Edges and switches, tunnels and bridges

Edges and Switches, Tunnels and Bridges

David Eppstein* Marc van Kreveld† Elena Mumford‡ Bettina Speckmann‡

Abstract

Edge casing is a well-known method to improve the readability of drawings of non-planar graphs. A cased drawing orders the edges of each edge crossing and interrupts the lower edge in an appropriate neighborhood of the crossing. Certain orders will lead to a more readable drawing than others. We formulate several optimization criteria that try to capture the concept of a "good" cased drawing. Further, we address the algorithmic question of how to turn a given drawing into an optimal cased drawing. For many of the resulting optimization problems, we either find polynomial time algorithms or NP-hardness results.

1 Introduction

Drawings of non-planar graphs necessarily contain edge crossings. The vertices of a drawing are commonly marked with a disk, but it can still be difficult to detect a vertex within a dense cluster of edge crossings. *Edge casing* is a well-known method—used, for example, in electrical drawings and, more generally, in information visualization—to alleviate this problem and to improve the readability of a drawing. A *cased drawing* orders the edges of each crossing and interrupts the lower edge in an appropriate neighborhood of the crossing. One can also envision that every edge is encased in a strip of the background color and that the casing of the upper edge covers the lower edge at the crossing. See Fig. 1 for an example.

If there are no application specific restrictions that dictate the order of the edges at each crossing, then we can in principle choose freely how to arrange them. Certain orders will lead to a more readable drawing than others. In this paper we formulate several optimization criteria that try to capture the concept of a "good" cased drawing. Further, we address the algorithmic question of how to turn a given drawing into an optimal cased drawing.

Definitions. Let G be a graph with n vertices and m edges and let D be a drawing of G with k crossings.

*Department of Computer Science, University of California, Irvine, eppstein@ics.uci.edu

†Department of Information and Computing Sciences, Utrecht University, marc@cs.uu.nl

‡Department of Mathematics and Computer Science, TU Eindhoven, e.mumford@tue.nl and speckman@win.tue.nl

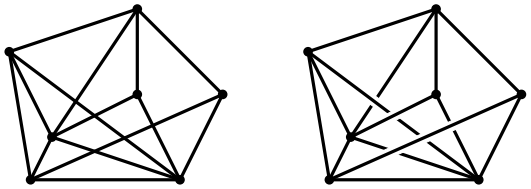


Figure 1: Normal and cased drawing of a graph.

We assume that no vertex v of D lies on (or very close to) an edge e of D unless v is an end-point of e. Further, no more than two edges of D cross in one point and any two crossings are far enough apart so that the casings of the edges involved do not interfere. With these assumptions we can consider crossings independently. We define the *edge crossing graph* G_{DC} for D as follows. G_{DC} contains a vertex for every edge of D and an edge for any two edges of D that cross.

Let C be a crossing between two edges e_1 and e_2. In a cased drawing either e_1 is drawn on top of e_2 or vice versa. If e_1 is drawn on top of e_2 then we say that C is a *bridge* for e_1 and a *tunnel* for e_2. In Fig. 2, C_1 is a bridge for e_1 and a tunnel for e_2. A pair of consecutive crossings C_1 and C_2 along an edge e is called a *switch* if C_1 is a bridge for e and C_2 is a tunnel for e, or vice versa. In Fig. 2, (C_1, C_2) is a switch.

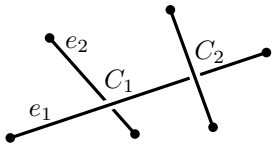


Figure 2: Tunnels and bridges.

Stacking and weaving. When we turn a given drawing into a cased drawing then we need to define a drawing order for every edge crossing. We can choose to either establish a global top-to-bottom order on the edges or to treat each edge crossing individually. We call the first option the *stacking model* and the second one the *weaving model*, since cyclic overlap of three or more edges can occur (see Fig. 3).

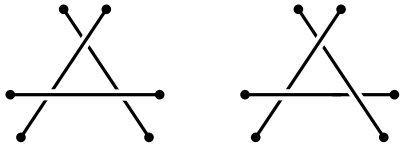


Figure 3: Stacking and weaving.

Quality of a drawing. Globally speaking two factors may influence the readability of a cased drawing in a negative way. Firstly, if there are many switches along an edge then it might become difficult to follow that edge. Drawings that have many switches can appear somewhat chaotic. Secondly, if an edge is frequently below other edges, then it might become hardly visible. These two considerations lead to the following optimization problems for a drawing D.

MinTotalSwitches Minimize the total number of switches.

MinMaxSwitches Minimize the maximum number of switches for any edge.

MinMaxTunnels Minimize the maximum number of tunnels for any edge.

MinMaxTunnelLength Minimize the maximum total length of tunnels for any edge; the length of a tunnel is $\mathit{casingwidth}/\sin\alpha$, where $\alpha \leq \pi/2$ is the angle of the edges at the crossing.

MaxMinTunnelDistance Maximize the minimum distance between any two consecutive tunnels.

Fig. 4 illustrates that the weaving model is stronger than the stacking model for MinTotalSwitches—no cased drawing of this graph in the stacking model can reach the optimum of four switches. For, the thickly drawn bundles of $c > 4$ parallel edges must be cased as shown (or its mirror image) else there would be at least c switches in a bundle, the four vertical and horizontal segments must cross the bundles consistently with the casing of the bundles, and this already leads to the four switches that occur as drawn near the midpoint of each vertical or horizontal segment. Thus, any deviation from the drawing in the casing of the four crossings between vertical and horizontal segments would create additional switches. However, the drawing shown is not a stacked drawing.

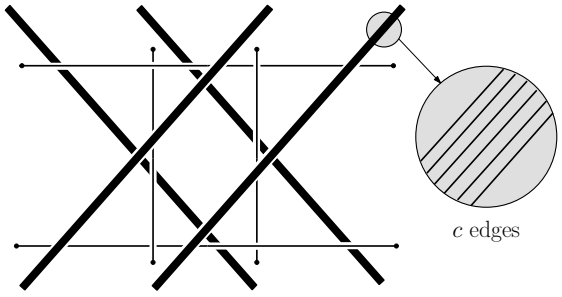


Figure 4: Optimal drawing in the weaving model for MinTotalSwitches.

Results. For many of the problems described above, we either find polynomial time algorithms or NP-hardness results in both the stacking and weaving models. We summarize our results in Table 1.

Model	Stacking	Weaving
MinTotalSwitches	*open*	polyn.
MinMaxSwitches	*open*	*open*
MinMaxTunnels	polyn.	polyn.
MinMaxTunnelLength	polyn.	NP-hard
MaxMinTunnelDistance	polyn.	polyn.

Table 1: Table of results.

2 Minimizing switches

In this section we discuss results related to the MinTotalSwitches and MinMaxSwitches problems. We first discuss some non-algorithmic results giving simple bounds on the number of switches needed, and recognition algorithms for graphs needing no switches. As we know little about these problems for the stacking model, all results stated in this section will be for the weaving model.

Lemma 1 *Given a drawing D of a graph we can turn D into a cased drawing without any switches if and only if the edge crossing graph G_{DC} is bipartite.*

Corollary 2 *Given a drawing D of a graph we can decide in $O((n+m)\log(n+m))$ time if D can be turned into a cased drawing without any switches.*

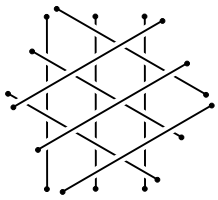
Proof. We apply the bipartiteness algorithm of [2]. Note that this does not construct the arrangement, so there is no factor of k in the runtime. □

Lemma 3 *Given a drawing D of a graph the minimum number of switches of any cased drawing obtained from D is at least half of the number of odd length face cycles in the arrangement of edges (we only count odd-length cycles that do not include a graph vertex).*

Proof. Every odd-length cycle must have a switch. If two odd-length cycles are adjacent, this switch may occur on their shared edge. □

Lemma 4 *For any n large enough, a drawing of a graph G with n vertices and $O(n)$ edges exists, for which any crossing choice gives rise to $\Omega(n^2)$ switches.*

Proof. A construction with three sets of parallel lines, each of linear size, gives $\Omega(n^2)$ triangles, and each triangle gives at least one switch. □



Lemma 5 *For any n large enough, a drawing of a graph G with n vertices and $O(n^2)$ edges exists for which any crossing choice gives rise to $\Omega(n^4)$ switches.*

Proof. We build our graph in the following way. Make a very elongated rectangle, place $n/6$ vertices equally spaced on each short edge, and make the complete bipartite graph. This graph has $(n/6)^2$ edges. One can prove that there is a strip parallel to the short side of the rectangle, such that the parts of the edges inside the strip behave in the same way as parallel ones do with respect to creating triangles when overlapped the way it is described in the previous lemma. This gives us the desired graph with $\Omega(n^4)$ triangles, and hence with $\Omega(n^4)$ switches. □

Theorem 6 MinTotalSwitches *can be solved in polynomial time in the weaving model.*

Proof. Let D be the drawing which we wish to case for the minimum number of switches. We may assume without loss of generality that each vertex of D has degree one, for we may replace any degree-k vertex v by a set of k degree-one vertices placed on a small circle surrounding v, minimize the number of switches in the resulting modified drawing, and then reconnect each edge to v, without changing the number of switches. Define a *segment* of the drawing to be a maximal component of an edge of D that does not include any crossing point with another edge, and define the *parity* of a cycle in D to be the number of segments along the cycle, plus the number of vertices of D contained within the cycle, modulo two.

We apply a solution technique related to the Chinese Postman problem, and also to the problem of via minimization in VLSI design [1]: form an auxiliary graph G, and include in G a single vertex for each odd-parity face cycle in D. Also include in G an edge connecting each pair of vertices, and label this edge by the number of segments of the drawing that are crossed in a path connecting the corresponding two faces in D that crosses as few segments as possible. We claim that the minimum weight of a perfect matching in G equals the minimum total number of switches in any casing of D.

In one direction, we can case D with a number of switches equal to or better than the weight of the matching, as follows: for each edge of the matching, insert a small break into each of the segments in the path corresponding to the edge. The resulting broken arrangement can be shown to have no odd face cycles, for the breaks connect pairs of odd face cycles in D to form larger even cycles. More strongly, it has no odd cycles at all, for in any graph drawing in which all vertices have degree one, the parity of any cycle is the sum (modulo two) of the face cycles within it. Therefore, it has a bipartite edge crossing graph, and can be drawn without switches. Forming a drawing of D by reconnecting all the break points adds at most one switch per break point, so the total number of switches equals at most the weight of the perfect matching.

In the other direction, from any casing of D we can derive a set of paths connecting odd cycles via switch points, showing that the number of switch points is at least as large as the weight of the optimal perfect matching; we omit the details. □

3 Minimizing tunnels

In this section we present three algorithms that solve MinMaxTunnels, MinMaxTunnelLength, and MaxMinTunnelDistance in the stacking model. We also present algorithms for MinMaxTunnels and MaxMinTunnelDistance in the weaving model. MinMaxTunnelLength is NP-hard in the weaving model.

3.1 Stacking model

In the stacking model, some edge e has to be bottommost. This immediately gives the number of tunnels of e, the total length of tunnels of e, and the shortest distance between two tunnels of e. The idea of the algorithm is to determine for each edge what its value would be if it were bottommost, and then choose the edge that is best for the optimization to be bottommost (smallest value for MinMaxTunnels and MinMaxTunnelLength, and largest value for MaxMinTunnelDistance). The other $m-1$ edges are stacked iteratively above this edge. It is easy to see that such an approach indeed maximizes the minimum, or minimizes the maximum. We next give an efficient implementation of the approach. The idea is to maintain the values of all not yet selected edges under consecutive selections of bottommost edges instead of recomputing it.

We start by computing the arrangement of edges in $O(m \log m + k)$ expected time, for instance using Mulmuley's algorithm [4]. This allows us to determine the value for all edges in $O(k)$ additional time.

For MinMaxTunnels and MinMaxTunnelLength, we keep all edges in a Fibonnacci heap on this value. One selection involves an extract-min, giving an edge e, and traversing e in the arrangement to find all edges it crosses. For these edges we update the value and perform a decrease-key operation on the Fibonnacci heap. For MinMaxTunnels we decrease the value by one and for MinMaxTunnelLength we decrease by the length of the crossing, which is $\mathit{casingwidth}/\sin\alpha$, where α is the angle the crossing edges make. For MinMaxTunnels and MinMaxTunnelLength this is all that we need. We perform m extract-min and k decrease-key operations. The total traversal time along the edges throughout the whole algorithm is $O(k)$. Thus, the algorithm runs in $O(m \log m + k)$ expected time.

For MaxMinTunnelDistance we use a Fibonnacci heap that allows extract-max and increase-

KEY. For the selected edge we again traverse the arrangement to update the values of the crossing edges. However, we cannot update the value of an edge in constant time for this optimization. We maintain a data structure for each edge that maintains the minimum tunnel distance in $O(\log m)$ time under updates. The structure is an augmented balanced binary search tree that stores the edge parts in between consecutive crossings in its leaves. Each leaf stores the distance between these crossings. Each internal node is augmented such that it stores the minimum distance for the subtree in a variable. The root stores the minimum distance of the edge if it were the bottommost one of the remaining edges. An update involves merging two adjacent leaves of the tree and computing the distance between two crossings. Augmentation allows us to have the new minimum in the root of the tree in $O(\log m)$ time per update. The whole algorithm therefore takes $O(m \log m + k \log m)$ expected time.

Theorem 7 *Given a straight-line drawing of a graph with n vertices, $m = \Omega(n)$ edges, and k edge crossings, we can solve* MinMaxTunnels *and* MinMaxTunnelLength *in $O(m \log m + k)$ expected time and* MaxMinTunnelDistance *in $O(m \log m + k \log m)$ expected time in the stacking model.*

3.2 Weaving model

In the weaving model, the polynomial time algorithm for MinMaxTunnels comes from the fact that the problem of directing an undirected graph, and minimizing the maximum indegree, can be solved in time quadratic in the number of edges [5]. We apply this on the edge crossing graph of the drawing, and hence we get $O(m^4)$ time. For minimizing tunnel length per edge, we can show:

Theorem 8 MinMaxTunnelLength *is NP-hard in the weaving model.*

In the remainder of this section we show how to solve MaxMinTunnelDistance. We observe that there are polynomially many possible values for the smallest tunnel distance, and perform a binary search on these, using 2-SAT instances as the decision tool.

Our algorithm first computes the arrangement of the m edges to determine all crossings. Only distances between two—not necessarily consecutive—crossings along any edge can give the minimum tunnel distance. One edge crosses at most $m-1$ other edges, and hence the number of candidate distances, K, is $O(m^3)$. Obviously, K is also $O(k^2)$. From the arrangement of edges we can determine all of these distances in $O(m \log m + K)$ time. We sort them in $O(K \log K)$ time to set up a binary search. We will show that the decision step takes $O(m + K)$ time, and hence the whole algorithm takes $O(m \log m + K \log K) = O((m + K) \log m)$ time.

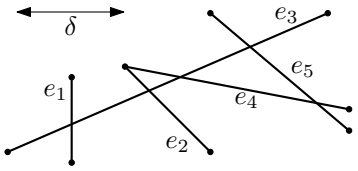


Figure 5: Example where the 2-SAT formula is $(\overline{x}_{13} \vee \overline{x}_{23}) \wedge (\overline{x}_{23} \vee x_{34}) \wedge (\overline{x}_{23} \vee x_{35}) \wedge (x_{34} \vee x_{35})$.

Let δ be a value and we wish to decide if we can set the crossings of edges such that all distances between two tunnels along any edge is at least δ. For every two edges e_i and e_j that cross and $i < j$, we have a Boolean variable x_{ij}. We associate x_{ij} with TRUE if e_i has a bridge at its crossing with e_j, and with FALSE otherwise. Now we traverse the arrangement of edges and construct a 2-SAT formula. Let e_i, e_j, and e_h be three edges such that the latter two cross e_i. If the distance between the crossings is less than δ, then e_i should not have the crossings with e_j and e_h as tunnels. Hence, we make a clause for the 2-SAT formula as follows (Fig. 5): if $i < j$ and $i < h$, then the clause is $(x_{ij} \vee x_{ih})$; the other three cases ($i > j$ and/or $i > h$) are similar. The conjunction of all clauses gives a 2-SAT formula that is satisfiable if and only if we can set the crossings such that the minimum tunnel distance is at least δ. We can construct the whole 2-SAT instance in $O(m+K)$ time since we have the arrangement, and satisfiability of 2-SAT can be determined in linear time [3].

Theorem 9 *Given a straight-line drawing of a graph with n vertices and $m = \Omega(n)$ edges, we can solve* MaxMinTunnelDistance *in $O((m + K) \log m)$ expected time in the weaving model, where $K = O(m^3)$ is the total number of pairs of crossings on the same edge.*

References

[1] R.-W. Chen, Y. Kajitani, and S.-P. Chan. A graph-theoretic via minimization algorithm for two-layer printed circuit boards. *IEEE Trans. Circuits and Systems*, 30(5):284–299, 1983.

[2] D. Eppstein. Testing bipartiteness of geometric intersection graphs. In *Proc. 15th ACM-SIAM Sympos. Discrete Algorithms*, pages 853–861, 2004.

[3] S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM J. Comput.*, 5(4):691–703, 1976.

[4] K. Mulmuley. *Computational Geometry: An Introduction through Randomized Algorithms.* Prentice Hall, 1994.

[5] V. Venkateswaran. Minimizing maximum indegree. *Discrete Applied Mathematics*, 143:374–378, 2004.