Crossing Minimization and Layouts of Directed Hypergraphs with Port Constraints

Markus Chimani[1,⋆], Carsten Gutwenger[2], Petra Mutzel[2], Miro Spönemann[3], and Hoi-Ming Wong[2,⋆⋆]

[1] Algorithm Engineering Group, Friedrich-Schiller-Universität Jena
markus.chimani@uni-jena.de
[2] Chair of Algorithm Engineering, Technische Universität Dortmund
{carsten.gutwenger,petra.mutzel,hoi-ming.wong}@tu-dortmund.de
[3] Real-Time and Embedded Systems Group, Christian-Albrechts-Universität zu Kiel
msp@informatik.uni-kiel.de

Abstract. Many practical applications for drawing graphs are modeled by directed graphs with domain specific constraints. In this paper, we consider the problem of drawing directed hypergraphs with (and without) port constraints, which cover multiple real-world graph drawing applications like data flow diagrams and electric schematics.

Most existing algorithms for drawing hypergraphs with port constraints are adaptions of the framework originally proposed by Sugiyama et al. in 1981 for simple directed graphs. Recently, a practical approach for upward crossing minimization of directed graphs based on the planarization method was proposed [7]. With respect to the number of arc crossings, it clearly outperforms prior (mostly layering-based) approaches. We show how to adopt this idea for hypergraphs with given port constraints, obtaining an upward-planar representation (UPR) of the input hypergraph where crossings are modeled by dummy nodes.

Furthermore, we present the new problem of computing an orthogonal upward drawing with minimal number of crossings from such an UPR, and show that it can be solved efficiently by providing a simple method.

1 Introduction

The visualization of directed graphs in an upward fashion is a central research field in graph drawing. Thereby we ask for a drawing where all arcs are drawn monotonously increasing in one direction, in order to make it easy to follow the overall direction in the process that is modeled by the graph. There is a large number of publications regarding this topic. Most known approaches follow the layer-based scheme originally proposed by Sugiyama et al. [18]. However, many important applications such as data flow diagrams or electric schematics require

⋆ Markus Chimani was funded by a Carl-Zeiss-Foundation juniorprofessorship.
⋆⋆ Hoi-Ming Wong was supported by the German Research Foundation (DFG), priority project (SPP) 1307 "Algorithm Engineering", subproject "Planarization Practices in Automatic Graph Drawing".

U. Brandes and S. Cornelsen (Eds.): GD 2010, LNCS 6502, pp. 141–152, 2011.

directed hypergraphs rather than traditional directed graphs; moreover they often come with further specific drawing constraints. Layer-based methods that consider these specialties [17] often suffer from too many edge crossings, and thus planarization-based methods—where minimizing the number of crossings is the main objective—may be preferable. In order to keep a consistent arc direction, the technique of *upward-planarization* is needed. Recent results for traditional graphs have shown that, by using this approach, the number of edge crossings can be reduced by 50% compared to layer-based methods [7, 8].

In this paper we consider the problem of drawing directed hypergraphs with port constraints. Basic definitions on hypergraphs are given in Sect. 1.1, and port constraints are introduced in Sect. 1.2. We describe how to adapt an existing upward-planarization method to handle directed hyperarcs as well as prescribed port positions. Our upward-planarization algorithm produces an upward-planar representation of the hypergraph and is covered in Sect. 2. Using this representation, we show how to construct an orthogonal layout that respects the arc directions using a derived layering in Sect. 3. By exploiting the topological information of the planarization phase, we can efficiently minimize the number of crossings with respect to the computed embedding. We conclude with Sect. 4.

1.1 Hypergraphs

A *directed graph* is a pair $G = (V, A)$, where V is a finite set of nodes and A is a set of ordered node pairs called *directed edges* or *arcs*. A directed acyclic graph (DAG) with exactly one source node is called an *sT-graph*. A node u *dominates* a node v in G if there exists a directed path from u to v in G. A *directed hypergraph* is a pair $H = (V, \mathcal{A})$, where V is a finite set of nodes and $\mathcal{A}$ is a set of pairs (S, T) with non-empty sets $S, T \subseteq V$. The elements of $\mathcal{A}$ are called *directed hyperedges* or *hyperarcs*, S are the *source* nodes, and T are the *target* nodes. While our definition conceptually allows $S \cap T \neq \emptyset$ (i.e., a hyperarc may be or contain a self-loop), we will not consider such a case in this paper.

Now let H be a self-loop free directed hypergraph and $\phi = (S, T)$ a hyperarc of H. A directed tree $\mathcal{T} = (V_\phi, A_\phi)$ with $V_\phi = (S \cup T \cup N)$ is an *underlying tree* of ϕ if: (i) for each source node $s \in S$ there is a node $n \in N$ with $(s, n) \in A_\phi$; (ii) for each target node $t \in T$ there is a node $n' \in N$ with $(n', t) \in A_\phi$; (iii) the degree of each $v \in S \cup T$ is exactly 1 within $\mathcal{T}$; and (iv) each $n \in N$ is only adjacent to vertices of V_ϕ and has degree at least 2. We call N the *hypernodes* of ϕ. Informally, the source and target nodes are the leaves and the hypernodes are the inner nodes of $\mathcal{T}$. $\mathcal{T}$ is called *confluent* if each source node dominates all target nodes. A *directed underlying graph* H' of a directed hypergraph H is obtained by substituting each hyperarc ϕ by an underlying tree T_ϕ, i.e., H' consists of the nodes of H together with the hypernodes and arcs of all underlying trees. If each underlying tree contains only one hypernode, we call H' a *star-based* underlying graph of H (cf. *tree-based* and *point-based* drawing style of hyperedges [6]). H' is *confluent* if all underlying trees are confluent.

In this paper we also consider the problem of orthogonal routing of hyperedges. Layer-based approaches for orthogonal routing of plain edges were given

by Sander [15] and Baburin [1]. Eschbach et al. showed that the orthogonal hyperedge routing problem using at most one horizontal line segment per hyperedge is NP-hard by revealing its equivalence to the minimal feedback arc set problem [10], and proposed a greedy assignment heuristic and a sifting heuristic. Sander proposed to use standard cycle breaking heuristics [16], still limited to at most one horizontal line segment for each hyperedge (except for hyperedges that span multiple layers).

Since our approach is based on planarization, we use the basic ideas for handling of hyperedges from previous work on hypergraph planarization [6].

1.2 Port Constraints

The *ports* of a hyperarc $\phi = (S, T)$ are the points in the drawing where ϕ touches the nodes in S and T. In many applications these ports have a specific semantic interpretation, such as being *inputs* or *outputs* for data tokens in data flow diagrams, or *pins* of electric components in circuit schematics. Thus in such applications the positioning of ports is not arbitrary, but may be subject to specific constraints [17]. The strictest variant of port constraints is the one where the exact position of each port, relative to the respective node, is prescribed. This implies that each port has an associated side of the node where it is drawn, i.e., the top, bottom, left, or right side.

Given a node v with incident arcs I, the application's model of port constraints may limit the set of admissible clockwise orders of the arcs I. A generic approach to model this set of orders for each node is by using *embedding constraints*, which define a tree structure of constraint nodes [12]. This structure can be considered by an extended planarization algorithm in order to obtain a planar embedding that respects the pre-defined constraints. The embedding constraints approach is compatible with the method described in this paper, but for the sake of brevity we mostly consider only the strict port positions variant in the following, which admits merely one port order for each node.

First approaches to include ports in layer-based drawings were given by Gansner et al. [11] and Sander [14]. A more advanced adaption considers different types of port constraints as well as hyperarcs [17], as they are required for the layout of data flow diagrams. That method leads to quite acceptable results for different types of hypergraphs with port constraints, but is still limited by the fact that a bad layering can lead to an unnecessarily high number of arc crossings. As an alternative that is based on planarization, Eiglsperger et al. [9] proposed a method to include constraints in the orthogonalization phase for the *topology-shape-metrics* approach.

2 Upward-Planarization

The currently strongest upward-planarization approach for sT-graphs is due to [7]. The key ingredients of this algorithm can be summarized as follows. Let $G = (V, A)$ be the graph to draw. Similar to the traditional undirected

planarization approach [2], we start with an upward-planar subgraph $U = (V, A')$ of G and then iteratively insert the edges $A \setminus A'$ into U with as few crossings as possible. All arising crossings on an inserted edge are replaced by dummy nodes (*crossing dummies*), such that the iteration always considers the problem of inserting a single arc into an otherwise upward-planar graph.

Yet, unlike for the undirected planarization approach, our upward requirement results in two subtle but central difficulties: not every upward-planar subgraph can be used as a starting point, and a crossing minimal insertion of some arc a may leave a graph in which the remaining, not yet inserted arcs cannot be inserted in an upward fashion anymore, see [7] for details.

In the following, we will investigate how this approach can be extended to hypergraphs and port constraints.

2.1 Preprocessing

We can assume that the given directed hypergraph H contains no hyperacrs with self-loops, as they could be easily reinserted as a postprocessing step without requiring any further crossings. Let H' be the star-based directed underlying graph of hypergraph H. As we require an sT-graph, we may have to insert dummy edges from an artificial super source node to the source nodes of H'. These edges will have weight 0 for the purpose of crossing minimization and can be removed for the final layout computation (Sect. 3).

It remains to make H' acyclic by reversing the direction of the arcs in a *minimal* feedback arc set. Although this problem is NP-hard, any heuristic finding a minimal (in contrast to a *minimum*) arc set suffices for our purpose. Also note that, under the assumption that an upward drawing is actually a suitable drawing paradigm for H, this set will typically be small or even empty. In the final drawing, we re-reverse these arcs again. Yet, for the upward-planarization approach, we still have to take special care of such reversed arcs, as we require each hypergraph to be drawn in a confluent way. Let *Rev* be the arcs that were reversed in the preprocessing step.

Our port constraints may require arcs $a = (u, v)$, with u being properly drawn below v, (a) to leave u downwards, or (b) to enter v from above. This clearly invalidates the pure upward drawing style. Nevertheless we want to allow such constructs, but have to ensure that such "misdirected" pieces of the arcs are only drawn close to u or v, respectively (see below). Hence any such arc requiring (a) and (b) is substituted by a *chain* of *subarcs* $(d_a^1, u), (d_a^1, d_a^2), (v, d_a^2)$, where d_a^1, d_a^2 are two new dummy nodes. Arcs only requiring either (a) or (b) are analogously replaced by simpler chains of only two subarcs and a single dummy node. For notational simplicity, we will continue to store the original unmodified arcs in the graph, denoted by the set *Chn*. Whenever we consider an arc $a \in Chn$, we in fact use the complete chain corresponding to a.

So after preprocessing we have a simple sT-graph $G = (V, A)$, with $Rev \subset A$ and $Chn \subset A$. Note that these two subsets may not be disjoint.

2.2 Feasible Upward-Planar Subgraph

Besides ensuring upward-feasibility of the initial upward-planar subgraph (i.e., ensuring that all temporarily removed arcs are re-insertable), our subgraph has to additionally be feasible with respect to the port constraints. As described in Sect. 1.2, embedding constraints can be seen as a specific model of admissible arc orders for planarization. We can handle such constraints by replacing each constrained node by its constraint-tree [12], thus in the following we only consider nodes with no constraints or strict port constraints.

We first compute a spanning tree $U' = (V, A'')$ of G, with $A'' \cap \mathit{Rev} = \emptyset$, which clearly exists due to the minimality of Rev.

We then insert the arcs $a = (x, y) \in A \setminus (A'' \cup \mathit{Rev})$ one by one into U'. After each insertion step we perform an upward-planarity test, obtaining some feasible embedding, and check whether the graph still allows the insertion of all remaining arcs within this embedding, using the merge-graph paradigm introduced in [7]. By the implicit transformation of the arcs Chn we thereby also ensure that the port constraints can be satisfied. If any of the above two tests fails, we remove the arc again, and store it in a set B instead.

After that, we fix the current embedding of the graph, and try to re-insert the arcs in B a second time, as the previously considered embedding might just have been a bad choice. In the end we have a maximal port-feasible upward-planar subgraph $U = (V, A')$ with a feasible embedding Γ, and the set $B = A \setminus A'$ of arcs not yet in U.

Remark. The runtime of the arc-wise test for insertion-feasibility is still dominated by the acyclicity test in the merge-graph, and we can hence obtain U in $\mathcal{O}(|A|^2)$ time.

2.3 Arc Insertion

Again, we will start our investigation by outlining how arc insertion works for traditional arcs without port constraints, cf. [7] for details. We will then use this algorithm as a building block when considering hypergraphs and port constraints. Sometimes we thereby require seemingly minor internal modifications to this algorithm, which in fact require intricate modifications in the proofs of the algorithm's validity. Due to space constraints, we will only touch upon these issues and focus on the overall idea.

Considering a fixed embedding, the non-upward edge insertion problem can be solved by considering the dual graph of the embedding, and finding a shortest path in this dual graph between two faces adjacent to the edge's start and end node. When considering inserting an arc $a = (u, v)$ in an upward fashion, this dual graph has to be substituted by a somewhat similar routing network that ensures that the identified insertion path is monotonously going upward and is non-self-intersecting.

Furthermore, not every such found insertion path is feasible with respect to the remaining, not yet inserted arcs. Therefore, during the shortest path

computation within the routing network, we additionally have to check feasibility of the intermediate merge-graph (simply put, the merge graph is the current graph, augmented with the not-yet inserted arcs in such a way that the remaining arcs are insertable if and only if the merge graph is acyclic).

Hyperarc Insertion. Starting from the subgraph U, we will now iteratively insert full hyperarcs, until we obtain a upward-planarization of G and hence of H. Note that hyperedge insertion in the tree-based paradigm is already NP-hard in the undirected, non-upward setting [6]. We therefore introduce a novel piecewise insertion strategy which realizes a low-crossing number hyperarc insertion, still ensuring confluency of the hyperarc drawing.

For any original hyperarc $\phi \in \mathcal{A}$ in H, G contains a set of arcs $A_\phi \subset A$ and a set of hypernodes V_ϕ. Initially, $|V_\phi| = 1$ since we started with the star-based underlying graph of H. We will see that both sets will grow during the subsequent insertion steps. Also note that, in general, some arcs of A_ϕ may be in A' and some in B.

Let $U^\circ = (V, A^\circ)$ be a port-feasible upward-planar planarization (i.e., crossings are modeled via dummy nodes) of some subgraph of G during the insertion process, B° the not yet inserted arcs, and $\phi \in \mathcal{A}$ the hyperarc to insert. The edge set $A^\circ_\phi := A_\phi \cap A^\circ$ forms a tree, corresponding to the partially tree-based-drawn hyperarc. Our induction hypothesis states that this tree is confluent: In the initial subgraph U, this tree is at most a star (and at least a single edge) and therefore clearly confluent. The arcs $B_\phi := A_\phi \cap B^\circ$ are the arcs corresponding to ϕ that have yet to be inserted in our current iteration step, extending the tree A°_ϕ in a confluent way.

We will insert these arcs one by one. Let $a = (x, y) \in B_\phi$ and assume for now that $a \notin \mathit{Rev} \cup \mathit{Chn}$. We first compute the minimal subtrees T_s, T_t of A°_ϕ that contain all source and target nodes, respectively, that are already connected by A°_ϕ. By the induction hypothesis on confluency, T_s and T_t are disjoint except for at most one hypernode. Let h_s (h_t) be the hypernode of T_s (T_t) closest to T_t (T_s, respectively) on the tree A°_ϕ.

If x is a hypernode, then y is a target vertex of ϕ, and we search for a shortest feasible upward-insertion path from h_s to y. Otherwise, y is a hypernode, x is a source node of ϕ, and we search for a shortest feasible upward-insertion path from x to h_t. Thereby we modify the routing network in three ways:

Port Constraints. If our port constraints require us to leave x or enter y at some special positions with respect to already inserted arcs, we can easily restrict the routing network to use only applicable start or end faces for the routing. Yet, there are additional augmentations necessary, if $a \in \mathit{Chn}$, see below.

Arc Reuse. We want to reuse already drawn (sub)arcs corresponding to ϕ, in order to generate hyperarc drawings with low total number of crossings. Therefore, our routing allows 0-cost crossings over A°_ϕ and next to crossing dummies in A°_ϕ. In other words, the new path may reuse already established paths of ϕ, cf. Fig. 1.

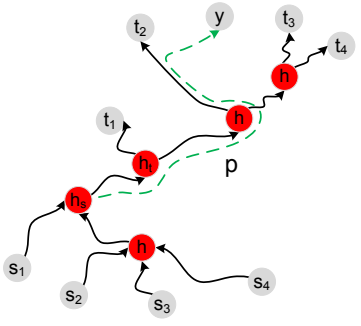

Fig. 1. Arc Reuse for the case when the source node x is one of the red colored hypernodes. The insertion path p for $a = (x, y)$ starts at node h_s.

Splitting other Hyperarcs/-nodes. The knowledge of the properties of hypernodes allows a further improvement, lifted from [6] where it was used in the context of undirected edge insertion for the so-called *minor-monotone* crossing number [3]. We can cross "through" a hypernode h of another hyperedge ψ, as long as we thereby do not separate both source and target nodes from each other, cf. Fig. 2: By our induction hypothesis on confluency, crossing through h means that we split h into two hypernodes h and h' and add the arc (h, h'). Then we change the source (or target) vertex of at least two arcs from h to h'. Let A_{in} and A_{out} be the arcs formerly entering and leaving h, respectively. To ensure confluency, we only allow a split where (a) all A_{in} remain incident to h, (b) all A_{out} become incident to h', or (c) neither of both, but either A_{in} or A_{out} is an empty set. After this split, our routing path can cross over the arc (h, h'). Note that, since we consider a fixed embedding, all valid hypernode-crossings can be modeled via arcs in the routing network directly connecting two faces (e.g., f_6, f_3 in Fig. 2). Such arcs have cost 1, as they induce a hypernode split such that a single crossing suffices.

Inserting chain-transformed arcs, i.e., $a \in Chn$. If $a = (x, y)$ has to leave x downwards or enter y upwards, we have to extend our routing network further. Assume that a only requires the second property (the first one is independent of the second and can be solved analogously). Usually, all arcs dominated by y will be *statically locked*, i.e., we may not cross through them. Now, we unlock the arcs that directly leave y, and search for a path entering y from there; depending on the exact port constraint, only a single face above y might be a valid entrance point into y. We now place the dummy node of the chain-transformation, where (x, y) was split into $(x, d), (y, d)$, into the face from which y is entered. Then all crossings are still drawn only between upward arcs. The small subarc (y, d), which will be reversed in the final drawing, is even drawn without any crossings. By forbidding crossings to happen over (y, d) in the subsequent steps, we therefore guarantee that the overall drawing is still upward, and the

(a) The insertion path p cross a hypernode h. (b) A realization of p.

Fig. 2. Splitting a hypernode by introducing an additional arc can reduce the crossings

small downward pieces of arcs, due to port constraints, are restricted to the direct neighborhood of the corresponding node.

Note that, after the full upward-planarization approach is completed, we can merge multiple such dummy nodes that correspond to the same hyperarc, when they lie in a common face.

Inserting reversed arcs, i.e., $a \in Rev$. To ensure confluency in the hyperarcs, we have to take special care for the arcs that where reversed in the preprocessing step to remove cycles. After the drawing is computed, we will have to reset their original direction. To avoid notational complexity, we will add such arcs only after all other arcs of the hyperarc are already inserted.

Assume the arc $a = (x, y)$ originally connected a source vertex to the hypernode of the star-based underlying graph, but got reversed and hence connects the hypernode to a target vertex. Nonetheless we have to ensure that it connects to the aforementioned subtree T_s, instead of T_t. Let S be the set of original sources in T_s before inserting a. Then a confluency-feasible upward insertion path for a can be found by selecting the minimal insertion path from any node in S to y. Thereby it may cross over $A^\circ_\phi \setminus T_t$ and next to crossing dummies of $A^\circ_\phi \setminus T_t$ at no cost. The analogous holds, if a originally connected a target vertex to the star-based hypernode.

Putting it together. So overall, after first computing a special feasible upward subgraph, our upward-planarization approach inserts one hyperarc after another. Each hyperarc is inserted by incrementally inserting the arcs of the star-based underlying graph, reusing the already established tree-based sub-drawing of the hyperarc as far as possible. By specially considering *original* arc directions, we thereby guarantee that all hyperarcs are drawn as confluent trees. The final result of the planarization is an upward-planar representation (planar, upward-feasible sT-graph) $\mathcal{R}$ of G, and hence of H, together with an embedding Γ of $\mathcal{R}$. Within $\mathcal{R}$, hyperarcs of H are represented as confluent trees, and crossings are represented as dummy nodes.

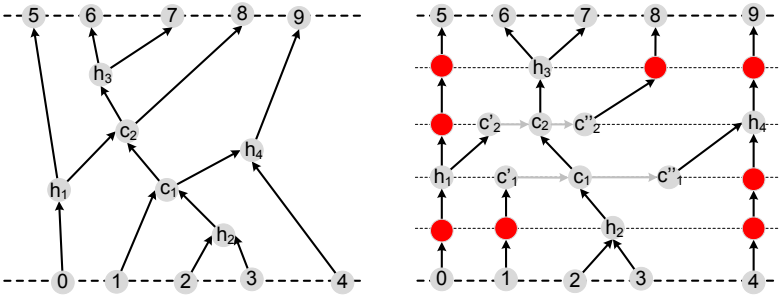

Fig. 3. Steps towards a final layout: (a) the subgraph between consecutive layers of an UPR $\mathcal{R}$, (b) fine-layering of the subgraph with included dummy nodes to split long arcs

3 Layout

Using the embedding Γ of the upward-planar planarization (UPR) $\mathcal{R}$ computed in the previous step, our layout procedure works in three steps:

1. A layering $\mathcal{L}$ of $\mathcal{R}$ is computed.
2. An initial orthogonal drawing of $\mathcal{R}$ is computed.
3. Optional step: Orthogonal compaction is applied to remove unnecessary bend points and improve layout quality.

We finally obtain an upward drawing of $\mathcal{R}$, which induces a drawing of H in a straight-forward way. The number of hyperarc crossings in this drawing equals the number of crossing dummies in $\mathcal{R}$, thus it is minimal with respect to the computed embedding. Our initial method for orthogonal upward drawing may produce an unnecessarily high number of bend points, but it reveals that such a drawing can be computed efficiently. We discuss the individual steps in more detail.

Layering. We compute a layering of $\mathcal{R}$ in two phases using the layering algorithm by Chimani et al. [8], which also induces a node ordering for each layer. In the first phase we compute a layering $\mathcal{L}'$ of the nodes of H, and in the second phase a layering $\mathcal{L}''$ of the subgraph between each two consecutive layers of $\mathcal{L}'$. Notice that the nodes of $\mathcal{L}''$ are either crossing dummies or hypernodes. For each crossing dummy c of $\mathcal{L}''$, we split c by adding new dummy nodes c' and c'' such that c' is the immediate left and c'' is the immediate right neighbor of c. These two nodes will be bend points in the later drawing and ensure the orthogonality of the arcs incident to c. We redirect the left incoming and the right outgoing arc of c such that c' is its new target and c'' is its new source node, respectively. We then merge $\mathcal{L}'$ and $\mathcal{L}''$ into a complete layering $\mathcal{L}$ of $\mathcal{R}$ and create dummy nodes to split long edges that span multiple layers. An example is shown in Fig. 3.

Initial orthogonal drawing. Using the layering $\mathcal{L}$, we apply an arbitrary coordinate assignment algorithm known for the third step of Sugiyama's framework (e.g., [4, 5, 11, 15]) to compute horizontal node coordinates. The orthogonal routing of the edges between each pair of consecutive layers can then be calculated using an existing method for layer-based routing, such as the one proposed by Sander [15]. Since the embedding of $\mathcal{R}$ is planar, such a routing can be constructed without introducing additional arc crossings. As a result, all incoming and outgoing arcs of hypernodes and dummy nodes end in the same point, where incoming arcs reach the nodes from below and outgoing arcs leave the nodes upwards. If there are multiple incoming or multiple outgoing arcs of a node u, the corresponding vertical line segments that touch u overlap each other. These overlapping line segments need to be merged to single line segments.

For the computation of the orthogonal layout, port constraints essentially need to be considered only in the routing algorithm, since they determine where the arcs shall touch the connected nodes. In case of ports that are situated on the left or right side of a node, we can artificially broaden the node prior to the horizontal coordinate calculation and add one bend point per arc such that incoming and outgoing arcs are redirected downwards and upwards, respectively (see [14], Sect. 7).

Applying orthogonal compaction. The resulting initial drawing may contain various unnecessary bends; many such bends can be removed using orthogonal compaction techniques (see [13] for an overview). First, we connect the nodes of each pair of consecutive layers L_{top} and L_{bottom} of $\mathcal{L}'$ by horizontal edges and connect the first nodes and the last nodes on these layers by edges with two bend points, such that all these additional edges form a surrounding rectangular frame (cf. Figure 4). We assign fixed edge lengths to the edges on this frame, so that the nodes on L_{top} and L_{bottom} will remain on their horizontal positions. Then, we compute the orthogonal representation induced by this drawing and apply orthogonal flow-based compaction. This allows us to assign costs to segments. Let $\mathcal{S}_0$ denote the set of line segments adjacent to two bend points with a 90 and a 270 degree angle in a face. The segments in $\mathcal{S}_0$ are assigned maximal cost and zero minimal length. The minimum length of the remaining segments is set to the minimum of their lengths in the initial drawing and a desired spacing, making sure that the initial drawing is a feasible solution for the compaction. Each segment in $\mathcal{S}_0$ for which the compaction achieves zero length is then removed from the orthogonal representation by merging its adjacent segments.

Final layout. The so constructed drawing is an orthogonal drawing and its arc crossings are exactly the crossings modeled by $\mathcal{R}$. In order to obtain a valid drawing of the original hypergraph H, we perform the following post-processing steps:

a) Replace each crossing dummy c and its corresponding neighboring nodes c' and c'' by a horizontal line segment from c' to c''.

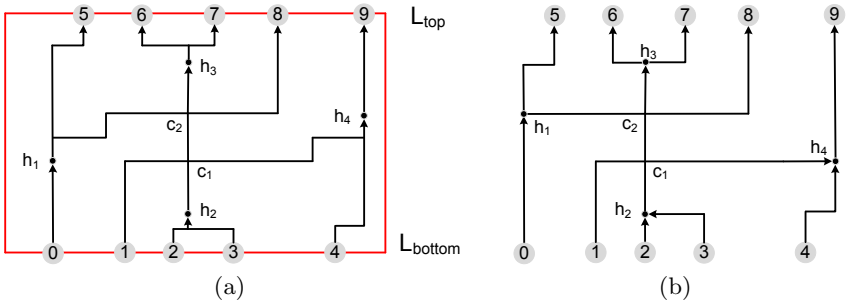

Fig. 4. Orthogonal compaction: (a) An initial drawing of Figure 3(a) with unnecessary bend points. Before starting the compaction, the drawing is framed by a rectangle (red edges) and the new edges are assigned fixed lengths. (b) The final drawing.

b) Eliminate all remaining dummy nodes and hypernodes by directly connecting the line segments of the incoming and outgoing arcs.
c) Reverse all arcs of *Rev*, which were previously reversed to break cycles.

We can add an additional compaction step on the whole layout—similar as before—to further improve the layout. Notice that when considering the whole layout, we do not need to fix the x-coordinates; we only have to ensure that the horizontal distance between ports is fixed.

4 Conclusion

We presented the first planarization approach for hypergraphs in the context of upward drawings. To this ends, we combined the known ideas of upward arc insertion and insertion of a single edge in the undirected minor crossing number setting with a novel heuristic method to assemble multiple adequately chosen insertion paths to a confluently drawn hyperarc. Furthermore, we dealt with the problem of laying out the so obtained upward-planar representation in an orthogonal upward drawing style. As our hyperarcs offer more freedom than prior approaches within the Sugiyama framework, their orthogonalization step is inapplicable for our needs, even after layering. We therefore introduced a new global scheme based on orthogonal compaction.

We also considered port constraints both in the upward-planarization as well as in the layout step, as they are integral to many hypergraph drawing applications such as pins in electrical circuits. We want to stress that our algorithm not only solves the upward drawing problem for hypergraphs with and without port constraints, but also is the first port-constraint-aware upward-planarization approach that is suitable for regular DAGs.

Based on the experience with prior upward-planarization methods, we expect that our methods should work well in practice—the implementation, together with a thorough experimental investigation comparing this approach to the more traditional hypergraph drawing algorithms within the Sugiyama framework, remains

as our next research step. Furthermore, we would be interested in more direct orthogonalization methods, along the lines of, e.g., [15, 10]. These methods solve the problem locally on a layer-by-layer basis, but extending them to allow multiple horizontally drawn hypernodes per hyperarc between two layers is non-trivial.

References

[1] Baburin, D.E.: Using graph based representations in reengineering. In: Proc. CSMR 2002, pp. 203–206 (2002)
[2] Batini, C., Talamo, M., Tamassia, R.: Computer aided layout of entity relationship diagrams. J. Syst. Software 4, 163–173 (1984)
[3] Bokal, D., Fijavz, G., Mohar, B.: The minor crossing number. SIAM J. Discrete Math. 20, 344–356 (2006)
[4] Brandes, U., Köpf, B.: Fast and simple horizontal coordinate assignment. In: Proc. Graph Drawing 2001, London, UK, pp. 31–44. Springer, Heidelberg (2002)
[5] Buchheim, C., Jünger, M., Leipert, S.: A fast layout algorithm for k-level graphs. In: Marks, J. (ed.) GD 2000. LNCS, vol. 1984, pp. 229–240. Springer, Heidelberg (2001)
[6] Chimani, M., Gutwenger, C.: Algorithms for the hypergraph and the minor crossing number problems. In: Tokuyama, T. (ed.) ISAAC 2007. LNCS, vol. 4835, pp. 184–195. Springer, Heidelberg (2007)
[7] Chimani, M., Gutwenger, C., Mutzel, P., Wong, H.-M.: Layer-free upward crossing minimization. ACM Journal of Experimental Algorithmics 15 (2010)
[8] Chimani, M., Gutwenger, C., Mutzel, P., Wong, H.-M.: Upward planarization layout. In: Eppstein, D., Gansner, E.R. (eds.) GD 2009. LNCS, vol. 5849, pp. 94–106. Springer, Heidelberg (2010)
[9] Eiglsperger, M., Fößmeier, U., Kaufmann, M.: Orthogonal graph drawing with constraints. In: Proc. SODA 2000, pp. 3–11. SIAM, Philadelphia (2000)
[10] Eschbach, T., Guenther, W., Becker, B.: Orthogonal hypergraph drawing for improved visibility. J. Graph Algorithms Appl. 10(2), 141–157 (2006)
[11] Gansner, E., Koutsofios, E., North, S., Vo, K.-P.: A technique for drawing directed graphs. Software Pract. Exper. 19(3), 214–229 (1993)
[12] Gutwenger, C., Klein, K., Mutzel, P.: Planarity testing and optimal edge insertion with embedding constraints. J. Graph Algorithms Appl. 12(1), 73–95 (2008)
[13] Klau, G.W., Klein, K., Mutzel, P.: An experimental comparison of orthogonal compaction algorithms. In: Proc. Graph Drawing 2000, London, UK, pp. 37–51. Springer, Heidelberg (2001)
[14] Sander, G.: Graph layout through the VCG tool. Technical Report A03/94, Universität des Saarlandes, FB 14 Informatik, 66041 Saarbrücken (October 1994)
[15] Sander, G.: A fast heuristic for hierarchical Manhattan layout. In: Brandenburg, F.J. (ed.) GD 1995. LNCS, vol. 1027, pp. 447–458. Springer, Heidelberg (1996)
[16] Sander, G.: Layout of directed hypergraphs with orthogonal hyperedges. In: Liotta, G. (ed.) GD 2003. LNCS, vol. 2912, pp. 381–386. Springer, Heidelberg (2004)
[17] Spönemann, M., Fuhrmann, H., von Hanxleden, R., Mutzel, P.: Port constraints in hierarchical layout of data flow diagrams. In: Eppstein, D., Gansner, E.R. (eds.) GD 2009. LNCS, vol. 5849, pp. 135–146. Springer, Heidelberg (2010)
[18] Sugiyama, K., Tagawa, S., Toda, M.: Methods for visual understanding of hierarchical system structures. IEEE Trans. Sys. Man. Cyb. 11(2), 109–125 (1981)