Counting Crossings for Layered Hypergraphs

Miro Spönemann, Christoph Daniel Schulze,
Ulf Rüegg, and Reinhard von Hanxleden

Department of Computer Science, Christian-Albrechts-Universität zu Kiel
{msp,cds,uru,rvh}@informatik.uni-kiel.de

Abstract. Orthogonally drawn hypergraphs have important applications, e. g. in actor-oriented data flow diagrams for modeling complex software systems. Graph drawing algorithms based on the approach by Sugiyama et al. place nodes into consecutive layers and try to minimize the number of edge crossings by finding suitable orderings of the nodes in each layer. With orthogonal hyperedges, however, the exact number of crossings is not determined until the edges are actually routed in a later phase of the algorithm, which makes it hard to evaluate the quality of a given node ordering beforehand. In this paper, we present and evaluate two cross counting algorithms that predict the number of crossings between orthogonally routed hyperedges much more accurately than the traditional straight-line method.

Keywords: edge crossings, hypergraphs, graph drawing, layered graphs.

1 Introduction

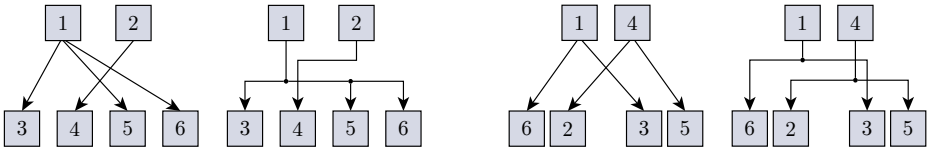
Many kinds of diagrams, e. g. data flow diagrams and circuit diagrams, can be formalized as *directed hypergraphs*. A directed hypergraph is a pair $G = (V, H)$ where V is a set of nodes and $H \subseteq \mathcal{P}(V) \times \mathcal{P}(V)$ is a set of hyperedges. Each $(S, T) \in H$ has a set of *sources* S and a set of *targets* T.

The layer-based approach to graph drawing proposed by Sugiyama et al. [1] has been extended for drawing hypergraphs orthogonally [2,3]. This approach can be structured in five phases: eliminate cycles by reversing edges, assign nodes to *layers*, reorder the nodes of each layer such that the number of crossings is low, determine concrete positions for the nodes, and finally route the hyperedges by computing bend points and junction points. The reordering phase is usually done with the *layer sweep* heuristic, where layers are processed one at a time with several iterations until the number of crossings is not further improved.

In order to determine whether an iteration of the the layer sweep heuristic has reduced the number of edge crossings, they need to be counted. A fundamental problem with this approach is that the actual number of crossings in orthogonal drawings does not depend only on the order of nodes in each layer, but also on the routing of edges. This routing in turn depends on the concrete positions of the nodes, which are unknown at the time the layer sweep heuristic is executed.

Standard algorithms for counting crossings in layered graphs assume that edges are drawn as straight lines [4]. We call this standard approach Straight,

T. Dwyer et al. (Eds.): Diagrams 2014, LNAI 8578, pp. 9–15, 2014.



(a) $c_s > c$ (b) $c_s < c$

Fig. 1. The number of crossings c_s resulting from a straight-line drawing can be (a) greater or (b) less than the actual number of crossings c resulting from an orthogonal hyperedge routing.

and denote its result as c_s. As noted by Eschbach et al. [5], there are simple examples where c_s is always different from the actual number of crossings c obtained after routing edges orthogonally (see Fig. 1). In order to quantify this difference, we measured c and c_s for a number of data flow diagrams from the Ptolemy project (see Sect. 3). The difference $c - c_s$ averaged -34 with a standard deviation of 190. As a general observation, the Straight approach tends to overestimate the crossing number, possibly compromising the quality of the resulting drawings.

Contributions. The number of crossings between straight-line edges is a bad predictor for the number of crossings between orthogonal hyperedges in the final drawing. We propose two methods for counting crossings that predict the number of crossings much more accurately, as our evaluation shows.

Related Work. Eschbach et al. [2] and Sander [3] proposed methods for the orthogonal routing of edges in the layer-based approach. They both noted that the number of crossings determined during the node ordering phase is only an approximation, but gave no proposals on how to solve this problem. In this paper, we will present algorithms that give much more accurate approximations.

Several authors have addressed the problem of counting straight-line crossings in layered graphs [4,6,7]. These methods produce exact results for normal graphs, but not for hypergraphs, as explained above. The "Straight approach" which we refer to in this paper represents any exact straight-line method. For our experiments we implemented the method of Barth et al. [4].

2 Counting Crossings

The main concept for representing a hyperedge in the layer-based approach is to replace the hyperedge by regular edges. Let (V, H) be a hypergraph and $h = (S, T) \in H$ be a hyperedge; for each $v \in S$ and each $v' \in T$ we generate an edge $e = (v, v')$. We call e a *representing edge* of h and define E_h to be the set of all representing edges of h. For instance, the hyperedge connected to node 1 in Fig. 1(a) would be represented by three edges $(1, 3)$, $(1, 5)$, and $(1, 6)$. These edges may partly overlap each other in the final drawing.

2.1 Lower Bound Method

Since counting straight-line crossings tends to yield rather pessimistic estimates when hyperedges are involved, we assumed that a more accurate approach might be to use a lower bound of the number of crossings.

In the following, let $G = (V, H)$ be a hypergraph with a set $E = \bigcup_{h \in H} E_h$ of representing edges and two layers L_1, L_2, i. e. $V = L_1 \cup L_2$, $L_1 \cap L_2 = \emptyset$, and all $h \in H$ have their sources in L_1 and their targets in L_2.

We propose an optimistic method MinOpt and denote its result as c_m. This method counts the minimal number of crossings to be expected by evaluating each unordered pair $h_1, h_2 \in H$: if any edge $e_1 \in E_{h_1}$ crosses an edge $e_2 \in E_{h_2}$ if drawn as a straight line, h_1 and h_2 are regarded as crossing each other once, denoted as $h_1 \bowtie h_2$. The result is $c_\mathrm{m} = |\{\{h_1, h_2\} \subseteq H : h_1 \bowtie h_2\}|$.

Observation 1. $c_\mathrm{m} \leq c_\mathrm{s}$.

Observation 2. *Let c be the number of hyperedge crossings in a layer-based drawing of G. Then $c_\mathrm{m} \leq c$.*

Theorem 1. *Let $q = |H|$ and $H = \{h_1, \ldots, h_q\}$. The time complexity of* MinOpt *is $\mathcal{O}\Big(\sum_{i=1}^{q-1} \sum_{j=i+1}^{q} |E_{h_i}| \cdot |E_{h_j}|\Big)$. If $|S| = |T| = 1$ for all $(S, T) \in H$, the complexity can be simplified to $\mathcal{O}\big(|H|^2\big)$.*

Proof. The result of MinOpt is $|\{\{h_i, h_j\} \subset H : h_i \bowtie h_j\}|$, which requires to check all unordered pairs $U = \{\{h_i, h_j\} \subset H\}$. This is equivalent to $U = \{(i, j) \in \mathbb{N}^2 : 1 \leq i < q,\ i < j \leq q\}$, hence $|U| = \sum_{i=1}^{q-1} \sum_{j=i+1}^{q} 1$. Whether $h_i \bowtie h_j$ is determined by comparing all representing edges of h_i with those of h_j, which requires $|E_{h_i}| \cdot |E_{h_j}|$ steps. In total we require $\sum_{i=1}^{q-1} \sum_{j=i+1}^{q} |E_{h_i}| \cdot |E_{h_j}|$ steps. The simplification follows immediately. □

2.2 Approximating Method

Theorem 1 shows that MinOpt has a roughly quadratic time complexity. In this section we propose a second method with better time complexity, which we call ApproxOpt. The basic idea is to approximate the result of MinOpt by checking three criteria explained below, hoping that at least one of them will be satisfied for a given pair of hyperedges if they cross each other in the final drawing.

Let again $G = (V, H)$ be a hypergraph with layers L_1, L_2. Let $\pi_1 : L_1 \to \{1, \ldots, |L_1|\}$ and $\pi_2 : L_2 \to \{1, \ldots, |L_2|\}$ be the permutations of L_1 and L_2 that result from the layer sweep heuristic for crossing minimization. We denote the result of ApproxOpt as c_a.

The ApproxOpt method is based on the four *corners* of a hyperedge: for each $h = (V_{h1}, V_{h2}) \in H$ and $i \in \{1, 2\}$, we define the left corners $\kappa_i^{\leftarrow}(h) = \min\{\pi_i(v) : v \in V_{hi}\}$ and the right corners $\kappa_i^{\rightarrow}(h) = \max\{\pi_i(v) : v \in V_{hi}\}$. The *virtual edges* are defined by $E^* = \{(\kappa_1^{\leftarrow}(h), \kappa_2^{\leftarrow}(h)) : h \in H\}$. The method consists of three steps:

Algorithm 1. Counting crossings with the ApproxOpt method

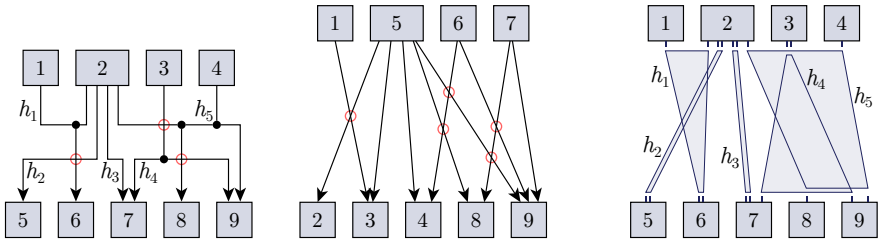
Input: L_1, L_2 with permutations π_1, π_2, hyperedges H with arbitrary order ϑ
for each $h \in H$ **do** *// Step 1*
  Add $(\kappa_1^{\leftarrow}(h), \kappa_2^{\leftarrow}(h))$ to E^*
$c_\mathrm{a} \leftarrow$ number of crossings caused by E^*, counted with a straight-line method
for $i = 1 \ldots 2$ **do** *// Steps 2 and 3*
  for each $h \in H$ **do**
    Add $(\kappa_i^{\leftarrow}(h), \kappa_i^{\rightarrow}(h), \vartheta(h), -1))$ and $(\kappa_i^{\rightarrow}(h), \kappa_i^{\leftarrow}(h), \vartheta(h), 1))$ to C_i
  Sort C_i lexicographically
  $d \leftarrow 0$
  for each $(x, x', j, t) \in C_i$ in lexicographical order **do**
    $d \leftarrow d - t$
    if $t = 1$ **then**
      $c_\mathrm{a} \leftarrow c_\mathrm{a} + d$
return c_a

1. Compute the number of straight-line crossings caused by virtual edges between the left corners.
2. Compute the number of overlaps of ranges $[\kappa_1^{\leftarrow}(h), \kappa_1^{\rightarrow}(h)]$ in the first layer for all $h \in H$.
3. Compute the number of overlaps of ranges $[\kappa_2^{\leftarrow}(h), \kappa_2^{\rightarrow}(h)]$ in the second layer for all $h \in H$.

The result c_a is the sum of the three numbers computed in these steps. A more detailed description is given in Alg. 1.

Step 1 aims at "normal" crossings of hyperedges such as h_1 and h_2 in Fig. 2. The hyperedge corners used in Steps 2 and 3 serve to check for overlapping areas, as shown in Fig. 2(c). For instance, the ranges spanned by h_4 and h_5 overlap each other both in the first layer and in the second layer. This is determined using a linear pass over the hyperedge corners, which are sorted by their positions. The sort keys are constructed such that the overlapping of two ranges is counted only if it actually produces a crossing. The variable d is increased whenever a left-side corner is found and decreased whenever a right-side corner is found. This variable indicates how many ranges of other hyperedges surround the current corner position, hence its value is added to the approximate number of crossings.

While MinOpt counts at most one crossing for each pair of hyperedges, ApproxOpt may count up to three crossings, since the hyperedge pairs are considered independently in all three steps. Fig. 3(a) shows an example where MinOpt counts a crossing and ApproxOpt counts none, while Fig. 3(b) shows an example where ApproxOpt counts a crossing and MinOpt counts none. Thus neither $c_\mathrm{m} \leq k c_\mathrm{a}$ nor $c_\mathrm{a} \leq k c_\mathrm{m}$ hold in general for any $k \in \mathbb{N}$. However, as shown in Sect. 3, the difference between c_m and c_a is rather small in practice.



(a) A hypergraph (b) Representing edges (c) Overlapping areas

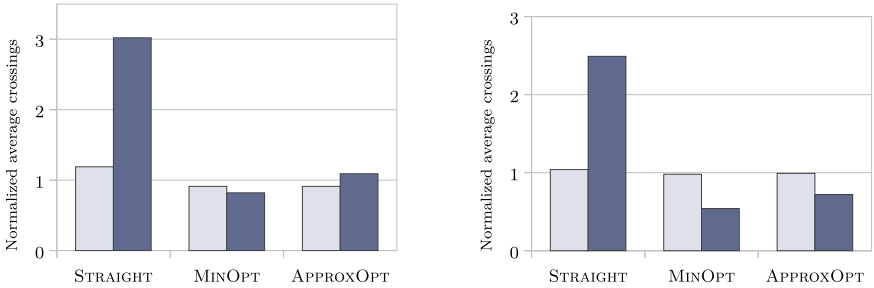
Fig. 2. The hypergraph (a) can be drawn orthogonally with $c = 3$ crossings. The straight-line crossing number (b) is $c_\mathrm{s} = 5$, the result of MinOpt is $c_\mathrm{m} = 2$, and the result of ApproxOpt is $c_\mathrm{a} = 4$. ApproxOpt counts three crossings between h_4 and h_5 (c) because the virtual edges $(2, 8)$ and $(3, 7)$ cross (Step 1 in Alg. 1) and the ranges spanned by the corners overlap both in the top layer and in the bottom layer (Steps 2 and 3).

(a) $c_\mathrm{m} = 1$, $c_\mathrm{a} = 0$ (b) $c_\mathrm{m} = 0$, $c_\mathrm{a} = 1$

Fig. 3. Differences between the MinOpt and ApproxOpt methods: (a) $c_\mathrm{m} = 1$ due to the crossing of $(1, 4)$ and $(2, 3)$, but $c_\mathrm{a} = 0$ since none of the three steps of ApproxOpt is able to detect that. (b) $c_\mathrm{m} = 0$ because $(2, 4)$ crosses neither $(1, 4)$ nor $(3, 4)$; $c_\mathrm{a} = 1$ because one crossing is detected in Step 2 of Alg. 1.

Theorem 2. *Let* $b = \sum_{(S,T)\in H}(|S|+|T|)$. *The time complexity of* ApproxOpt *is* $\mathcal{O}(b + |H|(\log|V| + \log|H|))$.

Proof. In order to determine the corners $\kappa_i^{\leftarrow}(h), \kappa_i^{\rightarrow}(h)$ for each $h \in H$, $i \in \{1, 2\}$, all source and target nodes are traversed searching for those with minimal and maximal index π_i. This takes $\mathcal{O}\left(\sum_{(S,T)\in H}(|S| + |T|)\right) = \mathcal{O}(b)$ time. The number of virtual edges created for Step 1 is $|E^*| = |H|$. Counting the crossings caused by E^* can be done in $\mathcal{O}(|E^*| \log|V|) = \mathcal{O}(|H| \log|V|)$ time [4]. Steps 2 and 3 require the creation of a list C_i with $2|H|$ elements, namely the lower-index and the upper-index corners of all hyperedges. Sorting this list is done with $\mathcal{O}(|C_i| \log|C_i|) = \mathcal{O}(|H| \log|H|)$ steps. Afterwards, each element in the list is visited once. The total required time is $\mathcal{O}(b + |H| \log|V| + |H| \log|H|) = \mathcal{O}(b + |H|(\log|V| + \log|H|))$. □



(a) Ptolemy graphs, $\bar{c} \approx 18.75$ (b) Random graphs, $\bar{c} \approx 1628$

Fig. 4. Average number of crossings when using the given cross counting algorithm (light) and average number of crossings predicted by the algorithm (dark).

3 Experimental Evaluation

For evaluating our algorithms, we used 171 diagrams taken from the set of demo models shipping with the Ptolemy open source project [8].[1] Since Ptolemy allows models to be nested using *composite actors* that represent subsystems composed of other actors, we eliminated the hierarchy by moving nested elements out of their composite actors. Diagrams unsuitable for evaluations were left out, e. g. those with very few nodes.

We executed our drawing algorithm once for each cross counting algorithm on each of the selected Ptolemy diagrams. For each execution, the actual number of crossings in the final diagram as well es the number predicted by the cross counting algorithm were measured. The results can be seen in Fig. 4(a). Our proposed methods are by far more accurate at predicting the number of crossings compared to the straight-line method. While the average difference of the actual and predicted numbers of crossings was 35.6 for Straight, the difference averaged 5.3 for MinOpt and 5.7 for ApproxOpt. A further important observation is that the average number of actual crossings is reduced by 23.6% when using MinOpt and by 23.8% when using ApproxOpt instead of Straight. These differences of mean values are significant: the p-values resulting from a t-test with paired samples are 4.5% for MinOpt and 4.0% for ApproxOpt.

We performed a second experiment with randomly generated bipartite graphs with 5 to 100 nodes and 2 to 319 hyperedges each. We performed the same measurements as for the Ptolemy diagrams, the results of which are shown in Fig. 4(b). They confirm the general observations made before. The average number of actual crossings is reduced by 5.6% when using MinOpt and by 4.6% when using ApproxOpt instead of Straight. Although the relative difference of mean values is lower compared to the Ptolemy diagrams, their significance is much higher: $p \approx 7.8 \times 10^{-43}$ for MinOpt and $p \approx 3.2 \times 10^{-32}$ for ApproxOpt.

[1] `http://ptolemy.eecs.berkeley.edu`

Performance evaluations conducted on a set of 100 randomly generated large bipartite graphs (500 nodes, 3 edges per node) confirmed our theoretical results: Straight (mean time 0.3ms) was significantly faster than ApproxOpt (1.7ms), which in turn was significantly faster than MinOpt (24ms).

4 Conclusion

We proposed two methods for counting crossings in orthogonal hypergraph drawings more accurately. Our experiments indicate that the algorithms lead to significantly fewer edge crossings both with real-world and with random diagrams.

We see two main areas for future research. First, the number of crossings between orthogonal hyperedges depends not only on the results of the crossing minimization, but also on the exact placement of nodes. However, current node placement algorithms only try to minimize either edge length or the number of bend points. And second, limiting the routing of each hyperedge to one horizontal segment reduces the number of bend points at the expense of edge crossings. Future research could address routing algorithms that reduce the number of edge crossings as well by creating multiple horizontal segments.

References

1. Sugiyama, K., Tagawa, S., Toda, M.: Methods for visual understanding of hierarchical system structures. IEEE Transactions on Systems, Man and Cybernetics 11(2), 109–125 (1981)
2. Eschbach, T., Guenther, W., Becker, B.: Orthogonal hypergraph drawing for improved visibility. Journal of Graph Algorithms and Applications 10(2), 141–157 (2006)
3. Sander, G.: Layout of directed hypergraphs with orthogonal hyperedges. In: Liotta, G. (ed.) GD 2003. LNCS, vol. 2912, pp. 381–386. Springer, Heidelberg (2004)
4. Barth, W., Mutzel, P., Jünger, M.: Simple and efficient bilayer cross counting. Journal of Graph Algorithms and Applications 8(2), 179–194 (2004)
5. Eschbach, T., Guenther, W., Becker, B.: Crossing reduction for orthogonal circuit visualization. In: Proceedings of the 2003 International Conference on VLSI, pp. 107–113. CSREA Press (2003)
6. Nagamochi, H., Yamada, N.: Counting edge crossings in a 2-layered drawing. Information Processing Letters 91(5), 221–225 (2004)
7. Waddle, V., Malhotra, A.: An $E \log E$ line crossing algorithm for levelled graphs. In: Kratochvíl, J. (ed.) GD 1999. LNCS, vol. 1731, pp. 59–71. Springer, Heidelberg (1999)
8. Eker, J., Janneck, J.W., Lee, E.A., Liu, J., Liu, X., Ludvig, J., Neuendorffer, S., Sachs, S., Xiong, Y.: Taming heterogeneity—the Ptolemy approach. Proceedings of the IEEE 91(1), 127–144 (2003)