

Inserting a Vertex into a Planar Graph

Markus Chimani* Carsten Gutwenger* Petra Mutzel* Christian Wolf*

Abstract

We consider the problem of computing a crossing minimum drawing of a given planar graph $G = (V, E)$ augmented by a star, i.e., an additional vertex v together with its incident edges $E_v = \{(v, u) \mid u \in V\}$, in which all crossings involve E_v . Alternatively, the problem can be stated as finding a planar embedding of G , in which the given star can be inserted requiring the minimum number of crossings. This is a generalization of the crossing minimum edge insertion problem [15], and can help to find improved approximations for the crossing minimization problem. Indeed, in practice, the algorithm for the crossing minimum edge insertion problem turned out to be the key for obtaining the currently strongest approximate solutions for the crossing number of general graphs. The generalization considered here can lead to even better solutions for the crossing minimization problem. Furthermore, it offers new insight into the crossing number problem for almost-planar and apex graphs.

It has been an open problem whether the star insertion problem is polynomially solvable. We give an affirmative answer by describing the first efficient algorithm for this problem. This algorithm uses the SPQR-tree data structure to handle the exponential number of possible embeddings, in conjunction with dynamic programming schemes for which we introduce *partitioning cost* subproblems.

1 Introduction

The *crossing number* problem is to find the smallest number of edge crossings necessary when drawing a graph into the plane. Various aspects of the problem have been studied for over half a century, see [20] for an extensive bibliography, yet relatively little is known about many of its properties. Garey and Johnson [11] showed that the problem is NP-hard, even for cubic graphs as shown by Hliněný [16]; Grohe [13] showed that it is fixed parameter tractable, even in linear time as shown by Kawabayashi [19]. Unfortunately, these FPT algorithms are only of theoretical interest due to the high constants involved.

An *almost-planar graph* G_e is a graph with an edge e whose removal leaves a planar graph. Analogously, an *apex graph* G_v is a graph with a vertex v whose removal (together with its incident edges) leaves a planar graph.

Let $G = (V, E)$ be the planar graph after the removal, $e = \{u, v\} \notin E$ the removed edge of the almost-planar graph, and $v \notin V$ and $W \subseteq V$ the removed vertex and its adjacent vertices in the apex graph. Surprisingly little is known about the crossing numbers $\text{cr}(G_e)$ and $\text{cr}(G_v)$. In particular, it is unclear whether these problems are polynomially solvable even though it is conjectured that the former lies in \mathcal{P} and the latter only in \mathcal{NP} [17].

Regarding approximations for the general crossing number problem, the best known polynomial algorithm approximates not directly the crossing number but $n + \text{cr}(G)$ within a factor of $\log^3 n$ [10]. Thereby n is the number of graph vertices, and the graph has to have bounded degree. The only constant factor approximations for $\text{cr}(G)$ known are for projective [12] and almost-planar [17, 4] graphs with bounded degree.

When we want to compute the crossing number in practice, we can nowadays solve the problem to provable optimality using integer linear programs and branch-and-cut-and-price techniques [2, 7]. Yet, this approach only works for relatively sparse graphs with up to 100 vertices, which hence allow a comparably small crossing number. For larger or more complex graphs, we have to resort to approximate solutions. The currently most successful algorithm—which also constitutes the aforementioned approximation algorithm for almost-planar graphs with bounded degree, and often gives (near)optimal solutions [5] for general graphs—works as follows: in the first step we compute a large planar subgraph of the given graph. In the second step we reinsert the temporarily removed edges one by one; thereby the crossings are replaced by dummy vertices of degree four such that the edge insertion is always performed on a planar graph. The same approach can be followed by starting with a vertex-induced subgraph [9], and iteratively inserting vertices together with their incident edges, so-called *stars*. These approaches give rise to multiple insertion problems:

DEFINITION 1.1. (EIF, SIF, EIV, SIV) *Given G and a fixed embedding Π of G . Insert e (or v with all its incident edges $\{v\} \times W$, $W \subseteq V$) into this embedding with the minimum number of crossings. This variant is known as edge (star) insertion with fixed embedding EIF (SIF), respectively.*

*Chair for Algorithm Engineering, Faculty of Computer Science, TU Dortmund, Germany. {markus.chimani, carsten.gutwenger, petra.mutzel, christian.wolf}@tu-dortmund.de

We achieve smaller crossing numbers by considering the more complex edge (star) insertion with variable embedding EIV (SIV), respectively: given G , find a planar embedding such that the insertion requires the minimum number of crossings.

In the literature, the star insertion problem is sometimes also called *vertex insertion*. Clearly, solving the star insertion problem is superior to iteratively solving edge insertion problems for the individual star's edges: adding the first of these edges will always be possible without any crossings, but the optimal solution for the star insertion might require a crossing on this edge.

We can further generalize the above problems to a (planar) subgraph insertion problem with variable embeddings, i.e., given a planar graph G , find an embedding Π of G such that the insertion of a (planar) subgraph S requires the minimum number of crossings, while preserving the induced embedding Π . Such problems are of further theoretical interest: if we could solve the (planar) subgraph insertion problem, we could also compute the exact crossing number for almost-planar and apex graphs: let e be the edge that makes the almost-planar graph non-planar and insert $S = G_e - e$ into the trivial embedding of e . Analogously, we can start with the star incident to v that makes the apex graph non-planar, and insert $G_v - v$ into it. This is further true for graphs that are planar if you remove a 3-cycle: the 3-cycle will never cross itself and can therefore be used as G .

Based on the conjecture [17], it is very likely that the general planar subgraph problem is NP-hard, even for fixed embeddings; it is clearly NP-hard if S is non-planar. Hence the question arises: How complex can the subgraph S become before the insertion problem becomes untractable? Until now, the only known polynomial case was with S being a single edge. We show that the problem remains polynomially solvable if S is a star.

Solving insertion problems. EIF can be trivially solved by computing a shortest path in the graph's dual. Reusing this idea, we can solve SIF: Let D be the dual graph of Π , cf. Figure 1: We assign a distance label $d(f)$ to each face f in Π , which is initially 0. We then start a BFS in D from each vertex $w \in W$, temporarily adding edges from w to its incident faces. We add the BFS index (minus 1) of the visited face to its distance label. In the end, the label $d(f)$ holds the numbers of required crossings if v is inserted at f , and we can choose a minimal one.

On the other hand, optimizing over the exponentially large set of all possible embeddings turns out to be far more challenging. After being a long-standing

open problem by itself, it was shown in [15] that EIV can be solved in linear time with the help of SPQR-trees. Later, it has been shown that this algorithm actually approximates the crossing number of almost-planar graphs with bounded degree [17].

Yet, it remained an open problem whether SIV is NP-hard, as most properties which made EIV solvable do not carry forward to SIV. Interestingly, when considering the *minor-monotone crossing number* [1], the star insertion problem becomes NP-hard even for fixed embeddings [3]. In this paper we show that the SIV problem can be solved in polynomial time, using SPQR-trees and dynamic programming techniques.

Overview. In the next section, we recapitulate some basic definitions concerning SPQR-trees, and describe the main complications why SIV is harder than EIV. Afterwards, we present the algorithm for efficiently solving SIV for 2-connected graphs in Sect. 3 and generalize this result to not necessarily 2-connected graphs in Sect. 4. Sect. 5 concludes with stating two interesting open problems.

2 Preliminaries & Complications

SPQR-trees. A *block* is a maximal 2-connected subgraph of a graph G . If G is a 2-connected graph, the *SPQR-tree* \mathcal{T} of G represents its decomposition into triconnected components [18]. We only describe the idea of SPQR-trees briefly, please refer to [8] for a formal definition. The SPQR-tree \mathcal{T} reflects the triconnectivity structure of G which is comprised of

- serial structures (*S-nodes*);
- parallel structures (*P-nodes*); and
- triconnected structures (*R-nodes*).

With each node μ of \mathcal{T} , a *skeleton* graph G_μ is associated. According to the type of μ , its skeleton graph is either a cycle of at least three vertices (S-node), a bundle of at least three parallel edges (P-node), or a triconnected simple graph (R-node). See Figure 2 for an example, ignoring the dashed lines as they will be discussed later.

A skeleton can be seen as a sketch of G in the following sense. An edge (u, v) in G_μ is either a *real* edge corresponding to an edge (u, v) in G , or a *virtual* edge corresponding to a uv -component of G . The respective uv -component of a virtual edge is determined by a neighbor η of μ whose skeleton G_η contains an edge (u, v) as well; η is also called the *pertinent node* of the virtual edge (u, v) . Hence, the skeletons of two adjacent nodes μ and η can be merged by contracting the edge (μ, η) , identifying the corresponding virtual edges in the skeletons, and finally removing the resulting

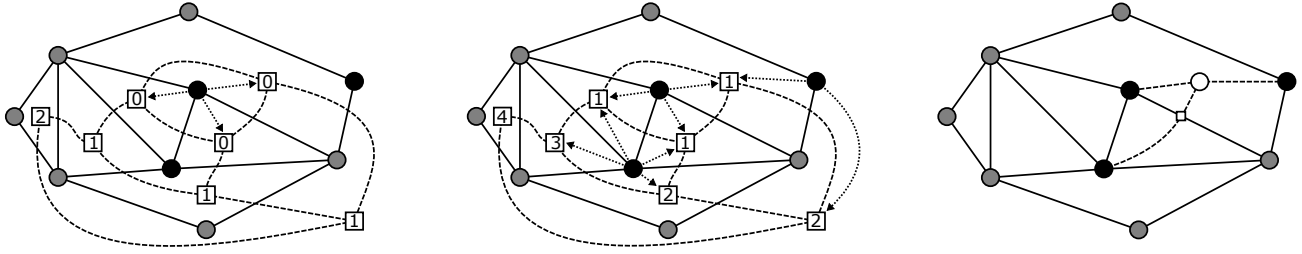


Figure 1: Solving SIF in the dual graph: the given graph is denoted by circular vertices and solid edges; the affected vertices are black. The dual graph induced by the shown embedding is drawn with square vertices and dashed edges. **(left)** the face labels after the first BFS computation starting at the affected vertex in the center of the graph; **(center)** the face labels after all BFS runs; **(right)** the graph after the star insertion (including the small square dummy vertex).

virtual edge. Exhaustively merging skeletons recreates the graph G .

We call the uv -component corresponding to a virtual edge e the *expansion graph* of e , and replacing a virtual edge e by its expansion graph is referred to as expanding e . If we consider \mathcal{T} as a rooted tree, the *reference edge* of a node μ in \mathcal{T} is the virtual edge e_{ref} in the skeleton graph G_μ whose pertinent node is the parent of μ , and the graph obtained by expanding all virtual edges in G_μ is the *pertinent graph* of μ .

Additionally, each edge of G can be represented by a Q-node whose skeleton is simply a 2-cycle (one virtual and one real edge); in this case, the former real edge in a skeleton graph is replaced by a virtual edge whose pertinent node is this Q-node. Since Q-nodes do not carry structural information, we rarely use them.

SPQR-trees and Insertion Problems. A central property of SPQR-trees is that they have linear size and can be obtained in linear time [14]. Nonetheless, if G is planar, they implicitly encode all exponentially many planar embeddings of G : note that a simple 3-connected graph allows only a single planar embedding and its mirror. We can enumerate all possible embeddings by choosing between the two embeddings for each R-node, and enumerating all possible orderings of the edges in each P-node. We obtain $2^r \cdot \prod_{i=1}^p (t_i - 1)!$ embeddings, where r and p are the number of R- and P-nodes, respectively, and t_i gives the number of edges in the skeleton of the i -th P-node.

For solving the EIV problem, as described in [15], certain properties of the SPQR-tree representation can be taken into account. The following observation is essential for both inserting an edge as well as inserting a star. Consider a virtual edge e in a skeleton graph. If we want to route a newly inserted edge such that it crosses e , this means that this edge has to cross through the

expansion graph of e . The number of crossings required to cross a particular embedding Π of this graph can be obtained by computing a shortest path in the dual graph of Π . The following lemma has been shown in [15]:

LEMMA 2.1. *The number of crossings required to cross through the expansion graph G' of $e = \{s, t\}$ are independent of the embedding Π . It can be computed as the minimum s - t -cut in G' .*

Therefore, this number is also called the *traversing costs* of e . But on the other hand, many properties exploited for efficiently solving EIV do not hold in the case of SIV:

- The insertion path of the inserted edge for EIV only goes through tree nodes which lie on a simple sub-path within \mathcal{T} . The embedding of any tree node not part of this path is irrelevant and we do not need to compute the minimum-cut through their skeletons. This property does clearly not hold for SIV, as the nodes of W may be arbitrarily distributed over the full SPQR-tree. Hence we cannot reduce our search to a simple path.
- Only considering the tree nodes on the aforementioned path in \mathcal{T} , EIV can be decomposed into sub-problems in which we ask for a shortest insertion path through one of these nodes. Each subproblem can be solved independently, thus allowing a polynomial algorithm. For SIV, such subproblems are no longer independent: as we will see in the following, routing multiple edges through the skeleton of some node $\mu \in \mathcal{T}$ depends on the continuation of the insertion paths within the child tree nodes. Conversely, the routing within the child tree nodes also depends on the routing of the edges in μ .
- Solving the insertion path for a P-node μ is trivial in the context of EIV: let G' be the skeleton of

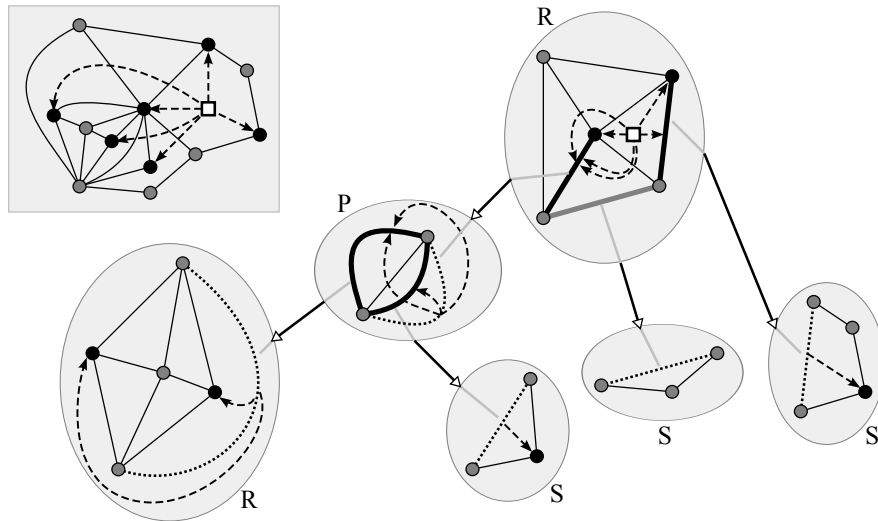


Figure 2: The given graph (in the top-left box), and its SPQR-tree decomposition. Virtual edges are thick, reference edges are dotted lines. Affected vertices are black; so are virtual edges that represent substructures with affected vertices. In the figure, we insert the new vertex (square) at some face in the root’s skeleton; the dashed lines denote insertion paths for the edges connecting the new vertex with the affected vertices.

μ , then the two edges in G' whose expansions hold the source and the target vertex of the new edge, respectively, can simply be placed next to each other. The ordering of the other edges is irrelevant, as they will not be crossed. For SIV, the P-nodes turn out to constitute a major challenge, as multiple edges contain vertices of the affected nodes W : even the ordering of only some of these edges can directly influence the crossing number of all paths considered in μ .

3 SIV in 2-Connected Graphs

In the following, assume we are given a 2-connected planar graph $G = (V, E)$ with its SPQR-tree \mathcal{T} . We want to solve the SIV problem for a new vertex v that shall be adjacent to the vertices $W \subseteq V$. We call these vertices the *affected* vertices. To solve SIV, we want to identify an embedding into which we can later insert v using the trivial SIF algorithm.

The overall idea of the algorithm is to successively root \mathcal{T} at all of its R-nodes and all Q-nodes that are adjacent to a P-node; for simplicity, consider \mathcal{T} with all Q-nodes pruned (except for the root node). Considering any possible face f in the skeleton of \mathcal{T} ’s root node, we compute the required number of crossings when inserting v into f .

Let ρ be the root node of \mathcal{T} , and let G_ρ be the skeleton of ρ . Since ρ is either 3-connected or trivial, G_ρ has only one embedding (and its mirror). It is easy to enumerate and visualize all faces, cf. Figure 2. One

after another, we fix a face f in G_ρ and compute the crossing number when inserting v in f . Generally, some vertices of W directly correspond to vertices in G_ρ . The other vertices of W lie within some components that are replaced by virtual edges in G_ρ . Considering the dual graph of G_ρ , we can conceptually—see Sect. 3.2 for details—compute the number of required crossings by summing up the distances to (a) the vertices $V(G_\rho) \cap W$, (b) to the virtual edges which “hide” the other vertices of W , and (c) within those shrunk components to the actual node. The challenge for the latter cases is that we do not know the arising crossing number in advance, when the edge is expanded into its actual subgraph. In particular, the arising crossings differ depending on whether insertion paths going “into” the virtual edge all come from the same side, or from different ones. We call this central subproblem the *partitioning cost* (PC) problem, see below. With a solution to the PC problem for each edge in the skeleton of \mathcal{T} ’s root node (Sect. 3.1), we can solve the insertion cost with respect to any face f in G_ρ (Sect. 3.2).

3.1 The Partitioning Cost (PC) Problem.

Given a rooted SPQR tree \mathcal{T} and a non-root node μ of \mathcal{T} with reference edge e . Let $W' \subseteq W$ be the set of affected vertices in the subtree rooted at μ , disregarding the vertices incident to e . Consider a subdivision of e , introducing a new temporary node u , and insert the edges $\{u, w\}$ for all $w \in W'$. We denote these new edges by E' . Considering the cyclic order of the edges

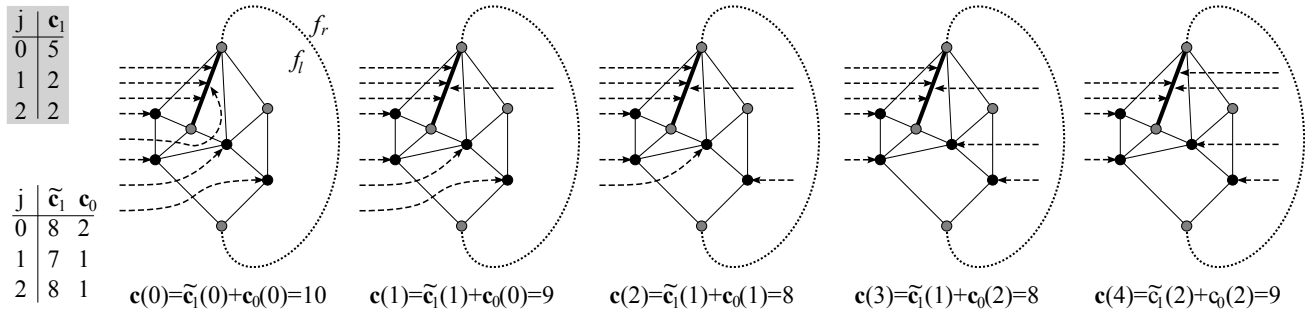


Figure 3: Example of computing the cost vector for some R-node, containing only a single virtual edge (bold) with precomputed cost vector \mathbf{c}_1 .

incident to u , we can observe that the subdivided edge e bipartitions E' into edges on one side and other edges on the other side of e . This observation leads to:

DEFINITION 3.1. (PARTITIONING COST PROBLEM)

Let \mathcal{T} , μ , e , W' , u , and E' be defined as above, and let $k = |W'|$. For each $0 \leq j \leq \lfloor \frac{k}{2} \rfloor$, find an embedding which allows the smallest number of edge crossings when inserting E' into the pertinent graph of μ where j edges lie on one side of e , and $|W'| - j$ edges on the other.

In this paper we will concentrate on the computation of the required number of crossings. We state that saving the extra information necessary for reconstructing the actual embedding is straight-forwardly possible. The solution of a PC problem will henceforth be a cost vector \mathbf{c} of length $\lfloor \frac{k}{2} \rfloor + 1$. We denote the i -th component of this vector by $\mathbf{c}(i)$.

We solve the PC problem recursively in a bottom-up traversal of the SPQR-tree. Hence, when considering μ , we already know the cost vector \mathbf{c}_g for any virtual edge g in the skeleton of μ , and only have to solve the problem on the skeleton of μ with all these cost vectors for the virtual edges. Furthermore, we can use Lemma 2.1 to compute the arising number of crossings when an inserted edge crosses a virtual edge. We differentiate based on the type of μ :

μ is an S- or R-node. Let G_μ be the skeleton graph of μ , cf. Figure 3. Some of the vertices in W' are already contained in G_μ ; we denote this subset with W_0 . The remaining vertices are contained in the expansion graphs of edges in G_μ . Let e_1, \dots, e_ℓ be these skeleton edges, and W_k the corresponding vertices of $W' \setminus W_0$ contained in the expansion graph of e_k ($1 \leq k \leq \ell$).

Suppose we have already solved the PC problem for each pertinent node of e_k and W_k , giving us cost vectors $\mathbf{c}_1, \dots, \mathbf{c}_\ell$. We compute the cost vector \mathbf{c} for μ and W' in two steps:

1. We compute a cost vector \mathbf{c}_0 that only covers the costs of inserting edges to the vertices in W_0 , i.e., we solve the PC problem for μ and W_0 .
2. We apply a dynamic programming approach computing the cost vectors $\tilde{\mathbf{c}}_i$ for the PC problem for μ and $W_0 \cup W_1 \cup \dots \cup W_i$ ($1 \leq i \leq \ell$). Then, the cost vector \mathbf{c} for μ and W' is simply $\tilde{\mathbf{c}}_\ell$.

Ad 1. Recall that since μ is either an S-node or an R-node, G_μ has a unique embedding (and its mirror). Hence, it is sufficient to consider only a fixed embedding of G_μ in any case; if μ is an R-node, the cost vector \mathbf{c} can be obtained from the two solutions by taking the componentwise minimum, i.e., if $\mathbf{c}^{(1)}$ and $\mathbf{c}^{(2)}$ are the solutions for the two possible embeddings, the resulting cost vector \mathbf{c} is given by $\mathbf{c}(i) = \min(\mathbf{c}^{(1)}(i), \mathbf{c}^{(2)}(i))$.

Let f_l and f_r be the two faces adjacent to the reference edge e in a fixed embedding Π_μ of G_μ . For solving the first step, we compute for each $w \in W_0$ the minimal number of crossings required to insert edges leading from w to the face f_l and f_r , respectively; we denote the number of crossings with $c_l(w)$ and $c_r(w)$. This can be done by computing a weighted shortest path (from a face adjacent with w to f_l or f_r , respectively) in the dual graph of Π_μ , where the traversing costs of the skeleton edges are the weights of their dual edges.

Let $W_0 = \{w_1, \dots, w_p\}$. We compute a sequence $\mathbf{c}'_1, \dots, \mathbf{c}'_p$ of cost vectors, such that $\mathbf{c}'_i(j)$ denotes the minimum number of crossings required to insert edges to $\{w_1, \dots, w_i\}$ such that j edges lead to f_l and $i - j$ edges lead to f_r . We have $\mathbf{c}'_1(0) = c_r(w_1)$ and $\mathbf{c}'_i(i) = \sum_{j=1}^i c_l(w_j)$ ($1 \leq i \leq p$). For $1 < i \leq p$ and $0 \leq j < i$, we get

$$\mathbf{c}'_i(j) = \min(\mathbf{c}'_{i-1}(j) + c_r(w_i), \mathbf{c}'_{i-1}(j-1) + c_l(w_i)).$$

To obtain \mathbf{c}_0 , we need to observe that the PC problem does not distinguish left and right, but only demands any bipartition of the inserted edges. Therefore, we

obtain \mathbf{c}_0 from \mathbf{c}'_p by

$$\mathbf{c}_0(j) = \min(\mathbf{c}'_p(j), \mathbf{c}'_p(p-j))$$

for $0 \leq j \leq \lfloor \frac{p}{2} \rfloor$. This solves the first step, i.e., the PC problem for μ and W_0 .

Ad 2. In the second step, we first convert each cost vector \mathbf{c}_k ($1 \leq k \leq \ell$) to a cost vector $\tilde{\mathbf{c}}_k$ that gives the costs for connecting the vertices in W_k in the pertinent graph of μ . Similar to the previous step, we calculate the number of crossings required when inserting an edge from f_l or f_r to the skeleton edge e_k (again imagine that e_k is subdivided with a vertex acting as endpoint of the inserted edges). In contrast to the previous case, we need to distinguish if we leave e_k to the left or the right face; therefore we need to compute four costs $c_{ll}(e_k), c_{lr}(e_k), c_{rl}(e_k), c_{rr}(e_k)$ for each e_k (the first index refers to the face f_l or f_r , and the second if we leave e_k to the left or to the right). We obtain $\tilde{\mathbf{c}}_k$ by taking the minimum over all edge cardinalities that can arise by partitioning the insertion paths based on the side at which they leave e_k , while exactly j edges go to f_l :

$$\tilde{\mathbf{c}}_k(j) = \min_{\substack{j_{ll}+j_{lr}+j_{rr}+j_{rl}=|W_k| \\ j_{ll}+j_{lr}=j}} \sigma_{j_{ll},j_{lr},j_{rr},j_{rl}}$$

with

$$\begin{aligned} \sigma_{j_{ll},j_{lr},j_{rr},j_{rl}} &= \mathbf{c}_k(j_{ll} + j_{lr}) + \\ &\quad + c_{ll}(e_k) \cdot j_{ll} + c_{rl}(e_k) \cdot j_{rl} + \\ &\quad + c_{rr}(e_k) \cdot j_{rr} + c_{lr}(e_k) \cdot j_{lr}. \end{aligned}$$

It remains to show how to obtain the final cost vector \mathbf{c} from $\mathbf{c}_0, \tilde{\mathbf{c}}_1, \dots, \tilde{\mathbf{c}}_\ell$. We again apply dynamic programming, computing the cost vectors $\bar{\mathbf{c}}_i$ for the PC problems for μ and $W_0 \cup W_1 \cup \dots \cup W_i$. We have $\bar{\mathbf{c}}_0 = \mathbf{c}_0$ and get

$$\bar{\mathbf{c}}_i(j) = \min_{\substack{0 \leq p \leq |W_0 \cup \dots \cup W_{i-1}| \\ 0 \leq q \leq |W_i| \\ p+q=j}} (\bar{\mathbf{c}}_{i-1}(p) + \tilde{\mathbf{c}}_i(q))$$

for $1 \leq i \leq \ell$ and $j \leq |W_0 \cup \dots \cup W_i|$.

μ is a P-node. In this case, the skeleton graph G_μ consists of two vertices u_1 and u_2 and $m+1$ parallel edges e_0, \dots, e_ℓ , where e_0 denotes the reference edge. Since any permutation of e_1, \dots, e_ℓ constitutes a different embedding of G_μ , we cannot afford to enumerate all embeddings of G_μ . However, we can also establish a dynamic programming approach to handle this type of skeletons.

Let $W_k = W' \setminus \{u_1, u_2\}$ be the affected vertices in the expansion graph of e_k and $E_i = \{e_1, \dots, e_i\}$ for $1 \leq$

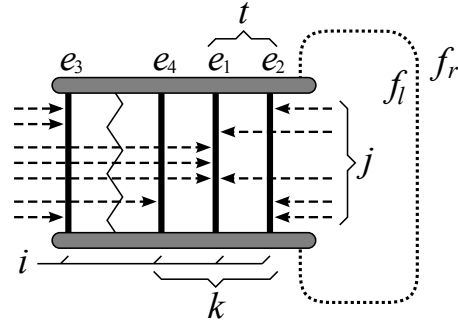


Figure 4: Dynamic programming variables for the PC problem at a P-node. Its two nodes are drawn as gray rounded rectangles. Suppose we already computed \mathbf{c}_4^t for all t , i.e., we solved the problem for the subproblem that the P-node contains only the virtual edges e_1, \dots, e_4 , each with thickness $\theta_1, \dots, \theta_4$, respectively. Consider the problem of adding the edge e_5 . Then the entry $N_i^t(j, k)$ ($N_4^{\theta_1+\theta_2}(5, \theta_4 + \theta_1 + \theta_2)$) gives the number of crossings required to draw the P-node with edges e_1, \dots, e_i , if (a) e_i was inserted such that it has thickness t to its right, (b) j insertion paths go to f_l , and (c) we can add the edge e_{i+1} (zigzag-line) such that we have thickness k to its right.

$i \leq \ell$. Suppose we have already solved the PC problem for each pertinent node of e_k with $W_k \neq \emptyset$, giving us a cost vector \mathbf{c}_k . Consider an edge e_i of G_μ with $1 \leq i \leq \ell$. The traversing costs of e_i reflect the number of crossings required to cross through the expansion graph of e_i . In the following, we call these costs the *thickness* of e_i , denoted by $\theta(e_i)$, and we further define the thickness of E_i as the sum of the thicknesses of all the edges in E_i , i.e., $\theta(E_i) = \sum_{1 \leq k \leq i} \theta(e_k)$. Hence, $\theta(E_i)$ is the number of crossings required to cross through all the edges in E_i .

As subproblems, we solve the PC-problem \mathcal{P}_i^t for the subgraph G_μ^i of G_μ consisting only of the edges e_0, \dots, e_i with the additional requirement that e_i is inserted such that the thickness of the edges to the right of e_i is t , giving us cost vectors \mathbf{c}_i^t . Hence, we compute (at most) $m \cdot T$ cost vectors, where $T = \theta(E_{\ell-1})$. We need the various possible thicknesses as additional parameter, since the crossings required to route the inserted edges from e_i to the left and right only depends on the thicknesses of the edges to the left and right of e_i , respectively, thus being independent on their actual permutation. This is one key observation to solve the P-node case in polynomial time. The cost vector \mathbf{c} for μ and W is then obtained from all the \mathbf{c}_ℓ^t ($0 \leq t \leq T$)

by choosing the best t in each component:

$$\mathbf{c}(j) = \min_{0 \leq t \leq T} \mathbf{c}_\ell^t(j)$$

However, we need additional information that allows us to compute the number of crossings that occur when inserting e_{i+1} at a certain position, caused by edges crossing through e_{i+1} . Consider a possible solution, i.e., a permutation of the edges e_1, \dots, e_i . Any position between two neighboring edges in this permutation is a possible position for inserting the next edge e_{i+1} . For every subproblem \mathcal{P}_i^t with j edges leaving to the left, and each possible position with thickness k to the right, we compute in $N_i^t(j, k)$ the number of edges running through such a position. Observe that the insertion paths will never cross, since—in the final drawing—they are all connected to the same vertex. Figure 4 illustrates our notation with an example.

We now show how to compute the \mathbf{c}_i^t and N_i^t . W.l.o.g, we can assume that $W_1 \neq \emptyset$. Then, the initial values are $\mathbf{c}_1^0 = \mathbf{c}_1$ and

$$N_1^0(j, k) = \begin{cases} j & \text{if } k = 0 \\ \infty & \text{if } 0 < k < \theta(e_1) \\ |W_1| - j & \text{if } k = \theta(e_1) \end{cases}$$

For $i \geq 1$, we compute \mathbf{c}_i^t from \mathbf{c}_{i-1}^t and N_{i-1}^t as follows. If $W_i = \emptyset$, we compute:

$$\mathbf{c}_i^t(j) = \min_{0 \leq t' \leq \theta(E_{i-1})} (\mathbf{c}_{i-1}^{t'}(j) + N_{i-1}^{t'}(j, t))$$

$$N_i^t(j, k) = \begin{cases} N_{i-1}^{t'}(j, k) & \text{if } 0 \leq k \leq t \\ \infty & \text{if } t < k < t + \theta(e_i) \\ N_{i-1}^{t'}(j, k - \theta(e_i)) & \text{if } t + \theta(e_i) \leq k \\ & \wedge k \leq \theta(E_i) \end{cases}$$

In the equation for $\mathbf{c}_i^t(j)$, the t' refers to the index where the minimum in the equation above for $\mathbf{c}_{i-1}^{t'}$ occurred. If $W_i \neq \emptyset$ we have:

$$\mathbf{c}_i^t(j) = \min_{\substack{0 \leq t' \leq \theta(E_{i-1}) \\ 0 \leq j' \leq |W_1 \cup \dots \cup W_{i-1}| \\ 0 \leq j - j' \leq |W_i|}} \chi(t', j')$$

with

$$\chi(t', j') = \mathbf{c}_{i-1}^{t'}(j') + \mathbf{c}_i(j - j') + N_{i-1}^{t'}(j', t) \cdot \theta(e_i) + t(j - j') + (\theta(E_{i-1}) - t)(|W_m| - j + j'),$$

and

$$N_i^t(j, k) = \begin{cases} N_{i-1}^{t'}(j', k) + j - j' & \text{if } 0 \leq k \leq t \\ \infty & \text{if } t < k < t + \theta(e_i) \\ N_{i-1}^{t'}(j', k - \theta(e_i)) + |W_i| - j + j' & \text{if } t + \theta(e_i) \leq k \\ & \wedge k \leq \theta(E_i) \end{cases}$$

where t' and j' refer to the indices where the minimum in the equation above for \mathbf{c}_i^t occurred.

3.2 Combining the Partitioning Cost Subproblems.

Having a polynomial solution for the PC problem, we can discuss how to combine all these results to solve SIV, as sketched in the beginning of Sect. 3. Assume the skeleton G_ρ of \mathcal{T} 's root node ρ has a unique embedding Π_ρ , and fix any face f in Π_ρ . We can compute the number of required crossings when inserting the new vertex into f as follows:

1. First, we compute all solutions $\mathbf{c}_1, \dots, \mathbf{c}_\ell$ for the PC problems of the virtual edges e_1, \dots, e_ℓ in G_ρ , whose expansions contain affected vertices $W_1, \dots, W_\ell \subseteq W$.
2. In the dual graph of Π_ρ , we compute for each virtual edge e_k ($1 \leq k \leq \ell$) the insertion costs c_k^l and c_k^r from f to the face left and right of e_k , respectively.
3. We compute the number of required crossings r_k for connecting the new vertex to W_k as

$$r_k = \min_{0 \leq j \leq \lfloor \frac{|W_k|}{2} \rfloor} \mathbf{c}_k(j) + j \cdot \max(c_k^l, c_k^r) + (|W_k| - j) \cdot \min(c_k^l, c_k^r),$$

i.e., we identify the best combination of partitioning the inserted edges to leave the virtual node on the left or right hand side, taking the final routing within G_ρ , as well as the possibility of mirroring the expansion of e_k , into account.

4. Finally, we solve the SIF $_\rho$ problem for the remaining nodes $W_0 \subseteq W$ directly contained in G_ρ , obtaining r_0 crossings. The final crossing number is then $\sum_{i=0}^\ell r_i$.

Hence we can solve the insertion problem for any fixed insertion face in polynomial time, and it remains to enumerate all possible insertion faces. Although there generally is an exponential number of embeddings and faces, there are only polynomially many faces in the skeletons of \mathcal{T} . In particular, even though the skeleton of a P-node allows an exponential number of embeddings, it has only a quadratic number of different faces. Recall that a Q-node—consisting of an original edge and a virtual edge, representing the rest of the graph—has only a single embedding and the solution for inserting the new vertex in any of its two faces will be equivalent. Hence we can solve the SIV problem by enumerating all *interesting* faces I , applying the above algorithm to each of them, and choose the minimal solution as the optimal one. Recalling the properties

of SPQR-trees we have: the interesting faces I clearly are all the faces within any R-node and one face of each Q-node which is adjacent to a P-node. Considering any possible embedding Π of G and any face f , at least one of the faces of I will represent f in some skeleton in \mathcal{T} .

Therefore we obtain a polynomial algorithm to solve SIV. We can deduce the worst case running time by looking at the necessary nested loops for the face enumeration and the construction of the dynamic programming arrays. This gives (cf. [21] for a detailed derivation):

THEOREM 3.1. *For 2-connected graphs, the SIV problem can be solved to optimality in time $\mathcal{O}(\theta^2 \cdot |V|^3 \cdot |W|^2)$, where θ is the maximal thickness over all P-node skeletons of the SPQR-tree \mathcal{T} of G . If \mathcal{T} has no P-nodes, the runtime improves to $\mathcal{O}(|V| \cdot |W|^3 + |V|^2 \cdot |W|) = \mathcal{O}(|V|^2 \cdot |W|^2)$.*

4 SIV in General Graphs

Having a polynomial solution strategy for two-connected graphs, we can now describe how to extend this result to general graphs, which may contain several, only 1-connected components. Assume we can solve the problem for connected graphs, then the generalization to graphs with several components is trivial: we can solve the problem for each component independently; the faces in which to insert the new vertex then become the components' outer faces. Placing all these solutions next to each other, the components do not cross each other and the validity and optimality of this strategy is obvious.

Hence we concentrate on the case where we are given a connected graph G , which can be decomposed into several blocks B_1, \dots, B_b . We consider the BC-tree \mathcal{B} of G , i.e., the tree structure consisting of the graph's blocks and its cut vertices. Reusing the rooting concept used for the two-connected case, we will iteratively choose each block as the tree's root and construct a solution by (a) inserting the new node in a face of an embedding of this block, and (b) pasting the other blocks into the embedding of the root block. Thereby we will in particular show that we do not require any two blocks to cross and we can therefore compute the insertion problems for the blocks independently.

W.l.o.g., assume B_1 is the root node of \mathcal{B} . Let c_1, \dots, c_k be the cut vertices in B_1 , and H_1, \dots, H_k the subgraphs of G induced by the subtrees rooted at these cut vertices, respectively. After choosing an optimal embedding for B_1 , we will insert each H_i at a single face incident to the common vertex c_i of B_1 and H_i . We solve the problem in two steps:

1. First, we recursively solve the SIV problem for each

H_i , $1 \leq i \leq k$, restricting the enumeration of the insertion faces to the ones incident to c_i .

2. We then solve the problem in the block B_1 . For each vertex $w \in W$ which is not part of B_1 but of some H_i , we add c_i to the list of affected vertices. Note that the affected vertices thereby become a multiset, rather than a regular set. We have to ensure that all edges routed to c_i are routed through the same face directly before reaching c_i . This can be straight-forwardly ensured by using a deterministic algorithm for the shortest path computation, as well as forbidding combinations in the dynamic programming computation that would split these edges apart.

Using the solutions of the above two steps, we can construct a solution for G starting with the solution of B_1 . Let Π be its constructed embedding. We then paste each subgraph H_i into Π as follows, cf. Figure 5: let Π_i be the embedding for H_i obtained in step (1). We choose the computed insertion face to be the outer face of Π_i . We paste Π_i into the face of Π that is last traversed by the edges routed towards c_i and unifying the embeddings at their common vertex c_i . For any affected vertex, we can then join the insertion paths computed in B_1 and in H_i without introducing additional crossings.

Conceptually, we perform this operation for each block chosen as the root node. It is clear that we can streamline the algorithm by not solving SIV for each block b times, but only k_i —the number of cut vertices of the block B_i —times. We can compute the blocks independently. Since $\mathcal{O}\left(\sum_{i=1}^b \text{poly}(|\bar{V}_i|, |\bar{W}_i|)\right) = \mathcal{O}(\text{poly}(|V|, |W|))$ —where \bar{V}_i (\bar{W}_i) denotes the number of (affected) nodes in B_i —for any superlinear polynomial $\text{poly}(\cdot, \cdot)$, we obtain:

THEOREM 4.1. *The SIV problem can be solved to optimality in time $\mathcal{O}(\theta^2 \cdot |V|^3 \cdot |W|^2)$, where θ is the maximal thickness over all P-node skeletons in the SPQR-tree of any block of G .*

5 Conclusion

In this paper, we showed that the star or vertex insertion problem over all embeddings can be solved in polynomial time, which answers a previous open question. It is the first class of subgraphs, other than a trivial edge, for which it was hence shown that it can be efficiently and optimally inserted into a planar graph.

Very recently [6], it has been shown that a solution of the vertex insertion problem in fact approximates the crossing number of apex graphs. Hence our described algorithm furthermore constitutes the best known poly-

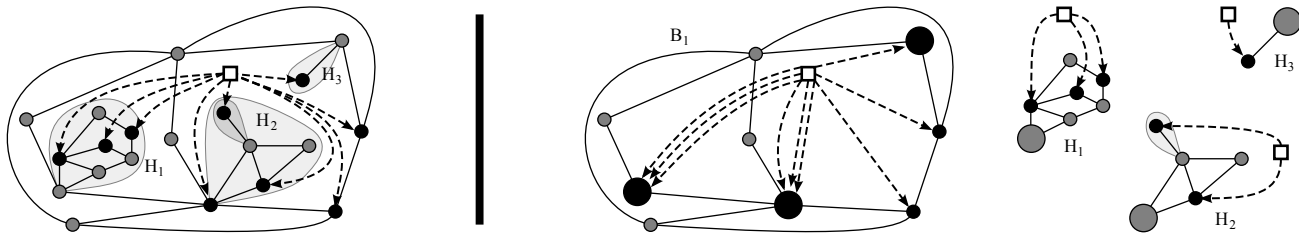


Figure 5: Inserting a vertex (square) into a non-2-connected graph (left), cf. text: we solve the problem for the root block (B_1), and for the subgraphs H_1, \dots, H_3 (right). The latter is done recursively as, e.g., H_2 consists of 2 blocks. The vertices of W are black; large circles in B_1 denote cut vertices that are contained in W multiple times; a large circle in H_i denotes the cut vertex c_i at which they are attached to B_1 : the vertex inserted into H_i has to be placed in a face adjacent to c_i .

nomial approximation algorithm for the crossing number of such graphs, even guaranteeing a constant factor approximation in case of bounded degree.

We conclude with an open question: Are there other (planar) subgraphs that can be optimally inserted into a planar graph in polynomial time?

References

- [1] D. Bokal, G. Fijavz, and B. Mohar. *The minor crossing number*. SIAM J. Discrete Math., 20:344–356, 2006.
- [2] 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.
- [3] M. Chimani and C. Gutwenger. *Algorithms for the hypergraph and the minor crossing number problems*. In Proc. ISAAC '07, volume 4835 of LNCS, pages 184–195. Springer, 2007.
- [4] S. Cabello and B. Mohar. *Crossing and Weighted Crossing Number of Near-Planar Graphs*. In Proc. Graph Drawing '08, to appear in LNCS, Springer.
- [5] M. Chimani, C. Gutwenger, and P. Mutzel. *Experiments on exact crossing minimization using column generation*. In Proc. WEA '06, volume 4007 of LNCS, pages 303–315. Springer, 2006.
- [6] M. Chimani, P. Hliněný, and P. Mutzel. *Vertex Insertion approximates the Crossing Number for Apex Graphs*. submitted. A preliminary version titled *Approximating the Crossing Number of Apex Graphs* appeared as a poster at Graph Drawing '08.
- [7] M. Chimani, P. Mutzel, and I. Bomze. *A new approach to exact crossing minimization*. In Proc. ESA '08, volume 5193 of LNCS, pages 284–296. Springer, 2008.
- [8] G. Di Battista and R. Tamassia. *On-line planarity testing*. SIAM Journal on Computing, 25:956–997, 1996.
- [9] K. Edwards and G. Farr. *An algorithm for finding large induced planar subgraphs*. In Proc. Graph Drawing '01, volume 2265 of LNCS, pages 75–83. Springer, 2002.
- [10] G. Even, S. Guha, and B. Schieber. *Improved approximations of crossings in graph drawing*. In Proc. STOC '00, pages 296–305, 2000.
- [11] M. R. Garey and D. S. Johnson. *Crossing number is NP-complete*. SIAM Journal on Algebraic and Discrete Methods, 4(3):312–316, 1983.
- [12] I. Gitler, P. Hliněný, J. Leanos, and G. Salazar. *The crossing number of a projective graph is quadratic in the face-width*. Electronic Notes in Discrete Mathematics, 29:219–223, 2007.
- [13] M. Grohe. *Computing crossing numbers in quadratic time*. In Proc. STOC '01, pages 231–236, 2001.
- [14] C. Gutwenger and P. Mutzel. *A linear time implementation of SPQR-trees*. In Proc. Graph Drawing '00, volume 1984 of LNCS, pages 77–90. Springer, 2001.
- [15] C. Gutwenger, P. Mutzel, and R. Weiskircher. *Inserting an edge into a planar graph*. Algorithmica, 41:289–308, 2005.
- [16] P. Hliněný. *Crossing number is hard for cubic graphs*. J. Combin. Theory Ser. B, 96:455–471, 2006.
- [17] P. Hliněný and G. Salazar. *On the crossing number of almost planar graphs*. In Proc. Graph Drawing '06, volume 4372 of LNCS, pages 162–173. Springer, 2007.
- [18] J. E. Hopcroft and R. E. Tarjan. *Dividing a graph into triconnected components*. SIAM Journal on Computing, 2(3):135–158, 1973.
- [19] K. Kawarabayashi and B. Reed. *Computing crossing number in linear time*. In Proc. STOC '07, pages 382–380, 2007.
- [20] I. Vrt'o. *Crossing numbers of graphs: A bibliography*. See <ftp://ftp.ifi.savba.sk/pub/imrich/crobib.pdf>, 2008.
- [21] C. Wolf. *Inserting a vertex into a planar graph*. Master's thesis, Dortmund University of Technology, 2008. See http://ls11-www.cs.uni-dortmund.de/people/gutweng/diploma_thesis_wolf.pdf.