ImPrEd: An Improved Force-Directed Algorithm that Prevents Nodes from Crossing Edges

Paolo Simonetto, Daniel Archambault, David Auber, Romain Bourqui

Eurographics / IEEE Symposium on Visualization 2011 (EuroVis 2011)
H. Hauser, H. Pfister, and J. J. van Wijk
(Guest Editors)

Volume 30 (2011), Number 3

ImPrEd: An Improved Force-Directed Algorithm that Prevents Nodes from Crossing Edges

Paolo Simonetto[1], Daniel Archambault[2], David Auber[1], and Romain Bourqui[1]

[1]LaBRI, Université Bordeaux 1 — INRIA, Bordeaux Sud-Ouest
[2]University College Dublin

Abstract

PrEd [Ber00] is a force-directed algorithm that improves the existing layout of a graph while preserving its edge crossing properties. The algorithm has a number of applications including: improving the layouts of planar graph drawing algorithms, interacting with a graph layout, and drawing Euler-like diagrams. The algorithm ensures that nodes do not cross edges during its execution. However, PrEd can be computationally expensive and overly-restrictive in terms of node movement.

In this paper, we introduce ImPrEd: an improved version of PrEd that overcomes some of its limitations and widens its range of applicability. ImPrEd also adds features such as flexible or crossable edges, allowing for greater control over the output. Flexible edges, in particular, can improve the distribution of graph elements and the angular resolution of the input graph. They can also be used to generate Euler diagrams with smooth boundaries. As flexible edges increase data set size, we experience an execution/drawing quality trade off. However, when flexible edges are not used, ImPrEd proves to be consistently faster than PrEd.

Categories and Subject Descriptors (according to ACM CCS): G.2.2 [Discrete Mathematics]: Graph Theory—Graph Algorithms

1. Introduction

PrEd [Ber00] is a force-directed algorithm [Ead84, FR91] that preserves the edge crossing properties of an input layout. The algorithm computes the maximal displacement of nodes and restricts their movement to guarantee this property.

The author of PrEd suggested two possible applications: improving the layout [Pur97] of planar, straight-line graph drawing algorithms [DFPP90] and driving force-directed graph layout interactively. By realising that not only does PrEd preserve all edge crossings in an existing layout but actually prevents nodes from crossing edges, we can apply this algorithm to a wider range of problems beyond those stated in the original paper. For example, PrEd can be used to improve Euler-like diagrams [SAA09] where nodes are set elements and edges are set boundaries. In general, we can use PrEd for tasks where we would like to bound portions of a layout, extending the applicability beyond "preserving edge crossing properties" [Ber00].

The contribution of this paper is ImPrEd: an improved force-directed algorithm that prevents nodes from crossing edges. ImPrEd is faster than PrEd and grants greater output control. Our test cases provide some evidence that it has improved visual quality. The modifications made, initially motivated by the problem of drawing Euler-like diagrams, can also be useful for general graph drawing.

2. Previous and Related Work

Many approaches aim to constrain or restrict node movement in force-directed algorithms. Mental map preservation techniques, described in Section 2.1, restrict node movement for dynamic graph drawing. Section 2.2 describes constraint-based techniques, which offer guarantees on properties in the output of the graph. Section 2.4 discusses planar graph drawing – an application area of PrEd. Finally, PrEd is described in Section 2.5.

2.1. Mental Map Preservation

In the field of dynamic graph drawing, a number of force-directed algorithms have included in their objective functions ways to keep similar nodes in similar areas of the plane.

Erten *et al.* [EHK*04] considered inter-timeslice edges that link the same node across timeslices to drive the positions of nodes to certain area of the plane. Weighting functions have been used to drive, but not constrain, the movement of nodes [FT04, FT08]. The PIGALE [dFOdM02] library also includes a spring embedder which preserves the planar map. Additionally, certain force-directed methods have impeded the movement of nodes via geometric restriction [SP08]. For energy-based approaches, penalising terms can be added to discourage nodes from moving too far from their previous positions [BBP08].

These approaches modify their respective algorithms to drive nodes in the layout towards certain positions. However, they do not guarantee that a node will stay in a particular region of the plane or that it will not cross edges.

2.2. Constraint-Based Graph Layout

Several approaches use hard constraints to guarantee that certain properties of an initial graph layout are not broken [DK05, DKM06, DMW09, Dwy09]. Properties such as the horizontal or vertical alignment of nodes, non-overlapping rectangular boundaries, containment of nodes within a boundary, and downward pointing edges in a directed graph can be preserved. The general technique has been shown to be applicable in diverse settings, including edge crossing properties. However, with `ImPrEd`, we aim at the specific application of node containment which may be handled more simply with a targeted algorithm.

2.3. Topology-Driven, Force-Directed Algorithms

Didimo *et al.* [DLR11] proposed a framework, combining force-directed and planarisation-based approaches for graph drawing. This hybrid algorithm aims at drawing graphs with a small number of crossings and high angular resolution. Although the algorithm shares similarities with `PrEd` and `ImPrEd`, there are significant differences between the approaches. For example, Didimo *et al.* has forces for geodesic edge tendency which `ImPrEd` does not. However, this algorithm does not preserve all crossings in the initial layout.

2.4. Planar Graph Drawing and Visualisation

Graph visualisation systems are often less interested in strict graph planarity [HMM00], since a large graph rarely admits a strict, planar embedding. However, planar graphs are often created to provide an overview of the data. Also, they have recently become of interest to the problem of drawing Euler-like diagrams [SAA09]. Current planar graph drawing algorithms [DFPP90, GM98] can produce sub-optimal results in terms of graph aesthetics, when considering angular resolution. `PrEd` has been proposed to improve the output produced by planar graph drawing algorithms, but it can have slow execution time and may produce sub-optimal results for some planar graphs.

Algorithm 1 Pseudo code for the original `PrEd` approach.

$\forall v$, calculate all F_v^r, F_v^a, and F_v^e
$\forall v$, compute $\mathcal{M}_v$ (Figure 1)
$\forall v$, move nodes using min of forces or $\mathcal{M}_v$

2.5. An Introduction to `PrEd`

Like most force-directed approaches, `PrEd` iteratively refines the layout. Each iteration has three stages. Algorithm 1 gives an overview of the approach. First, a force $\mathcal{F}_v$ is computed for every node $v \in V$: using node-node repulsion $\mathcal{F}_v^r$, edge attraction $\mathcal{F}_v^a$, and non-incident edge repulsion $\mathcal{F}_v^e$. Secondly, the algorithm computes the maximal movement permitted $\mathcal{M}_v$ for each node as shown in Figure 1. This movement is computed by dividing the directions around each node v into eight sectors and computing the shortest distance between v and the first edge that it could potentially cross. Finally, the positions of the nodes are updated according to the resultant force, ensuring the displacement is less than $\mathcal{M}_v$ in the appropriate direction.

`PrEd` has three forces: node-node repulsion, edge attraction, and node-edge repulsion. The first two forces are:

$$\mathcal{F}_v^r(u,v) \leftarrow \left(\frac{\delta}{||p_u - p_v||}\right)^2 (p_u - p_v)$$

$$\mathcal{F}_v^a(u,v) \leftarrow \left(\frac{||p_u - p_v||}{\delta}\right)^1 (p_v - p_u)$$

while the node-edge repulsive force is:

$$\mathcal{F}_v^e(v,(a,b)) \leftarrow \frac{(\gamma - ||p_v - v_e||)^2}{||p_v - v_e||}(p_v - v_e)$$
$$v_e \in (a,b), a \neq v, b \neq v, ||p_v - v_e|| < \gamma$$

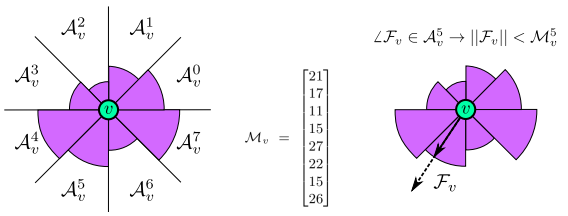
In these formulae, p_x is the position of node x, δ is the optimal distance between two nodes, and γ is the optimal distance between a node and an edge. The value of v_e is the projection of node v onto the line defined by the edge (a,b). These forces are computed for every pair of nodes, every edge, and every node-edge pair respectively.

To compute $\mathcal{M}_v$, `PrEd` divides the 2π angle around each node v into eight congruent angles $\mathcal{A}_v = (\mathcal{A}_v^0 \dots \mathcal{A}_v^7)$. The maximal movement is an array of eight values $\mathcal{M}_v^0 \dots \mathcal{M}_v^7$ corresponding to the maximal distance a node can move in the direction of any vector in the sector. Figure 1 depicts these sectors and how node movement is restricted.

3. Definitions

Before describing the algorithm, we review a few definitions and introduce some notation. A more detailed explanation can be found in graph theory and drawing textbooks [NR04].

Let $G = (V,E)$ be a simple undirected graph. An *embedding*, or planar drawing, of G is a representation of the graph



(a) (b) (c)

Figure 1: Maximal movement in PrEd. (a) The maximal movement sectors of v. (b) The corresponding $\mathcal{M}_v$, containing the radius of each sector. (c) Movement limitation during the node displacement phase. $\mathcal{F}_v$ belongs to $\mathcal{A}_v^5$, so its module will be clamped to $\mathcal{M}_v^5$.

where the nodes $v \in V$ are assigned to coordinates p_v in the plane. The edges $e = (u,v) \in E$ are lines connecting nodes at p_u and p_v and two edges can only cross at their extremities. Some authors refer to this concept as *planar graph embedding*, whereas a general embedding omits the non-crossing condition. A graph that has an embedding is called *planar graph*, and a particular embedding is a *plane graph*.

Let G be a plane graph. In its embedding, the edges of the graph divide the plane into regions called *faces*. Let Φ be the set of faces of G and $F \in \Phi$. The boundary $\mathcal{B}(F)$ of a face is the set of nodes $\mathcal{B}^n(F)$ and the set of edges $\mathcal{B}^e(F)$ that enclose F. The boundary of a face is a collection of nodes and edges for a disconnected graph, a closed walk for a connected graph, and a cycle when the graph is 2-connected. We will use the notation $\bar{F}$ when F has a connected boundary.

Two embeddings of G are *equivalent* if the boundary of each face in an embedding corresponds to the boundary of a face in the other. The *planar maps* of G are equivalence classes of the planar embeddings of G.

A *combinatorial embedding* of G is a data structure that lists in order the edges incident to the graph nodes. When G is connected, its planar maps correspond to its combinatorial embeddings. When G is not connected, we can obtain non-equivalent embeddings by swapping a connected component of G into a different face, and so the same combinatorial embedding might correspond to different planar maps.

4. Improvements to the PrEd Approach

ImPrEd is designed to improve the performance and extend the features of PrEd. This section presents the execution time improvements that affect the three steps of Algorithm 1. The surrounding edge computation reduces the number of edges checked when computing the maximal movement of a node. QuadTrees efficiently determine the nodes and edges in a region, reducing the time required for force and maximal displacement computations. New rules for maximal displacement allow nodes to move more freely, enabling faster convergence. Finally, the force system adapts, based on the iteration, leading to improved aesthetics and reliability of input parameters.

4.1. Surrounding Edges Computation

The author of PrEd observed that, in certain cases, edges can be crossed only after other edges are crossed. Thus, considering all edges in the maximal movement computation might be redundant. Consider a node v inside a face $\bar{F}$ of a connected plane graph $G = (V,E)$. By construction, the node must cross an edge of $\mathcal{B}^e(\bar{F})$ before it can cross any other edge in G. Therefore, we can safely test only the edges of $\mathcal{B}^e(\bar{F})$. Moreover, these edges are the most influential in the node-edge repulsion force $\mathcal{F}_v^e$ and can be the only edges considered without significantly affecting the final layout. These edges, the *surrounding edges* ($\mathcal{S}_v$) of the node v, will be the only edges considered in the calculation of F_v^e and $\mathcal{M}_v$ in Algorithm 1. Since the algorithm preserves layout topology, $\mathcal{S}_v$ can be computed in pre-processing step and does not need to be updated afterward.

Computing $\mathcal{S}_v$ requires us to analyse the plane graph as shown in Figure 2. For now, assume that the graph G is a plane graph, but we will later extend this result to non-plane graphs (a non-plane graph can either be a non-planar graph or a planar graph with a non-planar embedding). The surrounding edges of a node $v \in V$ can be computed as the union of all edges in the faces that contain v (see Figure 2c):

$$\mathcal{S}_v = \bigcup_{F \in \Phi_v} \mathcal{B}^e(F) \text{ where } \Phi_v = \left\{ F \in \Phi \mid v \in \mathcal{B}^n(F) \right\}$$

In a general plane graph, the boundary of F might not be connected and can depend on the embedding of the graph. We compute the connected components and the boundary of their faces $\bar{F}$. As we are dealing with connected components, we can compute their faces using their combinatorial embedding. Finally, we merge them to obtain the boundary of the face F (see Figure 2d).

We decompose the graph into its connected components and compute the faces expressed by the embedding. Each component, C_i, has a single unbounded or external face, $\bar{F}_i^{\text{ext}}$, and a set $\bar{F}_i^{\text{int}}$ of zero or more bounded or internal faces $\bar{F}_{i,j}^{\text{int}}$. Moreover, as there are no edge crossings, each connected component is contained in exactly one face.

We construct a partial ordering between components, so that the components C_i and C_j are in the relationship $C_i < C_j$ if and only if C_i is contained by an internal face of C_j and C_j is contained by the external face of C_i. The partial ordering organises the components into a hierarchy. The resulting forest has unrelated components as roots and all other components ordered by containment (see Figure 2b).

There are $1 + \sum_{\forall i} |\bar{F}_i^{\text{int}}|$ faces. Each face F is determined by an internal face $\bar{F}_{i,j}^{\text{int}}$ of a component C_i, and all the external faces $\bar{F}_k^{\text{ext}}$ of the components C_k directly contained

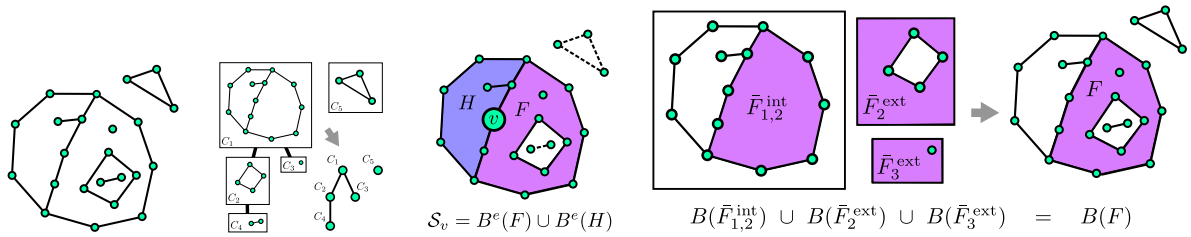


Figure 2: Surrounding edge computation. (a) An example of plane graph. (b) The graph is divided into connected components. These components are organised into a hierarchy via containment. (c) The surrounding edges $\mathcal{S}_v$ are computed as the edges of the faces in $\Phi_v = \{F, H\}$, that contain v. The edges $e \notin \mathcal{S}_v$ are dashed. (d) The disconnected boundary of the face F is computed from the connected boundaries $\bar{F}$, according to Figure 2b.

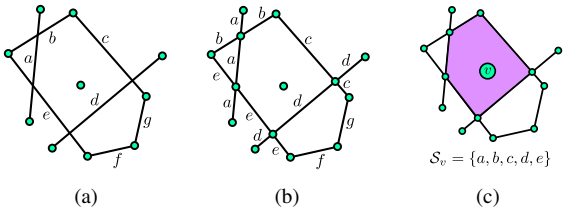


Figure 3: Surrounding edge detection for non-plane graphs. (a) A non-plane graph G. (b) Generation and labelling of $\hat{G}$. (c) The surrounding edges are computed using the labels.

in $\bar{F}_{i,j}^{\text{int}}$. The additional face of the previous total is instead determined by the external faces $\bar{F}_k^{\text{ext}}$ of the components C_k that are root of the hierarchy. The boundary of a face F can be computed as the union of $\mathcal{B}^n(\bar{F})$ and $\mathcal{B}^e(\bar{F})$ of the faces in $\bar{F}$ (see Figure 2d).

Let us assume that G is not a plane graph. To apply the above-described method, G is reduced to a plane graph, $\hat{G}$, through planar augmentation [DBETT98]. We label all nodes and edges of $\hat{G}$ using elements that generated them, and these labels are used to compute $\mathcal{S}_v$. The nodes of $\hat{G}$ that do not correspond to a node in G are not labeled and are not considered as part of the face boundary (Figure 3).

The complexity of the surrounding edges computation depends on the initial layout configuration. If G is a plane graph, each edge is considered twice. Since the number of edges and faces is linear with respect to the number of nodes, the overall complexity is $O(|V|)$. If G is not a plane graph, in worst case, we have $O(|E|^2)$ crossings. As each crossing produces at most one node and at most two segments and outputs a plane graph, the overall worst case complexity is $O(|V| + |E|^2)$. In general, the average number of surrounding edges is much smaller $|E|$. However, there are cases (for example, trees) where this worst case is realized. It is important to note, however, that this step is pre-processing and does not influence the main cycle of the algorithm.

4.2. QuadTrees

We use QuadTrees to determine with sublinear complexity the nodes $\mathcal{N}_v^d$ and edges $\mathcal{E}_v^d$ at a distance d from a node v. The data structure has been used in force-directed algorithms [QE01] as distant elements can often be approximated or ignored. In `ImPrEd`, when computing $\mathcal{F}^r$ and $\mathcal{F}^e$, we only consider nodes in $\mathcal{N}_v^{3\delta}$ and the edges $\mathcal{E}_v^{\gamma}$.

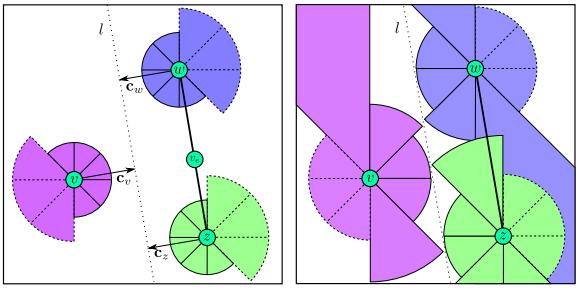
The set of nearby edges can also be used to reduce the running time of the maximal movement computation. For stability reasons, `ImPrEd` defines a global maximal distance $\bar{d}$ that no node can exceed when moving (see Section 4.4). Consider a node v and an edge e that cannot be crossed. If v and the extremities of e move closer to each other, they cross only if v and e are closer than $2\bar{d}$. We can then safely consider only the edges $\mathcal{S}_v \cap \mathcal{E}_v^{2\bar{d}}$ in the computation of $\mathcal{M}_v$.

The edge sets complement each other, overcoming many of their individual limitations. For example, when using `ImPrEd` on a tree, we have $\mathcal{S}_v = E$ as all edges are surrounding edges for every node. In this case, $\mathcal{E}_v^{2\bar{d}}$ helps alleviate the problem. Conversely, when using `ImPrEd` on a graph with an unbalanced layout, $\mathcal{S}_v$ often keeps the number of tested edges small.

4.3. New Rules for the Maximal Movement Calculation

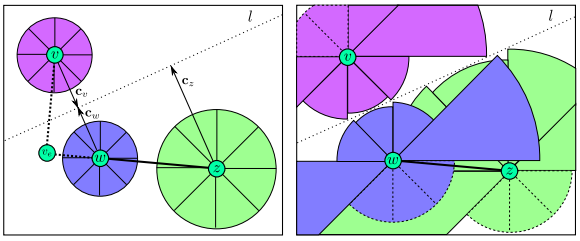
`PrEd` guarantees that no new crossings can be incurred or undone. To enforce this property, the algorithm considers each node-edge pair (v, e), $v \in V$, $e = \{w, z\}$ to ensure that v, w and z do not create new or undo crossings.

`PrEd` can overly restrict node movement, impeding algorithm convergence. Let us consider, for example, the configuration of Figure 4a. The movement of v is heavily bounded even when moving along the perpendicular to c_v,



(a) (b)

Figure 4: Maximal movement computation when $v_e \in e$. The dashed sectors are unlimited. (a) The result with PrEd's rules. (b) The result with ImPrEd's rules.



(a) (b)

Figure 5: Maximal movement computation when $v_e \notin e$. The dashed sectors are unlimited. (a) The result with PrEd's rules. (b) The result with ImPrEd's rules.

which is perfectly safe. Moreover, a node inside an acute angle or between two edges will have its movement unnecessarily constrained. By allowing nodes to escape these high-stress configurations, we are likely to obtain a stable configuration quickly, with additional positive effects on drawing quality. Therefore, we modify the node restriction rules to maximise node movement, while preserving the crossings in the initial layout. Figure 4 and 5 compare the movement sectors computed with the old and the new rules. This improvement affects the computation of $\mathcal{M}_v$ in Algorithm 1.

Each node v has an associated collision vector $\vec{c}_v$. The vector is the minimum movement of v before crossings are incurred. Let v_e be the projection of v in the line where e lies. The collision vector $\vec{c}_v$ is defined as:

$$\text{Case } v_e \in e: \qquad \vec{c}_v = \overrightarrow{vv_e}$$
$$\text{Case } v_e \notin e: \qquad \vec{c}_v = \min(\overrightarrow{vw}, \overrightarrow{vz})$$

The node v can move at most $||\vec{c}_v||$ in any direction without crossing e. However, both w and z can move as well. Consider a line l perpendicular to $\vec{c}_v$ and crossing it in its midpoint. We redefine $\vec{c}_v$ and define $\vec{c}_w$ and $\vec{c}_z$ as the vectors connecting w and z to their projections on l. Again, if nodes move less than their collision vectors, they will not cross l and therefore no crossing between e and v can occur.

To permit maximal movement, each sector can be extended until it touches, but does not become collinear with, the line l. This result can be achieved by multiplying the length of the collision vector by σ, defined as the secant of the angle between $\vec{c}$ and closest radius of the sector:

$$\mathcal{M}_x^j < \|\vec{c}_x\| \cdot \sigma(\vec{c}_x, \mathcal{A}_x^j)$$

$$\sigma(\vec{c}_x, \mathcal{A}_x^j) = \begin{cases} 1 & i-j \equiv 0 \mod 8 \\ \sec\left(\angle\vec{c}_x - (j+1)\pi/4\right) & i-j \equiv 1,2 \mod 8 \\ \sec\left(\angle\vec{c}_x - j\pi/4\right) & i-j \equiv 6,7 \mod 8 \\ \infty & i-j \equiv 3,4,5 \mod 8 \end{cases}$$

where $\angle\vec{c}_x$ is the angle of the given vector and where i is the index into $\mathcal{A}_x^i$ in which $\vec{c}_x$ lies. In the first case, the vector lies on the sector and the closest radius coincides with $\vec{c}_x$, thus $\sec(0) = 1$. The second case considers the two sectors in clockwise order from $\mathcal{A}_x^i$, where the closest radius corresponds to the sector border of $\mathcal{A}_x^j$ that has angle $(j+1)\pi/4$. The third case handles the two sectors in anticlockwise order that have the closest border with angle $j\pi/4$. The forth case handles the sectors not facing l that are unrestricted.

To prove the correctness of the algorithm, we first prove that when a node moves it cannot cross an edge. Given a node v and a non-incident edge e, the line l (see Figure 4 and 5) divides the plane into two half planes: the first, $R_v(v,e)$, contains v and the second, $R_e(v,e)$, contains $e = (w,z)$. For all other $\lambda \in E$ and λ non-incident to v, we compute the half planes that contain v using λ and v . Similarly, for all other $u \in V$ and u non-incident to e, we compute the half planes that contain e using e and u. The regions defined by the intersection of these half planes are R_v and R_e respectively and are computed for all nodes and edges in the graph. The above-described formula bounds the sectors around every v to be in R_v and every w and z to be in R_e because the nodes must remain on the same side of all defining half planes of the region. Therefore, R_v contains any future position of any node v and R_e, being convex, as it is the intersection of convex regions [DBCVKO00], fully contains any final position of the edge e. Since $R_v \subseteq R_v(v,e)$, $R_e \subseteq R_e(v,e)$, and $R_v(v,e) \cap R_e(v,e) = \emptyset$ for every node v and every non-incident edge e, no node can cross an edge when moving.

The property also guarantees that, given two edges, no crossings can be incurred or undone, as both cases require a node to cross an edge. Finally, $R_v(v,e)$ and $R_e(v,e)$ can always be computed, so long as v and e are not initially collinear, cannot become collinear with l, by definition, when moving, and we are working in a continuous space.

4.4. Force System Cooling

In PrEd, nodes repel each other with inversely quadratic forces and edges attract incident nodes with linear forces. By

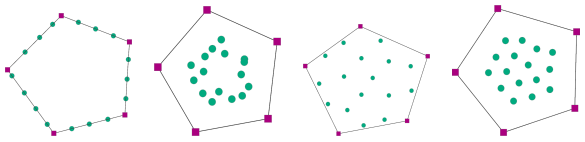


Figure 6: Stability to the input parameters of PrEd and ImPrEd. (a) PrEd with $\delta = 5$ and $\gamma = 2$. (b) PrEd with $\delta = 2$ and $\gamma = 5$. (c) ImPrEd with $\delta = 5$ and $\gamma = 2$. (d) ImPrEd with $\delta = 2$ and $\gamma = 5$.

design, distant, adjacent nodes have large attractive forces to reduce edge length. In our opinion, this behaviour can be contradictory to the aims of PrEd, as the extremities of a long edge may be tightly constrained, resulting in large, irreducible forces that delay convergence.

We overcome this problem by modifying the exponents of the force system in a way similar to previous work in revealing graph cluster structure [vHvW04]. Consider an optimal distance of δ between graph elements. Increasing the exponent of the repulsive force strengthens the repulsion of elements closer than δ and decreases the effect of elements farther than δ. Decreasing the exponent of the attractive force decreases the effect of distance, reducing the attraction of distant elements. These modifications de-emphasise the influence of distant elements.

Empirically, the best results were obtained with a gradual transition from global to local that was dependent on the iteration. Initially, we would like global forces to drive the diagram roughly towards its final shape. On the other hand, during later stages of the computation, local forces should be preferred in order to optimise the final position of elements. In the current implementation, the exponents of $\mathcal{F}^r$ and $\mathcal{F}^e$ (see Section 2.5) range from 2 to 4, and the exponent of $\mathcal{F}^a$ ranges from 1 to 0.4. All exponents are increased/decreased linearly at each iteration.

As increasing exponents can generate forces of a high magnitude, we define a global distance $\bar{d}$ that cannot be exceeded when moving. This distance ranges from $[3\delta, 0)$ and is linearly decreased at each iteration. The effect of these changes increase the stability of the force system and the reliability of the parameters. They also contribute to a more even distribution of graph elements (see Figure 6).

5. ImPrEd New Features

In ImPrEd, several new features extend the applicability of the algorithm. With these modifications, ImPrEd can deal with a wider range of input with greater output control.

5.1. Extended Input Class

PrEd was designed to work on undirected, simple graphs. We also believe that PrEd works on connected graphs only, as some configurations of disconnected graphs can produce results that use space inefficiently. In fact, the force system of PrEd might push disconnected components apart indefinitely as there is only a repulsive force between these elements. Inspired by other force-directed algorithms [FLM95], ImPrEd has a gravity force $\mathcal{F}^g$ that attracts nodes to the barycentre of the graph with constant magnitude. This force prevents the previously described behaviour and helps maintain a good aspect ratio [NR04].

ImPrEd can also operate on multigraphs with polyline edges. In these situations, the input graph is converted into a simple graph by substituting edge bends with paths. The core algorithm is then applied, and the result is transformed back to its input format.

5.2. Crossable and Uncrossable Edges

In ImPrEd, each edge of the input graph can be marked as uncrossable or crossable, influencing the computation of $\mathcal{M}_v$ in Algorithm 1. These labels define the edge as a PrEd edge or as a crossable edge. Crossable edges contribute to the attractive force $\mathcal{F}^a$ only. We found these edges useful for embedding graphs inside Euler diagrams. Crossable edges are temporarily removed when computing $\mathcal{S}$, excluding them from $\mathcal{M}$ and $\mathcal{F}^e$ computations.

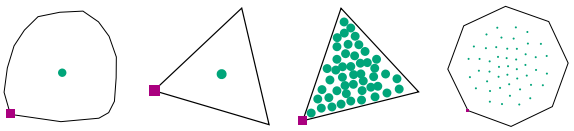
5.3. Flexible and Rigid Edges

The edges λ can also be labeled as rigid or flexible. These labels define if the edges can increase or decrease their number of bends. This property is useful when edges enclose elements of the graph: by marking these edges as flexible, they can expand or contract according to the stress applied to them by their containing elements.

During the iterations of Algorithm 1, each flexible edge is polled at regular intervals to verify whether it should be expanded or contracted. Let λ be a flexible edge mapped into the path $v_1, e_1, v_2, e_2 \dots v_n$ of nodes $v_i \in V$ and edges $e_i \in E$.

When contracting λ, we look for two nodes on the path, v_i, v_{i+2} for some i, closer than a distance α. If two such nodes exist and nodes are not inside the triangle v_i, v_{i+1}, v_{i+2}, we remove v_{i+1} and connect v_i to v_{i+2}. When expanding λ, we check if there are edges longer than a threshold β. If such an edge exists, we split the edge into two segments and insert them into the path. This operation is also safe, as it does not change the shape of λ. Examples of contraction and expansion of flexible boundaries are shown in Figure 7.

To obtain the best results with flexible edges, we remove node repulsion between consecutive nodes on a path. As a consequence, the attractive force exerted by an edge does



(a) (b) (c) (d)

Figure 7: Contraction and expansion of flexible edges. The pictures show a flexible loop containing nodes. a) An over-dimensioned flexible edge. c) An under-dimensioned flexible edge. b) and d) The result of ImPrEd.

not reach equilibrium and acts like a tension force that helps remove unnecessary bends. Empirically, we determined that $\alpha = 2 \cdot \delta$ and $\beta = 3 \cdot \delta$ are good threshold values, where δ is the optimal distance between two nodes. However, during early iterations of the algorithm, this value can cause a sharp increase in the number of bends introduced, as the graph is likely expanding under the effect of low attractive and high repulsive forces (see Section 4.4). This increase in the number of bends essentially increases the size of the data set and can affect the execution time of the algorithm substantially. As an execution time/diagram quality trade-off, we could clamp the number of bends introduced into the diagram or adjust the value of α and β. For our experiments that use flexible edges, we do not limit the number of bends introduced, providing diagrams of maximum quality.

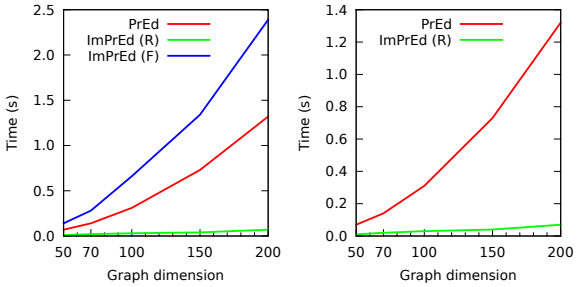
Also, it is important to note that edges should not be labeled as both flexible and crossable simultaneously. As nodes near a flexible edge often introduce bends, if the edge is also crossable, these bends would be incurred with little benefit. Therefore, in Section 6, none of our edges are simultaneously flexible and crossable. A similar argument can be made for flexible edges that initially cross, which should be avoided as well.

6. Results

In this section, we compare the drawing quality and execution time of PrEd and ImPrEd, on both synthetic and real data. ImPrEd (R) and ImPrEd (F) are applications of ImPrEd with all edges marked as rigid or flexible respectively.

We ran PrEd and ImPrEd on randomly generated plane and non-plane graphs in order to test execution time stability. The plane graph data set consists of 10 plane graphs with 50, 75, 100, 150, and 200 nodes and respectively 144, 204, 294, 444, 594 edges. The non-plane data set consists of the previous data sets with 20 randomly added crossings. In all, we test 100 randomly generated graphs.

We also compared PrEd and ImPrEd on real world data. In Simonetto *et al.* [SAA09], PrEd is used to generate Euler-like diagrams. First, PrEd improves the initial layout of a planar graph called *iGraph*, representing the skeleton



(a) (b)

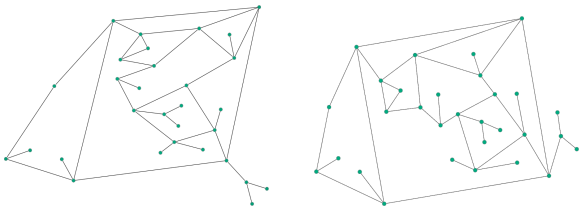
Figure 8: Growth of the average computation time for single iteration, with respect to the size of the random graphs presented in Section 6. a) Random plane graphs. b) Random non-plane graphs. ImPrEd (F) has not be applied to the non-plane graphs as flexible edges should not cross.

Graph		iGraphA	gGraphA	iGraphB	gGraphB
Nodes		149	2601	23	220
Edges		169	507	22	362
PrEd	100	17.50	2721.79	0.51	—
	250	40.91	6792.53	1.28	—
	500	80.46	13585.36	2.53	—
ImPrEd (R)	100	4.90	131.55	0.30	15.65
	250	11.81	308.83	0.73	33.49
	500	23.34	597.55	1.42	65.34
ImPrEd (F)	100	6.68	97.22	0.47	9.25
	250	22.02	215.62	1.73	18.07
	500	35.29	409.18	3.79	30.81
ImPrEd (PP)	—	0.13	5.64	0.02	0.14
ImPrEd (R)	Gain	3.49	21.81	1.75	—
ImPrEd (F)	Gain	2.25	30.90	0.73	—

Table 1: Average running times of PrEd and ImPrEd for 100, 250 and 500 iterations, over the Euler diagrams generation graphs described in Section 6. ImPrEd (PP) is the pre-processing step (Section 4.1). We only applied ImPrEd to gGraphB as the input includes edges marked as crossable, which cannot be handled by PrEd. Green numbers indicate improvements, red numbers indicate slower execution.

of the final diagram. The resulting graph is used to build *gGraph*, which is a first approximation of the set boundaries. Finally, PrEd improves the layout of gGraph to define smoother and clearer set boundaries.

Here, we consider two diagrams. The first diagram consists of 20 sets and over 2000 elements and is Figure 8 in Simonetto *et al.* [SAA09]. Sets are films extracted from the IMDb top-chart and contain the full credited actor list. The second diagram (10 sets, 176 elements) is a small protein-protein interaction network [IMMS09] where nodes are placed into multiple sets with network edges included. The first diagram's graphs are iGraphA and gGraphA, and the second diagram's graphs are iGraphB and gGraphB.



(a) (b)

Figure 9: Layout improvement of PrEd's example graph. Both algorithms executed 100 iterations. a) Using PrEd [Ber00, Figure 1a]. b) Using ImPrEd (R).

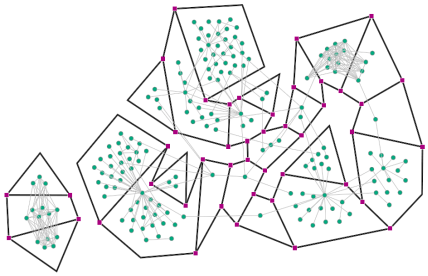


Figure 10: Result of the application of ImPrEd on gGraphB. In this graph, black edges represent the set boundaries and have been labeled as flexible. Grey edges represent element relationships and have been labeled as crossable.

6.1. Execution Time

Figure 8 plots running time vs. number of nodes for random planar and non-planar graphs respectively. ImPrEd (R) is consistently faster than PrEd on both data sets. On the smallest graphs, 50 nodes, ImPrEd (R) is 5 times faster than PrEd. On the largest graphs, 200 nodes, ImPrEd (R) is more than 20 times faster. On the other hand, ImPrEd (F) is consistently twice as slow as PrEd for all graph sizes with only small variations. As these random graphs, on average, have a high node to edge ratio, ImPrEd (F) must insert a high number of bends, slowing it considerably.

Table 1 reports the running times of PrEd and ImPrEd on the Euler diagram graphs. Again, ImPrEd (R) performs up to 20 times faster than PrEd. However, this time ImPrEd (F) outperforms ImPrEd (R) and is up to 30 times faster than PrEd. Most probably, this is due to the high number of initial bends in the diagram that ImPrEd (F) is able to remove, effectively reducing data set size.

On both the random data sets and Euler diagram graphs, pre-processing time is, for the most part, negligible (Table 1). On the smallest graphs, it can count for only 5% to 10% of the total running time. In the larger diagrams, it counts for only 1% of the total running time.

6.2. Drawing Quality

Figure 9 shows the results of PrEd and ImPrEd (R) on Bertault's example graph [Ber00, Figure 1]. ImPrEd seems to reach a more stable configuration with the same number of iterations, possibly because of the new movement rules.

Figure 11 shows a random plane graph of 100 nodes optimised by PrEd, ImPrEd (R), and ImPrEd (F). Although the force cooling in ImPrEd (R) (Figure 11b) makes better use of space when compared to PrEd (Figure 11a), the drawings are very similar. The high connectivity of the graph restricts the improvements that can be made with straight-line edges. ImPrEd (F) (Figure 11c) inserts bends, which increase the angular resolution of high degree nodes.

Figure 12 shows the results of PrEd, ImPrEd (R), and ImPrEd (F) on iGraphA and gGraphA. PrEd (Figures 12a and 12d) produces drawings where the nodes seem forced far from the centre. In particular, Figure 12d illustrates that nodes are distributed along region boundaries. As few attractive and many repulsive forces act on disconnected nodes, we realise this effect. The original forces of PrEd are hardly usable for graphs with a large number of disconnected nodes. The results in Simonetto *et al.* [SAA09] are obtained by careful selection of algorithm parameters and by increasing the repulsive force exponent (Section 4.4). However, in many of the diagrams of this article, nodes have a tendency to accumulate on set borders.

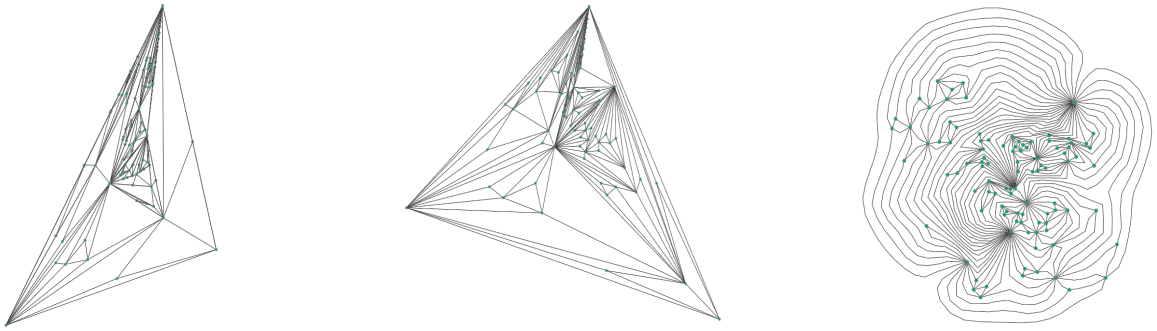
ImPrEd improves the distribution of disconnected nodes as seen in Figures 12b and 12e. The modified force system of ImPrEd enables this result. Also, the δ and γ parameters of ImPrEd drive, with higher reliability, the result towards optimal node-node and node-edge distances.

Flexible edges further improve the quality of these drawings. Figures 12c and 12f show improvements in the node distribution and more regular boundaries. Moreover, unnecessary bends are removed from the diagram, as shown in the close-ups, creating smoother set borders.

Finally, Figure 10 shows the results of ImPrEd (F) on gGraphB. The thick black edges, that represent the set boundaries, have smooth shapes proportional to the number of vertices contained. The grey edges, marked as crossable, helped determine the structure of the diagram and node position. However, they did not overly deform set boundaries.

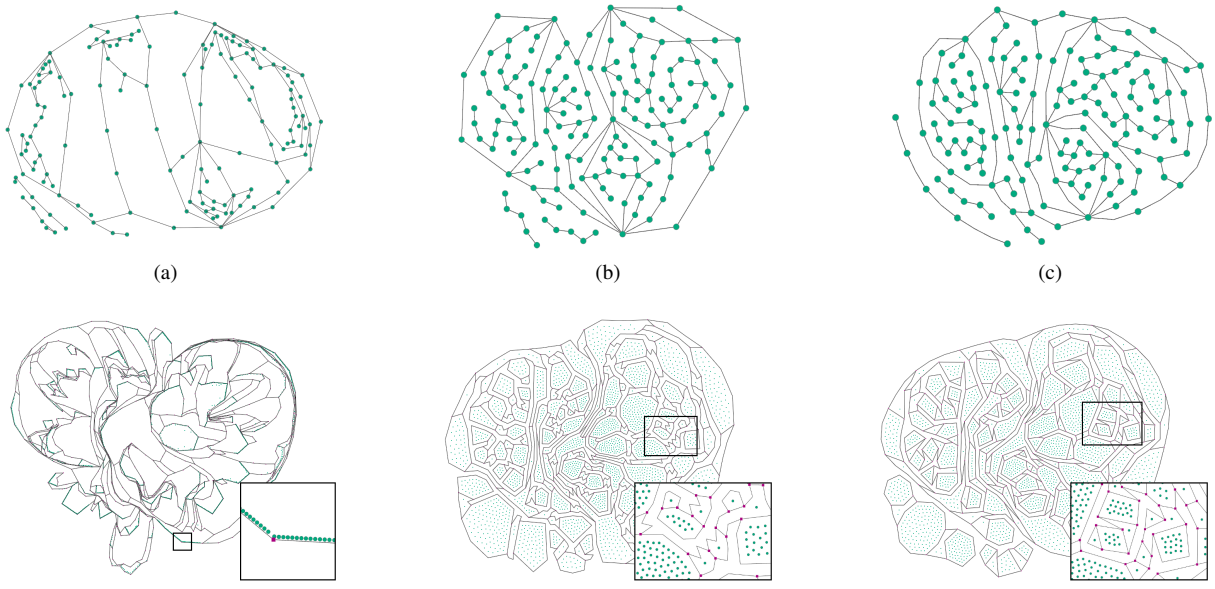
7. Conclusions and Future Work

We presented ImPrEd, an algorithm that improves an existing graph layout while preserving the graph map. Our test cases provide some evidence that ImPrEd improves upon PrEd in terms of both running time (surrounding edges computation and QuadTrees) and drawing quality (maximal movement computation and force cooling). Through flexible edges, ImPrEd can further improve diagram quality. However, there is an execution time/quality trade-off when a large number of bends are introduced into the drawing.



(a) (b) (c)

Figure 11: Layout improvement of a randomly generated plane graph. All algorithms used the parameters $\delta = 10$, $\gamma = 10$, 250 iterations. a) Using PrEd. b) Using ImPrEd (R). c) Using ImPrEd (F).



(a) (b) (c)

(d) (e) (f)

Figure 12: Layout improvement of graphs involved in the generation of Euler diagrams. In the first row, iGraphA. Parameters: $\delta = 27$, $\gamma = 21$, 250 iterations. The node diameter is set to 10 to better show the graph proportions. In the second row, gGraphA. Parameters: $\delta = 5$, $\gamma = 4$, 250 iterations. a,d) Using PrEd. b,e) Using ImPrEd (R). c,f) Using ImPrEd (F).

ImPrEd also extends PrEd to a larger input set, especially when portions of the graph lie in bounded regions. For these applications, ImPrEd can mark edges as crossable or uncrossable and as rigid or flexible, allowing for more customisable output. Finally, we showed ImPrEd can be applied to scenarios such as improving planar drawings of graphs and Euler-like diagram generation.

As future work, we would like to investigate new application fields for ImPrEd. For instance, the algorithm could be applied to combined graph-map drawings, such as GMaps [GHK10]. The running time improvement could also be further studied in terms of its performance with respect to edge density. However, ImPrEd still has problems scaling to large graphs. To overcome this limitation, its algorithm complexity should be further reduced as, in worst case, it coincides with that of PrEd.

Acknowledgments The second author would like to acknowledge support from Science Foundation Ireland Grant No. 08/SRC/I140.

References

[BBP08] BOITMANIS K., BRANDES U., PICH C.: Visualizing internet evolution on the autonomous systems level. In *International Symposium on Graph Drawing (GD07)* (2008), Hong S.-H., Nishizeki T., Quan W., (Eds.), vol. 4875 of *Lecture Notes in Computer Science*, Springer, pp. 365–376.

[Ber00] BERTAULT F.: A force-directed algorithm that preserves edge crossing properties. *Information Processing Letters 74*, 1–2 (Apr. 2000), 7–13.

[DBCVKO00] DE BERG M., CHEONG O., VAN KREVELD M., OVERMARS M.: *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 2000.

[DBETT98] DI BATTISTA G., EADES P., TAMASSIA R., TOLLIS I. G.: *Graph Drawing; Algorithms for the Visualization of Graphs*. Prentice Hall, July 1998.

[dFOdM02] DE FRAYSSEIX H., OSSONA DE MENDEZ P.: Public implementation of a graph algorithm library and editor (PIGALE), 2002. `http://pigale.sourceforge.net`.

[DFPP90] DE FRAYSSEIX H., PACH J., POLLACK R.: How to draw a planar graph on a grid. *Combinatorica 10*, 1 (1990), 41–51.

[DK05] DWYER T., KOREN Y.: Dig-CoLa: Directed graph layout through constrained energy minimization. In *IEEE Symposium on Information Visualization (InfoVis05)* (2005), Ward M., Stasko J., (Eds.), IEEE Computer Society, pp. 65–72.

[DKM06] DWYER T., KOREN Y., MARRIOTT K.: IPSep-CoLa: An incremental procedure for separation constraint layout of graphs. *IEEE Transactions on Visualization and Computer Graphics (InfoVis06) 12*, 5 (Sept. 2006), 821–828.

[DLR11] DIDIMO W., LIOTTA G., ROMEO S. A.: Topology-driven force-directed algorithms. In *International Symposium on Graph Drawing (GD10)* (2011), Brandes U., Cornelsen S., (Eds.), vol. 6502 of *Lecture Notes in Computer Science*, Springer, pp. 165–176.

[DMW09] DWYER T., MARRIOTT K., WYBROW M.: Topology preserving constrained graph layout. In *International Symposium on Graph Drawing (GD08)* (2009), Tollis I. G., Patrignani M., (Eds.), vol. 5417 of *Lecture Notes in Computer Science*, Springer, pp. 230–241.

[Dwy09] DWYER T.: Scalable, versatile and simple constrained graph layout. *Computer Graphics Forum (EuroVis09) 28*, 3 (June 2009), 991–998.

[Ead84] EADES P.: A heuristic for graph drawing. *Congressus numerantium 42* (1984), 149–160.

[EHK*04] ERTEN C., HARDING P. J., KOBOUROV S. G., WAMPLER K., YEE G. V.: GraphAEL: Graph animations with evolving layouts. In *International Symposium on Graph Drawing (GD03)* (2004), Liotta G., (Ed.), vol. 2912 of *Lecture Notes in Computer Science*, Springer, pp. 98–110.

[FLM95] FRICK A., LUDWIG A., MEHLDAU H.: A fast adaptive layout algorithm for undirected graphs. In *International Symposium on Graph Drawing (GD94)* (1995), Tamassia R., Tollis I. G., (Eds.), vol. 894 of *Lecture Notes in Computer Science*, Springer, pp. 388–403.

[FR91] FRUCHTERMAN T. M., REINGOLD E. M.: Graph drawing by force-directed placement. *Software: Practice and Experience 21*, 11 (Nov. 1991), 1129–1164.

[FT04] FRISHMAN Y., TAL A.: Dynamic drawing of clustered graphs. In *IEEE Symposium on Information Visualization (InfoVis04)* (2004), Ward M., Munzner T., (Eds.), IEEE Computer Society, pp. 191–198.

[FT08] FRISHMAN Y., TAL A.: Online dynamic graph drawing. *IEEE Transactions on Visualization and Computer Graphics 14*, 4 (2008), 727–740.

[GHK10] GANSNER E. R., HU Y., KOBOUROV S.: GMap: Visualizing graphs and clusters as maps. In *IEEE Pacific Visualization Symposium (PacificVis10)* (2010), North S., Shen H.-W., van Wijk J., (Eds.), IEEE Computer Society, pp. 201–208.

[GM98] GUTWENGER C., MUTZEL P.: Planar polyline drawings with good angular resolution. In *International Symposium on Graph Drawing (GD98)* (1998), Whitesides S., (Ed.), vol. 1547 of *Lecture Notes in Computer Science*, Springer, pp. 167–182.

[HMM00] HERMAN I., MELANÇON G., MARSHALL M. S.: Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics 6*, 1 (2000), 24–43.

[IMMS09] ITOH T., MUELDER C., MA K.-L., SESE J.: A hybrid space-filling and force-directed layout method for visualizing multiple-category graphs. In *IEEE Pacific Visualization Symposium (PacificVis09)* (2009), Eades P., Ertl T., Shen H.-W., (Eds.), IEEE Computer Society, pp. 121–128.

[NR04] NISHIZEKI T., RAHMAN M. S.: *Planar Graph Drawing*, vol. 12 of *Lecture Notes Series on Computing*. World Scientific, Sept. 2004.

[Pur97] PURCHASE H. C.: Which aesthetic has the greatest effect on human understanding? In *International Symposium on Graph Drawing (GD97)* (1997), Di Battista G., (Ed.), vol. 1353 of *Lecture Notes in Computer Science*, Springer, pp. 248–261.

[QE01] QUIGLEY A., EADES P.: FADE: Graph drawing, clustering, and visual abstraction. In *International Symposium on Graph Drawing (GD00)* (2001), Marks J., (Ed.), vol. 1984 of *Lecture Notes in Computer Science*, Springer, pp. 197–210.

[SAA09] SIMONETTO P., AUBER D., ARCHAMBAULT D.: Fully automatic visualisation of overlapping sets. *Computer Graphics Forum (EuroVis09) 28*, 3 (June 2009), 967–974.

[SP08] SAFFREY P., PURCHASE H. C.: The "mental map" versus "static aesthetic" compromise in dynamic graphs: A user study. In *Australasian User Interface Conference (AUIC08)* (Jan. 2008), Plimmer B., Weber G., (Eds.), vol. 76 of *Conferences in Research and Practice in Information Technology*, Australian Computer Society, pp. 85–93.

[vHvW04] VAN HAM F., VAN WIJK J. J.: Interactive visualization of small world graphs. In *IEEE Symposium on Information Visualization (InfoVis04)* (2004), Ward M., Munzner T., (Eds.), IEEE Computer Society, pp. 199–206.