Maximizing Ink in Partial Edge Drawings of k-plane Graphs

Matthias Hummel, Fabian Klute, Soeren Nickel, and Martin Nöllenburg(✉)

Algorithms and Complexity Group, TU Wien, Vienna, Austria
matthiashummel@ymail.com, {fklute,noellenburg}@ac.tuwien.ac.at, soeren.nickel@tuwien.ac.at

Abstract. Partial edge drawing (PED) is a drawing style for non-planar graphs, in which edges are drawn only partially as pairs of opposing stubs on the respective end-vertices. In a PED, by erasing the central parts of edges, all edge crossings and the resulting visual clutter are hidden in the undrawn parts of the edges. In symmetric partial edge drawings (SPEDs), the two stubs of each edge are required to have the same length. It is known that maximizing the ink (or the total stub length) when transforming a straight-line graph drawing with crossings into a SPED is tractable for 2-plane input drawings, but NP-hard for unrestricted inputs. We show that the problem remains NP-hard even for 3-plane input drawings and establish NP-hardness of ink maximization for PEDs of 4-plane graphs. Yet, for k-plane input drawings whose edge intersection graph forms a collection of trees or, more generally, whose intersection graph has bounded treewidth, we present efficient algorithms for computing maximum-ink PEDs and SPEDs. We implemented the treewidth-based algorithms and show a brief experimental evaluation.

1 Introduction

Visualizing non-planar graphs as node-link diagrams is challenging due to the visual clutter caused by edge crossings. The layout readability deteriorates as the edge density and thus the number of crossings increases [19]. Therefore alternative layout styles are necessary for non-planar graphs. A radical approach first used in applied network visualization work by Becker et al. [2] is to start with a traditional straight-line graph drawing and simply drop a large central part of each edge and with it many of the edge crossings. This idea relies on the closure and continuation principles in Gestalt psychology [17], which imply that humans can still see a full line segment based only on the remaining edge stubs by filling in the missing information. User studies have confirmed that such drawings remain readable while reducing clutter significantly [9,12] and Burch et al. [11] presented an interactive graph visualization tool using partially drawn edges combined with fully drawn edges.

The authors thank Michael Höller and Birgit Schreiber for the fruitful discussions during the "Seminar in Algorithms: Graphs and Geometry" held in 2017 at TU Wien. A preliminary abstract of this paper has been presented at EuroCG 2018.

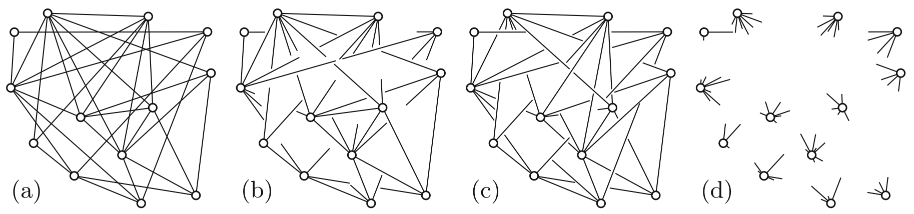


Fig. 1. Drawings of the same graph. (a) A straight-line drawing, (b) a maximum-ink SPED, (c) a maximum-ink PED, and (d) a maximum-ink SHPED.

The idea of drawing edges only partially has subsequently been formalized in graph drawing as follows [7]. A *partial edge drawing (PED)* is a graph drawing that maps vertices to points and edges to pairs of crossing-free edge stubs of positive length pointing towards each other. These edge stubs are obtained by erasing one contiguous central piece of the straight-line segment connecting the two endpoints of each edge. In other words each straight-line edge is divided into three parts, of which only the two outer ones are drawn (see Fig. 1). More restricted and better readable [4] variations of PEDs are *symmetric* PEDs, in which both stubs of an edge must have the same length (see Fig. 1(b)), and *homogeneous* PEDs, in which the ratio of the stub length to the total edge length is the same constant for all edges. Symmetric stubs facilitate finding adjacent vertices due to the identical stub lengths at both vertices, and symmetric homogeneous stubs (see Fig. 1(d)) additionally indicate the distance at which to find a neighboring vertex. Clearly, for very short stubs it is easy to hide all edge crossings, but reading adjacency information gets very difficult [12]. Therefore, the natural optimization problem in this formal setting is *ink maximization*, i.e., maximizing the total stub length, so that as much information as possible is given in the drawing while all crossings disappear in the negative background space.

We study the ink maximization problem for partial edge drawings (PEDs) and symmetric partial edge drawings (SPEDs) with a given geometric input drawing. These problems are known as MaxPED and MaxSPED, respectively [6,7]. Note that with a given input drawing, the ink maximization problem for symmetric homogeneous PEDs (SHPEDs) is trivial, as we can simply iterate over all crossings, choose the larger of the two stub ratios resolving the crossing and take the minimum of all these stub ratios, which yields the best solution.

Related Work. As a first result, Bruckdorfer and Kaufmann [7] presented an integer linear program for solving MaxSPED on general input drawings. Later, Bruckdorfer et al. [6] gave an $O(n \log n)$-time algorithm for MaxSPED on the class of 2-plane input drawings (no edge has more than two crossings), where n is the number of vertices. They also described an efficient 2-approximation algorithm for the dual problem of minimizing the amount of erased ink for arbitrary input drawings. The PhD thesis of Bruckdorfer [5] presents a sketch of an NP-hardness proof for MaxSPED, but left the complexity of MaxPED as an open problem, as well as the design of algorithms for MaxPED.

There are a number of additional results for PEDs without a given input drawing, i.e., having the additional freedom of placing the vertices in the plane. For example, the existence or non-existence of SHPEDs with a specified stub ratio δ for certain graph classes such as complete graphs, complete bipartite graphs, or graphs of bounded bandwidth has been investigated [6,7]. From a practical perspective, Bruckdorfer et al. [8] presented a force-directed layout algorithm to compute SHPEDs for stubs of 1/4 of the total edge length, but without a guarantee that all crossings are eliminated. Moreover, the idea of partial edge drawings has also been extended to orthogonal graph layouts [10].

Contribution. We extend the results of Bruckdorfer et al. [6] on 2-plane geometric graph drawings to k-plane graph drawings for $k > 2$, where a given graph drawing is *k-plane*, if every edge has at most k crossings. In particular, we strengthen the NP-hardness of MaxSPED [5] to the case of 3-plane input drawings without three (or more) mutually crossing edges. For MaxPED we show NP-hardness, even for 4-plane input drawings, which settles a conjecture of Bruckdorfer [5]. On the positive side, we give polynomial-time dynamic programming algorithms for both MaxSPED and MaxPED of k-plane graph drawings whose edge intersection graphs are collections of trees. More generally, we extend the algorithmic idea and obtain FPT algorithms if the edge intersection graph has bounded treewidth and also provide a proof-of-concept implementation. We evaluate the implementation using non-planar drawings from two classical layout algorithms, namely a force-based and a circular layout algorithm.

2 Preliminaries

Let G be a *simple graph* with edge set $E(G) = S = \{s_1, \dots, s_m\}$ and $\varGamma$ a straight-line drawing of G in the plane. We call $\varGamma$ *k-plane* if every edge $s_i \in S$ is crossed by at most k other edges from S in $\varGamma$. We often use edge in S and segment in $\varGamma$ interchangeably. Hence S can be seen as a set of line segments.

The *intersection graph* C of $\varGamma$ is the graph containing a vertex v_i in $V(C)$ for every $s_i \in S$ and an edge $v_i v_j \in E(C)$ between vertices $v_i, v_j \in V(C)$ if the corresponding edges $s_i, s_j \in S$ intersect in $\varGamma$. We also denote the segment in S corresponding to a vertex $v \in V(C)$ by $s(v)$. Observe that the intersection graph C of a k-plane drawing $\varGamma$ has maximum degree k. Using a standard sweep-line algorithm, computing the intersection graph C of a set of m line segments takes $O(m \log m + |E(C)|)$ time [3], where $|E(C)|$ is the number of intersections.

A *partial edge drawing* (PED) D of $\varGamma$ draws a fraction $0 < f_s \leq 1$ of each edge $s = uv \in S$ by drawing edge stubs of length $f_u|s|$ at u and $f_v|s|$ at v, s.t., $f_u + f_v = f_s$. The *ink* or *ink value* $I(D)$ of a PED D is the total stub length $I(D) = \sum_{s \in S} f_s|s|$. In the problem MaxPED, the task is to find for a given drawing $\varGamma$ a PED D^* such that $I(D^*)$ is maximum over all PEDs. A *symmetric partial edge drawing* (SPED) D of $\varGamma$ is a PED, s.t., $f_u = f_v = f_s/2$ for every edge $s = uv \in S$. Then the MaxSPED problem is defined analogously to MaxPED.

Treewidth. A *tree decomposition* [20] for a graph G is a pair $(T, \mathcal{X})$ with T being a tree and $\mathcal{X}$ a collection of subsets $X_i \subseteq V(G)$. For every edge $uv \in E(G)$ we find $t \in V(T)$ such that $\{u, v\} \subseteq X_t$ and for every vertex $v \in V(G)$ we get $T[\{t \mid v \in X_t\}]$ is a connected and non-empty subtree of T. To differentiate the vertices of G and T we call the vertices of T *nodes* and a set $X_i \in \mathcal{X}$ a *bag*. Now the *width* of a tree decomposition $(T, \mathcal{X})$ is defined as $\max\{|X_t| - 1 \mid t \in V(T)\}$. For a graph G we say it has *treewidth* ω, if the tree decomposition with minimum width has width ω. For a node $t \in T$ we denote with $V_t \subseteq V(G)$ the union of all bags $X_{t'} \in \mathcal{X}$ such that t' is either t or a descendent of t in T.

In our algorithms we are using the well known *nice tree decomposition* [14]. For a graph G a nice tree decomposition $(T, \mathcal{X})$ is a special tree decomposition, where T is a rooted tree and we require that every node in T has at most two children. In case $t \in V(T)$ has two children $t_1, t_2 \in T$, then $X_t = X_{t_1} = X_{t_2}$. Such a node is called *join node*. For a node $t \in T$ with a single child $t_1 \in T$ we find either $|X_t| = |X_{t_1}| + 1$, $X_{t_1} \subset X_t$ or $|X_t| = |X_{t_1}| - 1$, $X_t \subset X_{t_1}$. The former we call *insert node* and the latter *forget node*. A leaf $t \in T$ is called a *leaf node* and its bag contains a single vertex. Finally let $r \in T$ be the root of T, then $X_r = \emptyset$. It is known that a tree decomposition can be transformed into a nice tree decomposition of the same width ω and with $O(\omega|V(G)|)$ tree nodes in time linear in the size of the graph G [14].

3 Complexity

We first investigate the complexity of MaxSPED and MaxPED, and prove both problems to be NP-hard for 3-plane and 4-plane input drawings, respectively.

3.1 Hardness of MaxSPED for $k \geq 3$

We close the gap between the known NP-hardness of MaxSPED [5] for general input drawings and the polynomial-time algorithm for 2-plane drawings [6].

Theorem 1. MaxSPED *is* NP*-hard even for 3-plane graph drawings.*

Proof. We reduce from the NP-hard problem planar 3-Sat [18] using similar ideas as in Bruckdorfer's sketch of the hardness proof for general MaxSPED [5]. Here we specify precisely the maximum ink contributions of all gadgets needed for a satisfying variable assignment. Our variable gadgets are cycles of edge pairs that admit exactly two maximum-ink states. We construct clause gadgets consisting of three pairwise intersecting edges so that all crossings are between two edges only, while Bruckdorfer's gadgets have multiple edges intersecting in the same point. Let ϕ be a planar 3-Sat formula with n variables $\{x_1, \ldots, x_n\}$ and m clauses $\{c_1, \ldots, c_m\}$, each consisting of three literals. We can assume that ϕ comes with a planar drawing of its variable-clause graph H_ϕ, which has a vertex for each variable x_i and a vertex for each clause c_j. Each clause vertex is connected to the three variables appearing in the clause. In the drawing of H_ϕ all variable vertices are placed on a horizontal line and the clause vertices connect to the adjacent variable vertices either from above or from below the horizontal line. In our reduction (see Fig. 2) we mimic the drawing of H_ϕ by creating a 3-plane drawing Γ_ϕ as a set of line segments of two distinct lengths and define a value L such that Γ_ϕ has a SPED with ink at least L if and only if ϕ is satisfiable. The whole construction will be drawn onto a triangular grid of polynomial size.

All segments in the clause or variable gadgets are of length 8. The segments used for the connections are of length 4. We use pairs of intersecting segments, alternatingly colored red and green. The intersection point of each red-green segment pair is at distance 1 from an endpoint. Thus, the maximum amount of ink contributed by such a pair is 10 or 6, respectively (one full segment of length 8 or 4, respectively, and two stubs of length 1 each).

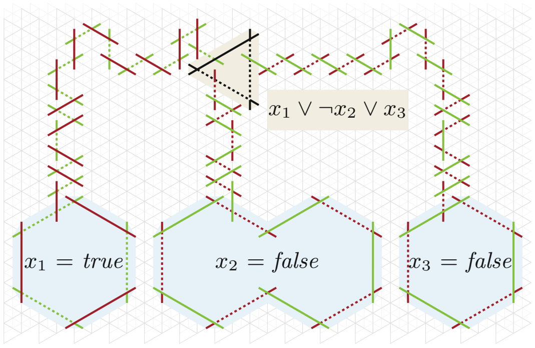


Fig. 2. Three variables and a satisfied clause gadget. Dotted parts do not belong to the SPED. (Color figure online)

Each variable gadget is a cycle of segment pairs, with (at least) one pair for each occurrence of the variable in ϕ, see Fig. 2. This cycle has exactly two ink-maximal SPEDs: either all red edges are full segments and all green edges are length-1 stubs or vice versa. We associate the configuration with green stubs and full red segments with the value *true* and the configuration with red stubs and full green segments with *false*.

For each clause we construct a triple of mutually intersecting segments, see the gadget on yellow background in the upper part of Fig. 2. Again, their intersection points are at distance 1 from the endpoints. It is clear that in such a clause triangle at most one of the three segments can be fully drawn, while the stubs of the other two can have length at most 1. Hence, the maximum amount of ink in a SPED contributed by a clause gadget is 12.

Finally, we connect variable and clause gadgets, such that a clause gadget can contribute its maximum ink value of 12 if and only if the clause is satisfied by the selected truth assignment to the variables. For a positive (negative) literal, we create a path of even length between a green (red) edge of the variable gadget and one of the three edges of the clause gadget as shown in Fig. 2. The first edge s of this path intersects the corresponding variable segment s' such that s' is split into a piece of length 3 and a piece of length 5, whereas s is split into a piece of length 1 and a piece of length 3. The last edge of the path intersects the corresponding clause edge again with a length ratio of 3 to 5. The path consists of a chain of red-green segment pairs, each contributing an ink value of at most 6.

It remains to argue that the resulting drawing has polynomial size and is a correct reduction. All segments are drawn on the underlying triangular grid and have integer lengths; all intersection points are grid points, too. Since the drawing of H_ϕ has polynomial size, so do the constructed gadgets. Additionally, no segment intersects more than three other segments, so the drawing is 3-plane.

For the correctness of the reduction, let L be the ink value obtained by counting 10 for each red-green segment pair in a variable, 6 for each red-green segment pair in a wire, and 12 for each clause gadget. First assume that ϕ has a satisfying truth assignment and put each variable gadget in its corresponding state. For each clause, select exactly one literal with value *true* in the satisfying truth assignment. We draw the clause segment that connects to the selected literal as a full segment and the other two as length-1 stubs. Recall that the literal paths are oriented from the variable gadget to the clause gadget. Since the last segment of the selected literal path must be drawn as two length-1 stubs, the only way of having a maximum contribution of that path is by alternating stubs and full segments. Hence, the first segment of the path must be a full segment. But because the variable is in the state that sets the literal to *true*, the intersecting variable segment is drawn as two stubs and the path configuration is valid. For the two non-selected literals, we can draw the last segments of their paths as full segments, as well as every segment at an even position, while the segments at odd positions are drawn as stubs. This is compatible with any of the two variable configurations and proves that we can indeed achieve ink value L.

Conversely, assume that we have a SPED with ink value L. By construction, every red-green segment pair and every clause gadget must contribute its respective maximum ink value. In particular, each variable gadget is either in state *true* or *false*. By design of the gadgets it is straight-forward to verify that the corresponding truth assignment satisfies ϕ. □

3.2 Hardness of MaxPED for $k \geq 4$

We adapt our NP-hardness proof for MaxSPED to show that MaxPED is NP-hard for k-plane drawings with $k \geq 4$; see [16] for the full proof.

Theorem 2. MaxPED *is NP-hard even for 4-plane graph drawings.*

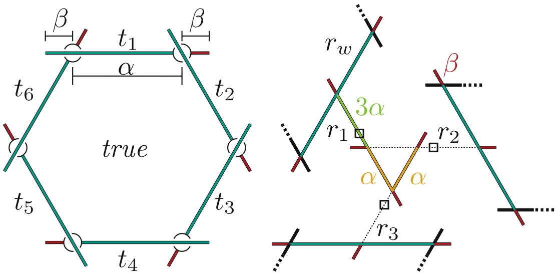
Proof (Sketch). As in the proof of Theorem 1, we show the result via a reduction from planar 3-Sat. The key change for MaxPED comes from the fact, that the two stubs are now independent from each other. Take two crossing edges as an example. We now can draw the two segments with almost full ink value by just excluding an ε-sized gap in one of the two segments for some small $\varepsilon > 0$. We will use this placement of a gap in the variable and wire gadgets, to create two possible states. As before we use an underlying triangular grid, which we omit in the figures of this section for ease of presentation.

Let ϕ be a planar 3-Sat formula. For a variable x of ϕ we construct a variable gadget consisting of a cycle of p line segments $t_1, \ldots, t_p$, see Fig. 3a. Such a cycle has exactly two maximum-ink drawings. One, where for each segment t_i the gap is placed at its intersection with $t_{i+1} \pmod p$ and another drawing, in which the gap is placed at its intersection with $t_{i-1} \pmod p$. Figure 3a shows a gadget in its true state. The length of each segment t_i is $\alpha + 2\beta$, where α is the distance between the two intersection points and β is the length of each stub sticking out.

A clause gadget is a cycle of three pairwise intersecting segments r_1, r_2, r_3, which we call triangle segments. All segments are elongated at one end, such that the total length of a segment r_i, $i \in \{1, 2, 3\}$, is $4\alpha + 2\beta$. Since the stubs are independent we could draw all three triangle segments. To avoid this we attach a big 4-cycle to each r_i. Then r_i intersects the 4-cycle at a segment r_w, see Fig. 3b. If we place the gap of r_w at its intersection with r_i, we lose more units of ink than we gain by drawing every triangle segment r_i completely. Hence it is never possible to draw more than one full triangle segment in an ink-maximal PED.

Finally, a wire is a chain of segments $s_1, \ldots, s_z$ for each variable occurrence in a clause c in ϕ. We place the wire such that s_1 intersects the corresponding variable gadget at some segment t_j, and s_z intersects the clause gadget of c at one if its triangle segments r_i. For the variable we place this intersection point at distance β to its intersection with s_{i+1}, if it occurs positively, or with s_{i-1}, if it occurs negated. At the clause gadget we place the intersection of s_z with the corresponding r_i at distance β from the intersection between r_i and its successor r_{i+1}, see the small squares in Fig. 3b. Each segment s_i has length $\alpha/2 + 2\beta$.

Correctness follows similarly to the proof of Theorem 1. Let Γ_ϕ be the set of line segments constructed as above for a planar 3-SAT formula ϕ. We determine an ink value L, s.t., Γ_ϕ has a PED D with $I(D) \geq L$ if and only if ϕ has a satisfying variable assignment. The key property is that for each clause we find one wire such that its last segment is forced to place its gap at the intersection with the clause gadget in an ink-maximal PED. Then for each other segment s_i, $i \in \{1, \ldots, z-1\}$, we must place its gap at the intersection with s_{i+1}. Otherwise we would have to remove either $\alpha/2$ units of ink in the middle part of some s_i, or remove $\alpha - \beta$ units of ink at the variable gadget intersected by s_1. Both can be avoided if and only if ϕ has a satisfying assignment. □



(a) Variable gadget (b) Clause gadget

Fig. 3. Gadgets of our reduction. Squares mark connection points for wires.

Corollary 1. MAXPED *and* MAXSPED *for k-plane drawings are not fixed-parameter tractable, when parameterized solely by k.*

4 Algorithms

Sections 3.1 and 3.2 showed that MAXSPED and MAXPED are generally NP-hard for $k \geq 3$ and $k \geq 4$ respectively. Now we consider the special case that the intersection graph of the k-plane input drawing is a tree or has bounded treewidth. In both cases we present polynomial-time dynamic programming algorithms for MAXSPED (Sects. 4.1 and 4.2) and MAXPED (Sect. 4.3).

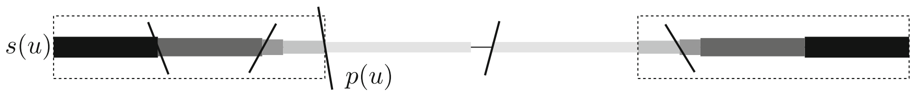


Fig. 4. A segment $s(u)$ with five intersecting segments and the induced stub lengths. The boxed stub lengths are considered in *short*(u) and do not affect $p(u)$.

Let C be the intersection graph of a given drawing Γ of a graph G as defined in Sect. 2. Let $u \in V(C)$ and $\delta_u = \deg(u)$. Then for the corresponding segment $s(u) \in S$ there are $\delta_u + 1$ relevant stub pairs including the whole segment, see Fig. 4. Let $\ell_1(u), \ldots, \ell_{\delta_u}(u) \in \mathbb{R}_+$ be the stub lengths induced by the intersection points of $s(u)$ with the segments of the neighbors of u, sorted from shorter to longer stubs. We define $\ell_0(u)$ as the length of the whole segment $s(u)$.

4.1 Trees

Here we assume that C is a rooted tree of maximum degree k. We give a bottom-up dynamic programming algorithm for solving MaxSPED on C. For each vertex $u \in V(C)$ we compute and store the maximum ink values $W_i(u)$ for $i = 0, \ldots, \delta_u$ for the subtree rooted at u such that $s(u)$ is drawn as a pair of stubs of length $\ell_i(u)$. For $u \in V(C)$ let $p(u)$ denote the parent of u in C and let $c(u)$ denote the set of its children. For $u \in V(C)$ let i_p be the index of the stub length $\ell_{i_p}(u)$ induced by the intersection point of $s(u)$ and $s(p(u))$. We define the following two values, which allow us to categorize the stub lengths into those not affecting the stubs of the parent and those that do affect the parent:

$$\mathit{short}(u) = \max\{W_1(u), \ldots, W_{i_p}(u)\} \quad \mathit{long}(u) = \max\{W_0(u), \ldots, W_{\delta_u}(u)\}.$$

Figure 4 highlights the stub lengths that are considered in *short*(u). We recursively define

$$W_i(u) = \ell_i(u) + \sum_{v \in c(u)} \begin{cases} \mathit{short}(v) & \text{if } s(u) \text{ with length } \ell_i(u) \text{ intersects } s(v) \\ \mathit{long}(v) & \text{otherwise.} \end{cases} \tag{1}$$

The correctness of Recurrence (1) follows by induction. For a leaf u in C the set $c(u)$ is empty and the correctness of $W_i(u)$ is immediate. Further, *short*$(u) = W_1(u)$ and *long*$(u) = W_0(u)$ are set correctly for the parent $p(u)$. For an inner vertex u with degree δ_u we can assume by the induction hypothesis that the values *short*(v) and *long*(v) are computed correctly for all children $v \in c(u)$. Each value $W_i(u)$ for $0 \leq i \leq \delta_u$ is then the stub length $\ell_i(u)$ plus the sum of the maximum ink we can achieve among the children subject to the stubs of u being drawn with length $\ell_i(u)$. Setting *long*(u) and *short*(u) as above yields the two maximum ink values that are relevant for $p(u)$.

Recurrence (1) can be solved naively in $O(mk^2)$ time, where $m = |V(C)|$. Using the order on the stub lengths we can improve this to $O(mk)$ time by computing all $W_i(u)$ for one $u \in V(C)$ in $O(k)$ time. Let $u \in V(C)$ be a vertex with degree δ_u. The values $W_0(u) = \ell_0(u) + \sum_{v \in c(u)} \mathit{short}(v)$ and $W_1(u) = \ell_1(u) + \sum_{v \in c(u)} \mathit{long}(v)$ for the whole segment $s(u)$ and the shortest stubs can be computed in $O(k)$ time each. Now $W_{j+1}(u)$ can be computed from $W_j(u)$ in $O(1)$ time as follows. Let v_j be the neighbor of u that induces stub length $\ell_j(u)$ and assume $v_j \neq p(u)$. In $W_j(u)$ we could still count the value $\mathit{long}(v_j)$, but in $W_{j+1}(u)$ the stub length of u implies that v_j can contribute only to $\mathit{short}(v_j)$. Then $W_{j+1}(u) = W_j(u) - \mathit{long}(v_j) + \mathit{short}(v_j)$. If $v_j = p(u)$ the two values $W_j(u)$ and $W_{j+1}(u)$ are equal, as the corresponding change in stub length has no effect on the children of u. Then computing $\mathit{short}(u)$ and $\mathit{long}(u)$ takes $O(k)$ time.

Using standard backtracking we are able to find an optimal solution to the MaxSPED problem on G with drawing Γ by solving Recurrence (1) in $O(mk)$ time. Furthermore, the intersection graph C with m edges can be computed in $O(m \log m)$ time. We obtain the following theorem.

Theorem 3. *Let G be a simple graph with m edges and Γ a straight-line drawing of G. If the intersection graph C of Γ is a tree with maximum degree $k \in \mathbb{N}$, then* MaxSPED *can be solved in $O(mk + m \log m)$ time and space.*

4.2 Bounded Treewidth

Now we extend the case of a simple tree to the case that the intersection graph C has treewidth at most ω; see [16] for the omitted proofs. Our algorithm and proof of correctness follow a similar approach as the weighted independent set algorithm presented by Cygan et al. [13]. Let $(T, \mathcal{X})$ be a nice tree decomposition of C and k the maximum degree in C. For $V' \subseteq V(C)$ we define the *stub set* $S(V') := \{(u, \ell_i(u)) \mid u \in V' \text{ and } i = 0, \dots, \delta_u\}$. For $(u, \ell_u), (v, \ell_v) \in S(V')$, $u \neq v$, we say (u, ℓ_u) *intersects* (v, ℓ_v) if $s(u)$ drawn with stub length ℓ_u intersects $s(v)$ drawn with length ℓ_v. Further we call $S(V')$ *valid* if $S(V')$ contains exactly one pair (u, ℓ) for each $u \in V'$ and no two pairs in $S(V')$ intersect, i.e., the stub lengths in $S(V')$ imply a SPED in the input drawing Γ. Further we define the ink of a stub set $S(V')$ as $I(S(V')) := \sum_{(u,\ell) \in S(V')} \ell$.

Lemma 1. *Let G be a simple graph, Γ a straight line drawing of G, C the intersection graph of Γ, and $(T, \mathcal{X})$ a tree decomposition of C. For any fixed $S \subseteq S(X_t), t \in T$, any two valid stub sets $S_1, S_2 \subseteq S(V(C))$ with maximum ink and $S_1 \cap S(X_t) = S_2 \cap S(X_t) = S$ have equal ink value $I(S_1 \cap S(V_t)) = I(S_2 \cap S(V_t))$.*

As a consequence of Lemma 1 we get that it suffices to store for every set of vertices V_t and a node $t \in T$ the value of a maximum-ink valid stub set for the choices of vertices in $S(X_t)$. Let $t \in T$ and $S \subseteq S(X_t)$ a stub set, then we define

$$W(t, S) = \max\{I(\hat{S}) \mid \hat{S} \text{ is a valid stub set, } S \subseteq \hat{S} \subseteq S(V_t), \text{ and } \hat{S} \cap S(X_t) = S\}.$$

If no such $\hat{S}$ exists, we set $W(t, S) = -\infty$. In other words, $W(t, S)$ is the maximum ink value achievable by any valid stub set in $S(V_t)$ choosing S as the stub set for the vertices in X_t. If the values $W(t, S)$ are computed correctly for every $t \in T$ we find the ink-value of a maximum-ink SPED by evaluating $W(r, \emptyset)$ with r being the root of T. Applying standard backtracking we can also reconstruct the stubs themselves. We now describe how to compute $W(t, S)$ for every node type $t \in T$ of a nice tree decomposition of C. All the recursion formulas are stated here. We provide the correctness proof for the introduce nodes; see [16] for the forget and join nodes.

Leaf Node. Let $t \in T$ be a leaf node and $v \in X_t$ the vertex contained in its bag, then we store $W(t, \{(v, \ell_i(v))\})$ for each pair $(v, \ell_i(v)) \in S(X_t)$ with $i \in [0, \delta_v]$.

Introduce Node. Suppose next $t \in T$ is an introduce node and t' its only child. Let $v \in X_t$ be the vertex introduced by t, $S \subseteq S(X_t)$, and $i \in [0, \delta_v]$, s.t., $(v, \ell_i(v)) \in S$. If S is not a valid stub set we set $W(t, S) = -\infty$, else

$$W(t, S) = W(t', S \setminus \{(v, \ell_i(v))\}) + \ell_i(v).$$

Correctness follows by considering a valid stub set $\hat{S}$ whose maximum is attained in the definition of $W(t, S)$. Then for the set $\hat{S} \setminus \{(v, \ell_i(v))\}$ it follows that it is considered in the definition of $W(t', S \setminus \{(v, \ell_i(v))\})$ and hence we get

$$W(t', S \setminus \{(v, \ell_i(v))\}) \geq I(\hat{S} \setminus \{(v, \ell_i(v))\}) = I(\hat{S}) - \ell_i(v) = W(t, S) - \ell_i(v)$$
$$W(t, S) \leq W(t', S \setminus \{(v, \ell_i(v))\}) + \ell_i(v).$$

On the other hand consider a valid stub set $\hat{S}'$ for which the maximum is attained in the definition of $W(t', S \setminus \{(v, \ell_i(v))\})$. We need to argue that $\hat{S}' \cup \{(v, \ell_i(v))\}$ is again a valid stub set. First, by assumption that S is a valid stub set, we immediately get that $(v, \ell_i(v))$ does not intersect any $(u, \ell_u) \in S \setminus \{(v, \ell_i(v))\} = \hat{S}' \cap X_{t'}$. Additionally, by the properties of the nice tree decomposition, we get that v has no neighbors in $V_{t'} \setminus X_{t'}$ and with $\hat{S}' \setminus X_{t'} \subseteq V_{t'} \setminus X_{t'}$ we can conclude that $\hat{S}' \cup \{(v, \ell_i(v))\}$ is a valid stub set. Furthermore it is considered in the definition of $W(t, S)$ and we have that

$$W(t, S) \geq I(\hat{S}' \cup \{(v, \ell_i(v))\}) = I(\hat{S}') + \ell_i(v) = W(t', S \setminus \{(v, \ell_i(v))\}) + \ell_i(v).$$

Forget Node. Suppose t is a forget node and t' its only child such that $X_t = X_{t'} \setminus \{v\}$ for some $v \in X_{t'}$. Let S be any subset of $S(X_t)$, if S is not a valid stub set we set $W(t, S) = -\infty$, else $W(t, S) = \max\{W(t', S \cup \{(v, \ell_i(v))\}) \mid i = 0, \ldots, \delta_v\}$.

Join Node. Suppose that t is a join node and t_1, t_2 its two children with $X_t = X_{t_1} = X_{t_2}$. Again let S be any subset of $S(X_t)$. If S is not a valid stub set we set $W(t, S) = -\infty$, else $W(t, S) = W(t_1, S) + W(t_2, S) - I(S)$.

It remains to argue about the running time. Let $m = |V(C)|$. We know there are $O(\omega m)$ many nodes in the tree T of the nice tree decomposition [14].

For each bag $t \in T$ we know it has at most $\omega + 1$ many elements and each element has at most $k + 1$ many possible stubs, hence we have to compute at most $(k + 1)^{\omega+1}$ values $W(t, S)$ per node $t \in T$. At each forget node we additionally need to compute the maximum of up to k entries. Consequently we perform $O((k + 1)^{\omega+2})$ many operations per node $t \in T$. All operations can be implemented in $O(k\omega)$ time. The only problematic one is to test a stub set for validity. We use a modified version of the data structure used in the independent set algorithm by Cygan et al. [13]. See [16] for the implementation details.

Theorem 4. *Let G be a simple graph with m edges and Γ a straight-line drawing of G. If the intersection graph C of Γ has treewidth at most $\omega \in \mathbb{N}$ and maximum degree $k \in \mathbb{N}$,* MaxSPED *can be solved in $O(m(k + 1)^{\omega+2}\omega^2 + m \log m)$ time and space.*

We remark that Theorem 3 shows a better running time in the case of C being a tree, than would follow from Theorem 4 with $\omega = 1$. Furthermore, since Theorem 4 is exponential only in the treewidth ω of C, it implies that MaxSPED is in the class XP[1] when parametrized by ω.

4.3 Algorithms for MaxPED

Let C be the intersection graph in a MaxPED problem. In contrast to MaxSPED we need to consider more combinations of stub lengths since they are not necessarily symmetric anymore. In fact there are $O(k^2)$ possible combinations of left and right stub lengths $\ell_i(u), \ell_j(u)$ for a vertex $u \in V(C)$. For the case of C being a tree our whole argumentation was based solely on the fact that we can subdivide the stub pairs into sets affecting the parent segment or not. This can also be done with the quadratic size sets of all stub pairs and we only get an additional factor of k in the running time.

Corollary 2. *Let G be a simple graph with m edges and Γ a straight-line drawing of G. If the intersection graph C of Γ is a tree with maximum degree $k \in \mathbb{N}$, then* MaxPED *can be solved in $O(mk^2 + m \log m)$ time and space.*

In case of C having bounded treewidth we again did never depend on the symmetry of the stubs, but only on them forming a finite set for each vertex. Consequently we can again just use these quadratic size sets of stub pairs, adding a factor of $k + 1$ compared to MaxSPED, and obtain the following.

Corollary 3. *Let G be a simple graph with m edges and Γ a straight-line drawing of G. If the intersection graph C of Γ has treewidth at most $\omega \in \mathbb{N}$ and maximum degree $k \in \mathbb{N}$,* MaxPED *can be solved in $O(m(k+1)^{\omega+3}\omega^2 + m \log m)$ time and space.*

[1] The class XP contains problems that can be solved in time $O(n^{f(k)})$, where n is the input size, k is a parameter, and f is a computable function.

5 Experiments

We implemented and tested the tree decomposition based algorithms.[2] To compute the nice tree decomposition we used the "htd" library [1] version 1.2, compiled with gcc version 8.3. The algorithm itself was implemented in Python 3.7, using the libraries[3] NetworkX 2.3 and Shapely 1.6. To run the experiments we used a cluster, each node equipped with an Intel Xeon E5-2640 v4 processor clocked at 2.4 GHz, 160 GB of Ram, and operating Ubuntu 16.04. Each run had a memory limit of at most 80 GB of RAM.

We generated random graphs using the NetworkX `gnm` algorithm. The graphs have $n = 40$ vertices and between $m = 40$ and 75 edges in increments of 5. This makes a total of 800 graphs, 100 for each $m \in \{40, 45, \ldots, 75\}$. For the layouts we used the NetworkX implementation of the spring embedder by Fruchterman and Reingold [15] and the graphviz[4] implementation "circo" of a circular layout, version 2.40.1. We could successfully run MaxSPED for all but four of the spring layouts and for all circle layouts with up to 60 edges.

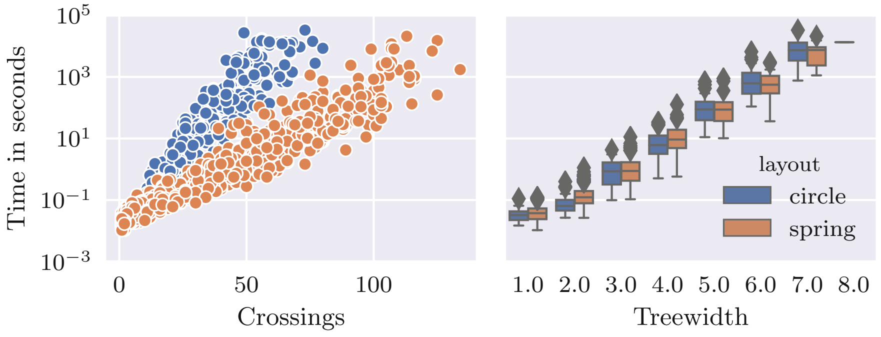


Fig. 5. Experimental results for the MaxSPED algorithm.

Since the time complexity of the algorithm depends exponentially on the treewidth of the intersection graph, we evaluated the running time relative to treewidth and number of crossings, see Fig. 5. The results are as expected, with the runtime quickly increasing to about 16 min (1,000 sec) for 80 crossings in case of the spring and 50 crossings in case of the circle layouts – or for a treewidth of 6 for both layouts. On the other hand we can handle up to 50 crossings and a treewidth of 4 for the spring layouts in about 10 seconds. The discrepancy in the runtime relative to the number of crossings between spring and circle layouts likely comes from different numbers of crossings per edge. To confirm this we took for each intersection graph its maximum degree k divided by the

[2] https://www.ac.tuwien.ac.at/partial-edge-drawing/.
[3] https://networkx.github.io/ and https://github.com/Toblerity/Shapely.
[4] https://www.graphviz.org/.

total number of input crossings. For the spring layouts this resulted in a ratio of 0.24 and for the circle layouts 0.33. Recall that the running time is dominated by $O((k+1)^{\omega+2})$. Hence an increase by a of factor 1.5 in the aforementioned value also results in an additional factor of $1.5^{\omega+2}$ in the asymptotic running time. Concerning ink, for the circle layouts an average of 84% ($\sigma = 0.09$) and for spring layouts an average of 90% ($\sigma = 0.06$) of the ink could be preserved.

For MaxPED we conducted the same experiments. In general one can say that the additional factor of $(k+1)^{\omega}$ makes a big difference. For details see [16].

In summary, our experiment confirmed the predicted running time behavior and showed that the amount of removed ink was moderate. Moreover, the "htd" library [1] performed very well for computing a nice tree decomposition so that we could focus on implementing the dynamic programming algorithm itself.

6 Conclusion

We extended the work by Bruckdorfer et al. [6] and showed NP-hardness for the MaxPED and MaxSPED problems, as well as polynomial-time algorithms for the case of the intersection graph of the input drawing being a tree or having bounded treewidth. For the latter, our proof-of-concept implementation worked reasonably well for small to medium-size instances.

An interesting open problem is to close the gap for MaxPED. While we showed it to be NP-hard for $k \geq 4$ and it is easy to solve for $k \leq 2$ [7], the case of $k = 3$ remains open. Another direction is to consider the existential question for homogeneous (but non-symmetric) PEDs with a fixed ratio δ, for which we can freely distribute the δ fraction of the ink to both stubs. We expect that our algorithms for trees and intersection graphs of bounded treewidth extend to that case, but we could not resolve if the problem is NP-hard or not.

References

1. Abseher, M., Musliu, N., Woltran, S.: htd - a free, open-source framework for (customized) tree decompositions and beyond. In: Salvagnin, D., Lombardi, M. (eds.) CPAIOR 2017. LNCS, vol. 10335, pp. 376–386. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-319-59776-8_30
2. Becker, R.A., Eick, S.G., Wilks, A.R.: Visualizing network data. IEEE Trans. Vis. Comput. Graph. **1**(1), 16–28 (1995). https://doi.org/10.1109/2945.468391
3. de Berg, M., Cheong, O., van Kreveld, M., Overmars, M.: Computational Geometry: Algorithms and Applications, 3rd edn. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-77974-2
4. Binucci, C., Liotta, G., Montecchiani, F., Tappini, A.: Partial edge drawing: homogeneity is more important than crossings and ink. In: Information, Intelligence, Systems Applications (IISA 2016). IEEE (2016). https://doi.org/10.1109/IISA.2016.7785427
5. Bruckdorfer, T.: Schematics of Graphs and Hypergraphs. Ph.D. thesis, Universität Tübingen (2015). http://dx.doi.org/10.15496/publikation-8904

6. Bruckdorfer, T., Cornelsen, S., Gutwenger, C., Kaufmann, M., Montecchiani, F., Nöllenburg, M., Wolff, A.: Progress on partial edge drawings. J. Graph Algorithms Appl. **21**(4), 757–786 (2017). https://doi.org/10.7155/jgaa.00438
7. Bruckdorfer, T., Kaufmann, M.: Mad at edge crossings? Break the edges! In: Kranakis, E., Krizanc, D., Luccio, F. (eds.) FUN 2012. LNCS, vol. 7288, pp. 40–50. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30347-0_7
8. Bruckdorfer, T., Kaufmann, M., Lauer, A.: A practical approach for 1/4-SHPEDs. In: Information, Intelligence, Systems and Applications (IISA 2015). IEEE (2015). https://doi.org/10.1109/IISA.2015.7387994
9. Bruckdorfer, T., Kaufmann, M., Leibßle, S.: PED user study. In: Di Giacomo, E., Lubiw, A. (eds.) GD 2015. LNCS, vol. 9411, pp. 551–553. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-319-27261-0_47
10. Bruckdorfer, T., Kaufmann, M., Montecchiani, F.: 1-bend orthogonal partial edge drawings. J. Graph Algorithms Appl. **18**(1), 111–131 (2014). https://doi.org/10.7155/jgaa.00316
11. Burch, M., Schmauder, H., Panagiotidis, A., Weiskopf, D.: Partial link drawings for nodes, links, and regions of interest. In: Information Visualisation (IV 2014), pp. 53–58 (2014). https://doi.org/10.1109/IV.2014.45
12. Burch, M., Vehlow, C., Konevtsova, N., Weiskopf, D.: Evaluating partially drawn links for directed graph edges. In: van Kreveld, M., Speckmann, B. (eds.) GD 2011). LNCS, vol. 7034, pp. 226–237. Springer, Heidelberg (2012).https://doi.org/10.1007/978-3-642-25878-7_22
13. Cygan, M., Fomin, F.V., Kowalik, Ł., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., Saurabh, S.: Parameterized Algorithms, vol. 3. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-319-21275-3
14. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Heidelberg (2012)
15. Fruchterman, T.M.J., Reingold, E.M.: Graph drawing by force-directed placement. Softw. Pract. Exper. **21**(11), 1129–1164 (1991). https://doi.org/10.1002/spe.4380211102
16. Hummel, M., Klute, F., Nickel, S., Nöllenburg, M.: Maximizing ink inpartial edge drawings of k-plane graphs. CoRR **abs/1908.08905** (2019). http://arxiv.org/abs/1908.08905
17. Koffka, K.: Principles of Gestalt Psychology. Routledge, Abingdon (1935)
18. Lichtenstein, D.: Planar formulae and their uses. SIAM J. Comput. **11**(2), 329–343 (1982). https://doi.org/10.1137/0211025
19. Purchase, H.: Which aesthetic has the greatest effect on human understanding? In: Di Battista, G. (ed.) GD 1997. LNCS, vol. 1353, pp. 248–261. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-63938-1_67
20. Robertson, N., Seymour, P.D.: Graph minors. III. Planar tree-width. J. Comb. Theory Ser. B **36**(1), 49–64 (1984). https://doi.org/10.1016/0095-8956(84)90013-3