Information Processing Letters 74 (2000) 7–13

Information Processing Letters

www.elsevier.com/locate/ipl

A force-directed algorithm that preserves edge-crossing properties

François Bertault [1]

Department of Computer Science and Software Engineering, University of Newcastle, Callaghan, 2308 NSW, Australia

Received 25 October 1999; received in revised form 29 January 2000
Communicated by K. Iwama

Abstract

We present an iterative drawing algorithm for undirected graphs, based on a force-directed approach, that preserves edge-crossing properties. This algorithm ensures that two edges cross in the final drawing if and only if these edges crossed in the initial layout. Therefore no new edge crossings are introduced. We describe applications of this technique to improve classical algorithms for drawing planar graphs and for dynamic graph drawing. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Algorithms; Graph drawing; Force-directed method

1. Introduction

Force-directed algorithms are commonly used in graph drawing because they are easy to implement and often achieve good results. These methods use an analogy with a physical model and are furthermore easy to understand. For example, in [4], the physical model is defined by representing nodes by magnetic rings and edges by springs between the rings. In this force model, any two rings repel each other, and two rings linked by a spring are attracted until the natural length of the spring is achieved. The equilibrium state of the physical system gives the position of the nodes in the drawing. Many other force-directed algorithms have been proposed, using different force models or convergence methods. Usually, the force model is chosen to obtain uniform edge lengths and show symmetries in the graph [4–6]. However, these algorithms can introduce a lot of edge crossings which reduce the readability of the drawing. Algorithms with more complicated force models that try to reduce these edge crossings have also been proposed. These algorithms use genetic methods or simulated annealing. However they are slow, difficult to parameterize, and do not ensure that the resulting drawing is planar when the graph is planar.

The best classical algorithms for drawing straight-line planar graphs require the graph to be biconnected [1,7,9]. For drawing general planar graphs, the first step is to modify the graph by using augmentation techniques [8], that is, by adding nodes and edges to the graph. The initial drawing in Fig. 1 shows an example of a planar graph, drawn using a planar graph and a basic augmentation technique. Note the lack of symmetry and variation in edge length. The basis of the approach presented in this paper is to combine the useful characteristics of both classical planar graphs and force-directed algorithms. The approach proceeds by first finding a planar drawing using classical algorithms and then iteratively applying a new force-

[1] Email: francois@cs.newcastle.edu.au.

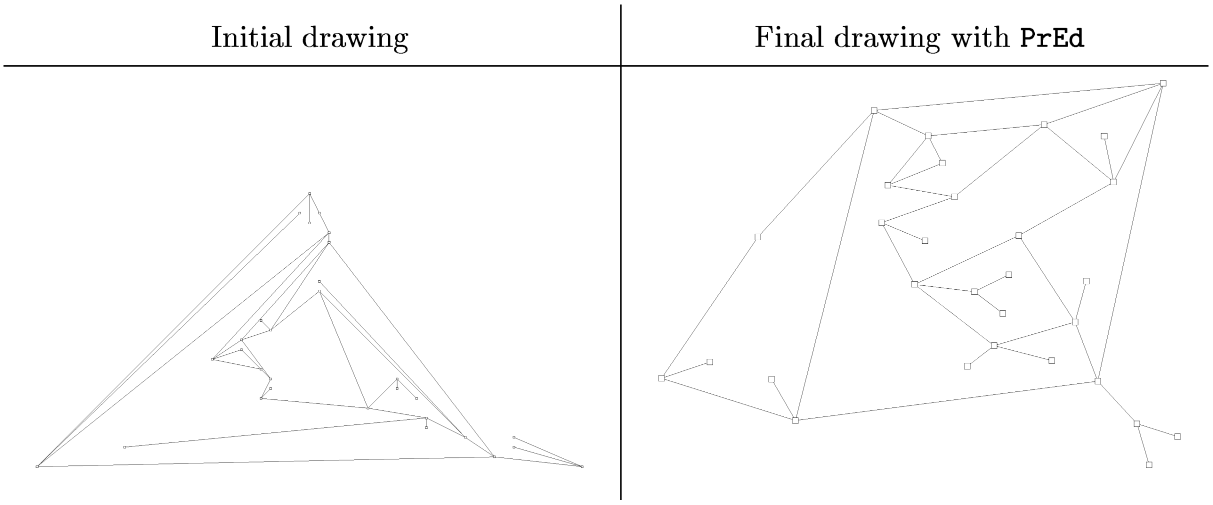

Fig. 1. The initial planar graph on the left is obtained using the LEDA planar graphs drawing algorithm. The drawing on the right is obtained by applying the `PrEd` algorithm to this initial layout. Note that the planar embedding remains the same.

directed algorithm, called `PrEd`. The final drawing improves symmetry and uniformity of edge lengths, while preserving initial edge-crossing properties of the graph. This ensures that no new edge crossings are introduced. The final drawing in Fig. 1 shows an example of a planar graph drawn using this method. Note that the planar embedding remains the same.

The `PrEd` algorithm is not limited to planar graphs. The intrinsic property of the algorithm is that two edges cross in the final drawing if and only if these edges crossed in the initial layout. This property is well suited to dynamic graph drawing. A user can easily indicate the desired relative position between nodes and edges by simply moving nodes interactively. The `PrEd` algorithm will then improve the graph symmetry and produce more uniform edge lengths.

2. `PrEd` algorithm

We consider a graph $G = (V, E)$, where V is a set of nodes, and E a set of undirected edges $(a, b) = (b, a) \in V \times V$. The number of nodes is denoted $|V|$ and the number of edges $|E|$. The `PrEd` algorithm works as follows. At each iteration, for each node v of the graph, a force $F(v)$ is computed, depending on the positions of the nodes and edges of the graph. Each node is then moved in the direction of $F(v)$. The main difference with classical force-directed algorithms is that we restrict, for each node, the maximum amplitude of the move such that the edge-crossing properties are preserved. This restriction depends on the direction of the force.

2.1. *Computation of forces*

The force model used here is very similar to force models employed in other force-directed algorithms. Three kinds of forces between nodes and edges are considered: the attraction forces between nodes linked by an edge; the repulsion forces between each pair of nodes, and the repulsion forces between nodes and edges.

The position of a node v is denoted $(x(v), y(v))$. The Euclidean distance between two nodes a and b is denoted $d(a, b)$ and δ is the edge length that we would like to obtain. The force applied to a node v is denoted $F(v) = (F_x(v), F_y(v))$. The attraction force $F^a(u, v)$ and repulsion force $F^r(u, v)$ between two nodes u and v of the graph, are defined as in [5]:

$$F_x^a(u, v) = \frac{d(u, v)}{\delta}\bigl(x(v) - x(u)\bigr),$$

$$F_x^r(u, v) = \frac{-\delta^2}{d(u, v)^2}\bigl(x(v) - x(u)\bigr).$$

The effectiveness of these force definitions was shown in [5].

For the computation of the repulsing force $F^e(v,(a,b))$ between a node v and an edge (a,b), we consider a virtual node i_v, defined by the orthogonal projection of the node v on the vector formed by the vertices of the edge. A force is applied to nodes v, a and b only if the virtual node i_v is located on the edge, and if the distance between v and i is smaller than a parameter γ. A value of $\gamma = 4\delta$ was used in the example presented in the paper. We ignore forces if $v = a$ or $v = b$.

$$F_x^e\big(v,(a,b)\big) = \begin{cases} -\dfrac{(\gamma - d(v,i_v))^2}{d(v,i_v)}(x(i_v) - x(v)) \\ \quad \text{if } i_v \in (a,b), d(v,i_v) < \gamma, \\ \qquad v \neq a, v \neq b, \\ 0 \quad \text{otherwise.} \end{cases}$$

The overall force applied to a node v is obtained by summing up the attraction and repulsion forces.

$$\begin{aligned} F_x(v) = {} & \sum_{(u,v)\in E} F_x^a(u,v) + \sum_{u\in V} F_x^r(u,v) \\ & + \sum_{(a,b)\in E} F_x^e\big(v,(a,b)\big) \\ & - \sum_{\substack{u\in V, w\in V \\ (v,w)\in E}} F_x^e\big(u,(v,w)\big). \end{aligned}$$

The $F_y(v)$ component is described by a similar formula.

2.2. Computation of the amplitude of moves

The node-edge repulsion force has been used in other algorithms in order to avoid overlapping between nodes and edges. However, since we consider discrete moves of the nodes during the steps of the algorithm, this does not guarantee that a node will not cross an edge and create a new crossing.

For preserving the crossing properties between edges, a *zone* $Z(v)$ is associated to each node v. $Z(v)$ indicates where the node v is allowed to move. A zone is defined by eight arcs $Z_1(v), \ldots, Z_8(v)$. The zone of a node v can be represented with only eight values $R_1(v), \ldots, R_8(v)$, corresponding to the radius of the arcs. The zone of a node is not the maximal area in which a node is allowed to move without changing crossing properties. The choice of representing zones by eight arcs is a good compromise between efficiency and liberty of move of each node.

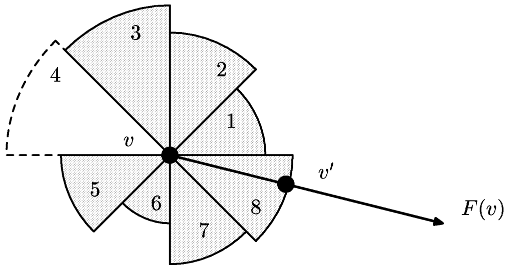

Fig. 2. Move of a node v according to its zone $Z(v)$.

The amplitude of move of a node v, with a force $F(v)$ applied to it, is bounded by the value of the arc that contains $F(v)$. For example, in Fig. 2, the node v moves in direction of the force $F(v)$, with an amplitude bounded by $R_8(v)$. An arc with an infinite radius is represented in white on the figure.

The computation of the zones is done in such a way that we avoid the creation of new crossings, while considering only one node and one edge at a time. For each pair of node v and edge (a,b), the idea is to consider two cases, depending on the position of the virtual node i_v defined earlier for the computation of repulsive forces between nodes and edges. In both cases, we restrict the move of the nodes v, a and b to avoid the creation of crossings. If the virtual node i_v is located on the edge (a,b), we also allow the node v to "escape" by increasing its distance to the edge (a,b). If the virtual node i_v is not located on the edge (a,b), we allow the node v to "turn around" the edge (a,b).

To compute the zones of each node of the graph, the eight values of each zone are first initialized to infinity. Then, for each pair of node v and edge (a,b), we consider the position of the virtual node i_v:

Case 1. If i_v is on the edge (a,b) (Fig. 3). We consider the segment $[v,i_v]$, and we determine which arc s of $Z(v)$ intersects the segment. We update the values of the zones as follows:

$$\begin{aligned} & R_j(v) = \min\big(R_j(v), d(v,i_v)/3\big), \\ & \quad j = r(s-2), \ldots, r(s+2), \\ & R_j(a) = \min\big(R_j(a), d(v,i_v)/3\big), \\ & \quad j = r(s+2), \ldots, r(s+6), \end{aligned}$$

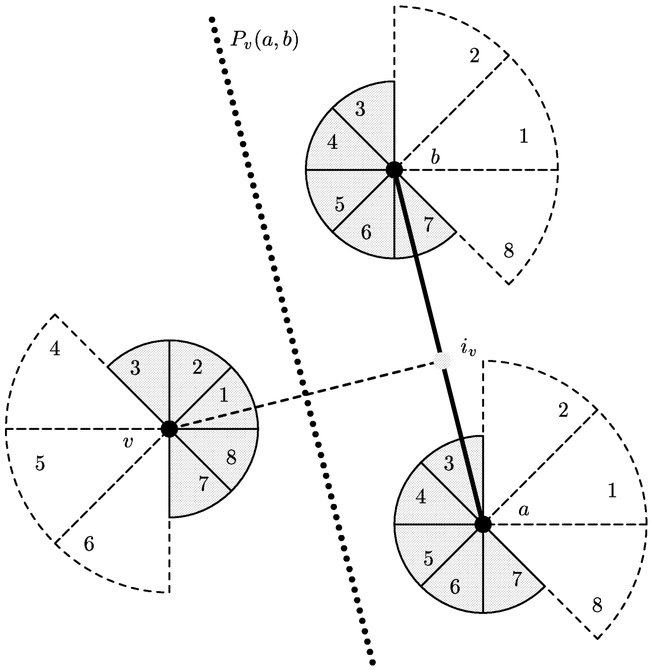

Fig. 3. Case 1: Computation of the zones for a node v and an edge (a,b). Note that the line $P_v(a,b)$ is used only in the proof of the correctness of the algorithm.

$$R_j(b) = \min\bigl(R_j(b), d(v, i_v)/3\bigr),$$
$$j = r(s+2), \ldots, r(s+6),$$

where $r(j) = 1 + (j \bmod 8)$.

Case 2. If i_v is not on the edge (a,b) (Fig. 4). We update the values of the zones as follow:

$$R_j(v) = \min\bigl(R_j(v), \min\bigl(d(a,v), d(b,v)\bigr)/3\bigr),$$
$$j = 1, \ldots, 8,$$
$$R_j(a) = \min\bigl(R_j(a), d(a,v)/3\bigr),$$
$$j = 1, \ldots, 8,$$
$$R_j(b) = \min\bigl(R_j(b), d(b,v)/3\bigr),$$
$$j = 1, \ldots, 8.$$

In summary, the steps of the `PrEd` algorithm are, for one iteration:

Step 1. Computation of the forces applied to each node: $O(|V|^2 + |V||E|)$.
Step 2. Computation the values of the zone of each node: $O(|V||E|)$.
Step 3. Move of each node, with the amplitude of the move bounded by its zone: $O(|V|)$.

The number of iterations is to be determined by the user. More sophisticated stopping criteria could also be included without changing the structure of the algorithm (see, for example, [2]).

The overall complexity of one step is $O(|V|^2 + |V||E|)$. In practice, the algorithm works well for small or sparse graphs. For example, Fig. 1 was obtained after 100 iterations and 3 seconds. In some cases, the distance between nodes can be small.

2.3. Correctness

Theorem 1. *The* `PrEd` *algorithm preserves edge-crossing properties of a graph.*

Proof. We first show that one step of the algorithm preserves non-crossing edges. For this, we consider two non-crossing edges (u,v) and (a,b). The idea of the proof is to show that we can construct two polytopes (i.e., an intersection of half planes), $P(u,v)$ and $P(a,b)$, such that we have:

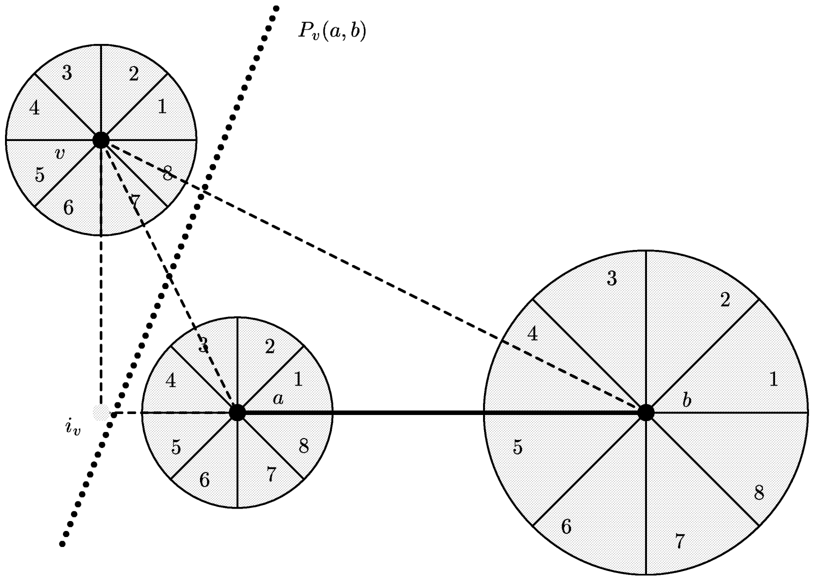

Fig. 4. Case 2: Computation of the zones for a node v and an edge (a,b).

$$P(u,v) \cap P(a,b) = \emptyset,$$
$$Z(u) \cup Z(v) \subset P(u,v), \qquad (2.1)$$
$$Z(a) \cup Z(b) \subset P(a,b).$$

Since a polytope is convex, and the ends of edges can only move inside their zone, the edges remain in their polytope after the moves of the ends.

The key point of the proof is to show, for a node v and an edge (a,b), that we can define a polytope $P_v(a,b)$ containing $Z(a)$ and $Z(b)$ and not $Z(v)$. We consider two cases, according to the position of the virtual node i_v. The other cases are deduced by renaming. The two cases are as follows:

Case 1. If $i_v \in (a,b)$. We define $P_v(a,b)$ by the half plane with a boundary orthogonal to the segment $[v,i_v]$ that passes through the middle of this segment as shown in Fig. 3.

Case 2. If $i_v \notin (a,b)$, with $d(i_v,a) < d(i_v,b)$. We define $P_v(a,b)$ by the half plane with a boundary that passes the midpoint of the segment $[a,b]$ and that forms an angle of $\pi/4$ with that segment, as shown in Fig. 4. This plane contains $Z(a)$ if the radius of $Z(a)$ is smaller than $d(a,v)/(2\sqrt{2})$. We ensure that $Z(a)$ is included in the plane by defining the radius of the zones to be $d(a,v)/3$. It is easy to verify that this half plane contains also $Z(b)$, since the radius of $Z(b)$ is given by $d(b,v)/3$.

The final step of the proof is to verify, according to the positions of the virtual nodes, that we can define the polytopes $P(u,v)$ and $P(a,b)$ that satisfy property (2.1). For clarity, we indicate explicitly the cases that can be deduced by renaming: $a \leftrightarrow b$ means that we exchange the names of the nodes a and b.

Case 1. If $i_u \in (a,b)$ and $i_v \in (a,b)$, we define $P(a,b) = P_u(a,b) \cap P_v(a,b)$, and $P(u,v) = \mathbb{R}^2 \setminus P(a,b)$.

Case 2. If $i_u \in (a,b)$ and $i_v \notin (a,b)$:

(a) If $i_a \in (u,v)$ and $i_b \in (u,v)$: $u \leftrightarrow a$ and $v \leftrightarrow b$.

(b) If $i_a \in (u,v)$, $i_b \notin (u,v)$ and $d(u,i_u) < d(a,i_a)$, we define $P(a,b) = P_u(a,b)$, and $P(u,v) = \mathbb{R}^2 \setminus P(a,b)$.

(c) If $i_a \in (u,v)$, $i_b \notin (u,v)$ and $d(u,i_u) > d(a,i_a)$: $a \leftrightarrow u$ and $b \leftrightarrow v$.

(d) If $i_a \notin (u,v)$, $i_b \in (u,v)$: $a \leftrightarrow b$.

(e) If $i_a \notin (u,v)$ and $i_b \notin (u,v)$, we define $P(a,b) = P_u(a,b)$, and $P(u,v) = \mathbb{R}^2 \setminus P(a,b)$.

Case 3. If $i_u \notin (a,b)$ and $i_v \in (a,b)$: $u \leftrightarrow v$.

Case 4. If $i_u \notin (a,b)$, $i_v \notin (a,b)$ and $i_a \in (u,v)$: $u \leftrightarrow a$ and $v \leftrightarrow b$.

Case 5. If $i_u \notin (a,b)$, $i_v \notin (a,b)$, $i_a \notin (u,v)$ and $i_b \in (u,v)$: $a \leftrightarrow v$ and $b \leftrightarrow u$.

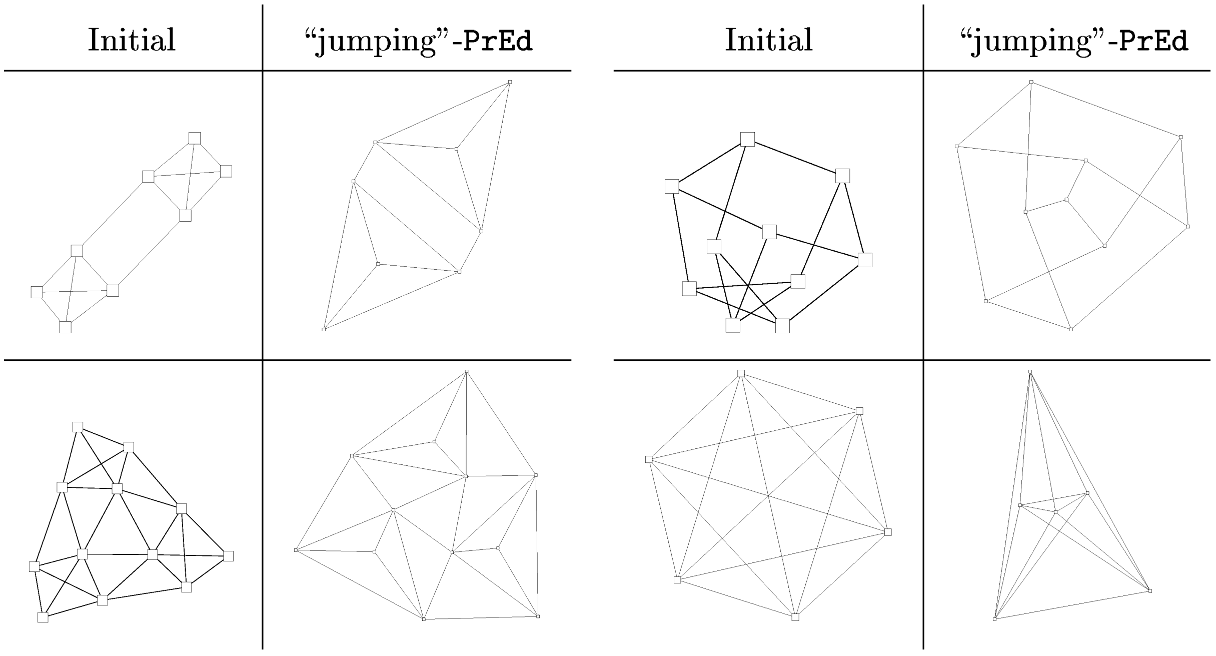

Fig. 5. Combination of a the PrEd algorithm with a "jumping" operation for reducing edge-crossings.

Case 6. If $i_u \notin (a,b)$, $i_v \notin (a,b)$, $i_a \notin (u,v)$ and $i_b \notin (u,v)$:

(a) If $d(a,u) \leqslant \min(d(a,v), d(b,u), d(b,v))$, we define $P(a,b) = P_u(a,b)$, and $P(u,v) = \mathbb{R}^2 \setminus P(a,b)$.

(b) If $d(a,v) \leqslant \min(d(a,u), d(b,u), d(b,v))$: $u \leftrightarrow v$.

(c) If $d(b,u) \leqslant \min(d(a,u), d(a,v), d(b,v))$: $a \leftrightarrow b$.

(d) If $d(b,v) \leqslant \min(d(a,u), d(a,v), d(b,u))$: $a \leftrightarrow b$ and $u \leftrightarrow v$.

In order to verify that each step of the algorithm preserves crossing edges, we use similar techniques. If two edges (u,v) and (a,b) cross, we remark that we can remove the crossing by moving the nodes only if we can obtain a crossing between the virtual edges (u,a) and (v,b), or between (u,b) and (v,a). $\square$

3. Further work

The speed of the algorithm could be greatly improved. We could, instead of considering all edges of the graph for defining the zone of a node, consider only a restricted set of edges that "surround" that node. The set of edges that we need to consider could be determine in a preprocessing step of the algorithm.

The PrEd algorithm can also be combined with a simple "jumping" heuristic, for reducing edge-crossings in general graph drawing. The initial positions of nodes are obtained using a classical force-directed algorithm. Then, for each pair of edge crossings, we test, during the iterations of the PrEd algorithm, if removing this crossing, by placing one end near the intersection, reduces the total number of crossing in the graph. This naive approach can give good results on simple graphs, but is very computationally expensive. Fig. 5 demonstrates this algorithm for removing edge crossings. Further work could be to investigate better heuristics based on this idea.

4. Conclusion

The PrEd algorithm takes $O(|V|^2 + |V||E|)$ time to complete one iteration. It preserves crossing and non-crossing edges during the iteration steps. This property is well suited for improving the drawing of straight-

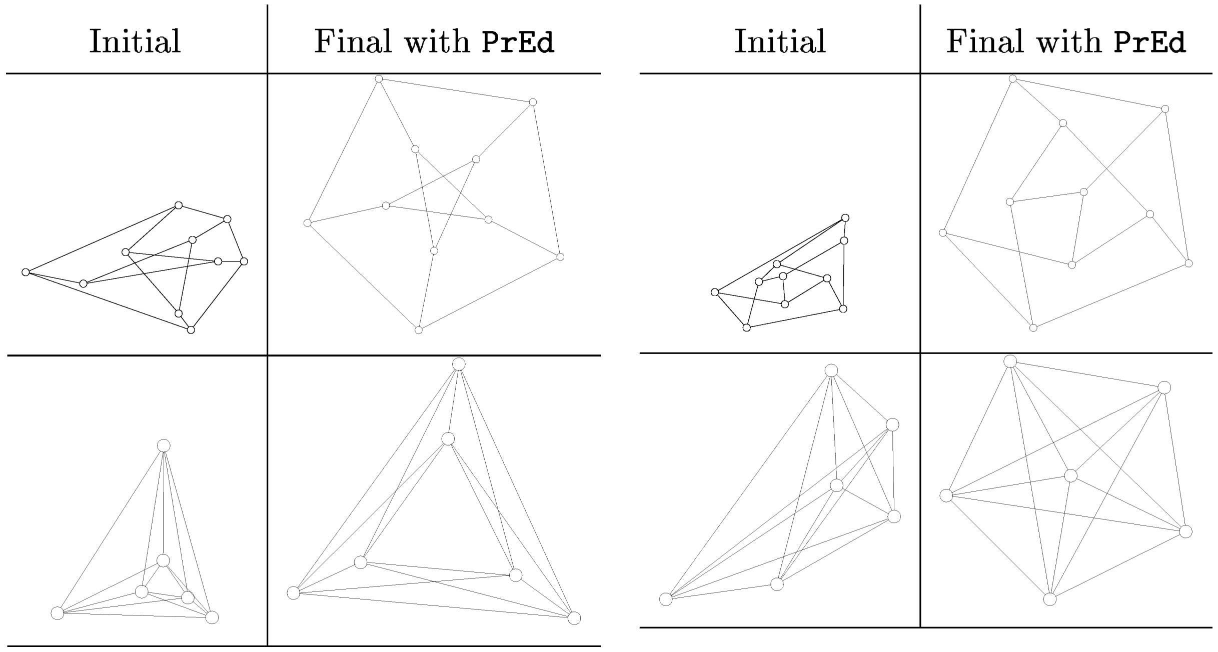

Fig. 6. Examples of dynamic drawing with the PrEd algorithm. The initial position of nodes is specified by the user, and the PrEd algorithm improves symmetry and edge length criteria without changing the relative positions of nodes.

line planar graphs, where the initial position of nodes is obtained from a classical algorithm.

The PrEd algorithm can also be applied for dynamic drawing of general graphs. The user can modify interactively the position of the nodes, and the algorithm improves the drawing criteria such as symmetry and uniform edges lengths. Relative positions of nodes are preserved by the algorithm. Small modifications of the graph induce small modifications on the new representation of the graph, that is, the mental map is preserved [3]. Examples of drawings obtained this way are shown in Fig. 6.

References

[1] H. de Fraysseix, J. Pach, R. Pollack, How to draw a planar graph on a grid, Combinatorica 10 (1990) 41–51.

[2] G. Di Battista, P.D. Eades, R. Tamassia, I.G. Tollis, Graph Drawing, Prentice-Hall, Englewood Cliffs, NJ, 1999.

[3] P. Eades, W. Lai, K. Misue, K. Sugiyama, Layout adjustment and the mental map, J. Visual Languages Comput. 6 (1995) 183–210.

[4] P.D. Eades, A heuristic for graph drawing, Congr. Numer. 42 (1984) 149–160.

[5] T. Fruchterman, E. Reingold, Graph drawing by force-directed placement, Software-Practice Exper. 21 (11) (1991) 1129–1164.

[6] T. Kamada, S. Kawai, An algorithm for drawing general undirected graphs, Inform. Process. Lett. 31 (1989) 7–15.

[7] G. Kant, Drawing planar graphs using the *lmc*-ordering, in: Proc. IEEE Symp. on Foundation of Computer Science, 1992, pp. 101–110.

[8] G. Kant, H.L. Bodlaender, Triangulating planar graphs while minimizing the maximum degree, Inform. and Comput. 135 (1) (1997) 1–14.

[9] P. Mutzel, A fast linear time embedding algorithm based on the Hopcropt–Tarjan planarity test, Technical Report, Institut für Informatik, Universität zu Köln, 1992.