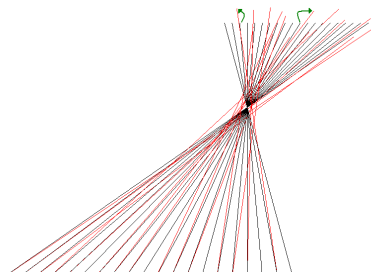
(d) Random line segments



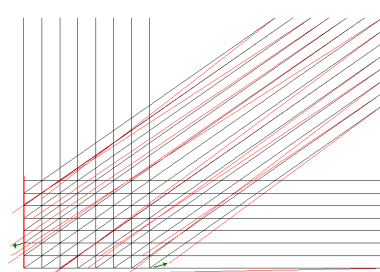
(e) Random clique



(f) Visibility graph inside a polygon

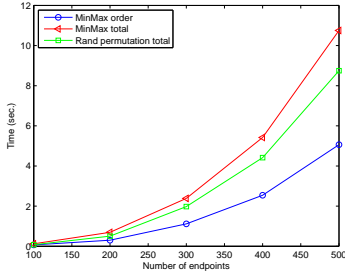


(g) Double wedge

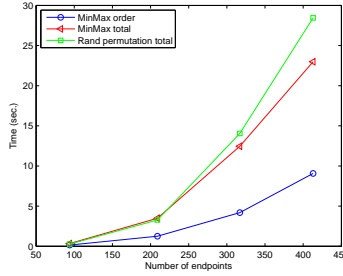


(h) Structured segments

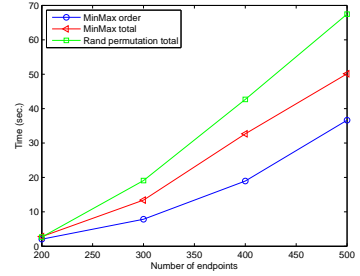
Figure 4: Experiment snapshots obtained with our software on different input sets. For visualization purposes, the input size is smaller than the sizes of our experiment data. The black edges are the input (but in (a) one triangulation is in black and the other is on green). Some of them were perturbed and their perturbed edges are painted in red. For convenience, some of the perturbations were arrowed. Subfigures (b) and (c) show a step in the robot motion planning problem. Subfigure (b) is the input, and Subfigure (c) shows the CP output of the Minkowski sum of the symmetric robot and the obstacles. For clarity, the input and output endpoints in Subfigure (e) are emphasized in blue and red respectively.



(a) Random line segments



(b) Robot motion planning



(c) Triangulation overlay

Figure 5: Time as a function of the input size. The data correspond to the average of many executions. We report the total time for RAN and MIN-MAX. For the latter, we also show the time it took to decide the order.

maximum perturbation, the MIN-MAX algorithm performed a little better than the other three on average. MIN-SUM-3 showed a small advantage for minimizing the sum in many experiments.

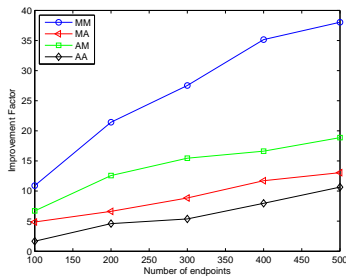
There are two opposite effects on the processing time when using smart ordering heuristics. While they consume extra time to compute, they save time in the perturbation process because they result in less trials to perform in order to find valid placements. It turned out that MIN-MAX was always faster than the rest of the smart ordering heuristics. Thus, it should be preferred if the processing time is important, as its result were similar to the others. Figure 5 compares the average time obtained with MIN-MAX and RAN. While in the random line segment experiments (Figure 5(c)), MIN-MAX increased the total processing time, it decreased the total processing when working on triangulation overlay and robot motion planning computations (Figure 5(a) and (b) respectively). The difference was smaller than 40% and usually much smaller. We note that as the tests had more involved degeneracies to solve and thus their total processing time increased, MIN-MAX became more effective than RAN in terms of time (this is well illustrated in Figure 5). Thus, MIN-MAX actually decreased the time of long computations. The other heuristics took more time than MIN-MAX to process, but generally run for less than double the time taken with RAN.

Based on the discussion above, we used MIN-MAX to show the improvements in the perturbation magnitudes against RAN. The graphs (a), (b) and (c) in Figure 6 correspond to the experiments illustrated in Figure 4(a),(c) and (d), respectively. The graphs show by how much MIN-MAX decreased the perturbation that was achieved with RAN. The graphs show that

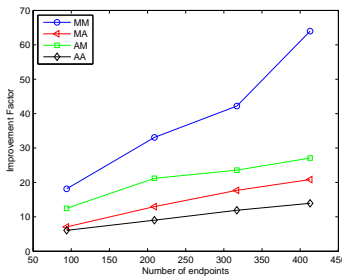
we experienced improvements, especially for decreasing the maximum perturbation. The improvements became more significant as the tests became more dense, thus having more involved degeneracies to solve and bigger perturbations. We emphasize that this is a desirable property, as it becomes more important to decrease the perturbation magnitudes when they become larger. It is worth noting that the improvement can be much bigger. For example, consider the case in Figure 3. If the forbidden discs move very close to  $b$  and RAN makes a bad decision (processing  $b$  before  $a$ ), the resulting perturbation will be huge in comparison to cases where the right choice is made. Figure 4(g) and (h) illustrate examples with involved potential degeneracies that result in huge perturbations if the wrong order is chosen—using our smart heuristics we always obtained good orders. In Figure 4 we show the results when using any of the smart heuristics, thus showing which vertices should be processed later (the ones that were perturbed).

## 7 Conclusions and Future Work

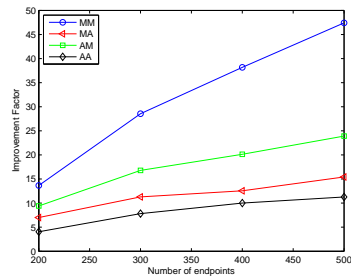
We presented a Controlled Perturbation scheme for sets of line segments in  $\mathbb{R}^2$ . The main drawback of the Controlled Perturbation paradigm is the possibility of large perturbation. We presented a novel technique to decrease the perturbation magnitude. The idea is to decide a smart ordering on the endpoint processing to decrease the perturbation magnitude. We designed and implemented several smart ordering algorithms. Our experiments showed improvements over using random order. We emphasize that the improvement factor (with regards to the output deviation) increases as the potential degeneracies become more involved and re-



(a) Random line segments



(b) Robot motion planning



(c) Triangulation overlay

Figure 6: Results of using MIN-MAX against RAN. The X-axis corresponds to the size of the input and the Y-axis corresponds to the factor of improvement that MIN-MAX achieved, compared to using RAN. The graphs in (a) correspond to tests with overlay of triangulations of polygons with 100 vertices. We performed many tests for each case. We illustrate the maximum perturbation and average perturbation improvements. For any kind of experiment, let  $k$  be the number of tests we performed. Let  $A_i$  and  $M_i$  be the average and maximum perturbation obtained in the  $i$ -th test, respectively. Then  $MM = \max_{1 \leq i \leq k} M_i$ ,  $MA = \max_{1 \leq i \leq k} A_i$ ,  $AM = \frac{\sum_{1 \leq i \leq k} M_i}{k}$  and  $AA = \frac{\sum_{1 \leq i \leq k} A_i}{k}$

quire more perturbation. Such cases that result in big perturbations are a serious problem for any Controlled Perturbation algorithm. Thus, using smart ordering could make the difference between a good and poor quality of the output.

We note that the idea of ordering the feature processing is general and may be used in other Controlled Perturbation schemes.

Our ordering heuristics depend on the order of two adjacent endpoints. It would be interesting to design more sophisticated ordering techniques that consider more involved relationships (for example, the relationship among three intersecting segments).

Another challenging direction is to develop graph algorithms to provably minimize the perturbation (or approximate the minimum perturbation), given the minimum required separation.

We believe that our scheme can be used for real-world applications that need to separate geometric features. As an example we give the following problem from the Graph Drawing field. We are given a graph  $G$  and we wish to draw it with captions surrounded by rectangles on the endpoints of  $G$ . For visualization purposes, the goal is to guarantee that no non-adjacent edges cross the caption rectangles. If we define any such case to be degenerate, our scheme can achieve this goal by perturbing the endpoints in a similar way we describe in this report.

An interesting experimental work is to compare the performance with exact arithmetic. The idea is to com-

pare the construction of arrangements of line segments (or any other algorithm that works on sets of line segments) with exact arithmetic against perturbing the input with our algorithm and then constructing the arrangement using fixed-precision datatypes.

## References

- [1] M. Abellanas, F. Hurtado, and P. Ramos. Tolerance of geometric structures. In *Proc. 6th Canad. Conf. Comput. Geom.*, pages 250–255, 1994.
- [2] N. Ailon, M. Charikar, and A. Newman. Aggregating inconsistent information: ranking and clustering. In *STOC '05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 684–693, New York, NY, USA, 2005. ACM Press.
- [3] L.-E. Andersson, S. M. Dorney, T. J. Peters, and N. F. Stewart. Polyhedral perturbations that preserve topological form. *Comput. Aided Geom. Design*, 12(8):785–799, Dec. 1995.
- [4] *The CGAL User Manual, Version 3.31*, 2007. [www.cgal.org](http://www.cgal.org).
- [5] D. Coppersmith, L. Fleischer, and A. Rudra. Ordering by weighted number of wins gives a good ranking for weighted tournaments. In *SODA '06: Proceedings of the seventeenth annual ACM-SIAM*

*symposium on Discrete algorithm*, pages 776–782, New York, NY, USA, 2006. ACM Press.

*book of Discrete and Computational Geometry*, chapter 41, pages 927–952. 2004.

- [6] C. Demetrescu and I. Finocchi. Combinatorial algorithms for feedback problems in directed graphs. *Inf. Process. Lett.*, 86(3):129–136, 2003.
- [7] S. Funke, C. Klein, K. Mehlhorn, and S. Schmitt. Controlled perturbation for delaunay triangulations. In *SODA '05: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1047–1056, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics.
- [8] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman And Company, New York, 1979.
- [9] L. J. Guibas, D. Salesin, and J. Stolfi. Epsilon geometry: building robust algorithms from imprecise computations. In *Proc. 5th Annu. ACM Sympos. Comput. Geom.*, pages 208–217, 1989.
- [10] D. Halperin and E. Leiserowitz. Controlled perturbation for arrangements of circles. *International Journal of Computational Geometry and Applications*, 14(4 and 5):277–310, 2004. Special issue, papers from SoCG 2003.
- [11] D. Halperin and C. R. Shelton. A perturbation scheme for spherical arrangements with application to molecular modeling. *Comput. Geom. Theory Appl.*, 10:273–287, 1998.
- [12] K. Mehlhorn and R. Osbild. Reliable and efficient computational geometry via controlled perturbation. *33rd International Colloquium on Automata, Languages and Programming*, 2006.
- [13] T. Ono. An eight-way perturbation technique for the three-dimensional convex hull. In *Proc. 6th Canad. Conf. Comput. Geom.*, pages 159–164, 1994.
- [14] S. Raab. Controlled perturbation of arrangements of polyhedral surfaces with application to swept volumes. In *Proc. 15th Annu. ACM Sympos. Comput. Geom.*, pages 163–172, 1999.
- [15] C. K. Yap. Robust geometric computation. In J. E. Goodman and J. O'Rourke, editors, *Hand-*

## A Computing the Perturbation Magnitudes

We start by formalizing the relationship between the perturbation radius, resolution bounds and the input. We formalize this idea next.

**Definition A.1** For any  $\lambda > 0$ , an input set is  $\lambda$ -approximated for CPLS if and only if  $\delta/l \leq \lambda$  where  $\delta$  is the perturbation radius and  $l$  is the length of the shortest line segment.

The idea is that the input must be  $\lambda$ -approximated to satisfy our analysis. If it is not, one should resort to larger  $\lambda$ 's. We note that finding a suitable  $\lambda$  could be either implemented using a binary search or be a user parameter. As we show later,  $\delta$  is larger than all the resolution bounds. Thus, if the input is  $\lambda$ -approximated then for each resolution bound  $\varepsilon$ ,  $\frac{\varepsilon}{\lambda} \leq \lambda$ .

In this section we derive an upper bound on  $\delta$ , the perturbation radius of our algorithm. Recall that this bound guarantees a possibility of at least  $\frac{1}{2}$  in finding degeneracy-free placement when perturbing a vertex randomly.

Let  $V = \{v_1, v_2, \dots, v_n\}$  be the input endpoints ordered using the algorithms in Section 5. We denote by  $v_i \in V$ ,  $1 \leq i \leq n$ , the endpoint that is currently processed. Let  $v'_i$  be the final placement of  $v_i$ , possibly perturbed. We denote by  $e$  a segment whose two endpoints have already been processed. For an endpoint  $v_i$ , let  $E_i$  and  $V_i$  be the set of all edges and endpoints that are processed by the time  $v_i$  is fully processed. Let  $V'_i = \{v'_1, v'_2, \dots, v'_i\}$  be the set of the final placements of the first  $i$  input endpoints.  $P(v)$  is the perturbation disc of  $v$ .

We maintain five resolution bounds. The reason for having several resolution bounds will be clear later. We denote them by  $\varepsilon_1 - \varepsilon_5$ . As we show later, the first four are ordered from large to small.

It is important to note that each atomic test for potential degeneracy takes  $O(1)$  time as it involves constant number of operations.

In the rest of this section we compute the value of  $\delta$  by considering the different potential degeneracy kinds and computing upper bounds on the area that they can occupy. Our analysis is from the point of view of some endpoint  $v_i$ , that is currently processed.

### A.1 Computing Forbidden Loci induced by Endpoints

Let  $\varepsilon_1$  be the minimum separation between two endpoints. Then, each  $v_j | j \neq i$  defines a forbidden disc of radius  $\varepsilon_1$  for  $v_i$ . Since at most  $2n$  endpoints may induce forbidden discs, we can bound the total forbidden loci of this type by

$$F_1 = 2n\pi\varepsilon_1^2 \quad (1)$$

### A.2 Computing Forbidden Loci Induced by $v_i$ and a Non-incident Edge

Each  $e \in E_{i-1}$  defines a forbidden locus for  $v_i$ . This region is the Minkowski sum of  $e$  and a disc centered at the origin with radius  $\varepsilon_2$ . After processing  $v_i$ , we are guaranteed that the separation between  $v_i$  and any non-incident edge (thus also between  $v_i$  and vertices along the edge) is at least  $\varepsilon_2$ . It is easy to show that the maximum area which the forbidden locus can cut from  $P(v_i)$  is when  $e$  passes through  $v_i$  and intersects  $P(v_i)$  twice. This area is bounded by a rectangle whose area is  $2\varepsilon_2 \times 2\delta$  (the rectangle  $abcd$  in Figure 7). There is an upper bound of  $n$  segments defining such loci. Let  $F_2$  be the maximum total forbidden loci in this case. We get

$$F_2 = 4n\varepsilon_2\delta \quad (2)$$

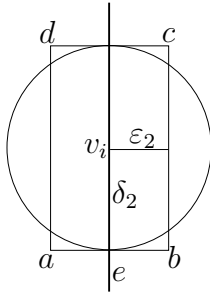


Figure 7: Forbidden loci induced by  $v_i$  and  $e \in E_{i-1}$

### A.3 Computing Forbidden Loci Induced by Intersections of Edges and Edges incident to $v_i$

Let  $e$  be an edge incident to  $v_i$ , such that the other endpoint of  $e$  has already been processed. Let  $v'$  be

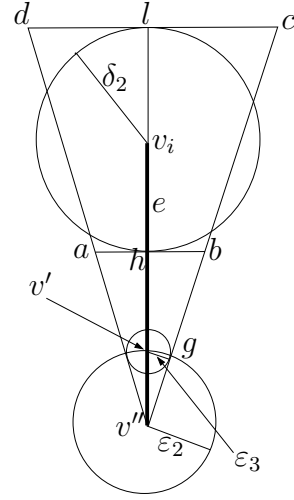


Figure 8: Forbidden loci of an intersection of two segments

an intersection point of the output of two edges (non-incident to  $v_i$ ) of  $E_{i-1}$ . We maintain a separation of  $\varepsilon_3$  between  $e'$  (the output of  $e$ ) and  $v'$  (thus also between  $e'$  and any vertex on  $e$ ). Here,  $e$  must not penetrate the disc of radius  $\varepsilon_3$  centered at  $v'$ . This is illustrated in Figure 8, where  $e$  is the thick line whose first endpoint,  $v''$ , has already been processed. The intersection point,  $v'$ , must be at least  $\varepsilon_3$  away from  $e$  in order not to induce potential degeneracies (we explain later why we place  $v'$  at the intersection between  $e$  and the disc of radius  $\varepsilon_2$  around  $v''$ ). It follows that in order to prevent  $e'$  (the output of  $e$ ) from penetrating the disc with radius  $\varepsilon_3$  around  $v'$ ,  $v'_i$  (the output of  $v_i$ ) must not be located inside the wedge  $dv''c$ . It defines a quadrilateral which bounds the part of the forbidden locus inside  $P(v_i)$  (quadrilateral  $abcd$  in Figure 8). The quadrilateral has maximum area when  $v'$  is located on  $e$  and on the disc with radius  $\varepsilon_2$  around  $v''$  (the later disc contains no non-incident segments since  $v''$  is the result of processing an endpoint) and when the length of  $e$  is maximum ( $L + \delta$ ). In this case  $abcd$  is a trapezoid. This explains the placement of  $v'$ . Note that  $g$  in Figure 8 is the place where the segment  $v''c$  is tangent to the disc centered at  $v'$ . We denote by  $D_1$  the maximum area of the trapezoid  $abcd$  (which bounds the forbidden locus). Next we compute its magnitude.

$$\begin{aligned} \Delta v''v'g &\approx \Delta v''hb \approx \Delta v''Lc \\ \frac{\varepsilon_3}{|v''g|} &= \frac{|\overline{ab}|/2}{L} = \frac{|\overline{dc}|/2}{L + 2\delta} \\ |\overline{ab}| &= \frac{2\varepsilon_3 L}{|v''g|} \end{aligned}$$

$$\begin{aligned} |\overline{dc}| &= \frac{2\varepsilon_3(L + 2\delta)}{|v''g|} \\ D_1 &= (|\overline{ab}| + |\overline{dc}|)\delta \end{aligned}$$

$$D_1 = \frac{4\varepsilon_3\delta(L + \delta)}{\sqrt{\varepsilon_2^2 - \varepsilon_3^2}} \quad (3)$$

Now it should be clear that  $\varepsilon_3$  must be much smaller than  $\varepsilon_2$ : if  $\varepsilon_3$  is not much smaller than  $\varepsilon_2$  then in Figure 8,  $\angle dv''c$  may be large. Thus, the size of the trapezoid  $abcd$  may be unacceptably large. On the other hand, making  $\varepsilon_2$  too much larger than  $\varepsilon_3$  means that  $\varepsilon_2$  would be very large itself, resulting in a large perturbation for the first endpoint—see Equation 2.

We next coordinate between  $\varepsilon_3$  and  $\varepsilon_2$  in order to compute  $\delta$  in terms of the input parameters. Let  $R$  be the ratio  $\frac{\varepsilon_2}{\varepsilon_3}$ . Then

$$\varepsilon_2 = R\varepsilon_3 \quad (4)$$

We set  $R$  by trying to minimize  $\delta$  (done by analyzing the equation of  $\delta$  and omitted from this report).

There are  $\binom{n}{2} = \frac{n(n-1)}{2}$  possible intersections that may take place here. Also, processing  $v_i$  may involve the processing of  $O(n)$  incident edges. Thus, the total forbidden loci is bounded by

$$F_3 = \frac{2n^2(n-1)\delta(L + \delta)}{\sqrt{R^2 - 1}}$$

Since the input set should be  $\lambda$ -approximated

$$F_3 = \frac{2n^2(n-1)\delta(L + \lambda l)}{\sqrt{R^2 - 1}} \quad (5)$$

Since  $\varepsilon_2$  must be larger than  $\varepsilon_3$ , the square root in equation 5 is real.

#### A.4 Computing Forbidden Loci Induced by $V'_{i-1}$ and Edges incident to $v_i$

This case is similar to the case in the previous subsection. The only difference is that we must maintain an empty disc with radius  $\varepsilon_2$  around any  $v' \in V'_{i-1}$  for future perturbations (note that in the previous subsection the disc around  $v''$  with radius  $\varepsilon_2$  had to be empty of non-incident segments). Here we assume for simplicity that all such endpoints are incident to edges that have not been processed yet. Thus, there is a separation of at least  $\varepsilon_2$  between  $v'$  and both  $e'$  and any

vertex on  $e'$ . Recall that we maintain a separation of at least  $\varepsilon_1$  between any pair of endpoints. The analysis of this case is similar to the one in the previous subsection (see Figure 9 for an illustration):

$$\begin{aligned} \Delta v''v'g &\approx \Delta v''hb \approx \Delta v''Lc \\ \frac{\varepsilon_2}{|v''g|} &= \frac{|\overline{ab}|/2}{L} = \frac{|\overline{dc}|/2}{L + 2\delta} \\ |\overline{ab}| &= \frac{2\varepsilon_2 L}{|v''g|} \\ |\overline{dc}| &= \frac{2\varepsilon_2(L + 2\delta)}{|v''g|} \\ D_2 &= (|\overline{ab}| + |\overline{dc}|)\delta \end{aligned}$$

$$D_2 = \frac{4\varepsilon_2\delta(L + \delta)}{\sqrt{\varepsilon_1^2 - \varepsilon_2^2}} \quad (6)$$

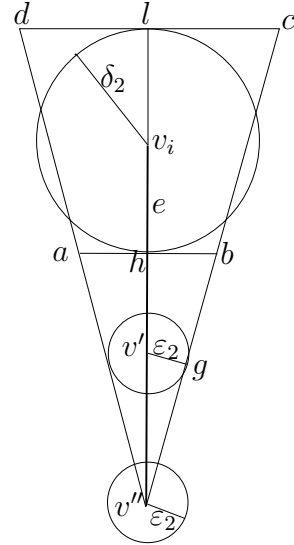


Figure 9: Forbidden loci of an endpoint

There are  $n$  possible incident edges and the test for each may involve testing  $2n$  endpoints. Thus, the total area of the forbidden loci in this case is bounded by

$$\frac{8n^2\varepsilon_2\delta(L + \delta)}{\sqrt{\varepsilon_1^2 - \varepsilon_2^2}}$$

and also by

$$F_3 = \frac{8n^2\varepsilon_2\delta(L + \lambda l)}{\sqrt{\varepsilon_1^2 - \varepsilon_2^2}} \quad (7)$$

For simplicity, we require that the angles  $dv''c$  in Figures 8 and 7 be equal. This determines the value of  $\varepsilon_1$ . From triangle similarity we get

$$R = \frac{\varepsilon_1}{\varepsilon_2} = \frac{\varepsilon_2}{\varepsilon_3}$$

$$\varepsilon_1 = \varepsilon_2 R \quad (8)$$

Then by using equations 8 in equation 7 we get

$$F_4 = \frac{8n^2\delta(L + \lambda l)}{\sqrt{R^2 - 1}} \quad (9)$$

#### A.4.1 A Lower Bound on another kind of distance between intersection point and edge

Let  $e_i$  be an edge incident to  $v_i$  whose other endpoint has already been processed. Let  $e'_j$  and  $e'_k$  be the output of two edges, where  $e'_j$  intersects  $e'_i$  ( $e_i$  after processing) at point  $f$  (see Figure 10). We next argue that if no other potential degeneracies are induced during the processing of  $e_i$  (as our algorithm guarantees), then a potential degeneracy of type intersection-edge involving  $f$  and  $e'_k$  cannot exist as well (and thus also no potential degeneracy between  $f$  and another intersection). We do so, by giving a lower bound on the distance between  $f$  and  $e'_k$ ; we denote this lower bound by  $\varepsilon_4$ .

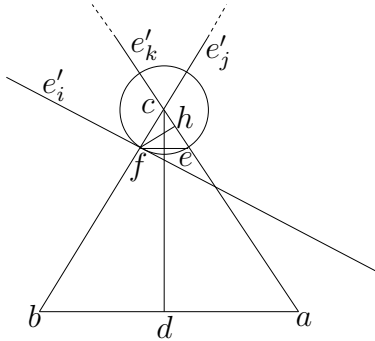


Figure 10: Minimum distance between  $f$  and  $e'_k$  when  $e'_j$  and  $e'_k$  intersect

We differentiate between two cases. The first one is when  $e'_j$  and  $e'_k$  do not intersect. Since the endpoints of  $e'_j$  are at least within distance of  $\varepsilon_2$  from  $e'_k$ , also any point on  $e'_j$  is at least as far from  $e'_k$  as these endpoints.

Then the distance between  $f$  to  $e'_k$  is at least  $\varepsilon_2$  and thus no potential degeneracy is induced.

The second case ( $e'_j$  and  $e'_k$  intersect) is illustrated in Figure 10.  $e'_j$  and  $e'_k$  force  $e'_i$  not to penetrate the disc  $C$ , centered at their intersection,  $c$ , with radius  $\varepsilon_3$ . Thus,  $f$ , which lies on  $e'_j$ , would be closest to  $e'_k$  if it is placed on the boundary of  $C$ . Moreover, the smaller the angle  $\angle bca$  (denoted by  $\alpha$ ) is, the smaller  $\varepsilon_4$  (the distance between  $f$  and  $e'_k$ ) is. In Figure 10,  $\varepsilon_4$  is the size of  $f\bar{h}$ .  $\alpha$  has minimum value when  $e'_j$  and  $e'_k$  have maximum length below  $C$  (bounded by  $L + 2\delta$ ) and the distance between their lower endpoints ( $a$  and  $b$  are the endpoints in the figure) is  $\varepsilon_2$  (the minimum distance between an endpoint and a non-incident edge). We next compute a lower bound on the length of the segment  $\overline{ef}$ .

By triangle similarity

$$\begin{aligned} \frac{|\overline{ef}|}{\varepsilon_2} &= \frac{\varepsilon_3}{L + 2\delta} \\ |\overline{ef}| &= \frac{\varepsilon_2\varepsilon_3}{L + 2\delta} \\ |\overline{ef}| &= \frac{R\varepsilon_3^2}{L + 2\delta} \end{aligned}$$

Let  $\sigma$  denote any resolution bound or perturbation radius. Then  $\sigma \leq \lambda l$ . Together with a simple trigonometric observation in Figure 10, we get

$$\begin{aligned} \cos\left(\frac{\pi}{2} - \frac{\alpha}{2}\right) &= \frac{\varepsilon_2/2}{L + 2\delta} \\ \cos^2\left(\frac{\pi}{2} - \frac{\alpha}{2}\right) &= \frac{\varepsilon_2^2}{4(L + 2\delta)^2} \\ \sin^2\left(\frac{\pi}{2} - \frac{\alpha}{2}\right) &= 1 - \frac{\varepsilon_2^2}{4(L + 2\delta)^2} \geq \\ 1 - \frac{(\lambda l)^2}{4(L + 2\delta)^2} &\geq 1 - \frac{(\lambda l)^2}{4L^2} \\ \sin\left(\frac{\pi}{2} - \frac{\alpha}{2}\right) &\geq \sqrt{1 - \frac{(\lambda l)^2}{4L^2}} \end{aligned}$$

In the same figure we also have

$$\begin{aligned} \sin\left(\frac{\pi}{2} - \frac{\alpha}{2}\right) &= \varepsilon_4/|\overline{ef}| \\ \varepsilon_4 &\geq |\overline{ef}| \sqrt{1 - \frac{(\lambda l)^2}{4L^2}} \\ \varepsilon_4 &\geq \frac{R\varepsilon_3^2 \sqrt{1 - \frac{(\lambda l)^2}{4L^2}}}{L + 2\delta} \end{aligned}$$

$$\varepsilon_4 \geq \frac{R\varepsilon_3^2 \sqrt{1 - \frac{(\lambda)^2}{4L^2}}}{L + 2\lambda} \quad (10)$$

Inequality 10 imposes a lower bound on  $\varepsilon_4$ .

For the same reasons it follows that  $\varepsilon_4$  is also a lower bound on the distance between any pair of intersections along  $e'_i$ .

The next lemma argues that  $\varepsilon_4$  is the smallest resolution bound.

**Lemma A.2**  $\varepsilon_4 < \varepsilon_3 < \varepsilon_2 < \varepsilon_1$

**Proof:** Since we fixed  $\varepsilon_3 < \varepsilon_2 < \varepsilon_1$ , we only have to prove that  $\varepsilon_4 < \varepsilon_3$ . Consider Figure 10. According to the construction above, the lengths of  $\overline{ca}$  and  $\overline{cb}$  are in the magnitude of  $l$ , and the length of  $\overline{ab}$  equals  $\varepsilon_2$ . If the input set is  $\lambda$ -approximated, then  $\angle bca$  will be sufficiently small so that  $\varepsilon_4$  (the length of segment  $\overline{fh}$ ) is smaller than  $\varepsilon_3$  (the length of segment  $\overline{ce}$ ).  $\square$

Since  $\varepsilon_4$  is the smallest among  $\{\varepsilon_1, \dots, \varepsilon_4\}$ , it depends on the input and the machine precision. On the other hand,  $\varepsilon_1$ ,  $\varepsilon_2$  and  $\varepsilon_3$  depend on  $\varepsilon_4$  and on the input, but not on the machine precision. Note that  $\varepsilon_4$  will be the maximum between two resolution bounds which involve the separation of a vertex and a non-incident edge and a separation between two vertices. Since  $\varepsilon_2$ ,  $\varepsilon_3$  and  $\varepsilon_4$  are larger than  $\varepsilon_2$ , their corresponding potential degeneracies will be solved for both potential degeneracy types.

Since we have a lower bound on  $\varepsilon_4$ , we can ignore the potential degeneracy we discuss here when perturbing endpoints although it affects the magnitudes of the resolution bounds and the perturbation radius as we show later.

We need to compute  $\varepsilon_3$  as a function of the input parameters and  $\varepsilon_4$ . If we transfer inequality 10 to an equation, we obtain a relationship between  $\varepsilon_3$  and  $\varepsilon_4$ .

$$\varepsilon_3 = \sqrt{\frac{(L + 2\lambda)\varepsilon_4}{R\sqrt{1 - \frac{(\lambda)^2}{4L^2}}}} \quad (11)$$

#### A.4.2 Computing the Forbidden Loci Induced by Endpoints for Robust Distance Computation

One of the tests we perform involves the computation of the distance between a point  $p$  and a segment  $s$ .

First we project  $p$  onto the line containing  $s$  (denote the projected point by  $q$ ). If  $q$  lies on  $s$  (including its endpoints) then the output is the distance between  $p$  and  $q$ . Otherwise, the output is the distance between  $p$  and one of the endpoints of  $s$  (the closer one to  $p$ ). Thus, we need to decide which one of these distances to consider. In order to have a robust decision,  $q$  has to be far enough from both endpoints of  $e$ . Alternatively,  $p$  has to be sufficiently far from the lines passing through the endpoints of  $e$  and perpendicular to it. These two lines introduce forbidden loci for  $v_i$  with a resolution bound denoted by  $\varepsilon_5$  (see Figure 11). They depend on the machine precision. The forbidden locus structure is similar to the one described in Section A.2. Since there are at most  $2n$  endpoints that may induce forbidden loci for  $v_i$ , the sum of forbidden loci in this case is bounded by

$$F_5 = 8n\varepsilon_5\delta \quad (12)$$

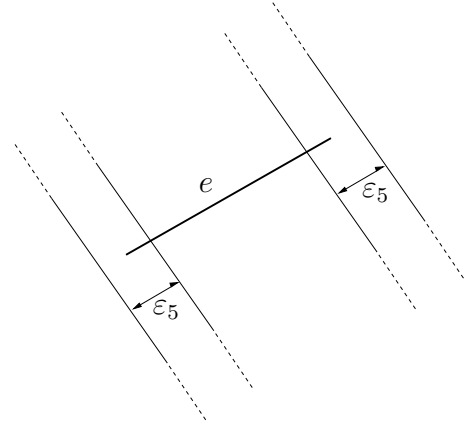


Figure 11: Forbidden loci induced by the endpoints of edge  $e$  for robust computation of a distance from a point. The forbidden loci are the infinite strips with width  $2\varepsilon_5$  that are perpendicular to  $e$  such that the endpoints of  $e$  lie on its medial axis

#### A.5 Computing $\delta$

Recall that we want the perturbation disc area to be twice as large as the total area of all the forbidden loci. Thus,  $\pi\delta^2 = 2(F_1 + F_2 + F_3 + F_4 + F_5)$ . By using formulas 1, 2, 4, 5, 8, 9, 11 and 12 we get the following quadratic equation on  $\delta$ :

$$a\delta^2 + b\delta + c = 0$$

where

$$\begin{aligned}
a &= \pi \\
b &= -4n(R\varepsilon_3 + 2\varepsilon_5) - \frac{2(L + \lambda)n^2(n + 3)}{\sqrt{R^2 - 1}} \\
c &= -2n\pi R^4 \varepsilon_3^2 \\
\varepsilon_3 &= \sqrt{\frac{(L + 2\lambda)\varepsilon_4}{R\sqrt{1 - \frac{(\lambda)^2}{4L^2}}}}
\end{aligned}$$

It is easy to verify that this equation has two real roots, one positive and one negative. Thus, the value of  $\delta$  is the positive one.

## B Strongly Connected Component Proof (Section 5.1)

We present a proof of theorem 5.1. The idea is that there must be an optimal solution in which the elements can be partitioned into groups that correspond to the strongly connected components.

**Theorem B.1** *Let  $S(G) = (S_1, S_2, \dots, S_k)$  be the strongly connected components of  $G(V, E)$ , in their topological order. Let  $G_i$  be the subgraph that includes the vertices of  $S_i$  and the edges connecting them. An optimal permutation for the MIN-SUM problem is in the form  $(\Pi_1, \Pi_2, \dots, \Pi_{k-1}, \Pi_k)$  where  $\Pi_i$  is an optimal permutation of  $G_i$ .*

**Proof:** Let  $SCC(v)$  be the strongly connected component (SCC) of any  $v \in V$ . Let  $\Pi'$  be an optimal permutation for the FAS problem and suppose it is not of the form  $(\Pi_1, \Pi_2, \dots, \Pi_{k-1}, \Pi_k)$ . We iteratively swap adjacent nodes in  $\Pi'$ ,  $v_1$  and  $v_2$ , where  $v_1$  appears before  $v_2$  in  $\Pi'$ , but  $SCC(v_1)$  is topologically after  $SCC(v_2)$ . By doing so, we are guaranteed that we do not harm  $\sum_{e \in E_\Pi} w(e)$ , because there is no directed edge from  $v_1$  to  $v_2$ . We continue in this fashion until there are no such adjacent nodes to swap. It follows that the final permutation is in the form  $(\Pi'_1, \Pi'_2, \dots, \Pi'_{k-1}, \Pi'_k)$  where  $\Pi'_i$  is a permutation of subgraph  $G_i$ . Note that at this point there are no backedges between any two subgraphs. Thus, there are no cycles that consist of nodes of more than one strongly connected component. Hence, we can analyze each subgraph independently. If  $\Pi'_i$  is not an optimal permutation of  $G_i$ , then we replace it with an optimal one,  $\Pi_i$ , and the result may only be improved. The claim follows.  $\square$

## C Proving that the Greedy Algorithm Produces an Optimal Solution (Section 5.2)

We present a proof that belongs to Section 5.2. The notations that are not defined here, are defined in that section.

We prove that our greedy algorithm finds an optimal solution. Let  $\Pi^i(V)$  be the last  $i$  elements in  $\Pi(V)$  and  $\Pi_i(V)$  be the first  $n - i$  elements of  $\Pi(V)$ , where  $n = |V|$ .

**Lemma C.1** *The values of  $w_\Pi(v) : v \in \Pi^i(V)$  do not depend on the order of  $\Pi_i(V)$ .*

**Proof:** According to the greedy algorithm, the elements of  $\Pi^i(V)$  are chosen before the elements of  $\Pi_i(V)$ , and thus appear after them in  $\Pi_i(V)$ . Therefore, the values of  $w_\Pi(v) : v \in \Pi^i(V)$  include all the weights of the halfedges directed from the elements of  $\Pi_i(V)$ . Moreover, the order of the elements of  $\Pi^i(V)$  is set by the time the algorithm works on the elements of  $\Pi_i(V)$ . From the above two observations, the values of  $w_\Pi(v) : v \in \Pi^i(V)$  are finalized by the time the algorithm determines the order of the elements in  $\Pi_i(V)$ . The claim follows.  $\square$

**Corollary C.2** *The optimal solution contains within it optimal solutions to subproblems (the Optimal Substructure property).*

For any permutation  $\Pi(V)$ , let  $\Xi(\Pi(V)) = \max\{w_\Pi(v) | v \in V\}$ . In other words,  $\Xi(\Pi(V))$  is the value of the MIN-MAX problem. We are now ready to prove the main idea.

**Theorem C.3** **MINIMIZE MAXIMUM PERTURBATION** *finds an order which minimizes the maximum  $w_\Pi(v)$  for all  $v \in V$ .*

**Proof:** Let  $OPT$  denote an optimal permutation and  $GREEDY$  denote the permutation obtained with our algorithm. Our goal is to prove that  $\Xi(OPT) = \Xi(GREEDY)$ . We prove it inductively by showing that whenever an element of  $OPT$  does not satisfy the greedy selection, we can safely replace the corresponding element by the element that would have been chosen with a greedy selection and still be optimal. Suppose  $OPT$  satisfies the greedy selection for its last  $i$  elements,  $1 \leq i \leq n - 1$  (the first  $i$  elements to be chosen). Further, suppose that it differs from the greedy

algorithm in the  $(n - i)$ 'th element (the one just before the last  $i$  elements). Let  $OPT_i$  be the first  $n - i$  elements of  $OPT$ . Following Corollary C.2,  $OPT_i$  is an optimal solution for the subproblem of  $OPT$  that involves the first  $n - i$  elements. Let  $v'$  be the last element of  $OPT_i$  and let  $v_{min}$  be the element that would be chosen by our greedy algorithm instead of  $v'$ . Let  $OPT'$  be defined as follows. Its last  $i$  elements are identical to the last  $i$  elements of  $OPT$  (in the same order). Its  $(n - i)$ 'th element is  $v_{min}$ . Finally, its first  $n - i - 1$  elements (we denote this list by  $\Psi$ ) are the same as the element of  $OPT_i$ , except of  $v_{min}$  which is removed (again, in the same order). From the construction of  $OPT'$  we get that

$$w_{OPT'}(v_{min}) < w_{OPT_i}(v') \leq \Xi(OPT_i) \quad (13)$$

Since  $\Psi$  is obtained by removing  $v_{min}$  from  $OPT_i$ ,  $w_{\Psi}(v) \leq w_{OPT_i}(v) : v \in \Psi$ . The reason is that by removing  $v_{min}$ , the sum of the weights of the endpoints are decreased by the weights of halfedges whose source is  $v_{min}$ . We get that

$$\Xi(\Psi) \leq \Xi(OPT_i) \quad (14)$$

From the construction of  $OPT'$  and Formulas 13 and 14, we get that  $\Xi(OPT') \leq \Xi(OPT)$ . Since  $OPT$  is an optimal permutation,  $\Xi(OPT') = \Xi(OPT)$ . Thus, we can arrive at an optimal solution by choosing the greedy choice and choosing an optimal permutation of the subproblem of the first  $n - i - 1$  elements. By induction on the elements of  $OPT$ , we get that we can safely make the greedy choice at each iteration and obtain an optimal solution. The claim follows.  $\square$