Information Processing Letters 112 (2012) 418–422

Contents lists available at SciVerse ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

Drawing trees in a streaming model ☆

Carla Binucci [a], Ulrik Brandes [b], Giuseppe Di Battista [c], Walter Didimo [a,*], Marco Gaertler [d], Pietro Palladino [e], Maurizio Patrignani [c,**], Antonios Symvonis [f], Katharina Zweig [g]

[a] Dipartimento di Ing. Elettronica e dell'Informazione, Università degli Studi di Perugia, Italy
[b] Department of Computer and Information Science, University of Konstanz, Germany
[c] Dipartimento di Informatica e Automazione, Università Roma Tre, Italy
[d] Institute of Theoretical Computer Science, University of Karlsruhe, Germany
[e] Dipartimento di Medicina Sperimentale e Scienze Biochimiche, Università degli Studi di Perugia, Italy
[f] Department of Mathematics, National Technical University of Athens, Greece
[g] Interdisciplinary Center for Scientific Computing (IWR), University of Heidelberg, Germany

ARTICLE INFO

Article history:
Received 29 September 2011
Received in revised form 10 January 2012
Accepted 14 February 2012
Available online 28 February 2012
Communicated by R. Uehara

Keywords:
Design of algorithms
Graph algorithms
Online algorithms
Graph drawing
Streaming
Large graphs

ABSTRACT

We pose a new visualization challenge, asking Graph Drawing algorithms to cope with the requirements of Streaming applications. In this model a source produces a graph one edge at a time. When an edge is produced, it is immediately drawn and its placement cannot be altered. The drawing has an image persistence, that controls the lifetime of edges. If the persistence is k, an edge remains in the drawing for the time spent by the source to generate k edges, and then it fades away. In this model we study the area requirement of planar straight-line grid drawings of trees and we assess the output quality of the presented algorithms by computing the competitive ratio with respect to the best known offline algorithms.

1. Introduction

We consider the following model. A source produces a graph one edge at a time. When an edge is produced, it is immediately drawn (i.e., before the next edge is produced) and its drawing cannot be altered. The drawing has an image persistence, that controls the lifetime of edges. If the persistence is k, an edge remains in the drawing for the time spent by the source to generate k edges, and then it fades away.

Studying this model, which we call *streamed graph drawing*, is motivated by the challenge of offering visualization facilities to streaming applications, where massive amounts of data, too large even to be stored, are produced and processed at a very high rate [2]. The data are available one element at a time and need to be processed quickly and with limited resources. Examples of application fields include computer network traffic analysis, logging of security data, stock exchange quotes' correlation, etc.

For the user of the visualization facility it is natural to associate any graphic change with a new datum coming from the stream. Hence, moving pieces of the drawing

☆ Work on this problem began at the BICI Workshop on Graph Drawing: Visualization of Large Graphs, held in Bertinoro, Italy, in March 2008. Work supported in part by the MIUR project AlgoDEEP prot. 2008TFBWL4. Part of the research was conducted in the framework of ESF project 10-EuroGIGA-OP-003 GraDR "Graph Drawings and Representations". An extended abstract of this paper appeared in the proceedings of the 17th International Symposium on Graph Drawing, GD 2009 (Binucci et al., 2009 [1]).

* Principal corresponding author.
** Corresponding author.
E-mail addresses: binucci@diei.unipg.it (C. Binucci), Ulrik.Brandes@uni-konstanz.de (U. Brandes), gdb@dia.uniroma3.it (G. Di Battista), didimo@diei.unipg.it (W. Didimo), marco.gaertler@kit.edu (M. Gaertler), pietropalladino@gmail.com (P. Palladino), patrigna@dia.uniroma3.it (M. Patrignani), symvonis@math.ntua.gr (A. Symvonis), katharina.zweig@iwr.uni-heidelberg.de (K. Zweig).

0020-0190/$ – see front matter
doi:10.1016/j.ipl.2012.02.011

would be potentially ambiguous. On the other hand, the drawing should have a size as small as possible.

Although streamed graph drawing is related to incremental and dynamic graph drawing, it is qualitatively different from both. In incremental graph drawing the layout is constructed step by step according to a precomputed vertex ordering that ensures invariants regarding, e.g., its shape [3,4]. In streamed graph drawing the order cannot be chosen. Dynamic graph drawing [5–7] usually refers to drawing sequences of graphs, where drawings of consecutive graphs should be similar. Insertions and/or deletions of vertices/edges are allowed and the current graph must be drawn without knowledge of future updates. However, the current layout is only weakly constrained by previous drawings. In streamed graph drawing modifications concern only single edges and previous layout decisions may not be altered.

While there is some work on computing properties of streamed graphs (see, e.g., [8–10]), as far as we know this is the first time that the problem of drawing the k most recent edges of a stream has been addressed.

In this paper, we concentrate on trees and we make some assumption on the ordering in which the edges of the tree are visited. Namely, we consider the area requirement for planar straight-line grid drawings of trees, and we assume that the edges are streamed corresponding to an Eulerian tour of the tree. A typical real-world scenario in which this kind of streamed trees occur is the live representation of procedure call trees in dynamic program analysis. Each procedure may call other procedures and each call suspends the calling procedure until the called procedure has terminated. Note that, even medium size programs may have billions of procedure calls during a single run, which motivates the design of visualization tools for trace exploration [11–14]. Also, drawing a graph in a small area is a typical goal in graph visualization (see, e.g., [15]).

Since a streamed graph drawing algorithm is a special case of an online algorithm, it is reasonable to assess its output quality in terms of its competitive ratio with respect to the best known offline algorithm.

This paper is organized as follows. In Section 2 we introduce the concept of streamed graph drawing. Area requirements for tree drawings are derived in Section 3, and we conclude with directions for future work in Section 4.

2. Framework

Let $G = (V, E)$ be a simple undirected graph. A *straight-line grid drawing* $\Gamma = \Gamma(G)$ is a geometric representation of G such that each vertex is drawn as a distinct point of an integer-coordinate grid, and each edge is drawn as a straight-line segment between the points associated with its end-vertices. A drawing is *planar* if no two edges cross. Since we only consider planar straight-line grid drawings we simply refer to them as *drawings* in the remainder.

Given a subset of edges $E' \subseteq E$, the *edge-induced* (*sub*)*graph* $G[E']$ contains exactly those vertices of V incident with edges in E', and the edges in E'. We study the problem of drawing a (potentially infinite) graph G described by a sequence of edges $(e_1, e_2, e_3, \ldots)$, which we call a *stream of edges*, where e_i is known at time i. Throughout this paper, let $W_i^k = \{e_{i-k+1}, \ldots, e_i\}$ denote a *window* of the stream of size k and let $E_i = \{e_1, \ldots, e_i\}$ denote the *prefix* of the stream of length i. Observe that $E_i = W_i^i$.

Our goal is to design online drawing algorithms for streamed graphs. An online drawing algorithm incrementally constructs a drawing of the graph, by adding one edge at a time according to the order in which they appear in the stream. Once a vertex is placed, however, its placement cannot be altered until the vertex is removed.

More formally, we address the following problem: Let Γ_0 be an initially empty drawing. At each time $i \geqslant 1$ and for some fixed parameter $k \geqslant 1$, called *persistence*, determine a drawing Γ_i of $G_i = G[W_i^k]$ by adding e_i to Γ_{i-1} and dropping (if $i > k$) e_{i-k} from Γ_i.

We assume that our memory is bounded by $O(k)$.

Since streamed graph drawing algorithms are special online algorithms, an important assessment of quality is their competitive ratio. For a given online drawing algorithm A and some measure of quality, consider any stream of edges $S = (e_1, e_2, \ldots)$. Denote by $A(S)$ the quality of A executed on S, and by $Opt(S)$ the quality achievable by an optimal offline algorithm, i.e. an algorithm that knows the streaming order in advance. Where possible, we measure the effectiveness of A by evaluating its *competitive ratio*: $R_A = \max_S \frac{A(S)}{Opt(S)}$.

In the remainder of the paper we restrict our attention to the case where G is a tree, the goal is to determine a planar straight-line grid drawing, and the measure of quality is the *area* required by the drawing, i.e., the number of grid points contained in the minimum bounding box for the drawing. We recall that, static algorithms to draw an n-vertex tree in $\Theta(n)$ area are known if the tree is a binary tree [16] or if its vertex-degree is bounded by $\sqrt{n}$ [17]. The best known area bound for general trees is $O(n \log n)$ [18,19].

3. Drawing a streamed tree

We consider the following scenario, corresponding to the intuitive notion of a user traversing an undirected tree: the edges of the stream are given according to an Eulerian tour of the tree where we suppose that the persistence k is much smaller than the number of the edges of the tree (the tree may be considered "infinite"). Each (undirected) edge (u, v) is traversed exactly twice, once from u to v and once from v to u: the direction in which the edge is traversed for the first time is called the *forward* direction; the other direction is called the *backward* direction.

This corresponds to a DFS traversal where backtracks explicitly appear. Observe that window W_i^k contains in general both forward and backward edges and that $G[W_i^k]$ is always connected. Fig. 1 shows an example of an Eulerian tour where several windows of size 5 are highlighted: window W_1 contains two forward and three backward edges; window W_5 contains all backward edges (in the figure, the DFS visit starts from the rightmost vertex of the drawing and proceeds counter-clockwise).

In this scenario a vertex may be encountered several times during the traversal. Consider edge $e_i = (u, v)$ and assume that the Eulerian tour moves from u to v. We

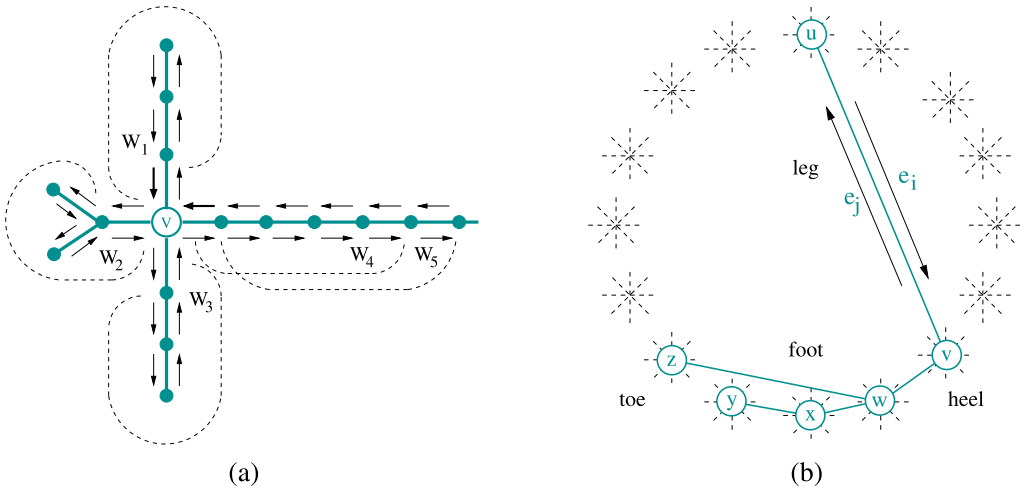

Fig. 1. (a) An Eulerian tour with a persistence $k = 5$. When W_5 is the current window, vertex v disappears from the drawing. (b) A leg of vertex u.

say that e_i *leaves* u and *reaches* v. Also, if v was already a vertex of G_{i-1} (and hence is already drawn in Γ_{i-1}) then, we say that e_i *returns* to v. Otherwise, v has to be inserted into the drawing Γ_i of G_i. Observe that if a vertex v, reached at time i, is reached again at time j, with $j > i + k + 1$, and is not reached at any intermediate time, then v has (in general) two different representations in Γ_i and Γ_j.

The first algorithm presented in this section is the following. Consider m integer-coordinate points $p_0, p_1, \ldots, p_{m-1}$ in convex position. An easy strategy is to use such points clockwise in a greedy way. At each time i, we maintain an index $next_i$ such that point p_{next_i} is the first unused point in clockwise order. The first edge e_1 is drawn between points p_0 and p_1 and $next_2 = 2$. Suppose that edge $e_i = (u, v)$ has to be added to the drawing. If v is not present in Γ_{i-1}, assign to v the coordinates of p_{next_i} and set $next_{i+1} = (next_i + 1) \bmod m$. We call this algorithm Greedy-Clockwise (GC).

Algorithm GC guarantees a non-intersecting drawing provided that two conditions are satisfied for all i.

Condition 1 Point p_{next_i} is not used in Γ_i by any vertex different from v.

Condition 2 Edge e_i does not cross any edge of Γ_i.

Lemma 1 and Lemma 2 show that satisfying Condition 1 implies satisfying Condition 2. For a vertex w of Γ_i, we denote by $i(w)$ the time when vertex w entered Γ_i.

Lemma 1. *Let Γ_i be a drawing of G_i constructed by Algorithm* GC *and let v_1, v_2, and v_3 be three vertices of G_i such that $i(v_1) < i(v_2) < i(v_3)$ in Γ_i. If there is a sequence of forward edges from v_1 to v_3, then there is a sequence of forward edges from v_1 to v_2.*

Proof. Consider edges $e_{i(v_1)} = (v_0, v_1)$ and $e_j = (v_1, v_0)$ of the stream. The Eulerian tour implies that the vertices reached by a forward path from v_1 are those vertices incident to some edge e_h, with $i(v_1) < h < j$. Suppose for a contradiction that v_2 is not reached by a forward path from v_1. Since v_2 was drawn after v_1, this implies $i(v_2) > j$. It follows that also $i(v_3) > j$. Hence, v_3 cannot be reached by a forward path from v_1. □

Lemma 2. *Let Γ_{i-1} be a drawing of G_{i-1} constructed by Algorithm* GC *and consider a vertex v that is not in G_{i-1} and should be added to G_{i-1} at time i. If Condition* 1 *is satisfied, then no crossing is introduced by drawing v at p_{next_i}.*

Proof. Let $e_i = (u, v)$. Draw v on p_{next_i}. Since Condition 1 is satisfied, p_{next_i} is not used by any vertex. Suppose for contradiction that Γ_i has a crossing. It follows that there exists an edge (x, y) in Γ_i such that vertices x, u, y, v appear in this relative order in the clockwise direction. By Condition 1 and since the points are used in a greedy way, $i(x) < i(u) < i(y) < i(v)$. Because of edge (x, y), there is a forward path from x to y and hence by Lemma 1 there is a forward path from x to u. Analogously, because of edge (u, v), there is a forward path from u to v and hence by Lemma 1 there is a forward path from u to y. Hence, there is an undirected cycle in G_i involving x, u, and y. This is a contradiction since we are exploring a tree. □

Consider two edges $e_i = (u, v)$ and $e_j = (v, u)$, with $j > i$. Observe that $j - i$ is odd. Edges $e_i, e_{i+1}, \ldots, e_j$ are a *leg* of u. Vertices discovered at times $i, i + 1, \ldots, j$, i.e., the $\frac{j-i+1}{2}$ distinct vertices incident to edges $e_{i+1}, \ldots, e_{j-1}$, are a *foot* of u. Node v is the *heel* of the foot and the last discovered vertex of the foot is the *toe*. Fig. 1(b) shows the drawing of a leg (and provides a hint of why its vertices are called a foot). A foot is itself composed of smaller feet, where the smallest possible foot is when a leaf of the tree is reached, that is, when its heel and its toe are the same vertex (as for vertex y of Fig. 1(b)).

Consider the case when $j - i \leqslant k$. This implies that u is present in all the drawings $\Gamma_{i-1}, \ldots, \Gamma_{j+k}$. In this case we say that the foot is a *regular foot* (or *R-foot*). Otherwise, we say that it is an *extra-large foot* (or *XL-foot*).

Property 1. A regular foot has maximum size $\lceil \frac{k}{2} \rceil$.

Observe that in any drawing constructed by Algorithm GC the vertices of a regular foot are contiguously placed after its heel, the toe being the last in clockwise order.

Property 2. Let i be the time when an extra-large foot of v is entered by the Eulerian tour. Vertex v disappears from the drawing at time $i+k$.

Now, we exploit the above properties and lemmas to prove that, if k is the persistence of the drawing and if the tree has maximum degree d (where a binary tree has $d=3$), then it suffices using $\lceil \frac{k}{2} \rceil \cdot (d-1)+k+1$ points in convex position to guarantee to GC that Condition 1 is satisfied. In order to prove this, we need the following lemma.

Lemma 3. *Consider Algorithm* GC *on m points in convex position. Suppose that for each vertex v it holds that during the time elapsing from when v is discovered and when it disappears from the drawing at most $m-1$ other vertices are discovered. Then Condition* 1 *holds at each time.*

Proof. Suppose, for a contradiction that there exists a vertex u, discovered at time i, for which Condition 1 does not hold because point p_{next_i} is used by vertex $w \neq u$. Since GC is greedy, after u has been inserted all the m points have been used. This implies that after w and before u, $m-1$ vertices have been discovered. Summing up, we have that w violates the condition of the statement. □

Theorem 4. *Let S be a stream of edges produced by an Eulerian tour of a tree of degree at most d. Algorithm* GC *draws S with persistence k without crossings on $\lceil \frac{k}{2} \rceil \cdot (d-1)+k+1$ points in convex position. Also $R_{GC} = O(d^3k^2)$.*

Proof. Due to Lemma 2 it suffices to show that Condition 1 holds at each time i. We exploit Lemma 3 to show that during the time elapsing from when a vertex v is discovered and when it disappears from the drawing at most $\lceil \frac{k}{2} \rceil \cdot (d-1)+k$ other vertices are discovered. Suppose v is discovered by edge $e_i=(u,v)$. Three cases are possible: (i) v is a leaf; (ii) all feet of v are regular; (iii) v has an XL-foot. Case (i) is simple: we have that v disappears from the drawing at time $i+k+1$. Hence, at most k vertices can be discovered before it disappears. In case (ii) since each R-foot can have at most $\lceil \frac{k}{2} \rceil$ vertices (Property 1) and since at most $(d-1)$ of them can be traversed, the maximum number of vertices that can be discovered after v enters the drawing and before it disappears is $\lceil \frac{k}{2} \rceil \cdot (d-1)+k$ (see Fig. 1(a) for an example with $k=5$). In case (iii), because of Property 2, after the XL-foot is entered, at most k vertices can be discovered before v disappears. Hence, the worst case is that the XL-foot follows $d-2$ R-feet. Overall, a maximum of $\lceil \frac{k}{2} \rceil \cdot (d-2)+k$ vertices can be discovered before v disappears.

Regarding the competitive ratio, m grid points in convex position take $O(m^3)$ area [20], and therefore the area of the drawing of our online algorithm is $\Theta(d^3k^3)$. Finally, any offline algorithm requires $\Omega(k)$ area for placing $O(k)$ vertices. □

Theorem 4 uses a number of points that is proportional to the maximum degree of the tree. In the following we introduce a second algorithm that uses a number of points that only depends on the persistence k.

Intuitively, the basic strategy is to alternate Algorithm GC with its mirrored version, called Greedy-Counter-Clockwise (GCC), where, at each step, $next_i$ is possibly decreased rather than increased. Namely, let $old(\Gamma_i)$ be the *oldest vertex of* Γ_i, that is, the vertex that appears in $\Gamma_i, \Gamma_{i-1}, \ldots, \Gamma_{i-j}$ with highest j. The decision of switching between GC and GCC (or vice versa) is taken each time you start to draw a new foot of $old(\Gamma_i)$. We begin with Algorithm GC and use points in the clockwise direction with respect to $old(\Gamma_i)$ until we have used enough of them to ensure that the points near to $old(\Gamma_i)$ in the counter-clockwise direction are available. At this point, we switch to Algorithm GCC, starting from the point immediately next to $old(\Gamma_i)$ in the counter-clockwise direction, and we use Algorithm GCC to draw the next feet of $old(\Gamma_i)$ until the last drawn foot of $old(\Gamma_i)$ has used enough points in the counter-clockwise direction to ensure that the points in clockwise direction are available. Fig. 2(a) shows an example where three feet were drawn by GC and the fourth foot is drawn by GCC.

Formally, Algorithm SnowPlow (SP) works as follows. Let old_i be the index of the point of Γ_i where $old(\Gamma_i)$ is drawn. Suppose that edge $e_i=(u,v)$ has to be added to the drawing. If v is present in Γ_{i-1} then $\Gamma_i = \Gamma_{i-1}$. Otherwise, if $u \neq old_i$ or $u = old_i$ but $(next_i - old_i) \bmod m \leqslant \lceil \frac{k}{2} \rceil$, place v on p_{next_i} and set $next_{i+1} = (next_i + 1) \bmod m$. If $u = old_i$ and $(next_i - old_i) \bmod m > \lceil \frac{k}{2} \rceil$, then switch to GCC, that is, place v on point $p_{(old_i-1) \bmod m}$ and set $next_{i+1} = (old_i - 2) \bmod m$.

A critical step is when $old(\Gamma_i) \neq old(\Gamma_{i-1})$. This happens when an XL-foot is drawn either by GC or by GCC. In this case the heel of such a foot becomes the oldest vertex (see Fig. 2(b) for an example).

We show in the following that SP needs $2k-1$ points in convex position to produce a non-crossing drawing of the stream of edges independently of the degree of the vertices.

Theorem 5. *Let S be a stream of edges produced by an Eulerian tour of a tree. Algorithm* SP *draws S with persistence k without crossings on $2k-1$ points in convex position. Also $R_{SP} = O(k^2)$.*

Proof. Without loss of generality, suppose that Algorithm SP is in its GC phase (the proof is symmetric if Algorithm SP is in its GCC phase). Also assume, without loss of generality, that $p_{old_i} = p_0$, and denote by $P^+ = \{p_1, p_2, \ldots, p_{\lceil \frac{k}{2} \rceil - 1}\}$ ($P^- = \{p_{-1}, p_{-2}, \ldots, p_{-\lceil \frac{k}{2} \rceil + 1}\}$) the points after p_{old_i} in clockwise (counter-clockwise) order. Consider the case when p_{old_i} has a sequence of R-feet. In order to switch to the GCC phase at least $\lceil \frac{k}{2} \rceil$ points and at most $2\lceil \frac{k}{2} \rceil - 1$ points of P^+ are used. Since at least $\lceil \frac{k}{2} \rceil$ points are used of P^+, at least the same amount of time elapsed from when the current GC phase started. Hence, points in P^- are not used by any vertex. □

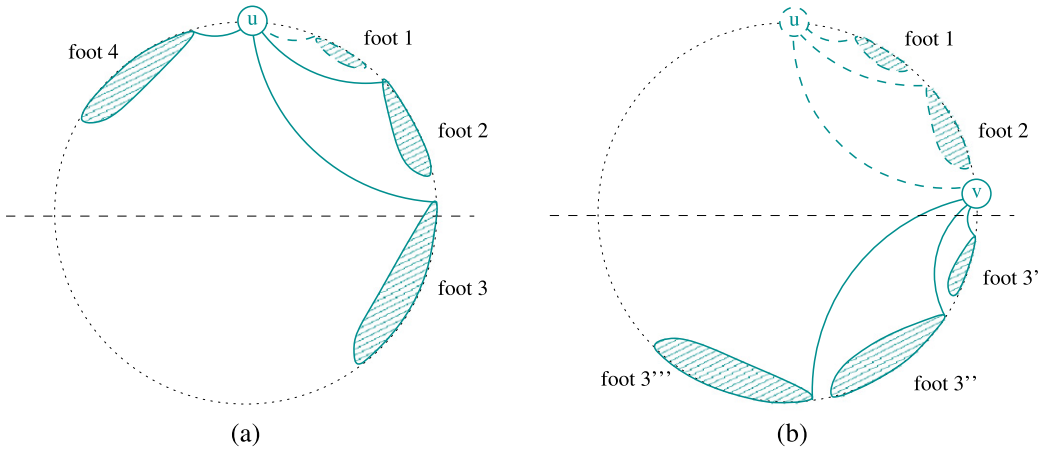

Fig. 2. (a) Feet 1, 2, and 3 are drawn by GC, foot 4 is drawn by GCC. (b) Foot 3 is an XL-foot of u. Its size is large enough to promote v as the oldest vertex in place of u.

4. Conclusions and open problems

This paper introduces a new model where Information Visualization is used to represent a stream of data and opens many possible research directions, including the following: (i) Streaming drawing algorithms could be better evaluated if tighter lower bounds on the area needed to draw streams of edges would be available. In this paper we compare with the (trivial) $\Omega(n)$ lower bound on the area requirement of n-vertex trees, which is unfair since no static algorithms are known for drawing general trees in linear area and since offline algorithms in our scenario are more constrained than static algorithms, as they are not allowed to change the vertex positions at each step. (ii) It would be interesting to extend the study to larger classes of planar graphs or even to general graphs. (iii) Other persistence models can be considered. For example we could have drawings where the persistence is $O(\log n)$, where n is the size of the stream. (iv) It would be interesting investigating models where the drawing algorithms can take advantage of a (limited) look-ahead of the stream of edges.

Acknowledgements

We wish to thank Camil Demetrescu and Irene Finocchi for useful conversations and especially for pointing out that dynamic program analysis would significantly benefit from streaming graph drawing techniques.

References

[1] C. Binucci, U. Brandes, G. Di Battista, W. Didimo, M. Gaertler, P. Palladino, M. Patrignani, A. Symvonis, K. Zweig, Drawing trees in a streaming model, in: D. Eppstein, E. Gansner (Eds.), GD'09, in: LNCS, 2009, pp. 292–303.
[2] S. Muthukrishnan, Data streams: Algorithms and applications, Found. Trends Theor. Comput. Sci. 1 (2) (2005) 117–236, doi:10.1561/0400000002.
[3] T. Biedl, G. Kant, A better heuristic for orthogonal graph drawings, Comput. Geom. 9 (1998) 159–180.
[4] H. de Fraysseix, J. Pach, R. Pollack, How to draw a planar graph on a grid, Combinatorica 10 (1990) 41–51.
[5] J. Branke, Dynamic graph drawing, in: M. Kaufmann, D. Wagner (Eds.), Drawing Graphs: Methods and Models, in: LNCS, vol. 2025, Springer, 2001, pp. 228–246.
[6] M.L. Huang, P. Eades, J. Wang, On-line animated visualization of huge graphs using a modified spring algorithm, J. Vis. Lang. Comput. 9 (6) (1998) 623–645.
[7] A. Papakostas, I.G. Tollis, Interactive orthogonal graph drawing, IEEE Trans. Comput. 47 (11) (1998) 1297–1309.
[8] Z. Bar-Yossef, R. Kumar, D. Sivakumar, Reductions in streaming algorithms, with an application to counting triangles in graphs, in: Proc. SODA, 2002, pp. 623–632.
[9] L. Buriol, D. Donato, S. Leonardi, T. Matzner, Using data stream algorithms for computing properties of large graphs, in: Proc. Workshop on Massive Geometric Datasets, MASSIVE'05, 2005, pp. 9–14.
[10] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, J. Zhang, On graph problems in a semi-streaming model, Theor. Comput. Sci. 348 (2–3) (2005) 207–216.
[11] T. Hill, J. Noble, J. Potter, Scalable visualizations of object-oriented systems with ownership trees, J. Vis. Lang. Comput. 13 (2002) 319–339, doi:10.1006/jvlc.2002.0238.
[12] A. Hamou-Lhadj, T. Lethbridge, A survey of trace exploration tools and techniques, in: Proceedings of the 2004 Conference of the Centre for Advanced Studies on Collaborative Research, ASCON'04, IBM Press, 2004, pp. 42–55.
[13] W. De Pauw, D. Lorenz, J. Vlissides, M. Wegman, Execution patterns in object-oriented visualization, in: Proceedings of the 4th Conference on USENIX Conference on Object-Oriented Technologies and Systems, vol. 4, COOTS'98, USENIX Association, 1998, pp. 219–236.
[14] J. Joshi, B. Cleary, C. Exton, Application of helix cone tree visualization to dynamic call graph illustration, in: Third Program Visualization Workshop, 2004, pp. 68–75.
[15] G.D. Battista, P. Eades, R. Tamassia, I.G. Tollis, Graph Drawing: Algorithms for the Visualization of Graphs, Prentice-Hall, 1999.
[16] A. Garg, A. Rusu, Straight-line drawings of binary trees with linear area and arbitrary aspect ratio, J. Graph Algorithms Appl. 8 (2) (2004) 135–160.
[17] A. Garg, A. Rusu, Straight-line drawings of general trees with linear area and arbitrary aspect ratio, in: ICCSA'03, in: LNCS, vol. 2669, 2002, pp. 876–885.
[18] P. Crescenzi, G. Di Battista, A. Piperno, A note on optimal area algorithms for upward drawings of binary trees, Comput. Geom. Theory Appl. 2 (1992) 187–200.
[19] Y. Shiloach, Arrangements of planar graphs on the planar lattice, Ph.D. thesis, Weizmann Institute of Science, 1976.
[20] I. Bárány, N. Tokushige, The minimum area of convex lattice n-gons, Combinatorica 24 (2) (2004) 171–185.