Maximizing Ink in Symmetric Partial Edge Drawings of k-plane Graphs

Michael Höller, Fabian Klute, Soeren Nickel, Martin Nöllenburg, and Birgit Schreiber

Algorithms and Complexity Group, TU Wien, Vienna, Austria
[fklute|noellenburg]@ac.tuwien.ac.at, firstname.lastname@student.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. We study symmetric partial edge drawings (SPEDs), in which 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 drawing with crossings into a SPED is tractable for 2-plane input drawings, but generally NP-hard. We show that the problem remains NP-hard even for 3-plane input drawings. Yet, for k-plane input drawings whose edge intersection graph forms a collection of trees or cacti we present efficient algorithms for ink maximization.

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. Therefore alternative layout styles are necessary for non-planar graphs. A radical approach first used in applied network visualization work by Becker et al. [1] 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 theory, which imply that humans can still see a full line segment based only on the remaining edge stubs by filling in the missing information in our brains. User studies have confirmed that such drawings remain readable while reducing clutter significantly [6, 7].

The idea of drawing edges only partially has been formalized in graph drawing as follows [5]. 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 [2] 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 for all edges. Symmetric stubs facilitate finding adjacent vertices due to the identical stub lengths at both vertices, and symmetric homogeneous stubs additionally indicate the distance at which to find a neighboring vertex. The natural optimization problem in this formal setting is *ink*

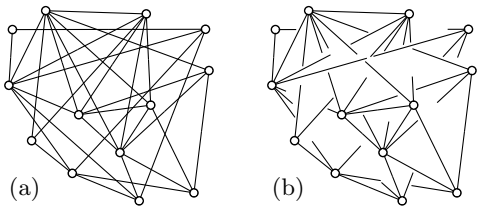

Figure 1 A straight-line graph drawing (a) and a maximum-ink symmetric partial edge drawing (b) of the same graph.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

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 symmetric partial edge drawings (SPEDs) with a given geometric input drawing. This problem is known as MaxSPED. Bruckdorfer and Kaufmann [5] presented an integer linear program for solving MaxSPED. Later, Bruckdorfer et al. [4] 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, and an efficient 2-approximation algorithm for the dual problem of minimizing the amount of erased ink for arbitrary input drawings. Bruckdorfer [3] further gives an NP-hardness proof for MaxSPED.

Contribution. We extend the results of Bruckdorfer et al. [4] on 2-plane geometric graph drawings to k-plane graph drawings for $k > 2$. In particular, we show that MaxSPED is NP-hard even for 3-plane input drawings. However, for k-plane graph drawings whose edge intersection graphs are collections of trees or cacti (which have maximum degree k), we give polynomial-time algorithms for solving MaxSPED.

2 Preliminaries

Throughout the paper let G be a *simple graph* with edge set $S = \{s_1, \ldots, s_m\}$ and Γ a straight-line drawing of G in the plane. We call Γ *k-plane* if every edge $s_i \in S$ is crossed by at most k other edges from S in Γ. We use the terms edge in S and segment in Γ interchangeably. Hence we can also interpret S as a set of line segments.

The *intersection graph* $C = (V, E)$ of Γ is the graph containing a vertex v_i in V for every $s_i \in S$ and an edge $v_iv_j \in E$ between vertices $v_i, v_j \in V$ if the corresponding edges $s_i, s_j \in S$ intersect in Γ. We also denote the segment in S corresponding to a vertex $v \in V$ by $s(v)$. Observe that the intersection graph C of a k-plane drawing Γ has maximum degree k. Using a standard sweep-line algorithm, computing the intersection graph C of a set of m line segments takes $O(m \log m + |E|)$ time [8], where $|E|$ is the number of intersections.

A *symmetric partial edge drawing* (SPED) D of Γ draws a fraction $0 < f_s \leq 1$ of each edge $s = uv \in S$ by drawing two symmetric edge stubs at u and v of length $f_s \cdot |s|/2$ each. No two stubs in D may intersect. The *ink* or *ink value* $I(D)$ of a SPED D is the total stub length $I(D) = \sum_{s \in S} f_s|s|$. In the problem MaxSPED, the task is to find for a given drawing Γ a SPED D^* such that its ink $I(D^*)$ is maximum over all SPEDs.

3 Hardness of MaxSPED for $k \geq 3$

In this section we close the gap between the known hardness of MaxSPED [3] and the polynomial-time algorithm for 2-plane drawings [4] as stated in the following theorem.

▶ **Theorem 3.1.** MaxSPED *is* NP*-hard even for 3-plane graph drawings.*

Proof. We reduce from the NP-hard problem Planar 3-Sat [9] using similar ideas as in Bruckdorfer's sketch of the hardness proof for general MaxSPED [3]. 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. Finally 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

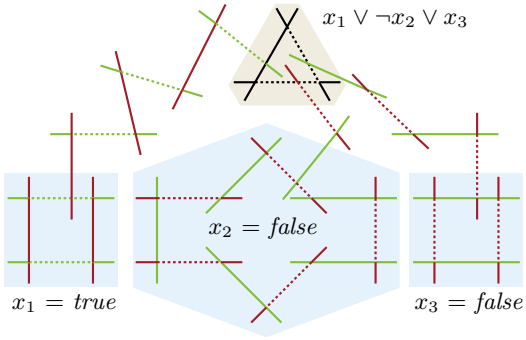

Figure 2 Example of three variable gadgets and a satisfied clause gadget. Dotted parts do not belong to the SPED.

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 uniform length and a value L such that Γ_ϕ has a SPED with ink at least L if and only if ϕ is satisfiable.

All segments in the gadgets are of length 5. 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 7 (one full segment of length 5 and the other one with two stubs of length 1 each).

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. Observe that 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 full green segments and red stubs with the value *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 9.

Finally, we connect variable and clause gadgets in such a way that a clause gadget can contribute its maximum ink value of 9 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 edge s' such that s' is split into a piece of length 2 and a piece of length 3, whereas s is split into a piece of length 1 and a piece of length 4. The last edge of the path intersects the corresponding clause edge with the same length ratios. The path itself consists of a chain of red-green segment pairs, so each pair contributes an ink value of at most 7.

Since the drawing of H_ϕ has polynomial size, the number of segments created in the reduction is polynomial. Further, no segment intersects more than three other segments, so the constructed drawing is 3-plane.

For the correctness of the reduction, let L be the ink value obtained by counting 7 for each red-green segment pair and 9 for each clause gadget. First assume that ϕ has a satisfying

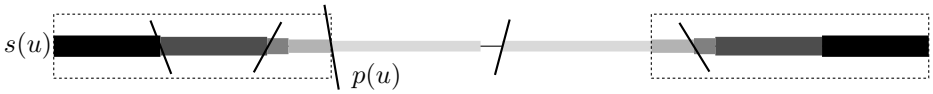

Figure 3 A segment $s(u)$ with five intersecting segments and the different induced stub lengths. The boxed stub lengths are considered in $short(u)$ and do not affect $p(u)$.

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 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. ◀

4 Two polynomial special cases

Section 3 showed that MaxSPED is generally NP-hard for $k \geq 3$. Now we consider the special case that the intersection graph of the k-plane input drawing is a tree or a cactus. In both cases we present polynomial-time dynamic programming algorithms. Let $C = (V, E)$ be the intersection graph of a given drawing Γ of a graph G as defined in Section 2. Let $u \in V$ and $\delta = \deg(u)$. Then for the corresponding segment $s(u) \in S$ there are $\delta + 1$ relevant stub pairs including the whole segment, see Fig. 3. Let $\ell_1(u), \ldots, \ell_\delta(u) \in \mathbb{R}_+$ 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 $C = (V, E)$ is a rooted tree of maximum degree k. We give a bottom up dynamic programming algorithm for solving MaxSPED on C. For each vertex $u \in V$ we compute and store the maximum ink values $T_i(u)$ for $i = 0, \ldots, \delta$ with $\delta = \deg(u)$ for the subtree rooted at u such that $s(u)$ is drawn as a pair of stubs of length $\ell_i(u)$. For $u \in V$ let $p(u)$ denote the parent of u in C and let $c(u)$ denote the set of its children. For $u \in V$ 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:

$$short(u) = \max\{T_1(u), \ldots, T_{i_p}(u)\}$$
$$long(u) = \max\{T_0(u), \ldots, T_\delta(u)\}.$$

Figure 3 highlights the stub lengths that are considered in $short(u)$. We recursively define

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

The correctness of Recurrence (1) follows by induction. For a leaf u in C the set $c(u)$ is empty and the correctness of $T_i(u)$ is immediate. Further, $short(u) = T_1(u)$ and $long(u) = T_0(u)$ are set correctly for the parent $p(u)$. For an inner vertex u with degree δ 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 $T_i(u)$ for $0 \leq i \leq \delta$ 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 $m = |V|$. Using the order on the stub lengths we can improve this to $O(mk)$ time by computing all $T_i(u)$ for one $u \in V$ in $O(k)$ time. Let $u \in V$ be a vertex with degree $\deg(u) = \delta$. The values $T_0(u) = \ell_0(u) + \sum_{v \in c(u)} short(v)$ and $T_1(u) = \ell_1(u) + \sum_{v \in c(u)} long(v)$ for the whole segment $s(u)$ and the shortest stubs can be computed in $O(k)$ time each. Now $T_{j+1}(u)$ can be computed from $T_j(u)$ 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 $T_j(u)$ we could still count the value $long(v_j)$, but in $T_{j+1}(u)$ the stub length of u implies that v_j can contribute only to $short(v_j)$. Then $T_{j+1}(u) = T_j(u) - long(v_j) + short(v_j)$. If $v_j = p(u)$, then the two values $T_j(u)$ and $T_{j+1}(u)$ are equal as the corresponding change in stub length has no effect on the children of u. Computing $short(u)$ and $long(u)$ clearly takes $O(k)$ time.

So by solving Recurrence (1) in $O(mk)$ time we find an optimal solution to the MaxSPED problem on G with drawing Γ using standard backtracking from $\max\{T_0(r), \ldots, T_{\deg(r)}(r)\}$ for the root r of C. Since the intersection graph C is a tree with $O(m)$ edges it can be computed in $O(m \log m)$ time and we obtain the following theorem.

▶ **Theorem 4.1.** *Let G be a simple graph with m edges and Γ a straight-line drawing of G. If the intersection graph $C = (V, E)$ of G is a tree with maximum degree $k \in \mathbb{N}$, then problem* MaxSPED *can be solved in $O(mk + m \log m)$ time and space.*

4.2 Cactus graphs

We now generalize our previous algorithm to cactus graphs. Due to space constraints we provide only a sketch of the arguments and omit further details. Let the intersection graph $C = (V, E)$ be a *cactus graph*, i.e. a simple graph in which no edge $e \in E$ is part of more than one cycle. For a cactus graph C the *block-cut tree* is an arbitrarily rooted tree $\mathcal{T} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ contains a node for every cycle and for every cut vertex in C. We call a node $u \in \mathcal{V}$ a *block node* if it represents an induced cycle on vertices $V(u) \subseteq V$ and an *articulation node* of the cut vertex $a(u) = v \in V$ otherwise. We have an edge $uv \in \mathcal{E}$ between a block node u and an articulation node v whenever $a(v) \in V(u)$ and an edge $uv \in \mathcal{E}$ between two articulation nodes whenever $a(u)a(v) \in E$. We re-use the notation $p(u)$ and $c(u)$ to refer to the parent and to the children of a node $u \in \mathcal{V}$.

We further define for each node $u \in \mathcal{V}$ two indices $i_p(u) \leq i_q(u)$. For an articulation node u, i_p and i_q are the indices corresponding to the stub lengths induced by the intersection points of $s(a(u))$ and its neighbors in C among the vertex set of $p(u)$. For a block node u, i_p and i_q are the indices corresponding to the intersection points of the parent's segment

$s(a(p(u)))$ with its neighbors in $V(u)$. Similar to Section 4.1 we now define three values for each articulation node $u \in \mathcal{V}$ with degree δ:

$$\begin{aligned} \mathit{long}(u) &= \max\{T_0(u), \ldots, T_\delta(u)\} \\ \mathit{mid}(u) &= \max\{T_1(u), \ldots, T_{i_q}(u)\} \\ \mathit{short}(u) &= \max\{T_1(u), \ldots, T_{i_p}(u)\}. \end{aligned}$$

Further, we define the recurrence $T_i(u)$ for all articulation nodes $u \in \mathcal{V}$:

$$T_i(u) = \ell_i(a(u)) + \sum_{v \in c(u)} \begin{cases} T_{i_p(v)}(v) & \text{if } 1 \le i < i_p(v) \\ T_{i_q(v)}(v) & \text{if } i_p(v) < i \le i_q(v) \\ T_0(v) & \text{otherwise.} \end{cases} \tag{2}$$

For a block node $u \in \mathcal{V}$ we cut open the cycle it represents at vertex $a(p(u))$ (refer to Bruckdorfer et al. [4] for details) and consider the three relevant stub lengths of the parent's segment $s(a(p(u)))$ induced by $i_p(u)$ and $i_q(u)$ and the whole segment. The resulting subgraph is a path on $V(u)$ with attached children that are articulation vertices in $\mathcal{T}$, so we can apply our algorithm of Section 4.1 with minor modifications. Once all values in the subgraph of block node u are computed, we can derive the values $T_i(u)$ for the block node u.

Correctness can be proved by modifying the inductive arguments of Section 4.1 accordingly. Combining the linear running time for cycles of Bruckdorfer et al. [4] with the running time for trees of Theorem 4.1 we can argue that cactus graphs can in fact be handled in the same time and space bounds as trees.

▶ **Theorem 4.2.** *Let G be a simple graph with m edges and Γ a straight-line drawing of G. If the intersection graph $C = (V, E)$ of G is a cactus with maximum degree $k \in \mathbb{N}$, problem* MaxSPED *can be solved in $O(mk + m \log m)$ time and space.*

References

1 Richard A. Becker, Stephen G. Eick, and Allan R. Wilks. Visualizing network data. *IEEE Trans. Visualization and Computer Graphics*, 1(1):16–28, 1995.

2 Carla Binucci, Giuseppe Liotta, Fabrizio Montecchiani, and Alessandra Tappini. Partial edge drawing: Homogeneity is more important than crossings and ink. In *Information, Intelligence, Systems Applications (IISA'16)*, pages 1–6. IEEE, 2016.

3 Till Bruckdorfer. *Schematics of Graphs and Hypergraphs.* PhD thesis, Uni Tübingen, 2015.

4 Till Bruckdorfer, Sabine Cornelsen, Carsten Gutwenger, Michael Kaufmann, Fabrizio Montecchiani, Martin Nöllenburg, and Alexander Wolff. Progress on partial edge drawings. *J. Graph Algorithms Appl.*, 21(4):757–786, 2017.

5 Till Bruckdorfer and Michael Kaufmann. Mad at edge crossings? Break the edges! In *Fun with Algorithms (FUN'12)*, volume 7288 of *LNCS*, pages 40–50. Springer, 2012.

6 Till Bruckdorfer, Michael Kaufmann, and Simon Leibßle. PED user study. In *Graph Drawing (GD'15)*, volume 9411 of *LNCS*, pages 551–553. Springer, 2015.

7 Michael Burch, Corinna Vehlow, Natalia Konevtsova, and Daniel Weiskopf. Evaluating partially drawn links for directed graph edges. In *Graph Drawing (GD'11)*, volume 7034 of *LNCS*, pages 226–237. Springer, 2012.

8 Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications.* Springer, 3rd edition edition, 2008.

9 David Lichtenstein. Planar formulae and their uses. *SIAM J Comput*, 11(2):329–343, 1982.