Strict confluent drawing

Soeren Nickel
Matr.: 01528974
Curriculum: 186.862 - Seminar in Algorithms Graphs and Geometry
e01528974@student.tuwien.ac.at

February 15, 2018

Abstract

This paper aims to present strict confluent drawings, a way of visualizing non-planar graphs in a crossing free way, and tries to place them in a current field of research. Adjacencies are modeled through a smooth path between two vertices, which can merge and split at junctions. An additional restriction (strictness) assures unique paths and no self loops.

1 Introduction

1.1 Motivation

Strict Confluent Drawings are a subclass of confluent drawings. In a confluent drawing adjacencies between two vertices are represented by a smooth path from one vertex to the other. This path is composed of arcs and junctions between these arcs. Every arc is adjacent to either a junction or a vertex at both ends. The resulting representation resembles a system of train tracks that merge and split at turnouts. Two arcs that meet at such a junction have, at the point of the junction, the same tangent. This leads to the possibility to enter the junction on one side and leave it on the other all on a smooth path, but makes a jump from one arc to another one, which enters the junction from the same side, impossible. A path from one vertex to another, that represents an adjacency, does not pass through another vertex.

Graph drawings can be optimized for better readability by optimizing various different characteristics and aesthetics of the drawing [17]. One of these aesthetics is to minimize the number of edge-crossings in a drawing. With confluent drawings we have the ability to visualize even very dense non-planar graphs in a visually improved manner, as we can see in figure 1. Confluent drawings can share a visual similarity to another method of graph drawing called edge bundling [2, 4, 9, 10, 12], in which edges are contracted to be near each other when they cross other edges in order to cluster multiple crossings in similar directions into something that might be recognized as one big crossing. However in contrast to edge bundling, confluent drawings have the advantage that they do not create false adjacencies, are unambiguous and stay “information faithful” [13].

The main contribution [7] which we will look at in this paper is about strict confluent drawings. The *strictness* adds the following restrictions to a drawing of a graph:

- If there exists a smooth path between two vertices, this path is unique. There cannot be another smooth path connecting these two vertices.
- There cannot be a smooth path from a vertex to itself

Through these restrictions, three things are intended to be achieved. First a better readability of a drawing since every path is unique. Secondly, by restricting the graph class like this, it is possible to completely characterize the complexity of the drawings, which is “the first such characterization for any form of confluence on arbitrary undirected graphs” [7]. And finally we get a clear path (in the drawing) to edge (in the graph) relation. The restrictions are visualized in Figure 1 (c) and (d).

We will talk in depth about the complexity of the drawings, as well as the complexity of recognizing if a given graph has a strict confluent drawing or not, in chapters 3 and 4.

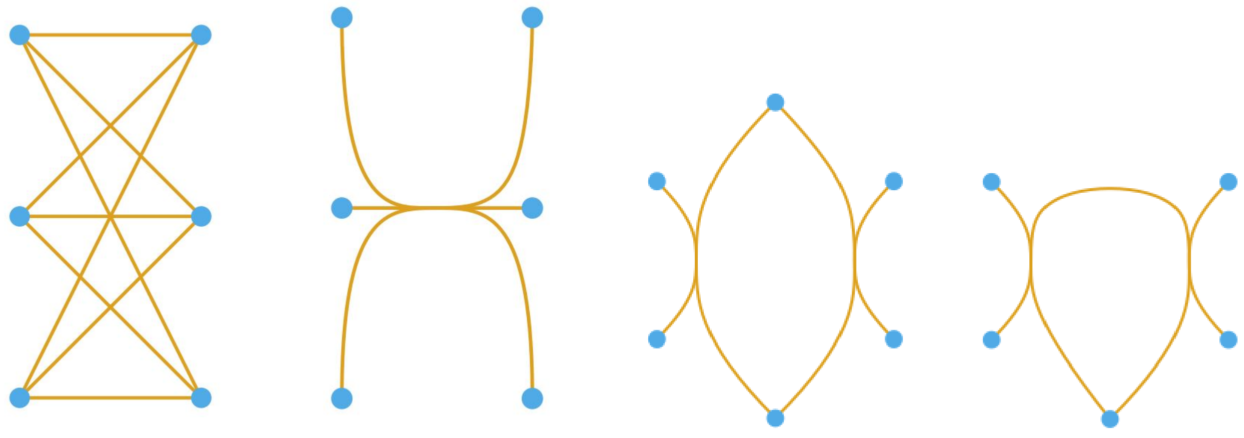


Figure 1: A complete Graph $K_{3,3}$ as a classic node-link drawing (a) and as a confluent drawing (b) and graph drawings that violate either uniqueness of paths (c) or prohibition of self loops (d)

1.2 Related Work

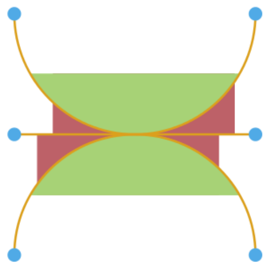
Confluent Drawings were introduced by Dickerson et al. [3], who provide two heuristic algorithms (for directed and undirected graphs), focusing on graphs with bounded arboricity. They also identify large graph classes (e.g. complement of trees, interval) which can be realized as a confluent drawing as well as a proof that there exist graphs for which such a drawing does not exist (e.g. Petersen graph, 4-dimensional hypercube). Additional work on confluent drawings include work on tree confluent drawings [11], where the confluent drawing of a graph is tree-like and an extension of tree confluent drawings called Δ-confluent drawings [5], where the junctions are allowed to join up to 3 arcs at the same time. These Drawings can be drawn in $\mathcal{O}(n \log n)$ area on a hexagonal grid according to Eppstein et al. [5]. Strong confluent drawings which are a subclass of confluent drawings can be recognized in polynomial time. Another extension are the so called layered confluent drawings [6]. Here the concept of layered drawings is combined with confluence by connecting adjacent layers with confluent bicliques.

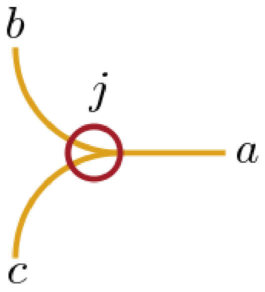
There are a couple of heuristic algorithms for creating confluent drawings. One method uses the biclique edge cover graph of a graph to create a confluent drawing [8], another one is based on rectangular duals [18]. Bach et al. [1] present an algorithm to construct confluent drawings from arbitrary directed and undirected graphs. They also compare confluent drawings with other graph drawing methods and found that readability for unexperienced users and the correctness of the percieved connectivity is increased, compared to other visualizations..

Even though confluent drawings are not yet used in real world applications [1, 13], there is potential for them to be used since a multitude of problems and applications contain non-planar graphs which can be realized in a readable way through a confluent way. One such example is the (partial) visualization of real-world multivariate datasets, that are too large to display in their entirety [13].

The restriction to strict confluent drawings, as mentioned above, did not see much work yet. The introductory paper [7] which is the main focal point of this paper as well as the report from Dagstuhl Seminar 13151 [14], from which it originated, are at the point of writing the only contribution to this topic. They introduce three different results:

- It is NP-complete to determine whether a given graph has a strict confluent drawing. To prove this, they reduce from planar 3-SAT and they use the fact that K_4 has exactly two strict confluent drawings.
- For a given graph, with a given cyclic ordering of its n vertices, there is an $\mathcal{O}(n^2)$-time algorithm to find an outer planar strict confluent drawing, if it exists: this is a drawing in a (topological) disk, with the vertices in the given order on the boundary of the disk. They define a canonical diagram which is a unique representation of all possible outer planar strict confluent drawings,


(a) Sharp angles



(b) Merge and Split

Figure 2: Left: *Sharp* (red) and *non-sharp* (green) angles at a *junction*
Right: *Junction* j is a split junction for arc a and conversely a merge junction for arc b and c

where some of the faces are marked as cliques. They compute a canonical diagram by first computing its junctions, and then computing the arcs between them. From a canonical diagram one can easily derive an outer planar strict confluent drawing.

- When a graph has an outer planar strict confluent drawing, an algorithm based on circle packing can construct a layout of the drawing in which every path is drawn using at most two circular arcs

2 Preliminaries

A graph $G = (V, E)$ consist of a set V of *vertices* and a set E of *edges*. *Edges* have elemnts of V as endpoints and represent adjacencies between *vertices*.

A *confluent drawing* $D = (N, J, \Gamma)$ consists of a set N of *nodes*, a set J of *junctions* and a set Γ of *arcs* whose endpoints are element of $J \cup N$. *Arcs* cannot intersect on another and can only meet at endpoints (*nodes* or *junctions*). *Arcs* meeting in *junctions* must have the same tangent. At every *junction*, *arcs* from the same direction merge into one *arc*, while the *arc* from the other direction splits. This can be seen in Figure 2b.

A *path* p_{ab} between two *nodes* a and b is a sequence of arcs such that the starting point of the first arc is a, the endpoint of the last arc is b and for two arcs i and $i + 1$, which are consecutive in the sequence, it holds that the endpoint of i equals the startpoint of $i + 1$. Since meeting *arcs* in junctions have the same tangent, the sequence is a *smooth path* from a to b and directly represents the corresponding *edge* in G between the corresponding *vertices* v_a and and v_b. A *path* consisting of exactly one *arc* with both enpoints $a, b \in N$ is called *direct*.

The angle between two *arcs* which meet at a *junction* is called *non-sharp* if the two arcs have the same tangent (which means the form a *smooth path*) otherwise the angle is called *sharp*. The angle bewteen two *arcs* which meet at a *node* is always *sharp*. Every junction has exactly two *non-sharp* angles. Figure 2a is an example of *sharp* and *non-sharp* edges at junctions.

Let j be a junction. Then the two non-sharp angles at j separate the incident arcs into two disjoint sets A and B. Let a be an arc in A and $|B| > 1$. Then we say that j is a split junction for a since the path entering j via a splits at j into two or more paths. Conversely, if $|A| > 1$, we say that j is a merge junction for a since at least one more path merges with a at j. Figure 2b shows the two cases.

Eppstein et al. showed the following two Lemmas for which we will provide proof sketches.

Lemma 2.1. *Let $G = (V, E)$ be a graph, and let $E_0 \subseteq E$ be the edges of E that are incident to at least one vertex of degree 2. If G has a strict confluent drawing D, then it also has a strict confluent drawing D_0 in which all arcs in E_0 are direct.*

The idea is that every *node* of degree 2 in D can ever only have two *smooth paths* to other *nodes*. If those *smooth paths* are entirely seperate we can achieve this through a transformation similar to

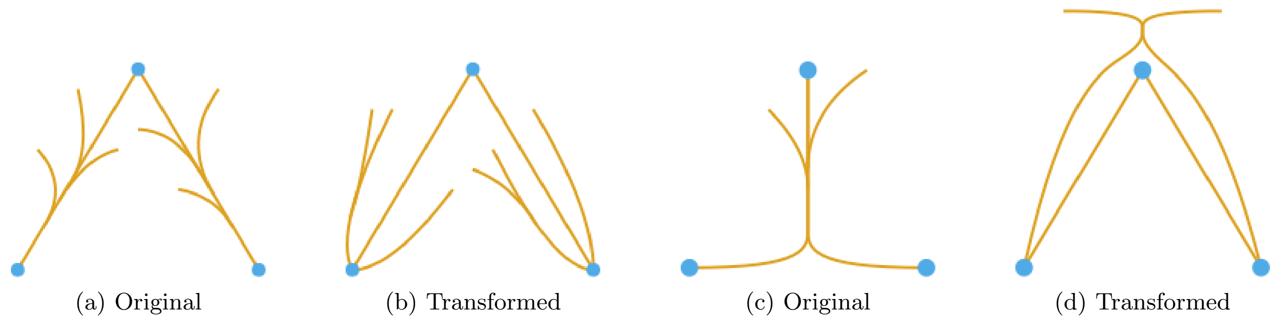


Figure 3: Transformation for two seperate paths ((a) to (b)) and two connected paths ((c) to (d))

what is depicted in Figure 3a and Figure 3b otherwise we can employ a transformation as depicted in Figure 3c and Figure 3d.

Lemma 2.2. *Let G be a graph. If G has no $K_{2,2}$ as a subgraph, whose vertices all have degrees ≥ 3 in G, then G has a strict confluent drawing if and only if G is planar.*

Since every planar drawing is also a strict confluent drawing, that implication is obvious. In the other direction the idea of the proof is to first show that an absence of a $K_{2,2}$ as a subgraph, whose vertices all have degrees ≥ 3 implies that every *smooth path* from one *node* to another can only visit a (possibly empty) sequence of *split junctions*, where an *arc* splits away from the *smooth path*, and after that a (also possibly empty) sequence of merge junctions, where another *arc* merges into the *smooth path*. Such a *path* can however be transformed into a *direct arc* by applying the transformation seen in Figure 3a and 3b from both ends of the *path*.

3 Combinatorial Complexity

This section considers strict confluent drawings with binary junctions. Let G be a graph with a given confluent drawing. Note that it's always possible to transform this drawing into a drawing where every junction has degree three and every vertex has degree one (that is one adjacent arc, not one adjacency to another vertex). This transformation maintains the number of faces and can only increase the number of arcs and junctions. Consider the following two points.

Lemma 3.1. *A strict confluent drawing D cannot contain any smooth closed curves (circles).*

Proof. this would lead to either multiple or irrelevant paths, since either:

- only one *arc* is connected to the circle, in which case the circle is not part of a *path* from one *node* to another one and is therefore irrelevant in this drawing, or
- more than one *arc* is connected to the circle, in which case some part of the circle is part of a *path* from one *node* to another. On this *path* we can always decide not to leave the circle at the *junction* where the *path* leaves the circle and instead go a full round around the circle. At the end of that round we will end up at the same *junction* again and we can still reach the end *node* of the path, which implies the existence of non-unique *paths*.

□

Lemma 3.2. *Every face in a strict confluent drawing D must have at least three sharp angles.*

This follows the following intuition. A face f without a sharp angle would be a closed loop, which is forbidden according to Lemma 3.1. If f has exactly one sharp angle, we can enter the circle on the same *path* on which we entered which results in a self loop. And if f has exactly 2 sharp angles we can complete any *path* that enters at the *junction* which forms the first sharp angle and leaves at the

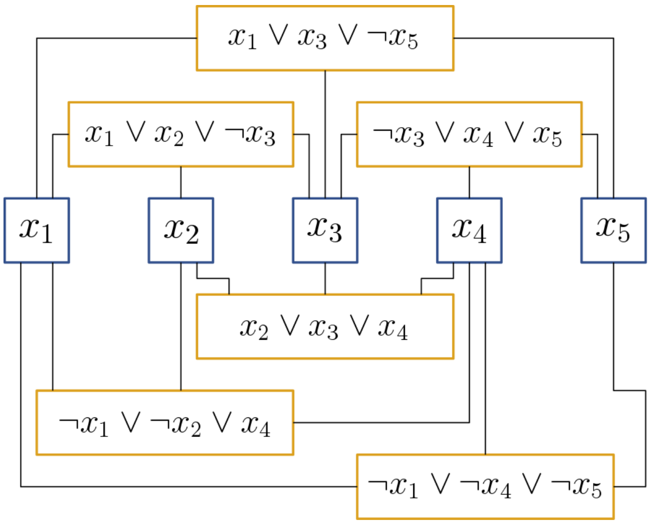

$x_1 \vee x_3 \vee \neg x_5$
$x_1 \vee x_2 \vee \neg x_3$
$\neg x_3 \vee x_4 \vee x_5$
x_1
x_2
x_3
x_4
x_5
$x_2 \vee x_3 \vee x_4$
$\neg x_1 \vee \neg x_2 \vee x_4$
$\neg x_1 \vee \neg x_4 \vee \neg x_5$

Figure 4: Instance of planar 3-SAT

junction which froms the second sharp angle by going either through one half of the boundary of f or through the other. Both of those halfs must be smooth *paths* since f only has two sharp angles. This would lead to non-unique *paths* between two *nodes*.

In the following let n be the number of *nodes*, k be the number of *junctions*, m be the number of *arcs*, F be the number of *faces*.

Using these results Eppstein et al. [7] show that F, k, m are in $\mathcal{O}(n)$ and give concrete bounds for the general case as well as for outerplanar strict confluent drawings which are strict confluent drawings contained in a disk, where the *nodes* are placed on the boundary of that disk.

4 Computational Complexity

This is the main part of this paper. It is NP-Complete to decide whether a graph G has a strict confluent drawing in which all edges incident to a vertex with degree 2 are direct, which by 2.1 is enough to show that it is also complete to decide whether G has any strict confluent drawing. This will be shown by reduction from planar 3-SAT [15].

Remark: It is important to emphasize that (similar to section 2) we aim to explain the proof, presented by Eppstein et al. [7] in our own terms. The goal is to provide another approach to an interesting and well thought out proof, in order to further understanding, not only of the proof but the general procedure of it, since this kind of proof is often encountered in the field of algorithmic geometry.

NP-Hardness An example of an instance of planar 3-SAT can be seen in Figure 4. The variables are represented by vertices and arranged on a spine in the middle of the graph. The clauses are also represented by vertices. Every clause is connected to all vertices that correspond to the variables that appear in the clause (positive or negated). The resulting graph has a planar embedding.

For the reduction we create a so called subdivided grid graph (a graph with an additional vertex on every edge). This additional vertex ensures that every *arc* in this graph is adjacent to one *node* with degree 2. We create such a grid graph for every variable of the original planar 3-SAT formula. These different grid graphs can be visualized in a rectangular fashion, as seen in Figure 5.

It is helpful to imagine the grid graphs starting at the points on the spine of the 3-SAT visualization and then extruding along the edges until they reach a clause vertex where they end up in a situation like the one depicted in Figure 6a. At this point it is important to mention that the outer boundary of the grid graphs is alternatingly colored red and green, however a change of color only occurs at a *node* of degree 3 or higher. The subdivision *nodes* lie therefore always on a uniformly colored *arcs*. At the

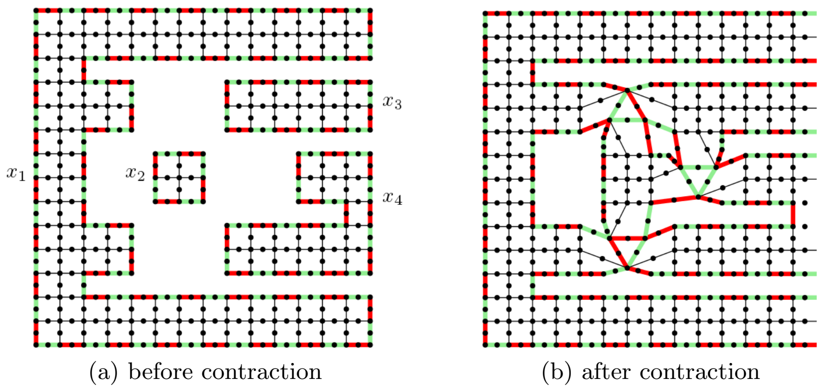


Figure 5: Example of a grid graph

created meeting points of the grid graphs (each of the three meeting grid graphs corresponds to an occurring variable in the clause we try to model) we select an *arc*, according to the occurrence of the variable in the clause - positive or negative occurrence corresponds to a green or red *arc*, respectively (see Figure 6b). Then we contract these *arcs* like in Figure 6c so that the three selected *arcs* build a triangle after contraction. The result is the contracted grid graph of the *frame* F of the construction. All *arcs* of F are adjacent to a degree-2 vertex and F has only one planar embedding (up to choice of the outer face).

The next step is the insertion of K_4's into the single cells of the Frame F. It should be paid attention to the fact that the cells are bounded by an 8-cycle which means that the *nodes* which are used to insert the K_4's are not the subdivision nodes. Note that a K_4 has only two different strict confluent drawings if all four *nodes* are drawn on the outside face. We can either construct a horizontal or a vertical *junction* in order to connect the diagonally separated *nodes*. After insertion in the frame, which then should be transformed into a strict confluent drawing, we can choose one of the two versions. However the neighboring cells are directly effected by this choice and have only one version left in order to maintain all adjacencies, as well as not creating multiple *paths* between *nodes*. This means that for a sufficiently large grid graph (at least 2 by 2 cells) the strict confluent drawing for the whole grid graph with inserted K_4's is determined by the choice for the first K_4 and there are only two different such drawings. Both can be seen in Figure 7.

We will use these two different embeddings as an indicator if the corresponding variable for this grid graph is set to true or to false in a given variable assignment of the original 3-SAT problem. Note that it can be easily checked which version of the embedding was choosing since both embeddings have either all outside *arcs* over a red (variable is false) or over a green *arc* (variable is true). These *arcs* could also be drawn inside the grid cells, the *arcs* over the opposite color (e.g. in the "true" assignment the *arcs* over the red edge) can not be chosen since they are needed on the inside to correctly model adjacencies.

We will use the fact that only *arcs* over *arcs* with the same color can be drawn outside to our advantage in the following way. We insert a certain graph (which can be seen in Figure 8). This graph has the important property of not having a confluent drawing, which has all three of the *arcs*, that connect the outer corners of the triangle, on the inside of the triangle. That means that at least one (possibly more) *arc* is drawn on the inside of the triangle in every possible strict confluent drawing. We will use this property to make a connection between variables which make this clause true and *arcs* drawn on the inside of the triangle.

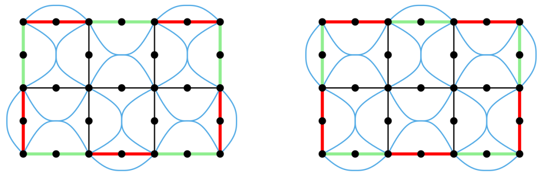
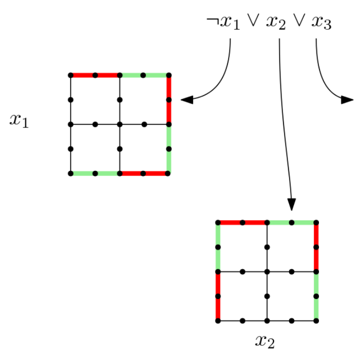
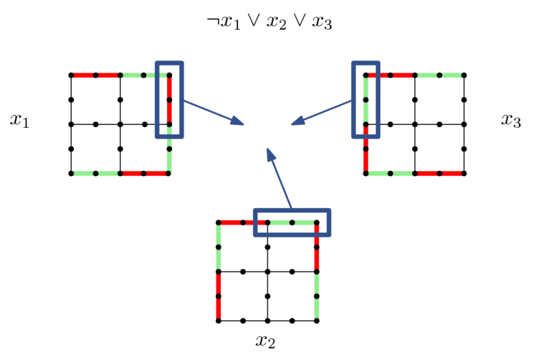


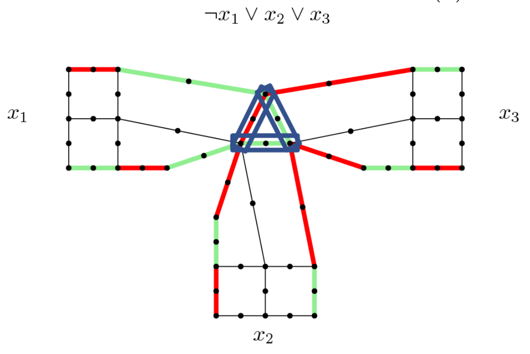
Figure 7: Two possibilities of making a grid graph with inserted K_4's strict confluent



(a) subsection



(b) corresponding edges



(c) connection of grid graphs

Figure 6: Grid graphs are connected corresponding to the occurrences of variables in a given clause

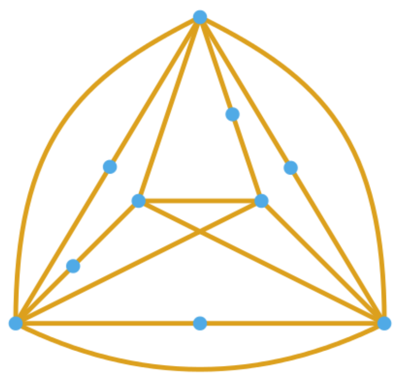


Figure 8: Clause graph

Lemma 4.1. *There is no strict confluent drawing of the clause graph in which all three triangle arcs are drawn outside. Moreover, there is a strict confluent drawing of the clause graph with two of these arcs outside, for every pair.*

The proof idea is that, by using Lemma 2.1 and Lemma 2.2, we can use the fact that the clause graph without the triangle *arcs* contains no $K_{2,2}$ with higher-degree vertices, but a K_5 as a subgraph, which is not planar and therefore the whole graph has no strict confluent drawing. The second part is illustrated in Figures 9a, 9b and 9c.

Note that the two *arcs* that are drawn outside in every drawing can easily be drawn on the inside without creating any crossings or violating any constraints, so this result in fact means, that one, two or all three of these *arcs* can be drawn inside, but always at least one.

This concludes the construction of the graph out of an instance of planar 3-SAT. What is left to show is that this created graph has a strict confluent drawing if and only if the planar 3-SAT formula from which it was created is satisfiable.

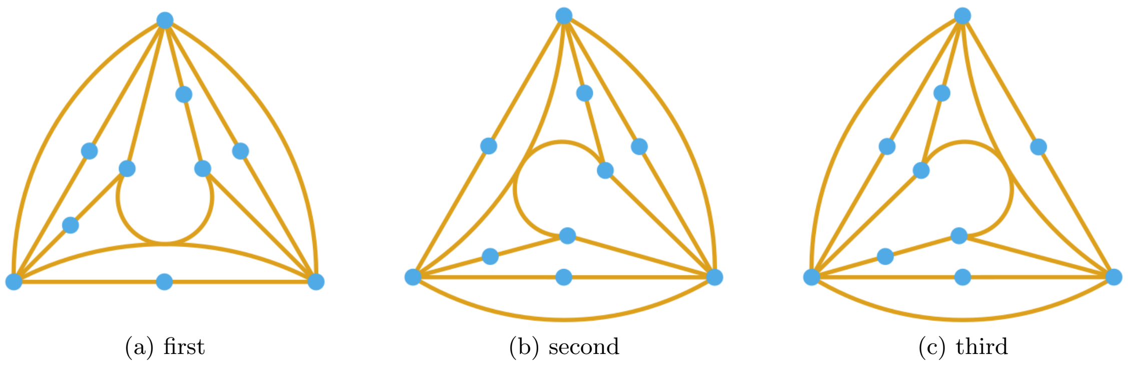


Figure 9: The three only possible confluent drawings of the clause graph depicted in Figure 8 embeddings

- If we have a satisfying variable assignment, the resulting graph has a strict confluent drawing. This can be shown by choosing the embedding option of the K_4's for every grid according to the truth value of the corresponding variable in the satisfying variable assignment If a variable x_i is set to *true*, we choose the embedding which enables the outer *arcs* to be drawn (exclusively) over the green *arcs* and we do the same for $false$ and the red *arcs*. Since every clause was modelled in such a way that the outer triangles correspond to the literals and their normal or negated occurrence in the clause, and we know that at least one variable in a positive literal is *true*, or one variable in a negative literal is $false$, we know that at least one *arc* can be drawn inside of the clause graph. According to Lemma 4.1 this means that the graph has a strict confluent drawing.
- If we have a strict confluent drawing, the planar 3-SAT has a satisfying variable assignment We use the same connection between *arcs* being drawn inside of clause graphs and the corresponding truth assignment of that variable making the corresponding literal and therefore those clauses true. Since in a strict confluent drawing of the constructed graph every clause graph must have at least one *arc* drawn on the inside (again by Lemma 4.1) we can choose the variable assignment according to the embedding of the K_4's. This variable assignment must fulfil the planar 3-SAT formula.

NP-Completeness To show that testing strict confluence is in NP, recall that the combinatorial complexity of the drawing is linear in the number of *nodes* [7]. Thus the existence of a drawing can be verified by guessing its combinatorial structure and verifying that it is planar and a drawing of the correct graph.

Theorem 4.2. *Deciding whether a graph has a strict confluent drawing is NP-complete.*

5 Algorithm

Here we will present a high-level Overview of the before mentioned algorithm, that runs in $\mathcal{O}(n^2)$ time and $\mathcal{O}(n^2)$ space and recognizes if a graph has a outer-planar strict confluent drawing. The general procedure is as follows. All funnels in the input are identified. From that a list of all junctions of the canonical diagram is created. These junctions are then connected into a planar drawing which is a subset of the canonical drawing. For all faces of the graph, which was created in the step before, all paths which would need to cross this face are identified. This information is represented in a graph for each face. With these graphs the algorithm decides which connections can be drawn directly and which faces need to be marked. The result of this step is the canonical diagram, which will be explained in the next paragraph. Next it is checked if the result of the previous step aren't in conflict with the input and finally the result is transformed into a strict confluent drawing. During this algorithm at

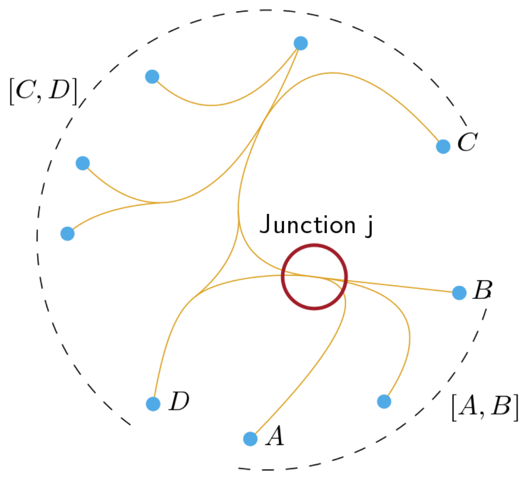


Figure 10: Funnel intervals of junction j

several points, we control if the graph exceeds bounds and, if so, abort the construction algorithm, answering that there is no outer-planar strict confluent drawing.

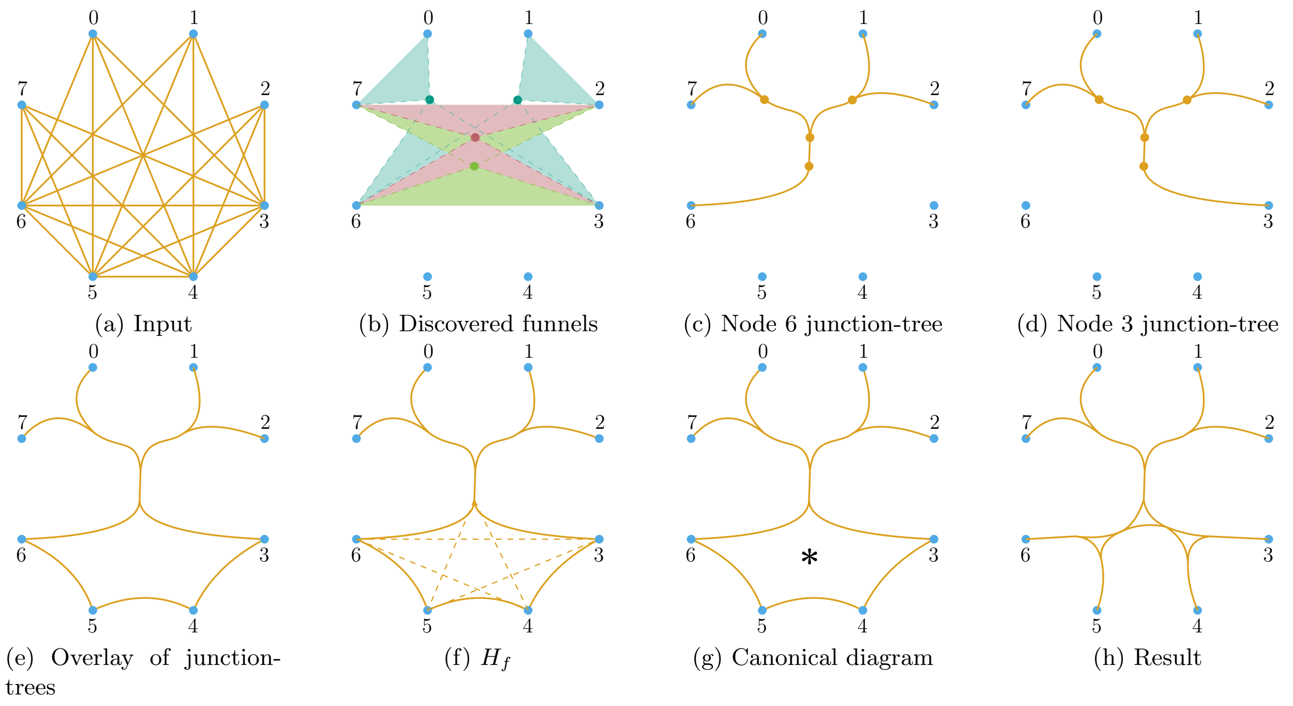
Canonical Diagrams Eppstein et al. [7] also present an algorithm that recognizes if a graph has an outer planar strict confluent drawing. The algorithm makes use of an intermediate data structure called "canonical diagram". A canonical diagram serves as a representation of all possible outer planar strict confluent drawings. A given graph G might have multiple strict confluent drawings, as we have already seen earlier, but a canonical diagram has the property of being unique, if it exists. Furthermore a canonical diagram exists if and only if the corresponding graph has a strict confluent drawing [7].

Canonical diagrams consist (just as confluent drawings) of vertices, which are connected through a system of arcs and junctions. Adjacencies are again represented through the existence of a smooth curve from one vertex to another. However there is an additional possibility to model adjacency in a canonical diagram. A face f can be marked (in our case depicted with a star) which makes it possible to cross this face from sharp angle of an adjacent junction to another one, as if every junction on the boundary of f was connected by an arc.

Also the following restrictions apply to canonical diagrams.

- Every arc is part of at least one smooth path from one vertex to another (which is allowed to cross marked faces from sharp angle to sharp angle).
- Any two smooth paths between the same two vertices must follow the same sequence of arcs and faces.
- Each marked face must have at least four angles, and all its angles must be sharp.
- Each arc must have either sharp angles or vertices at both of its ends.
- For each junction j with exactly two arcs in each direction, let f and f_0 be the two faces with sharp angles at j. Then it is not allowed for f and f_0 to both be either marked or a triangle (a face with three angles, all sharp).

And finally we need to define the funnel of a junction j of a canonical diagram D. A funnel is a 4-tuple of vertices a, b, c, and d where a and b are the vertex reached by a path that leaves j in one direction and continues as far clockwise/counterclockwise as possible, and c and d are the most clockwise/counterclockwise vertices reachable in the other direction. A funnel is depicted in Figure 10. Note that none of the paths from j to a, b, c, and d can intersect each other without contradicting the uniqueness of trails. We call the circular intervals of vertices $[a, b]$ and $[c, d]$ (in the counterclockwise direction) the funnel intervals of the respective funnel.



(a) Input (b) Discovered funnels (c) Node 6 junction-tree (d) Node 3 junction-tree (e) Overlay of junction-trees (f) H_f (g) Canonical diagram (h) Result

Figure 11: Visualization of the different steps of the algorithm

Algorithm We will in the following look at some detailed point however, this overview will not be exhaustive of the whole algorithm. The input graph can be seen in Figure 11a.

- First the algorithm iterates over all pairs of *nodes*, sorted by size. For every interval a set of conditions is checked to verify if they form a *funnel* with two other *nodes*. If they do, the interval is marked by the corresponding *junction*. Four such discovered *funnels* are depicted in Figure 11b. The coloured areas represent (counter-clockwise) the *funnelintervals*. If the number of *funnels* found in this step exceeds the linear bound described by [7], the algorithm is aborted.

- Next for every *node* n we find all junctions whose funnel intervals include the n. all of these *junctions* must be connected to n. These connections can be made for every *junction* which results in a tree (one per *node*). Two such trees can be seen in Figures 11c and 11d. Note that all orange *nodes* are drawn only for visual aid to represent junctions and are not part of the algorithm.

- By overlaying all those trees for all *nodes* we get a diagram which contains all *nodes*, all *junctions* and a subset of the arcs of the canonical diagram. This can be seen in Figure 11e. If this result is not planar, the algorithm is aborted.

- For every face f of the constructed diagram we create an auxiliary graph H_f. The *junctions* on the boundary of f are represented by vertices in H_f. The algorithm checks all pairs of *junctions* on the boundary of f if there exists a *path* which passes through both *junctions* of the pair. If so, the corresponding vertices in H_f are connected by an edge. A face f and its corresponding H_f can be seen in Figure 11f

- Next all uncrossed edges in H_f can be added to the diagram. Then all faces with crossing edges in H_f are marked as defined in the paragraph *Canonical Diagrams*. The result is the Canonical diagram and can be seen in Figure 11g. If this diagram contains multiple paths between two *nodes* or if the false adjacencies exist, the algorithm is aborted.

- The last step is to transform the canonical diagram into a strict confluent drawing. For every marked face we can number the edges in a clock-wise fashion. Then we repeatedly contract edges with an associated index i and j, such that $j = i + 2$. The result of these repeated contractions is the final strict confluent drawing and can be seen in Figure 11h.

Theorem 5.1 summarizes the properties of this algorithm.

Theorem 5.1. *For a given n-vertex graph G, and a given circular ordering of its vertices, it is possible to determine whether G has an outer-planar strict confluent drawing with the given vertex ordering, and if so to construct one, in time $\mathcal{O}(n^2)$.*

6 Conclusion

Eppstein et al. [7] have shown that, in confluent drawing, restricting attention to strict drawings allows us to characterize their combinatorial complexity. The recognition problem, whether a graph admits a strict confluent drawing has been shown to be NP-complete, but they have also shown that outer-planar strict confluent drawings with a fixed vertex ordering may be constructed in polynomial time.

There are multiple interesting open research questions. Perhaps the mos pressing problem [7] is to recognize the graphs that have an outer-planar strict confluent drawing without having a set order of the vertices and in what time they are recognizable. Which restrictions, other than outer planarity can we impose on strict confluent drawings such that they are still recognizable in polynomial time. Other question which have been mentioned are:

- Is every strict outer-planar confluent graph an alternation/circle polygon graph?
- Are graphs with rankwidth [16] 2, 3,. . . a subclass of outer-planar strict confluent?
- Has every permutation graph an outer-planar confluent drawing?

However, the overall most general question is, whether a given graph has a (not necessarily strict) confluent drawing, which is still unknown.

References

[1] Towards unambiguous edge bundling: Investigating confluent drawings for network visualization. In *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 2016. to appear.

[2] Weiwei Cui, Hong Zhou, Huamin Qu, Pak Chung Wong, and Xiaoming Li. Geometry-based edge clustering for graph visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1277–1284, 2008.

[3] Matthew Dickerson, David Eppstein, Michael T Goodrich, and Jeremy Yu Meng. Confluent drawings: Visualizing non-planar diagrams in a planar way. *J. Graph Algorithms Appl.*, 9(1):31–52, 2005.

[4] Tim Dwyer, Kim Marriott, and Michael Wybrow. Integrating edge routing into force-directed layout. In *International Symposium on Graph Drawing*, pages 8–19. Springer, 2006.

[5] David Eppstein, Michael T Goodrich, and Jeremy Yu Meng. Delta-confluent drawings. In *International Symposium on Graph Drawing*, pages 165–176. Springer, 2005.

[6] David Eppstein, Michael T Goodrich, and Jeremy Yu Meng. Confluent layered drawings. *Algorithmica*, 47(4):439–452, 2007.

[7] David Eppstein, Danny Holten, Maarten Löffler, Martin Nöllenburg, Bettina Speckmann, and Kevin Verbeek. Strict confluent drawing. In *Graph Drawing*, volume 8242, pages 352–363, 2013.

[8] Michael Hirsch, Henk Meijer, and David Rappaport. Biclique edge cover graphs and confluent drawings. In *International Symposium on Graph Drawing*, pages 405–416. Springer, 2006.

[9] Danny Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on visualization and computer graphics*, 12(5):741–748, 2006.

[10] Danny Holten and Jarke J Van Wijk. Force-directed edge bundling for graph visualization. In *Computer graphics forum*, volume 28, pages 983–990. Wiley Online Library, 2009.

[11] Peter Hui, Michael J Pelsmajer, Marcus Schaefer, and Daniel Stefankovic. Train tracks and confluent drawings. *Algorithmica*, 47(4):465–479, 2007.

[12] Christophe Hurter, Ozan Ersoy, and Alexandru Telea. Graph bundling by kernel density estimation. In *Computer Graphics Forum*, volume 31, pages 865–874. Wiley Online Library, 2012.

[13] TJ Jankun-Kelly, Tim Dwyer, Danny Holten, Christophe Hurter, Martin Nöllenburg, Chris Weaver, and Kai Xu. Scalability considerations for multivariate graph visualization. In *Multivariate Network Visualization*, pages 207–235. Springer, 2014.

[14] Stephen G Kobourov, Martin Nöllenburg, and Monique Teillaud. Drawing graphs and maps with curves (dagstuhl seminar 13151). In *Dagstuhl Reports*, volume 3. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.

[15] David Lichtenstein. Planar formulae and their uses. *SIAM journal on computing*, 11(2):329–343, 1982.

[16] Sang-il Oum and Paul Seymour. Approximating clique-width and branch-width. *Journal of Combinatorial Theory, Series B*, 96(4):514–528, 2006.

[17] Helen Purchase. Which aesthetic has the greatest effect on human understanding? In *International Symposium on Graph Drawing*, pages 248–261. Springer, 1997.

[18] Gianluca Quercini and Massimo Ancona. Confluent drawing algorithms using rectangular dualization. In *Graph Drawing*, volume 6502, pages 341–352. Springer, 2010.