An Interactive System for Drawing Graphs

Kathy Ryall[1], Joe Marks[2], and Stuart Shieber[1]

[1] Aiken Computation Lab
Harvard University
Cambridge, MA 02138, U.S.A.
E-mail: {kryall,shieber}@eecs.harvard.edu
[2] MERL
201 Broadway
Cambridge, MA 02139, U.S.A.
E-mail: marks@merl.com

Abstract. In spite of great advances in the automatic drawing of medium and large graphs, the tools available for drawing small graphs exquisitely (that is, with the aesthetics commonly found in professional publications and presentations) are still very primitive. Commercial tools such as Claris Draw or Microsoft's Powerpoint provide minimal support for aesthetic graph layout. At the other extreme, research prototypes based on constraint methods are overly general for graph drawing. Our system improves on general constraint-based approaches to drawing and layout by supporting only a small set of "macro" constraints that are specifically suited to graph drawing. These constraints are enforced by a generalized spring algorithm. The result is a usable and useful tool for drawing small graphs easily and nicely.

1 Introduction

Most small graphs (those with fewer than about 30 nodes) that appear in publications or presentations are still drawn with the aid of fairly primitive commercial drawing tools like Microsoft's PowerPoint or Claris Draw. Why do these tools not incorporate some of the advanced techniques that have been developed by either the graph-drawing or constraint-based-layout communities? One reason is that most graph-drawing algorithms cannot support the exquisite symmetries, spacings, and alignments that graphic designers utilize in professional-grade work. This kind of layout detail can be achieved in some constraint-based drawing systems, but the very general capabilities of such systems tend to make them cumbersome for the specific task of graph drawing.

Our system is based on constraints, but ones that are designed specifically for drawing graphs, not general graphics. These "macro" constraints, or *Visual Organization Features (VOFs)* [2], are listed in Figure 1; the application of each VOF is illustrated by "before" and "after" layouts. In our drawing tool, VOFs can be applied and removed interactively. Furthermore, the tool enforces syntactic constraints, such as preventing nodes overlapping other nodes and edges. The VOF and syntactic constraints are enforced by a generalized spring algorithm,

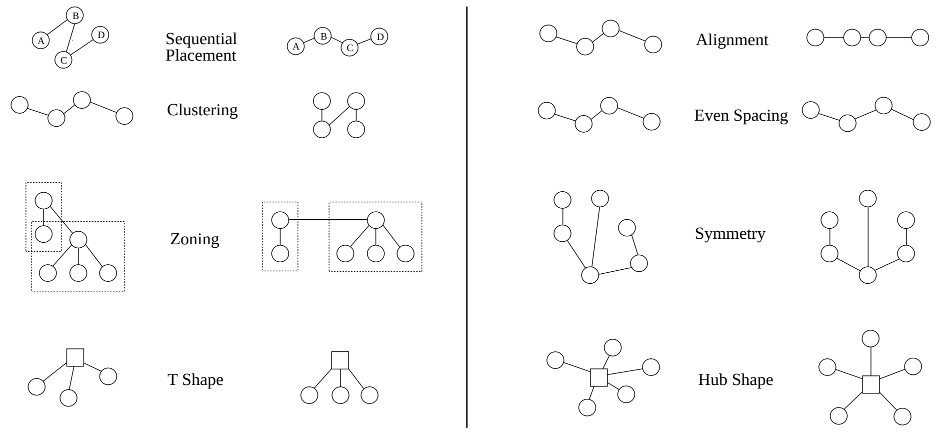

Fig. 1. Visual organization features (after [2]).

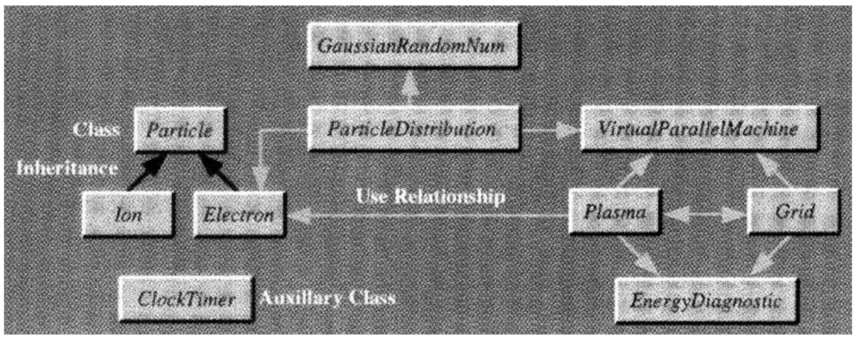

Fig. 2. Scanned diagram on which the example was based (from [3]).

an improved version of the one described by Dengler et al. [1]. We describe below how our tool can be used to replicate the drawing in Figure 2 [3]. The sample interactions shown illustrate the aptitude and convenience of our tool for drawing small graphs.

2 Example Interaction

Figures 3-9 show snapshots at various stages in the process of drawing a given graph. A screen dump of the entire interface to our system is shown in Figure 9; other figures show only the canvas area. The existence of active VOFs is indicated visually in our system by a user-responsive highlighting mechanism that

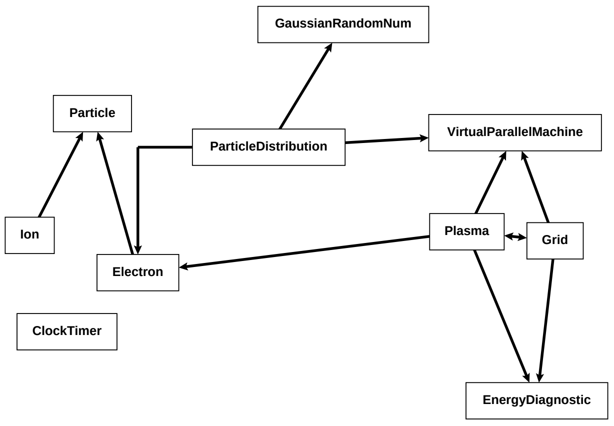

Fig. 3. The initial layout, with labeled nodes and edges.

cannot be replicated in static images. We therefore use shades of gray to indicate different VOFs in the figures.

The first step in the drawing process is typically to place the desired number of nodes in approximately the desired layout. To create a node, the user clicks the middle mouse button on the canvas area. Node characteristics (label, shape, font, background color, foreground color, border color, dimensions) can be modified via an edit dialog window. If a node's label becomes larger than the node itself, the system will automatically enlarge the node to accommodate its new label. Edges are added by dragging with the left mouse button from origin to destination node. Edges can be undirected, unidirectional, or bidirectional; they can also be direct or orthogonal. Figure 3 shows the layout after an initial node- and edge-placement phase.

To add a VOF to the layout, the user first selects a set of nodes using standard mouse techniques such as clicking or region-dragging. The user may then apply one or more VOFs to the set by pressing the appropriate push buttons, located on the right of the window in Figure 9. In Figure 4, the user has applied a horizontal *Alignment* VOF to the second, third, and fourth rows of nodes.

The system converts each VOF instance into a set of constraints, which it attempts to satisfy using a generalized spring algorithm. For example, an *Alignment* VOF mandates a set of zero-rest-length springs connecting each node to an axis-parallel line through the centroid of the points; a *Cluster* VOF places springs pairwise among the nodes with a short rest length. The physical simulation of the mass-spring model is continuously animated, indicating to the user the influence of the chosen VOFs. The user may move nodes and groups of nodes while the simulation proceeds in order to aid the system in finding better global solutions to the implicit constraint-satisfaction problem.

The use of a constraint-satisfaction scheme (mass-spring simulation) that

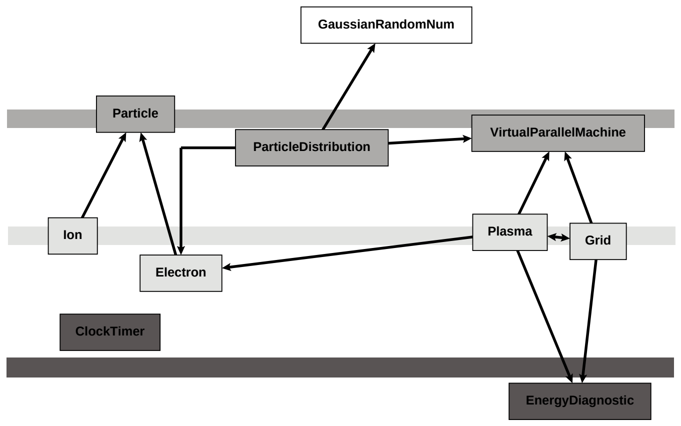

Fig. 4. User adds an *Alignment* VOF to each row.

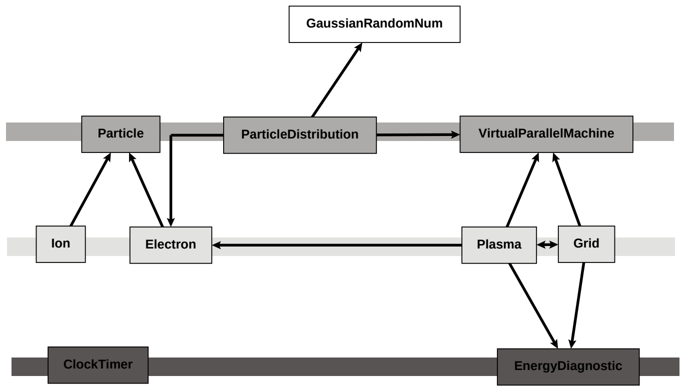

Fig. 5. A stable configuration, satisfying the *Alignment* VOFs.

is intuitive and predictable, rather than one better at finding global solutions, is deliberate. The tool is not intended to be good at globally satisfying the VOFs by itself. Rather, it is intended to provide an interface that allows a useful *collaboration* between user and computer in solving the layout problem. For this purpose predictability, simplicity, and the compelling nature of the animation are far more important than global optimality.

Figure 5 shows the stable configuration that ensues after the three *Alignment* VOFs have been satisfied.

Continuing with the example, the user adds three more VOFs, as illustrated in Figure 6. On the right, the user has added a *Hub Shape* VOF, indicated in

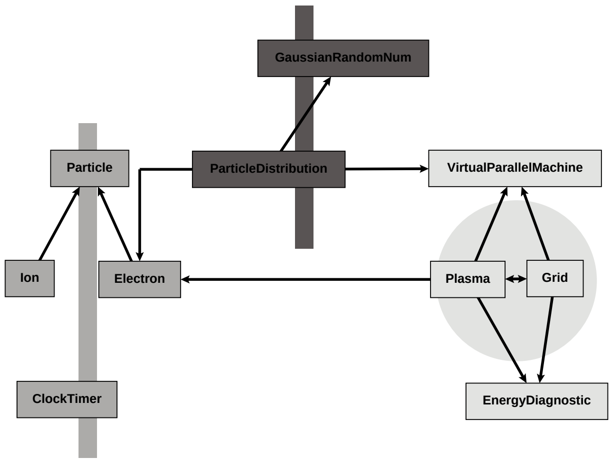

Fig. 6. The user adds three more VOFs: *Symmetry*, *Alignment*, and *Hub Shape*.

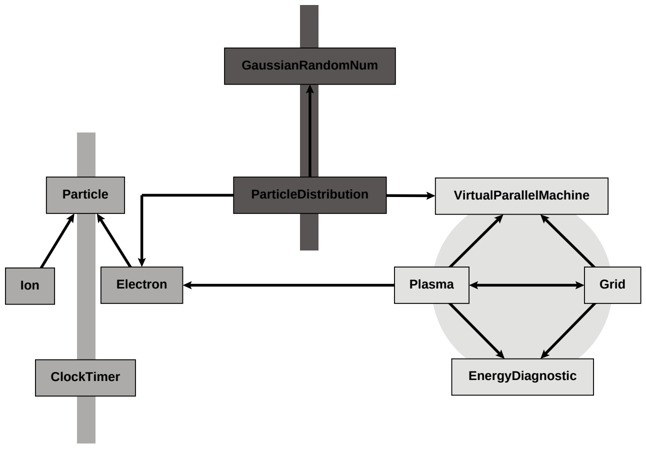

Fig. 7. The system satisfies both new and old constraints.

light gray. The center two nodes (dark gray) are to be vertically aligned. Finally, the four nodes on the left have had a *Symmetry* VOF applied to them.

Once again, the system converts each VOF instance into a set of constraints. It attempts to satisfy all constraints, both old and new, in determining node placement. Figure 7 shows the updated node positions. Each row is still aligned, and the new VOFs have been satisfied as well.

As a final step, the user adds text labels to the layout. Text labels are nodes

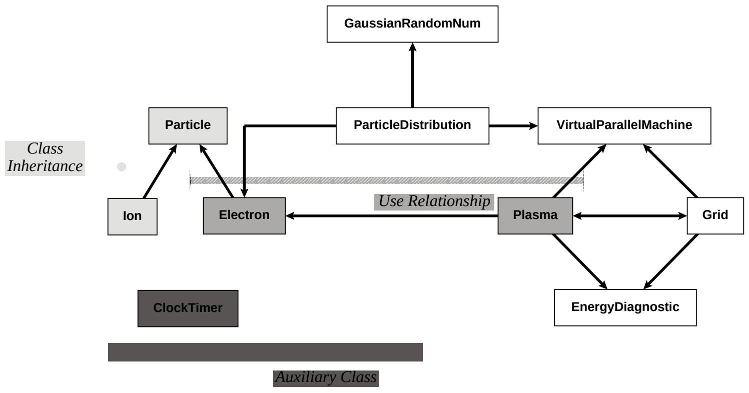

Fig. 8. The user adds three text labels, and three more VOFs: *Clustering*, *Even Spacing*, and *Alignment*.

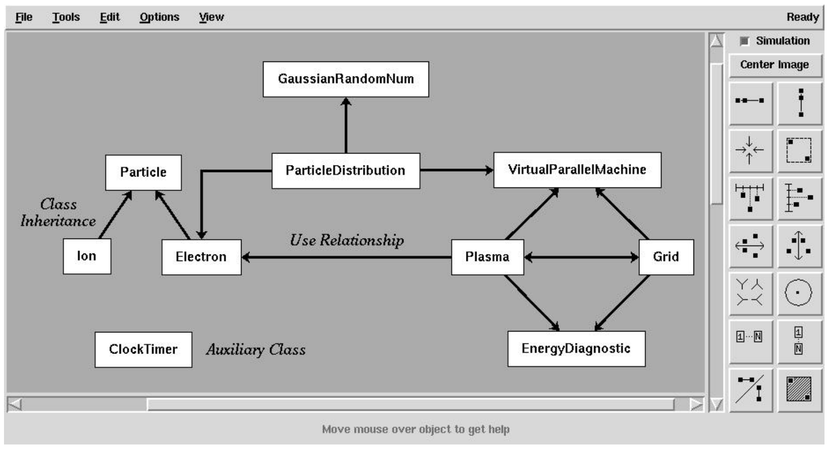

Fig. 9. The final layout (showing full system interface).

that have no background or border color, so that only the text string is visible. Any node can be designated a text label by selecting it and then pressing the *Text Label* push button. Text labels can also participate in VOFs, just like regular nodes. In Figure 8, the user has added three text labels, and applied a VOF to each one. The light-gray nodes on the left are subject to a *Cluster* VOF. The dark-gray nodes on the bottom are to be horizontally aligned. The middle three nodes, shaded medium gray, are to be evenly spaced. Figure 9 shows the final layout embedded in the system interface.

As an example of the flexibility of the system, Figure 10 presents successive snapshots of quiescent states of the interface as a user develops an alternative layout starting from the same initial layout (a) of the previous example graph. The user imposes a *T-shape* VOF on four nodes at the top. After the simulation stabilizes (b) the T-shape has been enforced. The user adds another *T-shape* VOF (c). The user moves the two root nodes of the T's roughly as indicated in (d) in order to invert the direction of the T's; the system maintains the T-shape constraints generating the layout in (e). The user swaps two nodes (f), and adds a *Vertical Alignment* and a *Symmetry* VOF (g), and two *Horizontal Alignment* VOFs (h). Finally, four nodes are specified to have *Equal Spacing* (i) to generate the final layout (j).

3 Conclusion

The VOFs supported by our system provide a natural and powerful vocabulary whereby users can express easily the desired characteristics of a graph layout, and the intuitive constraint-satisfaction method allows for a collaborative interaction between user and computer in solving graph-layout problems. An informal comparison with commercial drawing programs shows our system to be markedly superior for drawing small, aesthetic graphs.

References

1. E. Dengler, M. Friedell, and J. Marks. Constraint-driven diagram layout. In *Proceedings of the 1993 IEEE Symposium on Visual Languages*, pages 330–335, Bergen, Norway, August 1993.
2. C. Kosak, J. Marks, and S. Shieber. Automating the layout of network diagrams with specified visual organization. *IEEE Transactions on Systems, Man, and Cybernetics*, 24(3):440–454, March 1994.
3. C. D. Norton, B. K. Szymanski, and V. K. Decyk. Object-oriented parallel computation for plasma simulation. *CACM*, 38(10):88–100, October 1995. Figure 3.

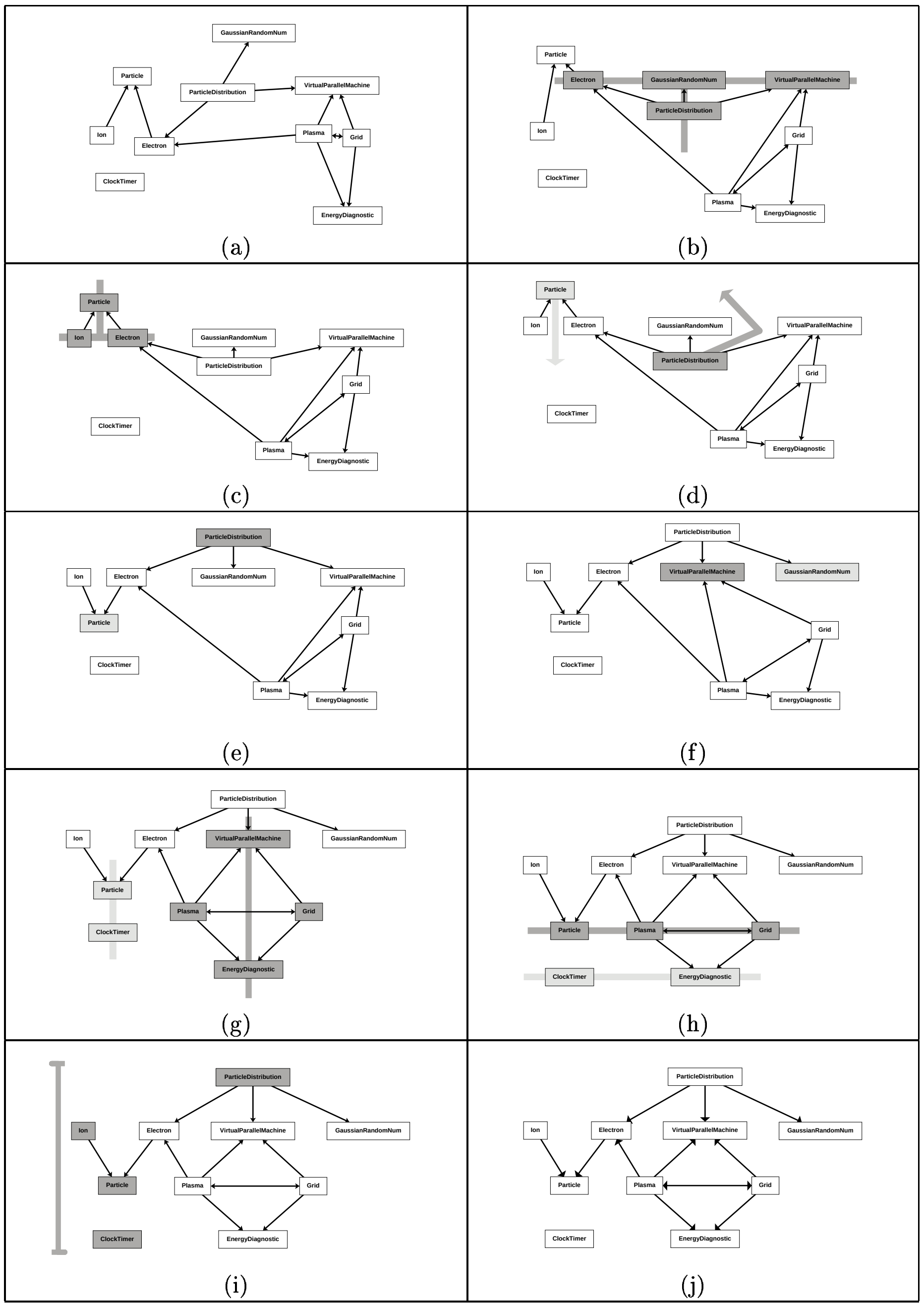

Fig. 10. Development of an alternative layout.