Drawing Graphs with GLEE*

Lev Nachmanson, George Robertson, and Bongshin Lee

Microsoft Research
One Microsoft Way, Redmond, WA 98052, USA
{levnach,ggr,bongshin}@microsoft.com

Abstract. This paper describes novel methods we developed to lay out graphs using Sugiyama's scheme [16] in a tool named GLEE. The main contributions are: a heuristic for creating a graph layout with a given aspect ratio, an efficient method of edge-crossings counting while performing adjacent vertex swaps, and a simple and fast spline routing algorithm.

1 Introduction

GLEE is a graph drawing tool that is being developed at Microsoft Research. Eiglsperger et al. [8] mention that most practical implementations of directed graph layout engines follow Sugiyama's scheme (or STT); GLEE is not an exception. In spite of the fact that there is lot of research devoted to the scheme, we were confronted with questions during the development process, for which we did not find answers in the literature. In the paper we address some of these questions. To our knowledge, nobody has solved the problem of creating a layered graph layout with a given aspect ratio. We developed a heuristic for creating such a layout. At an earlier stage of the development, GLEE's performance bottleneck was counting edge crossings while swapping adjacent nodes during the ordering step. We designed data structures and procedures to speed up the counting. Furthermore, several previous approaches for drawing splines did not give us satisfactory results. We developed a simple and efficient algorithm, producing aesthetic splines.

2 Layout with a Given Aspect Ratio

When laying out a graph, we would like to better utilize the available space and create an aesthetically pleasing layout. Here we present a heuristic of laying out graphs inside of a rectangle of a given aspect ratio. Figures 1 and 2 show two drawings of the same graph in rectangles of the same size. Both layouts were created by GLEE. We used the default algorithm for Fig. 2, and the heuristic for Fig. 1. The drawing in Fig. 1 better uses the available space and its larger nodes improve the readability.

* The full version of the paper is available at ftp://ftp.research.microsoft.com/pub/tr/TR-2007-72.pdf. GLEE can be downloaded at http://research.microsoft.com/~levnach/GLEEWebPage.htm.

S.-H. Hong, T. Nishizeki, and W. Quan (Eds.): GD 2007, LNCS 4875, pp. 389–394, 2007.

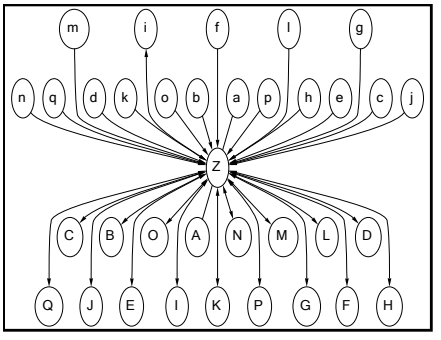

Fig. 1. Using the heuristic

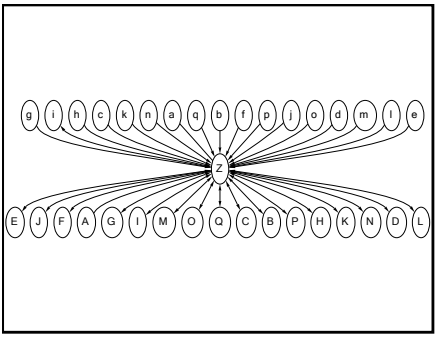

Fig. 2. Using the default layout

2.1 Description of the Heuristic

An algorithm for scheduling computer tasks for parallel processing on multiple processors can be used to compute the layering for STT. A scheduling algorithm deals with a DAG of computer tasks. If there exists a directed path in the DAG from task t_0 to task t_1, then t_0 is called a predecessor of t_1 and t_0 has to be completed before t_1 can be begun. Suppose each task takes a unit of time to complete on any processor. Then a task schedule corresponds to the layering, such that tasks scheduled to time 0 form the top layer, tasks scheduled to time 1 form a layer right below it, and so on. An upper limit on the sum of the node widths in a layer can play the role of the number of processors in a task scheduling algorithm. In particular, we could use Coffman-Graham algorithm [6] for our purposes. For a real number w let's call $A(w)$ a variation of Coffman-Graham algorithm where we require that the sum of node widths in a layer is not greater than w. In addition, $A(w)$ respects *Separation*. There is a tendency that for a larger w algorithm $A(w)$ produces a layout with the less height. We use it to find out the width w giving a good result. Let us call $B(w)$ the following procedure applied to DAG G:

Execute the layering step using $A(w)$
Execute the ordering step
Calculate node x-coordinates of the proper layered graph

We apply a binary search to find W such that $B(W)$ produces the aspect ratio which is close to the given one. Algorithm $B(w)$ can be repeated many times during the binary search. To speed up its execution, starting from graphs of a specific size, we apply the algorithm of [5] for calculation of node x-coordinates. We experimented with a variation of $A(w)$ where the width of edges crossing the layer is taken into account; surprisingly, the results were better when we ignored edge widths.

The application of $B(w)$ alone does not produce good layouts. To achieve a better quality of the layout we apply additional heuristics. These heuristics are node demotion and balancing of virtual and original nodes during the process of swapping of adjacent nodes.

Node demotion. When a processor is available and there is a task which is ready to be executed, Coffman-Graham algorithm immediately assigns the task to the processor. As

a result, some nodes can be positioned too high. We can improve the layout by pulling nodes down, or, in other words, by demoting them. The demotion step that we execute is the promotion step [14] being run in the opposite direction.

Balancing of virtual and original nodes. The heuristic helps in spreading uniformly edges passing a layer and nodes of the layer. We apply the heuristic during the ordering step. The ordering step starts when we already have a proper layered graph, but the order of nodes within a single layer is not yet defined. During the step we traverse the layers up and down several times applying the median method of [9] and create an ordering within the layers. The following sub-step of the ordering step is the swapping of nodes which are adjacent on the same layer. This is done to reduce the number of edge crossings. Here we utilize the sub-step for yet another purpose of spreading evenly virtual and original nodes. Let us describe the way we change the process of swapping of adjacent nodes. For a fixed layer, if the layer has fewer virtual nodes than original ones then we call virtual nodes separators and original nodes nulls. Otherwise we call virtual nodes of the layer nulls and original nodes separators. In the usual swapping process we proceed with a swap if it reduces the number of edge crossings, and do not proceed when it increases the number of edge crossings. In the case when the number of edge crossings does not change as a result of the swap, we have a freedom to apply the heuristic. Consider swapping of separator s with null m. Let K (M) be the set of all null nodes z to the left (right) of s such that no separator is positioned between z and s. Let K' and M' be sets defined the same way but as if s and m are swapped. If $||K'| - |M'|| < ||K| - |M||$ then we proceed with the swap.

Related work. Authors of Graphviz, a popular tool based on STT, mention Coffman-Graham algorithm as one of the approaches [2] to the aspect ratio problem. In [11] and [15] methods are developed to calculate *layering* for a directed graph with constraints on the width and the height of the layering. In the context of [11] and [15] the width of a layering is the maximum number of nodes in its layers, and the height of a layering is the number of its layers. Heuristics of [15] can be used instead of $A(w)$ in our approach, but we have not tried that.

3 Efficient Counting of Edge Crossings During Adjacent Swaps

Counting the crossings of edges connecting two neighboring layers at the ordering step is done by using the technique from [4] and works fast. However we observed a performance bottleneck in counting edge crossings at the phase of swapping adjacent nodes in a layer. The approach and data structures suggested here lead to an efficient implementation.

Proposition 1. *Swap of adjacent layer nodes u and v can be produced with the amortized cost $O(d(u) + d(v))$, where d is the degree of layered graph nodes.*

We give the details in the full version of the paper [1].

4 Spline Routing

In general, in our method we modify the given polyline to avoid nodes, straighten the polyline, and fit Bezier segments into its corners. Before describing the approach we need to define some notions. Let *PG* be the proper layered graph with already defined positions of nodes. For an edge $(u, v) \in E$ there is a unique sequence of nodes $U(e) = [u_0, ..., u_n]$ connecting u and v, such that; $u_0 = u$,$u_n = v$, (u_i, u_{i+1}) is an edge of *PG* for $i = 0, ..., n-1$, and nodes $u_1, ..., u_{n-1}$ are virtual. We call a polyline formed by positions of nodes from $U(e)$ the polyline of edge e. Let us define blocking nodes of an edge. Nodes u and v of *PG* belonging to the same layer are called non-blocking to each other if they are both virtual and some of edges adjacent to u cross some edges adjacent to v, as shown in Fig. 3. Otherwise u and v are called blocking to each other. The intuition is that if u is blocking for v, then a spline passing through v has to be disjoint from u. A node is called blocking for an edge e if it is blocking for some node of $U(e)$. For example, if u, v belong to the same layer and u is an original node, then u,v are blocking to each other. We build a spline for edge e of G and let p be a polyline of e. We proceed by the following steps which are explained below:

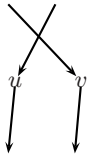

Fig. 3. Non-blocking u, v

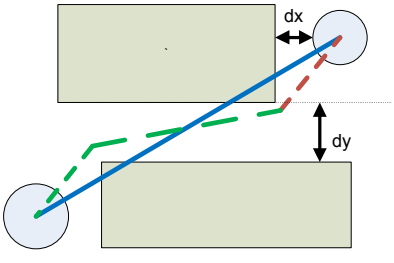

Fig. 4. Polyline refinement step

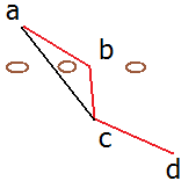

Fig. 5. Forbidden shortcut

1) Refine p if it crosses nodes blocking for e
2) Straighten p by using an inflection heuristic.
3) Smoothen p by fitting Bezier segments into p corners.

In the refinement step, using the node and layer separation, we replace each segment of p intersecting the blocking nodes with a polyine disjoint from the nodes. The inserted polyline can be split up into two connected pieces; one piece turns clockwise and the other counterclockwise as illustrated at Fig. 4. In the inflection heuristic we remove some polyline vertices by shortcutting them. Suppose that a, b, c and d are consecutive nodes of p. If at b polyline p turns, for example, clockwise, and at c it turns counter-clockwise, then we remove b from p, but only in the case when triangle a,b,c does not intersect a blocking node for e. Fig.5 shows a situation where we do not shortcut vertex b. Allowing such shortcuts can create additional edge spline crossings or allow the spline to intersect a blocking node.

Before smoothing corners we simplify the polyline, without actually changing its geometry, in a way that no three consecutive vertices of the polyline are collinear. Let

a, b and c be consecutive vertices of the polyline, and k be a real number. Let us set m to $a + k(b - a)$ and n to $c + k(b - c)$. We call $Bz(k)$ the cubic Bezier segment with control points m, $(m + 2b)/3$, $(n + 2b)/3$, and n. We find minimal k of the form $1 - 1/2^i$, for $i = 1, 2, ...$, such that the figure bounded by line segment $[m, b]$, line segment $[b, n]$ and $Bz(k)$ does not intersect blocking nodes for e. Such k exists since p does not intersect the blocking nodes. When we build $Bz(k)$ for each polyline corner, we reach our goal.

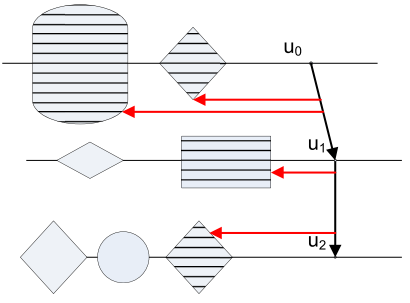

Fig. 6. Set *LS* is composed by dashed nodes

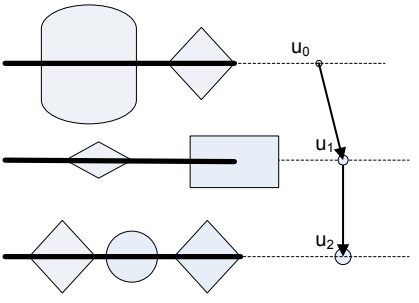

Fig. 7. Set *LT* is formed by thick horizontal line segments

We use the structure of *PG* to efficiently check for intersections. Namely, we do not intersect the polyline or the Bezier segment with all blocking nodes of edge e, but rather only with a subset of these nodes or with a specially constructed set of line segments. Let us show how to build these sets from the left of the edge. Denote by L all nodes of *PG* blocking for e and positioned to the left of e. The selected subset of nodes called *LS* is defined as the set of all nodes of L that are reachable by a horizontal ray starting at a point on p, as illustrated at Fig. 6. Set *LT* is formed by horizontal line segments starting at the left side of the *PG* bounding box and ending at the rightmost blocking node for u_i positioned the left of u_i, for $i = 0, ..., n$, as shown in Fig. 7. Sets *RS* and *RT* are defined by the symmetry. Step 1) checks intersections of the polyline only with nodes from sets *LS* and *RS*, while steps 2) and 3), in addition, intersect segment $[a, c]$ or $Bz(k)$ with sets *LT* and *RT* to detect intersections of triangle a, b, c or the figure bounded by $[m, b], [b, n], Bz(k)$ correspondingly with the blocking nodes. To speed up the calculation we build spatial trees on sets *LS*, *RS*, *LT* and *RT*, and utilize them in the crossing routine.

5 Conclusion and Future Work

We presented several novel methods producing good layouts for directed graphs using Sugiyama scheme. We developed a heuristic to lay out graphs inside of a rectangle of a given aspect ratio, which helps us better utilize the available space and create an aesthetically pleasing layout. We presented a fast edge-crossings counting method for adjacent node swaps. We described an efficient edge routing algorithm, which modifies a given edge polyline to avoid nodes, straightens the polyline, and fits Bezier segments into its corners.

We plan to introduce more balance and symmetry into GLEE layouts. Important features to add to the tool include interactive and incremental layout, and graph editing.

We would like to thank Stephen North and Yehuda Koren for fruitful discussions.

References

1. Drawing graphs with GLEE technical report, Lev Nachmanson, George Robertson and Bongshin Lee,
   `ftp://ftp.research.microsoft.com/pub/tr/TR-2007-72.pdf`
2. Graphviz todo list (December 22, 2005),
   `http://www.graphviz.org/doc/todo.html`
3. Abello, J., Gansner, E.R.: Short and smooth polygonal paths. In: Lucchesi, C.L., Moura, A.V. (eds.) LATIN 1998. LNCS, vol. 1380, pp. 151–162. Springer, Heidelberg (1998)
4. Barth, W., Jünger, M., Mutzel, P.: Simple and efficient bilayer cross counting. In: Goodrich, M.T., Kobourov, S.G. (eds.) GD 2002. LNCS, vol. 2528, pp. 130–141. Springer, Heidelberg (2002)
5. Brandes, U., Köpf, B.: Fast and simple horizontal coordinate assignment. In: Mutzel, P., Jünger, M., Leipert, S. (eds.) GD 2001. LNCS, vol. 2265, pp. 31–44. Springer, Heidelberg (2002)
6. Coffman, E.G., Graham, R.L.: Optimal Scheduling for Two-Processor Systems. Acta Informatica 1(3), 200–213 (1972)
7. Dobkin, D.P., Gansner, E.R., Koutsofios, E., North, S.: Implementing a general-purpose edge router. In: Di Battista, G. (ed.) Graph Drawing, Rome, Italy, September 18-20, 1997, pp. 262–271. Springer, Heidelberg (1998)
8. Eiglsperger, M., Siebenhaller, M., Kaufmann, M.: An efficient implementation of sugiyama's algorithm for layered graph drawing. In: Pach, J. (ed.) Graph Drawing, New York, pp. 155–166. Springer, Heidelberg (2004)
9. Gansner, E.R., Koutsofios, E., North, S.C., Vo, K.-P.: A Technique for Drawing Directed Graphs. IEEE Transactions on Software Engineering 19(3), 214–230 (1993)
10. Goodrich, M.T., Kobourov, S.G. (eds.): GD 2002. LNCS, vol. 2528. Springer, Heidelberg (2002)
11. Healy, P., Nikolov, N.S.: A branch-and-cut approach to the directed acyclic graph layering problem. In: Goodrich, M.T., Kobourov, S.G. (eds.) GD 2002. LNCS, vol. 2528, pp. 98–109. Springer, Heidelberg (2002)
12. Lutterkort, D., Peters, J.: Smooth paths in a polygonal channel. In: Symposium on Computational Geometry, pp. 316–321 (1999)
13. Myles, A., Peters, J.: Threading splines through 3d channels. Computer-Aided Design 37(2), 139–148 (2005)
14. Nikolov, N.S., Tarassov, A.: Graph layering by promotion of nodes. Discrete Applied Mathematics 154(5), 848–860 (2006)
15. Nikolov, N.S., Tarassov, A., Branke, J.: In search for efficient heuristics for minimum-width graph layering with consideration of dummy nodes. J. Exp. Algorithmics 10(2.7) (2005)
16. Sugiyama, K., Tagawa, S., Toda, M.: Methods for visual understanding of hierarchical system structures. IEEE Transactions on Systems, Man and Cybernetics SMC-11(2), 109–125 (1981)