Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 19, no. 1, pp. 191–222 (2015)
DOI: 10.7155/jgaa.00353

Algorithms for the Hypergraph and the Minor Crossing Number Problems

Markus Chimani[1] *Carsten Gutwenger*[2]

[1]Institute of Computer Science, Osnabrück University
[2]Department of Computer Science, TU Dortmund University

Abstract

We consider the problems of hypergraph and minor crossing minimization, and point out relationships between these two problems that have not been exploited before.

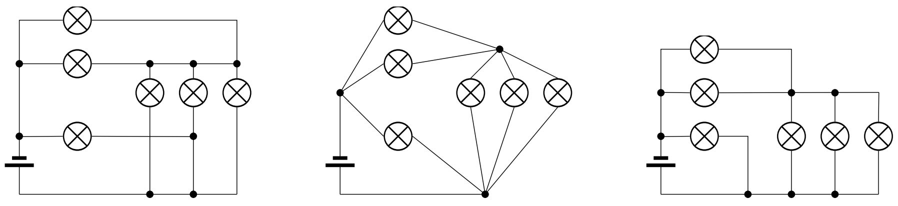
In the first part of this paper, we present new complexity results regarding the corresponding edge and vertex insertion problems. Based thereon, we present the first planarization-based heuristics for hypergraph and minor crossing minimization. Furthermore, we show how to apply these techniques to hypergraphs arising in real-world electrical circuits.

The experiments in this paper show the applicability and strength of this planarization approach, considering established benchmark sets from electrical network design. In particular, we show that our heuristics lead to roughly 40–70% less crossings compared to the state-of-the-art algorithms for drawing electrical circuits.

Submitted: January 2014 | Reviewed: April 2014 | Revised: July 2014 | Reviewed: September 2014 | Revised: January 2015
Accepted: March 2015 | Final: March 2015 | Published: March 2015
Article type: Regular paper | Communicated by: S.-H. Hong

A preliminary version of this paper appeared in the Proceedings of ISAAC 2007, LNCS 4835, Springer-Verlag 2007, pp. 184–195.

E-mail addresses: markus.chimani@uni-osnabrueck.de (Markus Chimani) carsten.gutwenger@tu-dortmund.de (Carsten Gutwenger)



(a) Electrical circuit C (b) Minor G (c) Realizing graph G'

Figure 1: The electrical circuit (a) cannot be drawn without crossings. By (b) computing a minor, and (c) considering a realizing graph, we obtain an equivalent but planar circuit.

1 Introduction

Crossing number research is a vivid field for over six decades; see [42] for an extensive bibliography. Most research was done with respect to the *traditional* crossing number: intuitively, given a graph, draw it into the plane with the least number of edge crossings. In recent years, several further crossing numbers have surfaced, either because of their theoretical appeal or their applicability in practical problems. In this paper, we bind together theoretical research based on the so-called *minor crossing number*, and practical demands often summarized under variants of *hypergraph crossing numbers*.

We will define those notions formally in the succeeding section. For now it shall suffice to say that the *minor crossing number* of G is the smallest crossing number of any graph G' that has G as its (graph) minor. This concept has been studied mostly only in the context of theoretical lower and upper bounds [2–4], but was never before tackled algorithmically. We will exploit the connection between this crossing number and those of hypergraphs.

Besides their theoretical appeal, these problems occur, e.g., for crossing minimal layouts of electrical circuits [4]. Consider Figure 1. Usually, the exact topology of such a circuit C is not interesting for the connected subgraphs that have the same electric potential. Hence we can "merge" these vertices into one vertex (which is exactly the central operation to obtain a minor G), compute the minor crossing number $\mathrm{mcr}(G)$ and expand the graph accordingly to obtain G'. In this example, we can observe the connection to hypergraphs: by seeing the impedances on the wires as vertices, we can interpret the wires on the same potential as *hyperedges*, i.e., edges with multiple incident vertices.

Outline and Contribution. We recapitulate the definition of the (minor) crossing number, and introduce formal definitions for the hypergraph crossing numbers in Section 2. Like the traditional crossing number, all crossing number variants considered herein are NP-hard to compute and we point out relationships between those measures.

For the traditional crossing number, its corresponding *edge* and *vertex inser-*

tion problems (see below for formal definitions) have turned out to be polynomial-time solvable and they have become a useful tool both in theory and in practice. After a brief introduction into this topic in Section 3, we define the corresponding insertion problems for our considered crossing numbers in Section 4. We prove some of them to be NP-hard, while obtaining exact polynomial algorithms for others.

Those algorithms then allow us to establish novel heuristics for our crossing number minimization problems in Section 5. Thereafter in Section 6, we outline additional properties and algorithmic adaptions to consider the practical application of drawing real-world electrical circuits. Both latter sections include experiments, where we demonstrate the algorithms' applicabilities and strengths in practice.

In the final section, we conclude with sketching how to adopt exact (exponential time) approaches based on integer linear programs to our crossing number variants and collect some open problems.

2 Minor and Hypergraph Crossing Numbers

A *drawing* of a graph G on the plane is a one-to-one mapping of each vertex to a point in $\mathbb{R}^2$ and each edge to a curve between its two endpoints. The curve is not allowed to contain other vertices than its two endpoints. A *crossing* is a common point of two curves, other than their endpoints, and no three edges cross at a common point. The (traditional) *crossing number* $\mathrm{cr}(G)$ then is the smallest number of crossings in any drawing of G.

2.1 Minor Crossing Number

A graph G is a minor of a graph G', denoted by $G \preceq G'$, if and only if G can be obtained from G' by a series of *minor operations*. Such an operation is to either (i) delete an edge or a vertex and its incident edges, or (ii) contract an edge v_1v_2, thereby unifying the two incident vertices into a new vertex v which is incident to all former neighbors of v_1 and v_2. The latter operation is called *edge contraction*.

Symmetrically, we can define the *inverse minor operations*. Graph G is a minor of G', if and only if we can obtain G' from G by a series of the following operations: We either (i) introduce a new edge or a new vertex, probably incident to some vertices in the graph, or (ii) we replace some vertex v by an edge v_1v_2, and for each neighbor $u \in N(v)$ of v we introduce an edge v_1u, v_2u, or both. We call the latter operation *vertex split*.

Definition 1 (Minor Crossing Number) The *minor crossing number* $\mathrm{mcr}(G)$, sometimes also called *minor-monotone crossing number*, is the smallest crossing number of any graph G' that has G as its minor, i.e., $\mathrm{mcr}(G) := \min_{G \preceq G'} \mathrm{cr}(G')$.

Let G' be a graph obtaining this minimum, i.e., $G \preceq G'$ and $\mathrm{cr}(G') = \mathrm{mcr}(G)$. We say G' is a *realizing graph* of $\mathrm{mcr}(G)$.

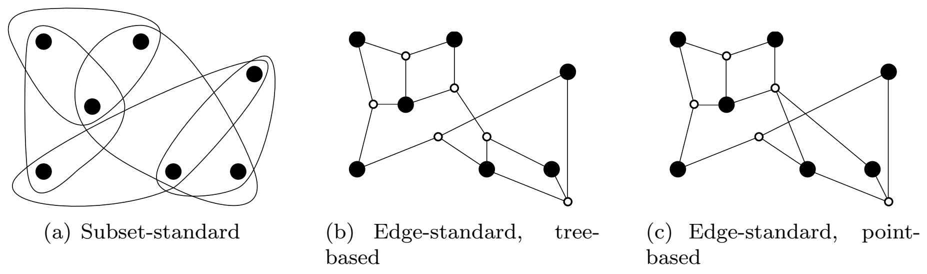

(a) Subset-standard (b) Edge-standard, tree-based (c) Edge-standard, point-based

Figure 2: A hypergraph, drawn using different drawing styles.

Clearly, we always have $\mathrm{mcr}(G) \leq \mathrm{cr}(G)$. A central property of this number is that, unlike the traditional crossing number, it is monotonously decreasing with respect to the minor relation; if $G \preceq G'$, we have $\mathrm{mcr}(G) \leq \mathrm{mcr}(G')$. The following observation is well-known and easy to see.

Observation 1 *Consider a cubic graph G, i.e., a graph where each vertex has degree 3. We have* $\mathrm{mcr}(G) = \mathrm{cr}(G)$.

In [32], Hliněný showed that the crossing number problem remains NP-complete when considering cubic graphs. Hence the minor crossing number problem is NP-complete as well.

2.2 Hypergraph Crossing Number

A hypergraph $\mathcal{G} = (V, \mathcal{E})$ differs from an ordinary graph in that instead of edges—which have exactly two incident vertices—we consider *hyperedges*: A hyperedge $h \in \mathcal{E}$ is a proper subset of V (i.e., $h \subset V$) with $|h| \geq 2$. See, e.g., [36] for details. Hypergraph crossing numbers have, sometimes implicitly, been used in various different variations before, but to our knowledge lack a clear over-arching definition.

There are two major variants on how to draw hypergraphs [38], cf. Figure 2: the *subset-standard* and the *edge-standard*. The first variant becomes very confusing with more hyperedges, and it is ambiguous how to define a consistent notion of crossings. Hence, most applications, e.g., [26, 37, 39], focus on the edge-standard, which allows two sub-variants: In the *tree-based* drawing style, each hyperedge h is drawn as a tree-like structure of lines whose leaves are the incident vertices of h. If we restrict the tree-like structure of every hyperedge to be a star, we obtain the *point-based* drawing style.

Based on these drawing styles, we can define a *tree-based transformation* to obtain a traditional graph L from $\mathcal{G}$. For each hyperedge $h \in \mathcal{E}$ we introduce a set of associated *hypervertices* V_h, which form the branching points of the line tree. Each vertex $v \in h$ is connected to exactly one $n \in V_h$, and all hypervertices V_h are tree-wise connected. We denote the set of all graphs L obtainable by such transformations by $\mathcal{L}(H)$ and can naturally define:

Definition 2 (Tree-based Hypergraph Crossing Number) Let $\mathcal{G}$ be a hypergraph. We define the *tree-based hypergraph crossing number* as $\mathrm{thcr}(\mathcal{G}) := \min_{L \in \mathcal{L}(\mathcal{G})} \mathrm{cr}(L)$.

We further define the *point-based transformation* $\Lambda(H)$ as the special tree-based transformation where each hyperedge has exactly one associated hypervertex, i.e., $\Lambda(\mathcal{G}) := (V \cup \mathcal{E}, E_{\mathcal{E}})$ with $E_{\mathcal{E}} := \{vh : v \in h \in \mathcal{E}\}$. Clearly, this leads to the point-based drawing style and the definition:

Definition 3 (Point-based Hypergraph Crossing Number) Let $\mathcal{G}$ be a hypergraph. We define the *point-based hypergraph crossing number* as $\mathrm{phcr}(\mathcal{G}) := \mathrm{cr}(\Lambda(\mathcal{G}))$.

Both hypergraph crossing numbers have the elegant property that they are equivalent to the traditional crossing number if all hyperedges have cardinality 2. Because of this property, computing $\mathrm{thcr}(H)$ and $\mathrm{phcr}(H)$ is NP-hard. Thinking in terms of graph minors, we furthermore observe:

Observation 2 *For any $L \in \mathcal{L}(\mathcal{G})$ we have $\Lambda(\mathcal{G}) \preceq L$, i.e., the point-based transformation of $\mathcal{G}$ is the minor of any tree-based transformation of $\mathcal{G}$.*

We can define *point-based hypergraph planarity* of $\mathcal{G}$ as $\mathrm{phcr}(\mathcal{G}) = 0$ straightforwardly, which is equivalent to Zykov planarity [36]. It can be efficiently tested by transforming $\mathcal{G}$ into $\Lambda(\mathcal{G})$ in linear time and applying any traditional linear-time planarity testing algorithm to $\Lambda(\mathcal{G})$. Analogously, *tree-based hypergraph planarity* can be defined as $\mathrm{thcr}(\mathcal{G}) = 0$. Since any $L \in \mathcal{L}(\mathcal{G})$ is planar if and only if $\Lambda(\mathcal{G})$ is planar, all three planarity definitions are equivalent.

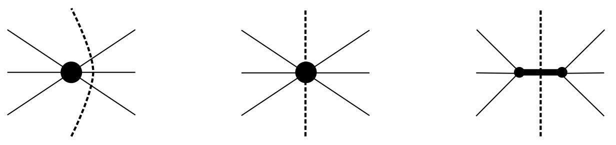
Obviously, the point-based hypergraph crossing minimization of $\mathcal{G}$ is equivalent to the traditional crossing minimization on the graph $\Lambda(\mathcal{G})$. Hence we will focus on computing $\mathrm{thcr}(\mathcal{G})$.

2.3 Restricted Minor Crossing Number and Relationships

Let $G \preceq G'$. Then, each vertex v in G corresponds to a subset of vertices in G' which we call *split vertices* of v. We may say v is *expanded* into these split vertices. Now, let $G = (V, E)$ be a graph and $W \subseteq V$ a vertex subset. We can define a special minor relation: G is a *W-minor* of G', denoted by $G \preceq_W G'$, if and only if we can obtain G' from G by only expanding vertices of W. This leads to the more general *W-restricted minor crossing number* $\mathrm{mcr}_W(G)$, i.e., the smallest crossing number over all graphs G' with $G \preceq_W G'$. Clearly $\mathrm{mcr}_V(G) = \mathrm{mcr}(G)$. Since vertices with degree less than 4 are irrelevant for the differences between the traditional and the minor crossing number, we have:

Theorem 3 *Let $\mathcal{G} = (V, \mathcal{E})$ be a hypergraph and $\hat{\mathcal{E}} := \{h \in \mathcal{E} : |h| \geq 4\}$. The tree-based hypergraph crossing number of $\mathcal{G}$ is equivalent to the $\hat{\mathcal{E}}$-restricted minor crossing number of $\Lambda(\mathcal{G})$, i.e., $\mathrm{thcr}(\mathcal{G}) = \mathrm{mcr}_{\hat{\mathcal{E}}}(\Lambda(\mathcal{G}))$.*

Hence, computing the tree-based hypergraph crossing number of $\mathcal{G}$ is equivalent to finding a realizing graph $\Lambda' \succeq_{\hat{\mathcal{E}}} \Lambda(\mathcal{G})$ with smallest crossing number, i.e.,


(a) Edge crosses vertex

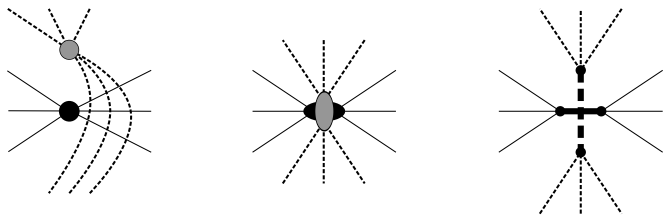

(b) Vertex crosses vertex

Figure 3: New types of crossings. Both (a) and (b) give three visualizations: On the left, we see a situation for the traditional crossing number. We require less crossings for the minor crossing number by allowing novel crossing types (middle) that lead to a realizing graph structure depicted on the right—the expansion trees, in these examples simple edges, are drawn bold.

we may obtain Λ' by only expanding hypervertices of degree at least 4. In the following, we will always consider an undirected graph $G = (V, E)$ with $W \subseteq V$, and we are interested in $\text{mcr}_W(G)$. We can assume $\deg(v) \geq 4$ for all $v \in W$.

For the following algorithms, there are two points of view which are helpful when discussing the problem of minor crossing numbers:

1. We can replace each vertex $v \in W$, with neighborhood $N(v)$, by an *expansion tree* T_v that is incident to all vertices—or their respective expansion trees—of $N(v)$. The vertices of T_v are exactly the split vertices of v. The W-restricted minor crossing number problem can then be reformulated as finding a *tree expansion* G', i.e., a graph obtained by such transformations, with smallest crossing number.

2. In the traditional crossing number problem, only edges are allowed to cross. For the minor crossing number, edges are also allowed to cross through vertices, and moreover vertices may even "cross" other vertices; cf. Figure 3. Such crossings are equivalent to crossings between an expansion tree and a traditional edge, or between two expansion trees, respectively.

3 Preliminaries for Insertion Problems

Edge and vertex insertions provide strong tools to tackle the traditional crossing number problem both in theory and in practice. They led to approximation algorithms for special graph classes and are the central ingredient of the *planarization method*, the family of the most successful crossing number heuristics. Recall that for crossing-free drawings (of connected planar graphs), we define *embeddings* as the equivalence classes over all possible drawings w.r.t. the cyclic order of the edges around their incident vertices [21].

In the *edge insertion* problem, we are given a planar graph G and an additional edge e not yet in G. We want to insert e into a planar embedding of G using the least possible number of crossings, i.e., find a drawing of $G + e$ with the least possible number of crossings subject to the constraint that the drawing induced by G (i.e., when deleting the image of e from the drawing) is planar. This is equivalent to the question for the smallest number of crossings of $G + e$ such that all crossings occur on e.

Analogously, we define the *vertex insertion* problem: Given a planar graph $G = (V, E)$ and a vertex subset $U \subseteq V$, draw G together with a new vertex, connected to all vertices of U, with the least number of crossings such that the drawing induced by G is planar. We will always denote the new vertex by u.

Both problems come in two flavors, with *fixed* and with *variable* embedding. In the former, the embedding of G in the final drawing is prespecified, and both the edge and the vertex insertion problem can be easily solved by considering shortest distances in the dual graph of G [21]. When considering the *variable* case, the problems include finding an embedding that allows the overall smallest number of crossings. Still, both edge [31] and vertex [16] insertions can be solved in polynomial time (the former even in linear time), but require much more complicated algorithms and data structures; see the next section.

Computing the exact crossing number is already NP-hard for a graph like $G + e$ where G is planar [8, 10]. However, we remark that an optimal solution to an insertion problem (with variable embedding) approximates the crossing number of the augmented graph [7, 9, 19, 33]; see also Section 7. In fact, the proof of [7, 9] (independently of our only slightly earlier publication [12]) even uses the concept of edge insertion w.r.t. the minor crossing number (although not algorithmically) to lower bound the number of required crossings for $G + e$.

3.1 Decomposition Trees

We briefly introduce two important graph decomposition structures, namely BC- and SPR-trees, which we will need in Section 4 for presenting our optimal minor edge insertion algorithm.

Let G be a connected graph. The *BC-tree* $\mathcal{B}$ of G is a tree with two different node types B and C: For each cut vertex in G, $\mathcal{B}$ contains a unique corresponding C-node, and for each *block*, i.e., a maximal two-connected subgraph or a bridge, in G a unique corresponding B-node. Two nodes in $\mathcal{B}$ are adjacent if and only if they correspond to a block b and a cut vertex c, such that $c \in b$. It is well-known that the size of $\mathcal{B}$ is linear in the size of G, and that $\mathcal{B}$ can be constructed in linear time, by computing the biconnected components of G; see [35, 41].

Based thereon, we can further decompose each non-trivial block G' (i.e., a block that is not a bridge) via an SPQR-tree [23, 24] into its triconnected components: While SPQR-trees are more complicated than BC-trees, they also only require linear size and can be constructed in linear time [29, 34]. This data structure is particularly interesting, as it directly encodes all (exponentially many) planar embeddings of G'. We use the definition from [11,18] which does not use Q-nodes, and therefore call the decomposition *SPR-tree* for conciseness. In a nutshell, each tree node corresponds to a *skeleton*, a "sketch" of G' where certain subgraphs are replaced by virtual edges; a skeleton's structure is restricted to only three simple types. By repeatedly merging the skeletons of adjacent nodes (at their virtual edges representing each other), we can obtain the original graph.

Definition 4 (SPR-tree) Let G' be a biconnected graph with at least three vertices. The *SPR-tree* $\mathcal{T}$ of G' is the (unique) smallest tree satisfying the following properties:

i. Each node ν in $\mathcal{T}$ corresponds to a *skeleton* $S_\nu = (V_\nu, E_\nu)$ which is a "sketch" (minor, in fact) of G': Certain subgraphs are replaced by single *virtual edges*. The non-virtual edges are referred to as *original edges*.

ii. The tree has three different node types with specific skeleton structure:

   S: The skeleton is a simple cycle; it represents a *serial* component.

   P: The skeleton consists of two vertices and multiple edges between them; it represents a *parallel* component.

   R: The skeleton is a simple triconnected graph. Note that a planar triconnected graph has a unique embedding (up to mirroring).

iii. For the edge $\nu\mu$ in $\mathcal{T}$ we have: S_ν contains a virtual edge e_μ that represents the subgraph described by S_μ, and vice versa.

iv. We can obtain the original graph G' by iteratively merging the skeletons of adjacent tree nodes: For the edge $\nu\mu$ in $\mathcal{T}$, let e_μ (e_ν) be the virtual edge in ν (μ) representing the subgraph described by S_μ (S_ν, respectively). Clearly, both edges e_μ and e_ν connect the same vertices, say u and v. We obtain a merged graph $(V_\nu \cup V_\mu, E_\nu \cup E_\mu \setminus \{e_\mu, e_\nu\})$ by gluing the graph together at u and v and removing e_μ and e_ν.

Observe that the merge operation guarantees that the end vertices of a virtual edge are in fact a 2-cut, i.e., their removal decomposes the graph into two or more components. In fact, the skeletons are exactly the triconnected components of G' discussed in [34].

4 Edge and Vertex Insertion for the Minor Crossing Number

We are now ready to present our core theoretical results on insertion problems in the context of our crossing number variants. In the following, we will always consider the W-restricted variant of the minor crossing number. Yet, all our results also hold for the special case that W is the complete set V, i.e., the pure minor crossing number. We summarize our results in the following table, subject to the precise definitions discussed below.

minor ... insertion	fixed embedding	variable embedding
edge	$O(\|V\|)$ [Theorem 4]	$O(\|V\|)$ [Theorem 5]
vertex	$O(\|V\| \cdot \|U\|)$ [Theorem 6]	**?**
tree	NP-hard [Theorem 7]	

Table 1: Summary of complexity results for insertion problems w.r.t. mcr_W. $G = (V, E)$ is the planar graph into which to insert an edge $e \notin E$ or a vertex $u \notin V$ (with edges incident to some $U \subseteq V$). Note that all entries hold independently whether $W = V$ or not.

To formally define our insertion problems, we need one additional definition. Two embeddings Γ of G and Γ' of G' ($G \preceq G'$) are *consistent*, if we can obtain Γ by performing the necessary minor operations stepwise on G' and Γ' in the natural way: Merge adjacent vertices v_1 and v_2 with their respective cyclic orders $\pi_{v_1} = \langle v_1v_2, e_1, \ldots, e_{\deg(v_1)-1}\rangle$ and $\pi_{v_2} = \langle v_2v_1, f_1, \ldots, f_{\deg(v_2)-1}\rangle$ of their incident edges. Then, the new vertex v will have the cyclic order $\pi_v = \langle e_1, \ldots, e_{\deg(v_1)-1}, f_1, \ldots, f_{\deg(v_2)-1}\rangle$.

4.1 Minor Edge Insertion

Definition 5 (Minor Edge Insertion) Given a planar graph $G = (V, E)$, two non-adjacent vertices $s, t \in V$, and a vertex subset $W \subseteq V$. The W-restricted minor edge insertion problem is to find the W-restricted minor crossing number of the graph $G + st := (V, E \cup \{st\})$ under the restriction that all crossings occur on the new edge st.

If we additionally require that the solution after removing st is an embedding *consistent* with some prespecified embedding Γ of G, we have the fixed embedding scenario, denoted by MEI-F; otherwise, we have the variable embedding scenario and denote the problem as MEI-V.

As noted before, the corresponding problems concerning the traditional crossing number can be solved in linear time (see [21, 31]). We will show that both MEI-F and MEI-V can also be solved to optimality in linear time. Our task is to find a tree expansion G' of G along with an insertion path connecting s and t, i.e., an ordered list of edges of G' that are crossed when inserting e. Observe that it is never necessary to expand s or t.

Theorem 4 *MEI-F can be solved to optimality in linear time.*

Proof: Let Γ be the prespecified embedding of G. We define a directed graph $D_{\Gamma,s,t} = (N, A)$ as follows. Its vertex set N contains a vertex n_φ for each face

φ in Γ and a vertex n_v for each vertex $v \in W \cup \{s,t\}$. Each arc $a \in A$ has an associated cost $c_a \in \{0,1\}$; we have the following arcs:

- For each pair φ, φ' of adjacent faces in Γ, we have two arcs $n_\varphi n_{\varphi'}$ and $n_{\varphi'} n_\varphi$ with cost 1.
- For each vertex $v \in W \setminus \{s,t\}$ and face φ incident to v we have an arc $n_v n_\varphi$ with cost 1 and an arc $n_\varphi n_v$ with cost 0.
- Finally, we have arcs $n_s n_\varphi$ for each face φ incident to s and $n_{\varphi'} n_t$ for each face φ' incident to t; all these arcs have cost 0.

Then, the solution to MEI-F is the length of a shortest path p in $D_{\Gamma,s,t}$ from n_s to n_t: Each arc $n_\varphi n_{\varphi'}$ in p corresponds to crossing an edge separating φ and φ'; each sub-path $n_\varphi n_v, n_v n_{\varphi'}$ corresponds to splitting vertex v and crossing the edge resulting from the split. We call p the *insertion path* for the new edge st.

In a planar graph, the number of edges and faces are both of order $O(|V|)$. Therefore both the number of vertices N and arcs A in $D_{\Gamma,s,t}$ are in $O(|V|)$ as well. Hence, we can apply breadth first search (BFS) for finding a shortest path in $D_{\Gamma,s,t}$ requiring only $O(|V|)$ time. We remark that BFS can easily be extended to graphs with 0/1-arc costs. $\square$

Theorem 5 *MEI-V can be solved to optimality in linear time.*

Proof: In order to solve MEI-V, we adapt the algorithm by Gutwenger et al. [31] which solves the problem for the traditional crossing number, i.e., $W = \emptyset$ and no vertex splits are possible. They showed that it is sufficient to consider the shortest path[1] $B_0, v_1, B_1, \ldots, v_k, B_k$ in the BC-tree $\mathcal{B}$ of G, such that B_0 (B_k) is a block containing s (t, respectively), and independently compute optimal edge insertion paths in the blocks B_i from v_i to v_{i+1} ($0 \leq i \leq k$, $v_0 = s$, and $v_{k+1} = t$). This is also true when we are allowed to split the vertices W: Assume we already found an optimal embedding and insertion path for each block. We obtain an embedding of the full graph and a full insertion path simply by concatenating the respective insertion paths in the blocks and identifying the respectively last visited faces of the adjacent blocks; alternately crossing edges from different blocks or splitting (and crossing through) a cut vertex v_i would result in unnecessary crossings.

Thus, we can restrict ourselves to a biconnected graph G. Let $\mathcal{T}$ be the SPR-tree of G. We consider the shortest path $p = \mu_1, \ldots, \mu_h$ in $\mathcal{T}$ from a node μ_1 whose skeleton contains s to a node μ_h whose skeleton contains t. Let S_i be the skeleton of μ_i ($1 \leq i \leq h$). The *representative* rep(v) of a vertex $v \in G$ in a skeleton S_i is either v itself if $v \in S_i$, or the virtual edge $e \in S_i$ whose expansion graph contains v. If $W = \emptyset$, the exact algorithm [31] only considers the R-nodes—triconnected skeletons with therefore unique embeddings—on p and independently computes optimal edge insertion paths in fixed embeddings

[1]Note that, if s and t are cut vertices, they both lie in multiple blocks; we are interested in the closest pair.

of the respective skeletons from rep(s) to rep(t). If the representative is an edge, we assume that a virtual vertex is placed on this edge and serves as start- or endpoint of the insertion path.[2]

This approach is invalid if $W \neq \emptyset$: An optimal insertion path in a skeleton S_i might cross through an endpoint a of the edge representing t in S_i, and continuing this path from a in S_{i+1} might save one or even more crossings.

As a first step, we compute and store up to 9 insertion paths for each R-node $\mu \in p$: If rep(s) in μ's skeleton S is an edge $e_s = ab$, we consider up to three sources: e_s is always a source, a (b) is a source if $a \in W$ ($b \in W$, respectively). Analogously, we have up to three targets e_t, c, d. Each source/target pair gives rise to a possible insertion path (considering the minor crossing number, of course). Each such path can be computed by slightly modifying the search network introduced for MEI-F: Recall that S allows only a unique embedding (up to mirroring), and hence MEI-F seems applicable. However, it may contain virtual edges other than e_s, e_t. Let $f = vw$ be such a virtual edge, representing a subgraph H that contains v, w. We know that $H + vw$ is planar. In [31] it was shown that any planar embedding of $H + vw$ allows the same minimum number of crossings when asking for an insertion path from one side of vw to the other, without crossing vw itself—called "crossing through H". This is clear by observing that removing a (v, w)-edge-cut in H in *any* embedding of H separates H into two disjoint subgraphs. Hence a minimum such cut resembles a possible insertion path. Since this also holds for (v, w)-{node,edge}-cuts, we can simply expand each virtual edge $f \notin \{e_s, e_t\}$ by an arbitrary embedding of the subgraph represented by f.

For all non-R-nodes along p, we know a crossing-free insertion path for any source/target combination. Now, with these up to 9 paths per node in p, we can deduce the full insertion path by simple dynamic programming over the length of p. Observe that at μ_1, we only have the unique source s. For increasing $i = 1, \ldots, h$, we compute the insertion paths $\mathcal{P}_i$ from s to the up to three targets e_t, c, d at μ_i. Clearly, $\mathcal{P}_1$ are the solutions stored at μ_1. We obtain $\mathcal{P}_i$, $i > 1$, by joining the solution paths $\mathcal{P}_{i-1}$ to those stored at μ_i[3] and storing the cheapest path for each target. The number of crossings is the sum of the so-joined paths, with an additional crossing if the source at μ_i was a vertex. $\mathcal{P}_h$ will consist of a single insertion path from s to t—our optimum insertion solution—and a corresponding optimum embedding is induced by the embeddings of the skeletons along p.

This algorithm can be implemented to run in linear time. □

4.2 Minor Vertex Insertion

Definition 6 (Minor Vertex Insertion) Given a planar graph $G = (V, E)$

[2]Recall that we will require no crossings for S- and P-nodes, as the respective insertion path will only consist of a single face adjacent to both rep(s) and rep(t): for S-nodes there is only a unique embedding of the skeleton cycle, for P-nodes we can pick an embedding where the considered virtual edges are consecutive.

[3]Clearly, in such a join, the target in $\mathcal{P}_{i-1}$ has to coincide with the source in μ_i.

and two vertex subsets $U, W \subseteq V$. Let $u \notin V$ be a new vertex, and E' the new edges connecting each vertex of U to u. The W-restricted minor vertex insertion problem is to find the W-restricted minor crossing number of the graph $G + u := (V \cup \{u\}, E \cup E')$ under the restriction that all crossings occur on the new edges E'.

Analogously to before, if we additionally require that the solution after removing u and E' is an embedding consistent with some prespecified embedding Γ of G, we have the fixed scenario, denoted by MVI-F. Otherwise, i.e., in the variable scenario, we denote the problem as MVI-V.

Note that in the above setting, the new vertex u cannot be an element of W and it is thus not allowed to be replaced by an expansion tree—we will consider the latter case in the next section under the term *tree insertion*.

The vertex insertion problem for the traditional crossing number where all embeddings are considered—and therefore a special case of MVI-V—has been shown to be polynomially solvable [16]. Yet, the algorithm's intricate structure seems to not readily allow a generalization towards minor crossing numbers. The complexity of MVI-V remains unknown. For the fixed embedding, the problem can be solved in $\mathcal{O}(|V| \cdot |U|)$ time when considering the traditional crossing number. An analogous algorithm, together with the ideas of Theorem 4, can be used to show:

Theorem 6 *MVI-F is solvable in* $\mathcal{O}(|V| \cdot |U|)$ *time.*

Proof: We can solve the vertex insertion problem with fixed embedding for the traditional crossing number by considering the dual graph G^D of G w.r.t. the given embedding Γ. Each vertex in G^D is labeled with a number which is initially 0. We then start a BFS for each $v \in U$, augmenting G^D with edges between v and its incident faces. The labels in G^D are incremented by their BFS-depth minus 1, for each different v. Finally, each vertex of G^D holds the sum of the shortest distances between itself and the vertices U. We then simply pick a vertex of G^D with smallest number and insert the new vertex v into the corresponding face in Γ.

Using the ideas from solving MEI-F, we can use the same algorithm but allow edges to cross through vertices. Since all inserted edges are incident to v, they will not cross each other in any optimal vertex insertion. Therefore, no conflicting edge-vertex crossings can occur, other than ones based on paths with equal length. Such conflicts can easily be resolved by choosing any of the conflicting paths for both inserted edges. Observe that crossing through a vertex w, when several inserted edges come from the same face φ, require only one crossing each: Let $e_1, \ldots, e_{\deg(w)}$ be the edges incident to w in the order specified by Γ and let $e_1, e_{\deg(w)}$ be those bordering φ. We can construct a realizing graph of embedded G, by replacing w by a path of vertices $w_1, .., w_{\deg(w)}$ where each edge e_i, $1 \leq i \leq \deg(w)$, becomes incident to vertex w_i. The correctness and running time of the algorithm follows. $\square$

4.3 Minor Tree Insertion

In the minor vertex insertion problem, the new vertex was not allowed to be part of W and could thus not be expanded. The following problem distinguishes itself from the minor vertex insertion exactly in the fact that it allows to replace u by an expansion tree.

Definition 7 (Minor Tree Insertion) Given a planar graph $G = (V, E)$ and two vertex subsets $U, W \subseteq V$. Let $u \notin V$ be a new vertex, and E' the new edges connecting each vertex of U to u. The W-restricted minor tree insertion problem is to find the $(W \cup \{u\})$-restricted minor crossing number of the graph $G + u := (V \cup \{u\}, E \cup E')$ under the restriction that all crossings occur on the new edges E' or the new vertex u.

Again we distinguish between the variants for fixed (MTI-F) and variable (MTI-V) embeddings. Like MVI-F and MVI-V, it is a natural generalization of the vertex insertion problem for the traditional crossing number. Yet, the seemingly simple extension to allow u to be expanded renders the problem NP-hard. In fact, it turns out to be already NP-hard for a fixed embedding with $W = \emptyset$, i.e., we want a tree-wise interconnection of U w.r.t. the traditional crossing number.

Theorem 7 *MTI-F and MTI-V are NP-hard, even for $W = \emptyset$ (i.e., traditional crossing number) and $W = V$ (i.e., unrestricted minor crossing number).*

Proof: We can restrict ourselves to MTI-F. Since planar 3-connected graphs have a unique planar embedding (and its mirror), NP-hardness for MTI-F (in particular also for 3-connected graphs) implies NP-hardness for MTI-V. Consider the following problem:

> *Planar Steiner Tree (PST).* Given a planar graph D, integral positive edge weights w, and a vertex subset $R \subset V(D)$ called *terminals*. Find a minimum-weight tree T that contains all terminals.

This problem was shown to be NP-hard in the strong sense [27], i.e., there are hard instances where the maximum edge weight $w_{\max}$ is bounded by a polynomial in the size of D.

We will transform any PST instance (with polynomially bounded $w_{\max}$) into a corresponding MTI-F instance with $W = \emptyset$. Choose any embedding of D and augment the graph greedily by additional edges to obtain a planar triangulated graph D' (i.e., each face is a triangle). The new edges get high enough costs such that they will not be chosen in any optimal solution, say $w'_{\max} := w_{\max} \cdot |E(D)| + 1$.

Let G be the dual graph of D'. Any edge in G obtains the same weight as its dual edge in D'. For each terminal $t \in R$, let φ_t be the corresponding face in G and we introduce a star of degree 3 into φ_t, i.e., we add a new vertex v_t with edges incident to the three vertices bordering φ_t. Collect all these new vertices in the set U, set the weight of all these new *star edges* to 0, and let G' be the graph resulting from all these additions.

Recall that D' and G' are triconnected and thus have unique embeddings. Now consider the traditional crossing number with edge weights, i.e., a crossing between two edges of weights w_1 and w_2 counts as $w_1 \cdot w_2$ crossings. In this setting, we can also ask the tree insertion problem (insert a given tree structure, where each inserted edge has weight 1) without the need for any graph minor operations. Then, this tree insertion problem w.r.t. (G', U) is equivalent to solving the given PST problem for D' and hence D.

It remains to transform the edge weighted insertion instance (which requires weight 0 for the star edges) into a proper non-weighted instance. First, observe that $w_{\max}$ and $w'_{\max}$ are both polynomial in the size of D. It is easy to see that any optimally inserted solution tree will cross at most 2 out of 3 star edges around any given vertex of U. We can hence scale all edge weights by a factor of $3|R|$, and set the weights of the star edges to 1. An optimum solution to this scaled problem induces an optimum solution to its unscaled variant. Still the weights in the scaled version are polynomially bounded, and we can replace each edge e in G' of weight w_e with w_e parallel edges. Subdividing these edges leaves a simple graph G''. The graph again only grows polynomially and we hence have NP-hardness for MTI-F with $W = \emptyset$. We can observe that the SPR-tree of G'' consists of a single R-node with skeleton G', multiple P-nodes adjacent to it, and (due to the subdivisions) S-nodes adjacent to these P-nodes. Hence all the possible embeddings of the graph only differ in the order of the subdivided parallel edges—since this order is irrelevant when mapping the solution back into the weighted problem, we also have NP-hardness for MTI-V.

Now, consider the case $W = V$, i.e., all vertices, not only the new vertex u, are allowed to be expanded. We replace each vertex w with degree > 3 by a large-enough grid of maximum degree 3—similar to the brick-wall style grids used in the proof of [32]—and attach w's incident edges to these substructures (at the outside of the grid, far-enough away from each other). We already know from Observation 1 that for graphs with maximum degree 3, expanding vertices does not influence the crossing number. □

Furthermore, recall that computing $\mathrm{thcr}(\mathcal{G})$ is a special case of a W-restricted minor crossing number. By the above proof we also get in particular:

Corollary 8 *Let $\mathcal{G} = (V, \mathcal{E})$ be a hypergraph and $h \notin \mathcal{E}$ a hyperedge not yet in $\mathcal{G}$. Computing $\mathrm{thcr}(\mathcal{G} + h)$ under the side condition that $\mathcal{G}$ has to be drawn planar is NP-hard, independent on whether a specific embedding of $\mathcal{G}$ is given or not.*

5 Heuristic (W-restricted) Minor Crossing Minimization

The *planarization method* is a well-known and successful heuristic for traditional crossing minimization; see [13, 30] for experimental studies. First, a planar subgraph is computed; then the remaining edges are inserted one after another

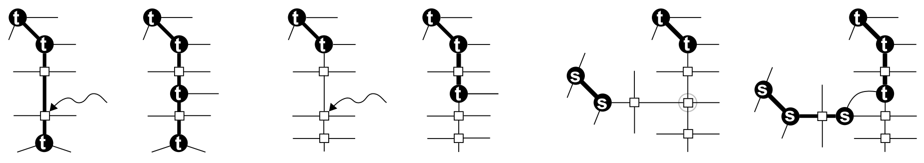

(a) Dummy on target (b) Dummy near target (c) Dummy near source and target

Figure 4: Modification of insertion paths ending at dummy vertices. Bold solid edges are part of expansion trees, dummy-vertices are denoted by squares. The lines with the arrow heads in the left figures ("before") show where the currently considered edge to insert attaches to T. The right figures ("after") show the new T after the insertion took place.

by computing edge insertion paths and inserting the edges accordingly, i.e., edge crossings are replaced by dummy vertices of degree 4.

In order to apply the algorithms from the previous section in a planarization approach for computing $\mathrm{mcr}_W(G)$, we need to generalize them, since the insertion of edges expands vertices into trees. Furthermore, edges of G and edges resulting from vertex splits get subdivided by dummy vertices during the course of the planarization. We call the resulting paths *edge paths* and *tree paths*, respectively. Hence, we are not simply given two vertices s and t but two vertex sets S and T, and we have to find an insertion path connecting a vertex of S with a vertex of T. Thereby, S (T) is the set of all split vertices of s (t) and all dummy vertices on tree or edge paths starting at a split vertex of s (t). The dummy vertices in these last sets have the property that a simple extension of the expansion tree is sufficient to connect an insertion path to a correct split vertex; see Figure 4 for a visual description.

Before we discuss the details, we give an overview of the planarization approach for $\mathrm{mcr}_W(G)$.

(1) Compute a planar subgraph $\bar{G} = (V, \bar{E})$ of G.

(2) For each edge $e = st \in E \setminus \bar{E}$:

  (a) Compute S and T.

  (b) Find an insertion path p from S to T in $\bar{G}$.

  (c) Insert e into $\bar{G}$ according to p by splitting vertices if required and introducing new dummy vertices for crossings.

It remains to show how to generalize the edge insertion algorithms. In the fixed embedding scenario, we simply introduce a super start vertex s^* connected to all vertices in S, and a super end vertex t^* connected from all vertices in T in the search network.

The following proposition shows the key property for generalizing the variable embedding case.

Proposition 9 *The following properties hold:*

1. *The blocks of $\bar{G}$ containing a vertex in S (T) and the cut vertices of $\bar{G}$ contained in S (T) form a subtree of the BC-tree of $\bar{G}$.*

2. *Let $\mathcal{T}$ be the SPR-tree of a block of $\bar{G}$. The nodes of $\mathcal{T}$ whose skeletons contain a vertex in S (T) form a subtree of $\mathcal{T}$.*

This allows us to compute the shortest paths in the BC- and SPR-trees in a similar way as described above (Theorem 5). The only difference is that we consider blocks and skeletons containing any vertex in S (or T). The computation of insertion paths in R-node skeletons is generalized as for the fixed case if several vertices of S or T are contained.

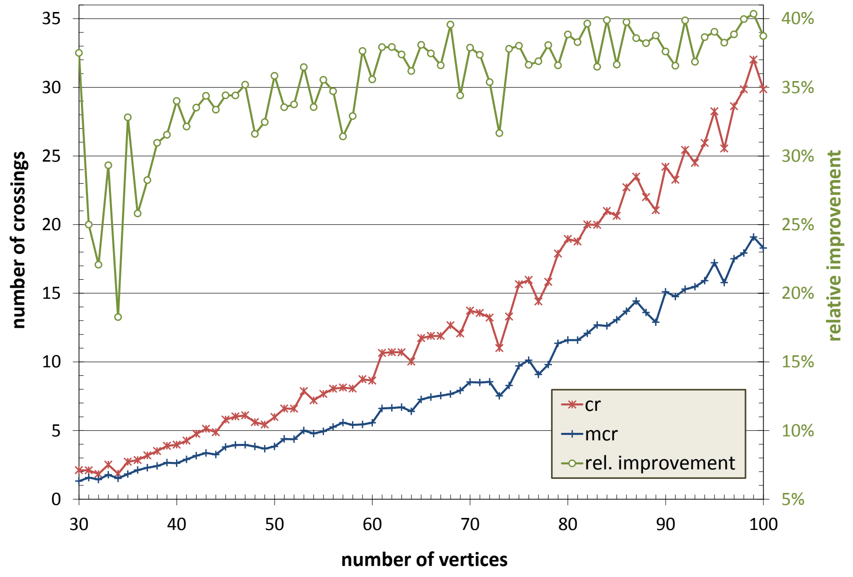
In [13,30], two improvement techniques for the traditional planarization approach are described which are both also applicable in our case. The *permutation* strategy calls step (2) several times and processes the edges in $E \setminus E'$ in random order. The *post-processing* strategy successively removes an edge path and tries to find a better insertion path. This can also be done for tree paths which in fact is a key optimization of our approach, since it allows us to introduce crossings between two tree expansions as well. Finally, we remark that we also contract tree paths during the algorithm if they no longer contain a dummy vertex and thus become redundant.

5.1 Experimental Results

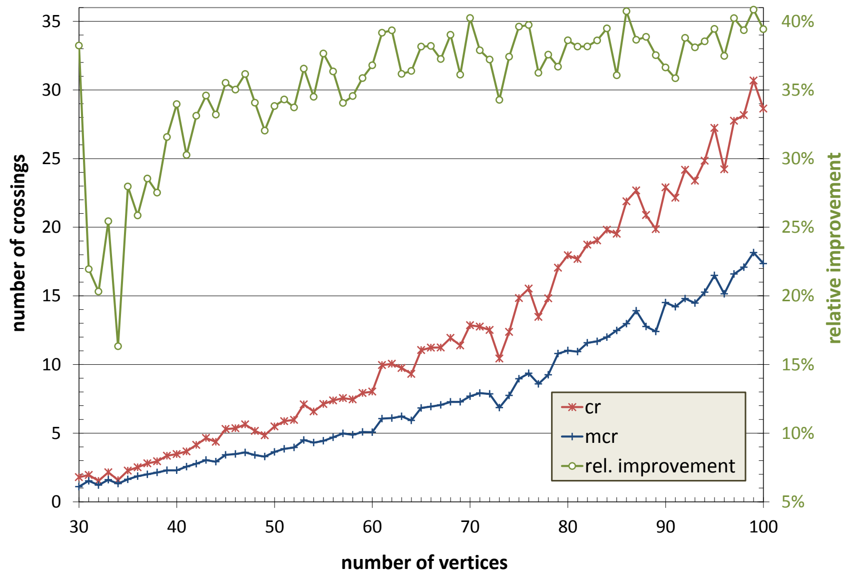
We implemented our algorithms as part of the open-source *Open Graph Drawing Framework* (OGDF, [14]). We conducted two series of experiments on an Intel Xeon E5430 (2.67 GHz) Linux system with 8 GB RAM.

The first experiment uses the well-known *Rome* benchmark set [22], which has been used for many studies on the traditional crossing number, e.g., [6,13, 30]. It consists of 11528 real-world graphs with 10–100 vertices. We restricted ourselves to the 8013 non-planar graphs with at least 30 vertices; they have an average density of 1.34. Our main focus was to investigate how the minor crossing number compares to the traditional crossing number in real-world settings. Figure 5 shows the average crossing numbers and minor crossing numbers per graph size, both for the fixed and variable embedding cases (using 25 permutations and post-processing). We can see that the minor crossing minimization leads to roughly 35% less crossings on average. Both diagrams look nearly identical, although the absolute crossing numbers for fixed embedding are of course a bit higher. In both cases, for the large graphs, the realizing graphs have about 10% more vertices than the original graphs, and roughly 8% of the graphs' vertices are substituted by expansion trees. All graphs can be solved clearly under half a second for the fixed embedding case and under 20 seconds for the variable case. For the latter, the 100-vertex graphs required 3.3 seconds on average.

We also considered the number of original vertices that were expanded and the total number of vertex splits. Figure 6 shows the results for the variable embedding case (the results for fixed embedding look quite similar). The average



(a) Fixed embedding



(b) Variable embedding

Figure 5: Results for the Rome graphs.

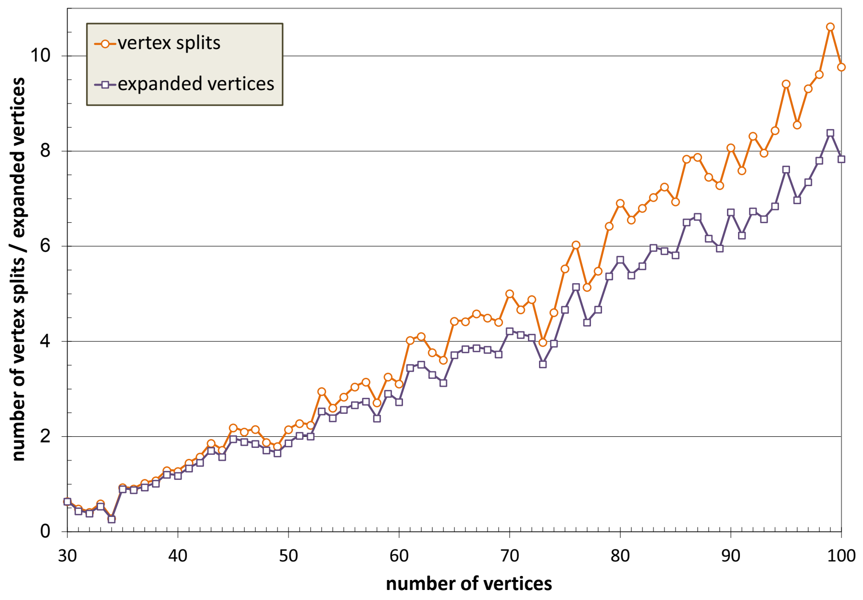


Figure 6: Number of expanded vertices and overall vertex splits for the Rome graphs (variable embedding).

number of expanded vertices grows to 8 for the largest graphs (that is also 8% of their vertices), and the total number of vertex splits is only slightly larger (by about 1.5 for the largest graphs), implying that only few vertices are split several times.

The second set of experiments deals with hypergraphs. Therefore we chose all hypergraphs from the Iscas'85, '89, and '99 benchmark sets of real-world electrical networks[4] with up to 500 vertices in their point-based expansion. Table 2 summarizes our heuristic results for these graphs, considering both phcr, using our traditional crossing minimizer [30], and thcr. The times are given in seconds. We can clearly see the benefit of considering the tree-based drawing style, as compared to the relatively large point-based crossing numbers. Furthermore, these fewer crossings even lead to faster computation times (by a factor of about 2) for the generally harder tree-based crossing number.

6 Applications to Electrical Circuits

In the introduction, we motivated the hypergraph crossing number by considering drawings of electrical circuit designs. On a chip, the most important criteria for realizing such circuits is a small required area, leading to compact but confusing edge routings. But as stated in [25], a readable drawing with few edge

[4]As some of these benchmark sets are hard to obtain, we collected all of them at `http://www.ae.uni-jena.de/Research/ElectricalNetworks.html`.

					FIX				VAR			
	name	$\|V\|$	$\|\mathcal{E}\|$	$\|E(\Lambda)\|$	**phcr**	time	**thcr**	time	**phcr**	time	**thcr**	time
ISCAS'85	*c17*	4	11	16	**0**	0.02	**0**	0.00	**0**	0.00	**0**	0.00
	c432	153	196	489	**323**	3.54	**169**	2.25	**314**	5:46.84	**170**	3:28.47
	c499	170	243	578	**543**	15.48	**215**	6.94	**486**	23:12.51	**200**	10:30.50
ISCAS'89	*s208a*	111	122	300	**29**	0.15	**20**	0.09	**28**	8.37	**18**	6.10
	s27a	12	17	33	**0**	0.00	**0**	0.00	**0**	0.00	**0**	0.00
	s298	127	136	385	**233**	4.03	**86**	1.66	**217**	5:08.03	**78**	2:06.11
	s344	164	184	448	**64**	0.47	**41**	0.38	**57**	39.03	**37**	27.08
	s349	165	185	453	**66**	0.60	**42**	0.48	**61**	50.01	**38**	32.61
	s382	173	182	500	**226**	1.79	**82**	1.31	**206**	3:09.76	**90**	1:47.50
	s386a	158	172	511	**820**	21.78	**300**	11.99	**805**	34:00.86	**266**	15:04.97
	s400	177	186	518	**234**	2.45	**95**	1.41	**224**	4:24.24	**88**	1:55.05
	s420a	233	252	632	**91**	0.69	**52**	0.57	**81**	1:31.69	**49**	50.66
	s444	196	205	569	**229**	2.62	**81**	1.54	**218**	3:38.78	**86**	2:07.42
	s510	210	236	640	**1237**	1:17.04	**524**	27.34	**1197**	111:59.36	**504**	38:14.45
	s526a	209	218	675	**631**	14.35	**247**	6.72	**621**	32:20.34	**224**	13:15.49
ISCAS'99	*b01*	43	47	128	**37**	0.11	**20**	0.10	**31**	5.32	**20**	3.26
	b02	25	27	73	**14**	0.01	**8**	0.02	**12**	0.79	**8**	0.64
	b03	148	156	432	**138**	0.88	**57**	0.74	**133**	1:32.56	**58**	45.59
	b06	42	50	134	**62**	0.16	**25**	0.12	**59**	7.88	**27**	4.22
	b08	166	179	493	**328**	7.47	**161**	3.85	**339**	10:09.06	**157**	3:52.97
	b09	167	169	472	**183**	1.38	**78**	0.80	**177**	2:04.45	**70**	57.48
	b10	183	200	553	**500**	10.79	**245**	6.16	**475**	14:17.70	**209**	8:40.14

Table 2: Results for real-world electrical circuits. $E(\Lambda)$ denotes the edges of the point-based transformation.

crossings is beneficial for debugging, teaching, presentation, and documentation purposes. This is further strengthened by the fact that gate-level descriptions may be automatically synthesized from other, e.g., register-transfer level, descriptions. Furthermore, according to [1], the crossing number of a graph seems to be a good estimate for the required area on a chip.

In this section, we will hence consider the real-world problem of drawing electrical circuit designs, in particular gate- and register-transfer-level networks. The truth is that there are often several additional properties required for such drawings. We will focus on these properties and show how to adopt our algorithms.

The problem of finding a crossing minimal drawing for such circuit designs was, e.g., tackled in [25, 26], the latter of which also gives a short overview on previous approaches. The currently best known approach [26] is based on Sugiyama's three-stage algorithm [40] of layering the graph, performing crossing minimizations between adjacent layers, and finally assigning coordinates.

6.1 Circuit Network

Gate-level networks are composed of several components; see Figure 7 for an example: *Logical gates* perform specified logical operations, like *NOT*, *AND*, *NOR*, etc. A corresponding electrical component takes one or more input signals

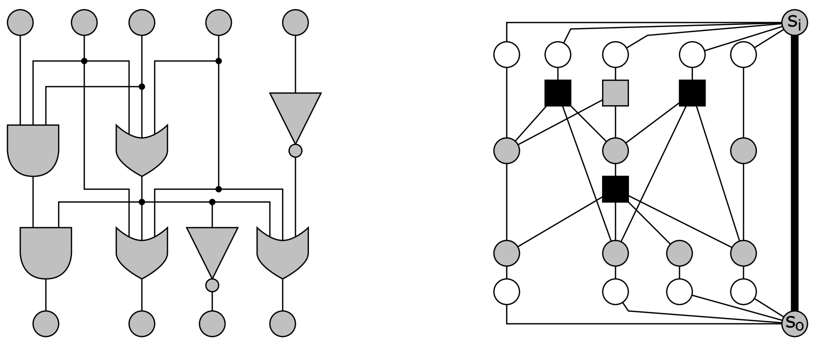


Figure 7: An electrical circuit (left) and its transformation into a graph for the restricted minor crossing number problem (right). Circles denote original vertices, squares denote the hypervertices from the point-based expansion. We allow the minor relations on the black vertices, the white vertices (input and output gates) are merged into the vertices s_i and s_o, respectively.

on its *in-ports* and outputs a single signal on its *out-port*. An *input gate* receives its input signal from outside of the network. Hence it has no explicit in-port and a single out-port. Conversely, an *output gate* outputs a resulting signal to the outside of the network. It has no explicit out-port and a single in-port. Gates are connected via *wires*. Each wire connects a single out-port to one or more in-ports. We will never have a wire directly connecting two out-ports: If the ports do not have the equivalent signal, this would result in a circuit failure.

For higher abstraction levels (register-transfer level, etc.), we may also consider *operational components*, which represent logical networks that perform more complex computations, like a 4-bit ADDER. Such components may have multiple out-ports, and the order of the in- and out-ports may be crucial.

Drawing requirements. A drawing of such a circuit has to follow certain established norms. The most common of which is that the input and output gates are drawn on opposing borders of the drawing, say inputs on the top, outputs on the bottom, and all other gates and wires in between. In terms of graph drawing we can deduce the following drawing requirements:

DR1. Input and output gates have to lie on the outer face of the drawing.

DR2. Consider the cyclic order of the input and output gates on the outer face. All input gates have to occur consecutively. This induces the same property for the output gates.

Consider a planarization and an embedding of a hypergraph resembling a circuit. If the embedding satisfies the above two properties, we can easily find a drawing of the circuit where the input and output gates are on opposing borders, and where all other wires and gates are in-between.

Furthermore, for each operational component the order of its ports has to be retained by the drawing. If no order is given, we always require that the component's in-ports are consecutive in the embedding (and hence so are its out-ports).

In the aforementioned currently best known crossing minimization algorithm for electrical circuits, the drawings furthermore have the property that they are drawn *upward*, i.e., the y-coordinate increases monotonically for each edge, when traversing it from its source vertex to one of its target vertices. This is an intrinsic property of Sugiyama-based algorithms, and it is unspecified whether this property is a requirement or a mere side-effect. In our context we will not take upwardness into account.

However, building on the results of this paper, Chimani et al. [15] describe an approach for drawing *directed* hypergraphs that is based on *upward planarization* [17]. If electrical circuits shall be drawn in an upward fashion, that approach is favorable; however, regarding crossing minimization of circuits without the upward property, the heuristics presented here are much stronger: For example—apart from the fact that the upward drawing restriction may already force more crossings—we use optimal edge insertion in the variable embedding scenario, whereas current upward planarization techniques can only consider fixed embeddings.

6.2 From Circuit Networks to Hypergraphs

Clearly, the gates in a (gate-level) circuit C correspond to vertices V in a hypergraph $\mathcal{G} = (V, \mathcal{E})$. We partition V into the sets I, O, and L, corresponding to the input, output, and logical gates, respectively. For each wire we have a hyperedge connecting the vertices corresponding to the gates connected by this wire; cf. Figure 7.

We start with the point-based transformation $\Lambda(\mathcal{G}) = (V \cup \mathcal{E}, E_{\mathcal{E}})$ of this hypergraph and would want to solve the $\hat{\mathcal{E}}$-restricted minor crossing number problem on $\Lambda(\mathcal{G})$, where $\hat{\mathcal{E}} := \{h \in \mathcal{E} : |h| \geq 4\}$. However, this translation itself would not yet guarantee the drawing requirements discussed above. Therefore we first modify $\Lambda(\mathcal{G})$ into the graph Λ^+ as follows: we unify all input vertices into a vertex s_i, all output vertices into a vertex s_o, and we introduce a new edge $s_i s_o$. We then have to solve the problem of finding "$\mathrm{mcr}_{\hat{\mathcal{E}}}^{s_i s_o}(\Lambda^+)$", which we define as the $\hat{\mathcal{E}}$-restricted minor crossing number of Λ^+ under the restriction that the edge $s_i s_o$ has no crossings. Assuming we (exactly or heuristically) solve this crossing minimization problem, we obtain a planarization of Λ^+, where hypervertices might be expanded into subtrees.

We can deduce a drawing for the circuit C by reinterpreting the hyperedges as wires; it remains to reintroduce the input and output vertices. Since the edge $s_i s_o$ has no crossings, we know that s_i and s_o lie in a common face incident to $s_i s_o$, which we choose as our outer face. We can then choose a (conceptually arbitrarily small) crossing free region around s_i and place the input vertices on their corresponding edges next to s_i; we do the analogous for the output vertices.

After removing $s_i s_o$, we finally obtain a drawing of the circuit C where all inputs and outputs lie on the outside (DR1), and the input vertices occur consecutively (DR2). Hence we obtain a valid drawing of C and can deduce:

Proposition 10 *Given a circuit C, let $\mathrm{ccr}(C)$ be the crossing number of the circuit C under the drawing requirements DR1 and DR2. Let Λ^+ be its transformed graph as described above. We have:* $\mathrm{ccr}(C) = \mathrm{mcr}_{\hat{\mathcal{E}}}^{s_i s_o}(\Lambda^+)$.

It remains to solve the restricted minor crossing number problem, as described in the section hereafter.

Note that more complex operational components with multiple in- and out-ports can be modeled by subgraphs as follows: Each port is represented by a vertex. If the order of the vertices is given, we connect the vertices accordingly via a cycle. If the order is specified and if it is furthermore important whether this order occurs clockwise or counterclockwise, we can reuse the machinery of *embedding constraints* and their planarization, as described in [28]. If, on the other hand the order is not specified at all, we connect all in-port vertices to a new vertex, all out-ports to a second new vertex, and then connect both new vertices. In any case, we have to forbid any crossings through these new subgraphs (using essentially the same technique described below for the edge $s_i s_o$). In the following, we will not consider complex ports and operational components, but note that the aforementioned upward planarization approach also allows port constraints, as described in [15].

6.3 Crossing Minimization of Circuit Designs

We have to augmented our above algorithms regarding $\mathrm{mcr}_W(G)$ in order to restrict the use of the edge $s_i s_o$. The simplest possibility to enforce the restriction is to assign a crossing weight $k = \min(|I|, |O|) + 1$ to the restricted edge, or to replace the edge by a parallel substructure of that thickness. An optimal solution will then never cross through $s_i s_o$ or its replacement, since it is cheaper to cross over all other edges incident to s_i or s_o, close to these vertices.

Although this strategy suffices, we prefer a method which does not require edge costs or the enlargement of the graph: The key concept of the insertion algorithm is to find a shortest path in the dual of the graph (or within subgraphs). To forbid the crossing of an edge then means to remove its dual from the routing network. It is obvious that this strategy still always allows us to find an insertion path. We have:

Proposition 11 *The edge insertion problem corresponding to* $\mathrm{mcr}_{\hat{\mathcal{E}}}^{s_i s_o}$ *can be solved in linear time, both in the fixed and the variable embedding scenario.*

6.4 Experimental Results

Again, we implemented our algorithms using OGDF [14] and ran our tests on the same Xeon Linux system we used in Section 5.1. We chose all hypergraphs from the established synthesized sequential benchmark set Synth [5],

			FIX			VAR				
	circuit	#gates	time	ccr-1	ccr-10	time	ccr-1	ccr-10	ccr [26]	red.
Synth	*add6*	229	0.14	**94**	**68**	4.98	**70**	**64**	112	42.9%
	adr4	147	0.06	**40**	**40**	1.29	**63**	**41**	74	44.6%
	alu1	85	0.02	**42**	**42**	0.78	**36**	**36**	60	40.0%
	alu2	189	0.13	**181**	**128**	9.15	**140**	**121**	243	50.2%
	alu3	218	0.20	**190**	**166**	12.34	**226**	**183**	331	44.7%
	co14	145	0.05	**36**	**33**	0.91	**33**	**32**	52	38.5%
	dk17	168	0.12	**159**	**138**	7.70	**131**	**116**	188	38.3%
	dk27	78	0.02	**43**	**40**	0.38	**43**	**34**	45	24.4%
	dk48	194	0.21	**204**	**170**	14.43	**171**	**166**	290	42.8%
	mish	215	0.10	**12**	**11**	0.43	**11**	**10**	49	79.6%
	radd	121	0.04	**38**	**29**	0.56	**30**	**24**	37	35.1%
	rckl	338	0.40	**192**	**186**	18.89	**164**	**157**		
	rd53	134	0.08	**85**	**69**	1.73	**75**	**75**	126	40.5%
	vg2	185	0.10	**92**	**86**	5.85	**74**	**74**	131	43.5%
	x1dn	186	0.10	**85**	**78**	6.34	**80**	**74**	134	44.8%
	x9dn	203	0.15	**115**	**100**	4.34	**77**	**77**	158	51.3%
	z4	125	0.04	**48**	**43**	0.72	**47**	**41**	66	37.9%
	Z9sym	438	7.81	**1074**	**917**	473.87	**824**	**763**	1802	57.7%
Iscas'85	*c17*	4	0.00	**1**	**1**	0	**1**	**1**		
	c432	153	0.41	**453**	**409**	23.92	**471**	**368**		
	c499	170	1.55	**708**	**574**	145.22	**578**	**546**		
	c880	357	3.18	**910**	**716**	471.94	**787**	**701**		
	c1196	516	75.37	**2195**	**1998**	3720.44	**2008**	**1765**		
	c1238	495	117.85	**2460**	**2353**	8968.46	**2252**	**2068**		
	c1355	514	2.99	**756**	**684**	240.23	**708**	**561**		
	c1908	855	20.16	**1571**	**1382**	1999.33	**1335**	**1328**		

Table 3: Test results for the Synth and Iscas'85 benchmark circuits. FIX and VAR denote whether the insertion algorithms assume a prespecified embedding or not, ccr-1 and ccr-10 are the obtained circuit crossing numbers after 1 and 10 runs, respectively. Where available, the last two columns give the previously best reported values for ccr in [26] and our relative reduction thereto.

and the Iscas'85, Iscas'89, and Itc'99 benchmark sets of real-world electrical networks[5]. In the latter, we considered all graphs with up to 1000 gates. This leads to hypergraphs with up to 1800 vertices and 2800 edges in their point-based transformations.

Tables 3 and 4 show our results. For both the fixed and the variable embedding versions, we give the runtime (in seconds) for one run, and the resulting number of crossings after one (ccr-1) and after 10 randomized runs (ccr-10). In each run, we also applied the postprocessing described in Section 5. The last two columns show the previously best known number of crossings reported by Eschbach et al. [26] and the relative reduction of this number achieved by our best result; empty cells in these columns refer to instances not considered in [26].

Our algorithm clearly outperforms the results summarized in [26]. The best

[5]We collected all these benchmark sets at `http://www.ae.uni-jena.de/Research/ElectricalNetworks.html`.

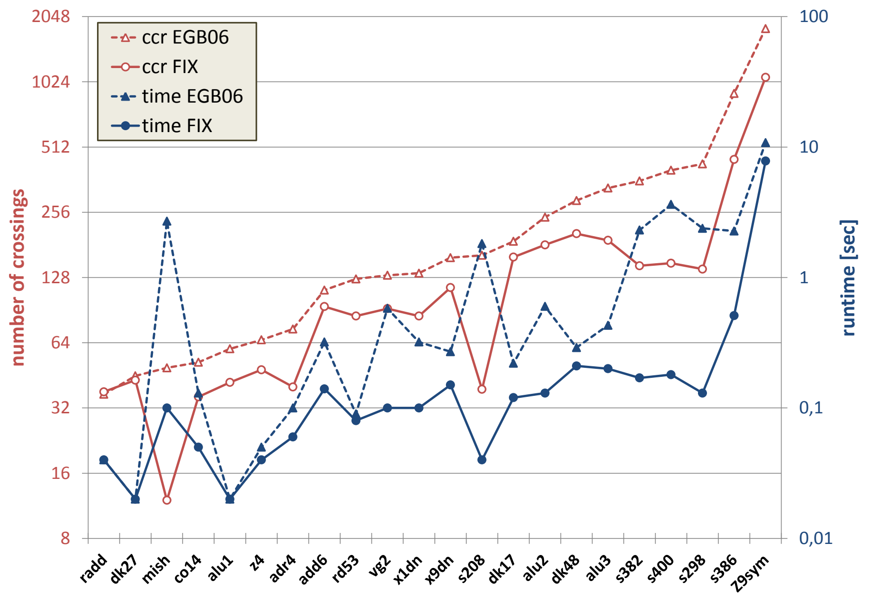


Figure 8: Comparison of computing ccr between a single run of our fixed embedding planarizer (FIX) and the algorithm of [26] (EGB06).

results we obtained for each circuit required on average 42% less crossings for the Synth instances, and even 71.3% less crossings for the Iscas'89 instances.

Comparing our experimental results with the results obtained by [26] regarding runtime is difficult, since the experiments were carried out on different machines.[6] They used a (not further specified) 2 GHz Linux PC with 1 GB RAM. Nevertheless, it is worthwhile to compare our fastest heuristic (fixed embedding with a single run, FIX) with their results (EGB06)—baring in mind that our machine is surely faster. Figure 8 shows the obtained number of crossings (left vertical axis) as well as the runtimes (right vertical axis) for all the circuits considered in [26] (the instances are sorted by increasing number of crossings with EGB06). FIX clearly dominates EGB06 with respect to number of crossings (for many graphs, it achieves 1.5–2 times less crossings) and is also faster (for some graphs up to a factor of 10). Even taking the different performance of the machines into account, FIX is definitely competitive w.r.t. running time.

Eschbach et al. [26] report on further experiments with an additional, time-consuming optimization technique called *windows optimization*, which required about 40–150 times more runtime for most instances. However, they could not achieve any significant improvements. On the other hand, our more time-consuming heuristics (using several permutations with fixed or variable embedding;

[6]Unfortunately, the source code of [26] is not publicly available, so we could not re-run their experiments on our machine.

			FIX			VAR				
	circuit	#gates	time	ccr-1	ccr-10	time	ccr-1	ccr-10	ccr [26]	red.
Iscas’89	*s27*	12	0.00	**1**	**1**	0.00	**1**	**1**		
	s208	102	0.04	**39**	**38**	0.48	**34**	**34**	162	79.0%
	s208a	111	0.03	**40**	**31**	0.44	**29**	**28**		
	s298	127	0.13	**140**	**116**	5.50	**128**	**116**	428	72.9%
	s344	164	0.09	**109**	**89**	3.27	**94**	**76**		
	s349	165	0.14	**94**	**80**	4.72	**83**	**76**		
	s382	173	0.17	**145**	**122**	4.60	**129**	**110**	357	69.2%
	s386	158	0.51	**449**	**360**	25.37	**385**	**310**	904	65.7%
	s400	177	0.18	**149**	**140**	7.18	**138**	**121**	400	69.8%
	s420	210	0.13	**120**	**83**	5.67	**96**	**84**		
	s420a	233	0.16	**87**	**83**	5.30	**78**	**75**		
	s444	196	0.19	**125**	**119**	7.63	**118**	**105**		
	s510	210	1.40	**857**	**764**	164.35	**716**	**678**		
	s526	208	0.38	**327**	**288**	26.93	**296**	**275**		
	s526a	209	0.60	**324**	**264**	30.38	**279**	**272**		
	s641	374	0.97	**394**	**394**	65.63	**425**	**385**		
	s713	389	1.10	**404**	**374**	49.45	**384**	**354**		
	s820	275	27.27	**1566**	**1378**	1129.80	**1603**	**1346**		
	s832	273	15.90	**1673**	**1527**	2058.63	**1360**	**1360**		
	s838	420	0.59	**281**	**269**	24.35	**277**	**247**		
	s838a	477	0.73	**187**	**170**	19.60	**172**	**158**		
	s953	401	14.47	**1936**	**1728**	1918.05	**1671**	**1555**		
	s1196	533	106.63	**2104**	**2016**	4333.75	**2005**	**1805**		
	s1423	726	2.23	**402**	**386**	90.63	**381**	**342**		
Itc’99	*b01*	40	0.01	**37**	**31**	0.38	**32**	**29**		
	b02	25	0.00	**16**	**11**	0.03	**11**	**9**		
	b03	137	0.06	**80**	**73**	1.77	**72**	**66**		
	b04s	570	3.79	**694**	**645**	407.00	**758**	**612**		
	b05s	872	20.58	**1567**	**1472**	4218.66	**1399**	**1312**		
	b06	46	0.02	**48**	**45**	0.46	**52**	**41**		
	b07s	403	1.34	**564**	**543**	138.64	**484**	**484**		
	b08	150	0.30	**250**	**196**	11.86	**232**	**181**		
	b09	156	0.10	**101**	**101**	2.57	**124**	**92**		
	b10	166	0.46	**355**	**299**	36.79	**313**	**300**		
	b11s	462	11.88	**1414**	**1182**	2016.01	**1366**	**1216**		
	b13s	309	0.30	**173**	**148**	12.84	**175**	**137**		

Table 4: Test results for the Iscas’89 and Itc’99 benchmark circuits. FIX and VAR denote whether the insertion algorithms assume a prespecified embedding or not, ccr-1 and ccr-10 are the obtained circuit crossing numbers after 1 and 10 runs, respectively. Where available, the last two columns give the previously best reported values for ccr in [26] and our relative reduction thereto.

using variable instead of fixed embedding) can reduce the number of crossings considerably (there are only a few exceptions when considering FIX vs. VAR).

We conclude with Figure 9, visually showcasing the benefit of our stronger crossing minimization in contrast to a published drawing of [25].

7 Conclusions and Open Problems

We have presented the first heuristics for minor and hypergraph crossing minimization based on the well-known planarization approach. To this end, we considered the complexity of insertion problems over graph minors. In particular, we showed how to insert edges optimally in polynomial time in these scenarios, both in the case of fixed and variable embeddings. Furthermore, we showed that while inserting a non-expandable vertex into a fixed embedding is polynomial-time solvable, inserting expandable vertices is NP-hard even in very restricted settings. An adaption to electrical circuits demonstrates the strength of our approach, leading to much better results than the current state-of-the-art for a large benchmark set of real-world circuits.

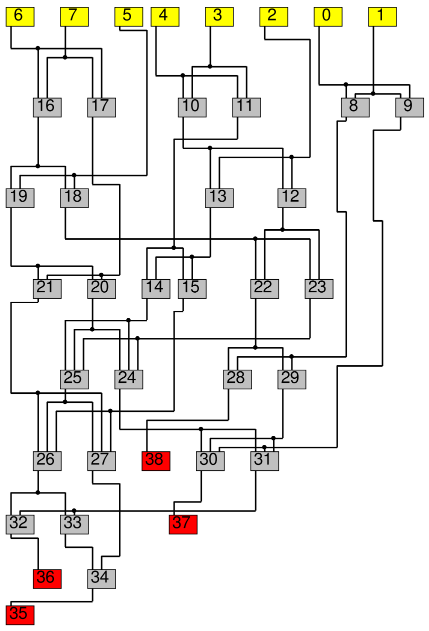
Considering the topics discussed in this paper there are (at least) three different aspects for further research we deem interesting:

Complexity of MVI-V. MVI-V with $W = \emptyset$ is known to be polynomial [16]. Thereby, despite the fact that the different inserted edges would prefer distinct embeddings, it turns out that one can find a single embedding satisfying all demands "well enough". When considering the case $W \neq \emptyset$, there is the additional problem that different inserted edges would want to split vertices differently. It is unclear if it is possible to find the best-possible tree expansion in polynomial time.

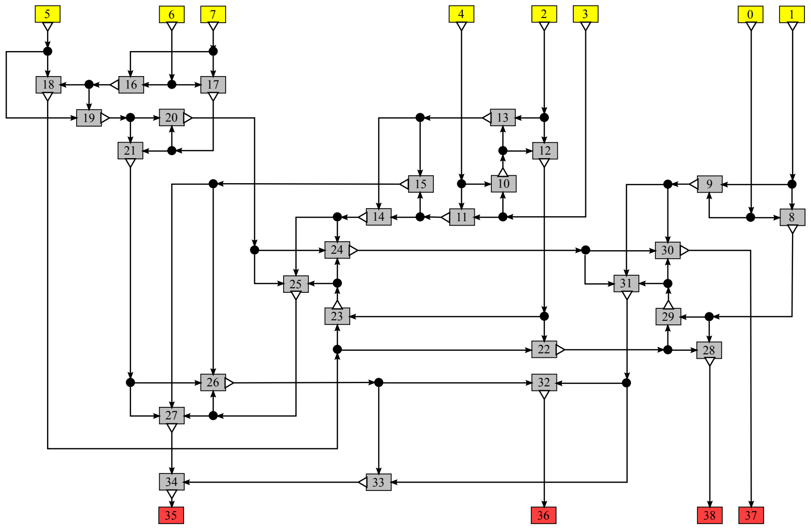
Approximations for minor crossing number. For the traditional crossing number, the solution of an edge or vertex insertion problem (G, st) or (G, U) (with variable embedding) was shown to approximate the crossing numbers of the augmented graphs $G+e$ [7,9,33] and $G+u$ [19], respectively. Does a similar connection hold for the minor crossing number and its insertion problems?

Exact algorithms based on integer linear programs (ILPs). In recent years, two different ILP formulations for solving the traditional crossing minimization problem to optimality in practice were presented [6,20]. Conceptually, they are based on binary variables for each edge pair, encoding whether these edges cross or not. Several types of constraints then encode the feasibility of the resulting solution.

Without going into details, we could extend both approaches using the following idea: any expandable vertex v is substituted by $\deg(v)$ vertices V'_v, one for each edge originally incident to v. Furthermore, we add additional $\deg(v)-2$ new vertices V''_v for each v. Adding edges between the nodes $V_v := V'_v \cup V''_v$ allows to build all possible treewise connections between V'_v. For each of these



(a) Layout from [25]; vertices recolored and background grid removed: 30 crossings



(b) A drawing realizing our computed planarization: 18 crossings

Figure 9: Example graph *rd84* from [25] (not part of the official Synth benchmark set).

possible edges we add a binary variable that encodes whether the edge is chosen or not. Two things remain. First, we add well-known undirected cut constraints $\mathcal{X}(V^\circ) \geq 1$ for all $\emptyset \neq V^\circ \subset V_v$ to ensure that each original v becomes a connected tree expansion. Thereby, $\mathcal{X}(V^\circ)$ denotes the sum over all variables corresponding to edges between vertices of V° and $V_v \setminus V^\circ$. Second, the original constraints of the ILP formulations have to become "deactivated" if any of the edges considered in them is one of our new edges and not chosen. This can be trivially achieved by subtracting the corresponding edge variable x_e (or its negation $1 - x_e$) from the constraint in a suitable fashion.

Yet, this naïve formulation is of no use in practice due to its sheer number of additional variables and the relatively weak constraints (due to the deactivation possibility). Hence the question is: Does there exist a practically relevant ILP formulation for the minor crossing number?

Acknowledgments

We thank Thomas Eschbach for kindly providing the Synth benchmark set.

References

[1] S. N. Bhatt and F. T. Leighton. A framework for solving VLSI graph layout problems. *Journal of Computer and System Sciences*, 28:300–343, 1984. doi:10.1016/0022-0000(84)90071-0.

[2] D. Bokal, M. Chimani, and J. Leaños. Crossing number additivity over edge cuts. *European Journal of Combinatorics*, 34(6):1010–1018, 2013.

[3] D. Bokal, E. Czabarkab, L. A. Székely, and I. Vrt'o. Graph minors and the crossing number of graphs. *Electronic Notes in Discrete Mathematics*, 28:169–175, 2007. doi:10.1016/j.endm.2007.01.024.

[4] D. Bokal, G. Fijavz, and B. Mohar. The minor crossing number. *SIAM Journal on Discrete Mathematics*, 20:344–356, 2006. doi:10.1137/05062706X.

[5] F. Brglez, D. Bryan, and K. Kozminski. Combinatorial profiles of sequential benchmark circuits. In *Proc. of IEEE International Symposium on Circuits and Systems, ISCAS 1989*, pages 1929–1934, 1989.

[6] C. Buchheim, M. Chimani, D. Ebner, C. Gutwenger, M. Jünger, G. W. Klau, P. Mutzel, and R. Weiskircher. A branch-and-cut approach to the crossing number problem. *Discrete Optimization*, 5(2):373–388, 2008. doi:10.1016/j.disopt.2007.05.006.

[7] S. Cabello and B. Mohar. Crossing and weighted crossing number of near planar graphs. In *Proc. of 16th International Symposium on Graph Drawing, GD 2008*, volume 5417 of *LNCS*, pages 38–49. Springer, 2008. doi:10.1007/978-3-642-00219-9_5.

[8] S. Cabello and B. Mohar. Adding one edge to planar graphs makes crossing number hard. In J. Snoeyink, M. de Berg, J. S. B. Mitchell, G. Rote, and M. Teillaud, editors, *Proc. of 26th ACM Symposium on Computational Geometry, SoCG 2010*, pages 68–76. ACM, 2010. doi:10.1145/1810959.1810972.

[9] S. Cabello and B. Mohar. Crossing number and weighted crossing number of near-planar graphs. *Algorithmica*, 60(3):484–504, 2011. doi:10.1007/s00453-009-9357-5.

[10] S. Cabello and B. Mohar. Adding one edge to planar graphs makes crossing number and 1-planarity hard. *SIAM Journal on Computing*, 42(5):1803–1829, 2013. doi:10.1137/120872310.

[11] M. Chimani. *Computing Crossing Numbers.* PhD thesis, TU Dortmund, Germany, 2008.

[12] M. Chimani and C. Gutwenger. Algorithms for the hypergraph and the minor crossing number problems. In *Proc. of 18th International Symposium on Algorithms and Computation, ISAAC 2007*, volume 4835 of *LNCS*, pages 184–195. Springer, 2007. `doi:10.1007/978-3-540-77120-3_18`.

[13] M. Chimani and C. Gutwenger. Advances in the planarization method: Effective multiple edge insertions. *Journal of Graph Algorithms and Applications*, 16(3):729–757, 2012. `doi:10.7155/jgaa.00264`.

[14] M. Chimani, C. Gutwenger, M. Jünger, G. W. Klau, K. Klein, and P. Mutzel. The Open Graph Drawing Framework (OGDF). In R. Tamassia, editor, *Handbook of Graph Drawing and Visualization*, chapter 17, pages 543–569. CRC Press, 2013. See also `http://www.ogdf.net`.

[15] M. Chimani, C. Gutwenger, P. Mutzel, M. Spönemann, and H.-M. Wong. Crossing minimization and layouts of directed hypergraphs with port constraints. In *Proc. of 18th International Symposium on Graph Drawing, GD 2010*, volume 6502 of *LNCS*, pages 141–152. Springer, 2010. `doi:10.1007/978-3-642-18469-7_13`.

[16] M. Chimani, C. Gutwenger, P. Mutzel, and C. Wolf. Inserting a vertex into a planar graph. In *Proc. of 20th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '09*, pages 375–383. ACM Press, 2009. `doi:10.1137/1.9781611973068.42`.

[17] M. Chimani, C. Gutwenger, P. Mutzel, and H.-M. Wong. Layer-free upward crossing minimization. *ACM Journal of Experimental Algorithmics*, 15:2.2:1–2.2:27, 2010. `doi:10.1145/1671970.1671975`.

[18] M. Chimani and P. Hliněný. A tighter insertion-based approximation of the crossing number. In *Proc. of 38th International Colloquium on Automata, Languages and Programming, ICALP 2011*, volume 6755 of *LNCS*, pages 122–134. Springer, 2011. Full version at ArXiv, id 1104.5039. `doi:10.1007/978-3-642-22006-7_11`.

[19] M. Chimani, P. Hliněný, and P. Mutzel. Vertex insertion approximates the crossing number for apex graphs. *European Journal of Combinatorics*, 33(3):326–335, 2012. `doi:10.1016/j.ejc.2011.09.009`.

[20] M. Chimani, P. Mutzel, and I. Bomze. A new approach to exact crossing minimization. In *Proc. of 16th Annual European Symposium on Algorithms, ESA 2008*, volume 5193 of *LNCS*, pages 284–296. Springer, 2008. `doi:10.1007/978-3-540-87744-8_24`.

[21] G. Di Battista, P. Eades, R. Tamassia, and I. Tollis. *Graph Drawing*. Prentice Hall, 1999.

[22] G. Di Battista, A. Garg, G. Liotta, R. Tamassia, E. Tassinari, and F. Vargiu. An experimental comparison of four graph drawing algorithms. *Comput. Geom. Theory Appl.*, 7(5-6):303–325, 1997. `doi:10.1016/S0925-7721(96)00005-3`.

[23] G. Di Battista and R. Tamassia. On-line maintanance of triconnected components with SPQR-trees. *Algorithmica*, 15(4):302–318, 1996. `doi:10.1007/BF01961541`.

[24] G. Di Battista and R. Tamassia. On-line planarity testing. *SIAM Jorunal on Computing*, 25(5):956–997, 1996. `doi:10.1137/S0097539794280736`.

[25] T. Eschbach, W. Günther, and B. Becker. Orthogonal circuit visualization improved by merging the placement and routing phases. In *Proc. of 18th International Conference on VLSI Design, VLSID 2005*, pages 433–438, 2005. `doi:10.1109/ICVD.2005.134`.

[26] T. Eschbach, W. Günther, and B. Becker. Orthogonal hypergraph drawing for improved visibility. *Journal of Graph Algorithms and Applications*, 10(2):141–157, 2006. `doi:10.7155/jgaa.00122`.

[27] M. R. Garey and D. S. Johnson. The rectilinear Steiner tree problem is NP-complete. *SIAM Journal on Applied Mathematics*, 32:826–834, 1977.

[28] C. Gutwenger, K. Klein, and P. Mutzel. Planarity testing and optimal edge insertion with embedding constraints. *Journal of Graph Algorithms and Applications*, 12(1):73–95, 2008. `doi:10.7155/jgaa.00160`.

[29] C. Gutwenger and P. Mutzel. A linear time implementation of SPQR trees. In J. Marks, editor, *Proc. of 8th International Symposium on Graph Drawing, GD 2000*, volume 1984 of *LNCS*, pages 77–90. Springer-Verlag, 2001. `doi:10.1007/3-540-44541-2_8`.

[30] C. Gutwenger and P. Mutzel. An experimental study of crossing minimization heuristics. In *Proc. of 11th International Symposium on Graph Drawing, GD 2003*, volume 2912 of *LNCS*, pages 13–24. Springer, 2004. `doi:10.1007/978-3-540-24595-7_2`.

[31] C. Gutwenger, P. Mutzel, and R. Weiskircher. Inserting an edge into a planar graph. *Algorithmica*, 41(4):289–308, 2005. `doi:10.1007/s00453-004-1128-8`.

[32] P. Hliněný. Crossing number is hard for cubic graphs. *Journal of Combinatorial Theory, Series B*, 96:455–471, 2006. `doi:10.1016/j.jctb.2005.09.009`.

[33] P. Hliněný and G. Salazar. On the crossing number of almost planar graphs. In *Proc. of 14th International Symposium on Graph Drawing, GD 2006*, volume 4372 of *LNCS*, pages 162–173. Springer, 2006. `doi:10.1007/978-3-540-70904-6_17`.

[34] J. E. Hopcroft and R. E. Tarjan. Dividing a graph into triconnected components. *SIAM Journal on Computing*, 2(3):135–158, 1973. `doi:10.1137/0202012`.

[35] J. E. Hopcroft and R. E. Tarjan. Efficient algorithms for graph manipulation. *Communications of the ACM*, 16(6):372–378, 1973. `doi:10.1145/362248.362272`.

[36] D. S. Johnson and H. O. Pollak. Hypergraph planarity and the complexity of drawing Venn diagrams. *Journal of Graph Theory*, 11(3):309–325, 1987. `doi:10.1002/jgt.3190110306`.

[37] H. Klemetti, I. Lapinleimu, E. Mäkinen, and M. Sieranta. A programming project: Trimming the spring algorithm for drawing hypergraphs. *SIGCSE Bulletin*, 27(3):34–38, 1995.

[38] E. Mäkinen. How to draw a hypergraph. *International Journal of Computer Mathematics*, 34:177–185, 1990.

[39] G. Sander. Layout of directed hypergraphs with orthogonal hyperedges. In *Proc. of 11th International Symposium on Graph Drawing, GD 2003*, volume 2912 of *LNCS*, pages 381–386. Springer, 2004. `doi:10.1007/978-3-540-24595-7_35`.

[40] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(2):109–125, 1981. `doi:10.1109/TSMC.1981.4308636`.

[41] R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972. `doi:10.1137/0201010`.

[42] I. Vrt'o. Crossing numbers of graphs: A bibliography, 2014. URL: `ftp://ftp.ifi.savba.sk/pub/imrich/crobib.pdf`.