Constraint-driven diagram layout

Constraint-Driven Diagram Layout *

Ed Dengler
Univ. of Waterloo

Mark Friedell
Lotus Development Corp.

Joe Marks
DEC CRL

Abstract

Taking both perceptual organization and aesthetic criteria into account is the key to high-quality diagram layout, but makes for a more difficult problem than pure aesthetic layout. Computing the layout of a network diagram that exhibits a specified perceptual organization can be phrased as a constraint-satisfaction problem. Some constraints are derived from the perceptual-organization specification: the nodes in the diagram must be positioned so that they form specified perceptual gestalts, i.e., certain groups of nodes must form perceptual groupings by proximity, or symmetry, or shape motif, etc. Additional constraints are derived from aesthetic considerations: the layout should satisfy criteria that concern the number of link crossings, the sum of link lengths, or diagram area, etc. Using a generalization of a simple mass-spring layout technique to "satisfice" constraints, we show how to produce high-quality layouts with specified perceptual organization for medium-sized diagrams (10–30 nodes) in under 30 seconds on a workstation.

1 Introduction

The layout problem for *network diagrams* (also known as *node-link diagrams* and *circle-and-arrow diagrams*) is to assign to each node, n_i, x and y locations in the plane, n_i^x and n_i^y.

This problem has been investigated extensively and a variety of layout techniques have been proposed [1]. Most techniques attempt to create a good layout by optimizing according to one or more aesthetic criteria, such as diagram area, the number of crossing links, or the total length of the links; many techniques are specialized for networks that satisfy certain topological restrictions, e.g., trees or acyclic graphs.

The approach reported in [5] takes a different tack. In order to have a truly general layout technique, the

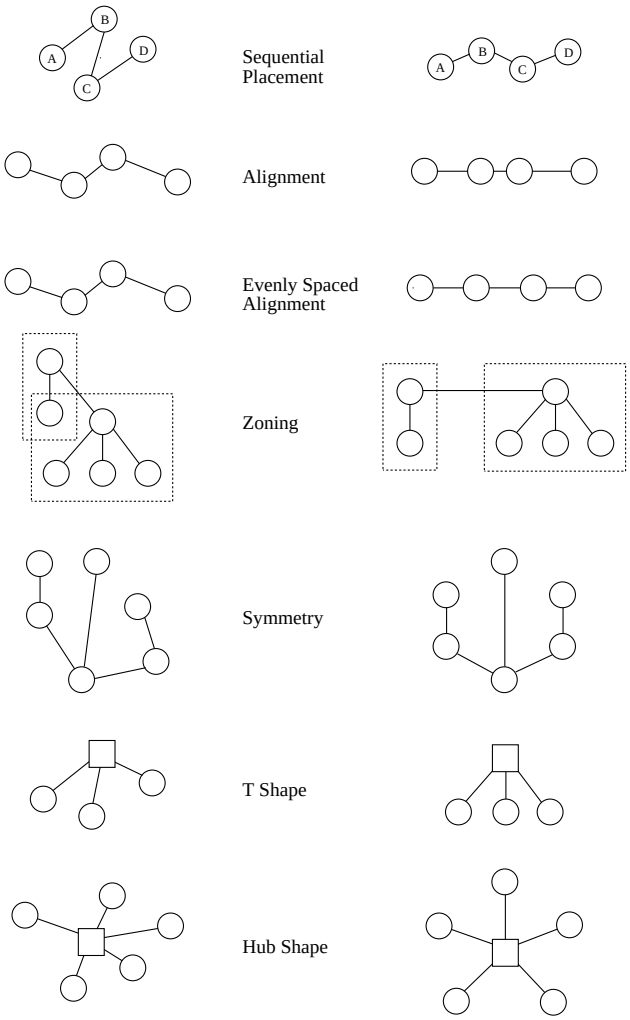


Figure 1: Visual Organization Features (VOFs)

*Reprinted from the *Proceedings of the 1993 IEEE Symposium on Visual Languages*, Bergen, Norway, August, 1993, pages 330-335.

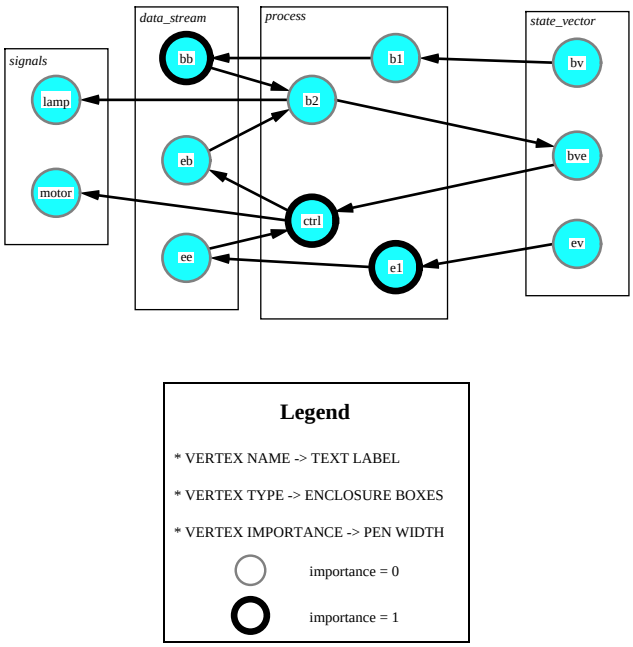


Figure 2: Zoning and symmetry VOFs illustrated

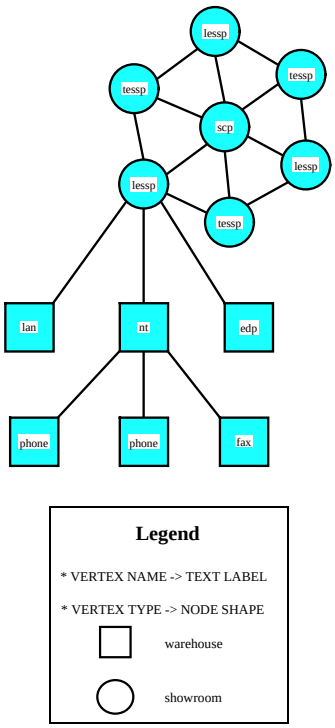


Figure 3: Shape-motif VOFs illustrated

topological characteristics of particular networks are not exploited. Furthermore, instead of optimizing relative to general aesthetics, Kosak et al. attempt to create a layout that exhibits the *Visual Organization Features* (VOFs) needed to convey effectively the message of the diagram. VOFs are arrangements of related nodes in the diagram; VOFs include horizontal and vertical alignment, axial and radial symmetries, various shape motifs (e.g., "T"-shaped and hub-shaped motifs), left-to-right and top-to-bottom sequential placement, and simple node proximity. As illustrated in Figure 1 (positive examples are shown on the right, negative examples on the left), VOFs are used routinely by human graphic designers and are indispensable in the creation of good diagram layouts [6, 8]. Three diagrams that exhibit various VOFs are shown in Figures 2, 3, and 4; these diagrams were laid out using the algorithm described here.

We believe the approach of Kosak et al. to be correct conceptually; however, it relies primarily on stochastic search and is implemented on a Connection Machine (a 4,096-processor CM-2), which can take several tens of minutes to produce a satisfactory layout.

The issue of where VOFs come from is not addressed here: we assume that they are specified by the user, or generated automatically by an intelligent diagram-design system.[1] The contribution of this paper is a layout technique incorporating VOFs that typically requires only a few tens of seconds to produce a good result on a DEC AXP workstation or similar computer. Our technique exploits the constraints imposed by a VOF specification to guide the search for a good layout.

2 The Layout Algorithm

The first attempt to compute layouts that exhibited the VOFs considered here used a spatial-grammar approach [5]. When this approach succeeded, the resulting layout was usually good, and was often produced quickly. Unfortunately, the layout rules were very specialized and highly interdependent. As a result, the system frequently failed to produce a satisfactory layout, and running time was sometimes very long as a result of extensive backtracking.

A second attempt involved a parallel genetic algorithm running on a Connection Machine [4]. This approach was very robust and almost always produced

[1] All aspects of the network diagrams included in this paper were designed automatically by the ANDD system [7].

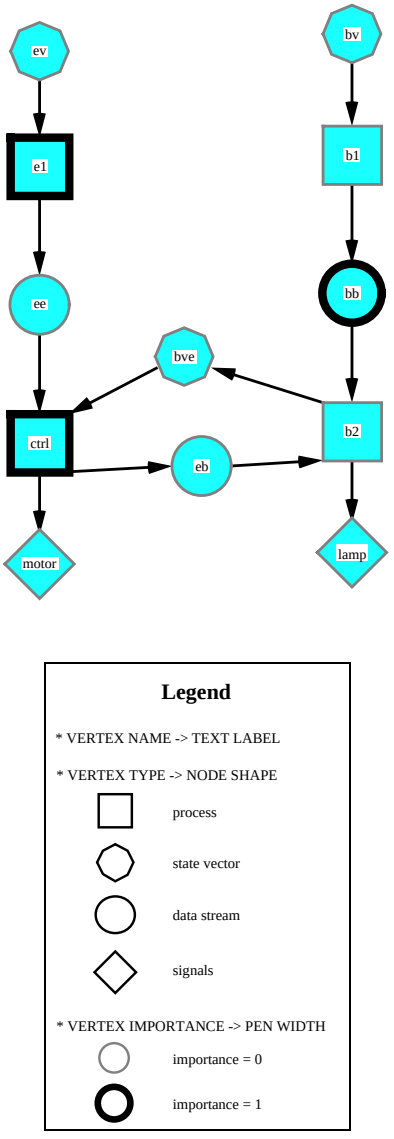


Figure 4: Alignment VOFs illustrated

a good layout. It did, however, depend on a rare and expensive compute engine, and it sometimes required several tens of minutes to produce a layout. We developed *constraint-driven diagram layout* in an attempt to combine, on a commonplace computing platform, the speed of the rule-based approach with the robustness of the genetic algorithm.

The underlying premise of constraint-driven diagram layout is that the VOF specification and certain aesthetic criteria can be stated as constraints that should be satisfied in the final layout. Furthermore, an additional premise is that the constraints derived from the VOFs and aesthetic criteria contain sufficient information to restrict the search for acceptable layouts considerably, thus permitting the rapid positioning of diagram elements in a satisfactory way. Our theory thus presumes that a good layout can be constructed quickly in the vast majority of practical cases, even though the general layout problem is known to be NP-complete [2].

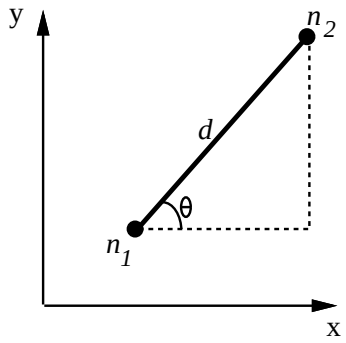
In our view of the problem, the set of N nodes in a diagram induces a high-dimensional *articulation space* with 2N dimensions, corresponding to the spatial attributes (x location and y location) of each node in the diagram's two-dimensional, Cartesian *layout space*. This formulation establishes a bijection between the set of possible diagram layouts and the set of points in the articulation space.

To evaluate the quality of a given layout, we form an objective function from the constraints. Each term in the objective function is itself a function, $c_j(n_0^x, n_0^y, \ldots, n_{i_{max}}^y)$, which returns a number that indicates the degree to which the jth constraint is satisfied (see Figure 5 for an example). The objective function,

$$LC = \sum_{j=0}^{j_{max}} \alpha_j c_j,$$

is then a measure of the degree to which the various constraints are satisfied, and, consequently, the degree to which the given layout varies from ideal. The α_j coefficients provide a means of expressing the relative importance of the various constraints. Our interest in producing functional, rather than simply good-looking layouts, leads us to weigh more heavily the terms that represent important VOFs, and to de-emphasize the terms corresponding to aesthetic criteria. This preference is reflected in the constraint-enforcement schedule described in Subsection 2.2.

A constraint c_j to maintain a minimum distance ϵ between nodes:



$$d = \sqrt{(n_2^x - n_1^x)^2 + (n_2^y - n_1^y)^2}$$

$$c_j(n_0^x, n_0^y, \ldots, n_{i_{max}}^y) = \begin{cases} 0 & \text{if } d \geq \epsilon \\ \epsilon - d & \text{if } d < \epsilon \end{cases}$$

$$\begin{aligned} \text{If } d = \tfrac{\epsilon}{2} \text{ then } \quad \Delta_{n_1^x}^{c_j} &= -\tfrac{\epsilon.cos(\theta)}{4} \\ \Delta_{n_2^x}^{c_j} &= \tfrac{\epsilon.cos(\theta)}{4} \\ \Delta_{n_1^y}^{c_j} &= -\tfrac{\epsilon.sin(\theta)}{4} \\ \Delta_{n_2^y}^{c_j} &= \tfrac{\epsilon.sin(\theta)}{4} \end{aligned}$$

Figure 5: Calculating difference corrections

2.1 Layout by Incremental Improvement

We use difference calculus to create a good layout by incrementally improving one that is chosen randomly.[2] If the value of c_j is the sum of unsigned displacements of the nodes in layout space from the nearest local minimum for c_j, then difference corrections for the $n_i^{a(=x \text{ or } y)}$, $\Delta_{n_i^a}^{c_j}$, can be calculated efficiently for most c_j. A simple example is shown in Figure 5.

Now consider a specific layout with $n_i^a = l_i^a$. Since

$$c_j\left(l_0^x, l_0^y, \ldots, l_i^a + \Delta_{n_i^a}^{c_j}, \ldots, l_{i_{max}}^y\right) \leq c_j\left(l_0^x, l_0^y, \ldots, l_{i_{max}}^y\right)$$

and

$$\left.\frac{\partial c_j}{\partial n_i^a}\right|_{\left(l_0^x, l_0^y, \ldots, l_i^a + \Delta_{n_i^a}^{c_j}, \ldots, l_{i_{max}}^y\right)} = 0,$$

a multidimensional, difference form of Newton's method can be used to incrementally drive LC towards 0. Let

$$\Delta_{c_j} = \left[\Delta_{n_0^x}^{c_j}, \Delta_{n_0^y}^{c_j}, \ldots, \Delta_{n_{i_{max}}^y}^{c_j}\right]$$

be the *correction vector* for constraint c_j, and define

$$\Delta_{LC}^k = \sum_{j=0}^{k} \Delta_{c_j}.$$

Given a point in articulation space, Φ, representing a given layout, we can compute an improved layout, represented by Φ', by

$$\Phi' = \Phi + \beta \Delta_{LC}^k. \quad (1)$$

This process may then be repeated to move towards an increasingly better layout, with β in Equation 1 being a scalar less than one which balances speed and robustness – a smaller value reduces the risk of divergence and oscillation, while a larger value increases the speed with which a good layout may be found. As discussed below, we advance k in Equation 1 from 0 to j_{max} as we incorporate increasingly more correction vectors during the process of incrementally improving the diagram layout.

2.2 Scheduling Constraint Enforcement

If all constraints are applied immediately and simultaneously, constraint-driven layout performs poorly. This is because the incremental search for a good layout tends to become "trapped" with LC at a local minimum very near to the starting configuration in articulation space. This difficulty can be avoided by enforcing the terms in the objective function according to a strategically devised schedule.

If we attempt to enforce a single constraint, we move directly towards the closest point in the articulation space at which the constraint is satisfied. To understand the interplay of enforcing multiple constraints, consider constraint A, which is very restrictive and satisfied in only a small number of small subspaces of the articulation space, and constraint B, which is very weak and satisfied in a large number of large subspaces.

What would happen if we enforced B, then enforced A? First, we would move directly to a point at which B was satisfied. We would then attempt to move directly to a point at which A was satisfied, but if we ever moved through a point at which B was violated, we would get "pushed back" towards the closest point satisfying B. We would only be able to satisfy A if we

[2] The approach we describe here can be thought of as a generalization of the simple "mass-spring model" of layout developed by Kamada and Kawai [3].

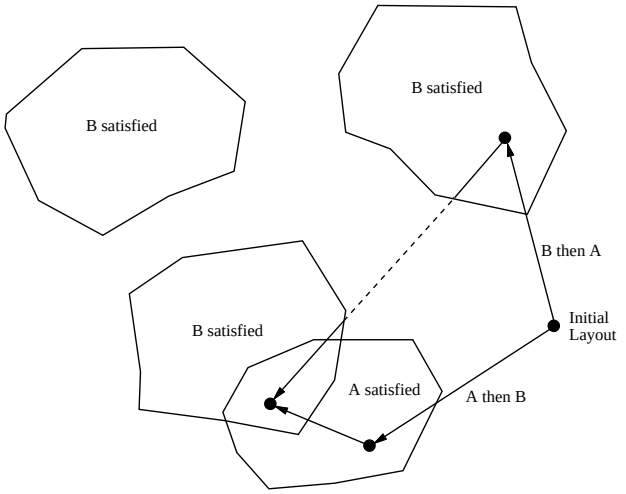


Figure 6: 2D cross-section of articulation space

could move to a point satisfying A through a region of points satisfying B. Such a path is unlikely to exist, because the regions of points that satisfy A are few and small. If, however, we satisfy A (the more restrictive constraint) first, we would have a better chance of moving to a point that satisfied B without passing through a point that violated A (since there are many big regions that satisfy B). These different search paths are illustrated in Figure 6.

Attempting to enforce A and B simultaneously is about as unlikely to be successful as solving B then A. We would attempt to move to the closest point that satisfied both A and B, but as soon as we passed though a region that satisfied either—most likely B–we would get trapped in that region.

In light of this phenomenon, we enforce constraints serially, in decreasing order of restrictiveness, by advancing k in Equation 1. Our sequence is:

1. separate nodes (this avoids degenerate solutions) and reduce edge lengths (to make layouts more compact);
2. form left-to-right and top-to-bottom sequences;
3. form horizontal and vertical alignments;
4. create "T" shapes;
5. create "hub" shapes;
6. form and separate zones;
7. establish horizontal symmetries and vertical symmetries;
8. establish bi-axial symmetries;
9. separate nodes from edges; and
10. uncross edges.

It is, of course, possible for constraint-driven layout to become trapped with LC at a local minimum, even if the constraints are applied in order of decreasing restrictiveness. Indeed, it is not uncommon to have a set of constraints which cannot all be satisfied, and for which some degree of "satisficing" is the best that can be achieved. We attempt to avoid the difficulties caused by local minima and mutually inconsistent constraints by creating several layouts from several initial, random configurations of nodes. We return the best layout found, as determined by LC. The speed of the incremental-improvement technique allows us to try several initial configurations in under 30 seconds on a workstation; the layouts shown in Figures 2–4 were the best that resulted from 50 independent trials, each of which took between 0.25 and 0.5 seconds to compute.

2.3 Uncrossing Edges

We do not know of a simple procedure for calculating a correction vector to uncross diagram edges. Instead, we use a highly constrained stochastic-search process. For each pair of nodes, n_1 and n_2, and edges, e_1 and e_2, such that

1. e_1 and e_2 cross;
2. n_1 is an endpoint of e_1 and n_2 is an endpoint of e_2, or n_1 is an endpoint of e_2 and n_2 is an endpoint of e_1;
3. for each zone, Z, n_1 is in Z $\Leftrightarrow$ n_2 is in Z;
4. for each symmetry group, SG, n_1 is in SG $\Leftrightarrow$ n_2 is in SG;
5. for each alignment, A, n_1 is in A $\Leftrightarrow$ n_2 is in A; and
6. for each sequence, S, n_1 is not in S and n_2 is not in S,

we swap the positions of n_1 and n_2 and re-evaluate LC to determine if the diagram was improved.

3 Results and Conclusions

Most attempts to design diagrams automatically focus on aesthetic criteria, such as the number of crossing lines or the total length of diagram edges. In our

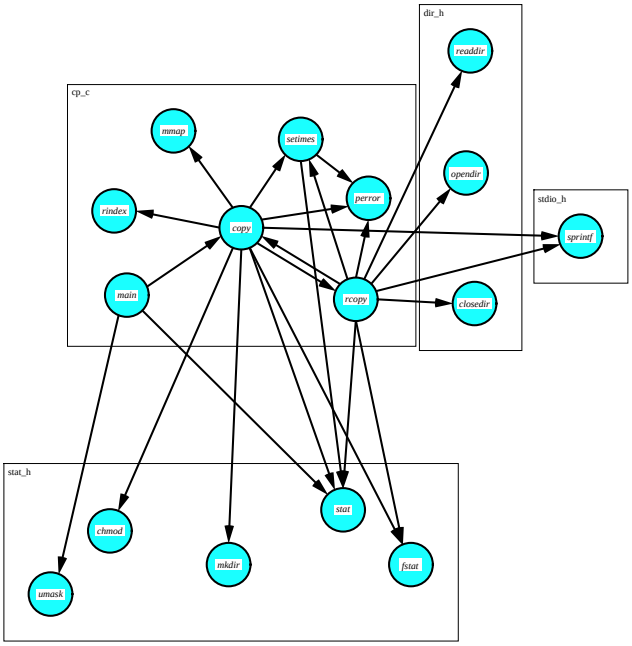


Figure 7: Calling relations for UNIX subroutines

work, we focus on creating structures in the diagram that convey efficaciously the diagram's message, treating aesthetics secondarily.

We have used this layout technique as a module in a system, ANDD (Automated Network Diagram Designer), for creating network diagrams that runs on a variety of commonplace workstations. It is composed of separate subsystems for two different tasks: design, which establishes a set of constraints on the appearance of the diagram [7], and articulation, which forms a diagram that satisfies the constraints. Layout is part of the articulation task. As an illustration of the performance of the ANDD system, the densely connected 16-node diagram in Figure 7 required 5 seconds for design and 30 seconds for articulation on a DEC AXP workstation; to compute the layout shown, 50 candidate layouts were generated, each taking less than 0.6 seconds to compute.

The diagram-layout problem is known to be NP-complete; however, we have developed a good solution to many practical diagram-layout problems. We recognize that our technique for incrementally improving randomly chosen layouts may, in theory, become trapped in unsatisfactory local maxima of layout quality as it searches for a good layout. In practice, however, this occurs only rarely. The risk can be reduced significantly by chosing the best of several alternative layouts developed from different initial random configurations.

In future work we hope to investigate the use of constraint-driven diagram layout as a means of improving user-drawn diagrams. We plan to derive the intended design of a diagram from the user's rough sketch, and to re-express it with our diagram-articulation techniques. By so doing, we would "clean up" the rough sketch and propose alternative design and articulation ideas.

4 Acknowledgements

This work benefited directly from discussions with Sandeep Kochhar, Computervision; Peter McMurry, Microsoft; Stuart Shieber, Harvard University; and Steve Sistare, Thinking Machines.

References

[1] P. Eades and R. Tamassia. Algorithms for drawing graphs: An annotated bibliography. Technical Report CS-89-09, Department of Computer Science, Brown University, October 1989. Revised version.

[2] M. R. Garey and D. S. Johnson. Crossing number is NP-complete. *SIAM Journal of Algebraic and Discrete Methods*, 4(3):312–316, September 1983.

[3] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31:7–15, April 1989.

[4] C. Kosak, J. Marks, and S. Shieber. A parallel genetic algorithm for network-diagram layout. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 458–465, UCSD, California, July 1991.

[5] C. Kosak, J. Marks, and S. Shieber. Automating the layout of network diagrams with specified visual organization. *IEEE Transactions on Systems, Man, and Cybernetics*, 1993. To appear.

[6] S. M. Kosslyn. Understanding charts and graphs. *Applied Cognitive Psychology*, 3:185–226, 1989.

[7] J. Marks. A formal specification scheme for network diagrams that facilitates automated design. *Journal of Visual Languages and Computing*, 2(4):395–414, December 1991.

[8] J. Marks and E. Reiter. Avoiding unwanted conversational implicatures in text and graphics. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI '90)*, pages 450–456, Boston, Massachusetts, August 1990.