Onion graphs: aesthetics and layout

Guttorm Sindre* Bjørn Gulla Håkon G. Jokstad

Faculty of Electrical Engineering and Computer Science
University of Trondheim, Norway

Abstract

In many engineering disciplines it is interesting to use diagrams which combine a graph notation with an onion notation. It is observed that such onion graphs lend themselves easily to some kinds of sophisticated tool support, but also that the need for tool support is significantly larger than for ordinary graphs. Most notably there is a need for supporting automatic layout to a much larger extent than for traditional graphs, particularly of the incremental kind. Some aesthetics and algorithms are outlined for the layout of onion graphs.

1 Introduction

Graphs — nodes connected by edges — are heavily used for visualization, for instance of engineering models. For hierarchical relationships, the *onion notation* [2, 6, 9][1] — putting nodes inside each other — is an alternative to using edges. Generally, since there are both hierarchical and non-hierarchical relations, a visual language might combine the onion and graph notations, yielding what we will call *onion graphs:* diagrams where nodes can be related to other nodes both by location and connecting edges.

There are basically three approaches to onion graphs:

- *The single onion approach:* The language uses the onion notation only for one kind of hierarchical construct. If there are several kinds of hierarchical relations, the other ones will have to be depicted in edge style. An example of this is shown in fig. 1, depicting part of a semantic data model for cars. Here, there are two kinds of hierarchical relations: aggregation and generalization: Wankel engines and cylinder-based engines are two subclasses of the more general class Engine. The example notation is similar to the data modeling language ERT [4]. To distinguish generalization from ordinary relations, a black circle is included to state that the two subclasses form a total partition of the superclass.
- *The Higraph approach:* This approach, suggested by Harel [2], models both generalization and aggregation in an onion fashion. However, the notation for aggregation is not a clean onion notation, achieved through the trick of dividing a node by dotted lines. The example of fig. 1(b) shows the same data model as that of (a).
- *The Hicon approach:* This approach [9] obtains a clean onion style for several hierarchical constructs at the same time by symbolic distinction of the nodes, e.g. squares for aggregation and circles for generalization, as in fig. 1(c), still showing the same data model. Of course, distinction does not have to be done by node shapes — it can also be indicated by features such as color, line types, or annotation.

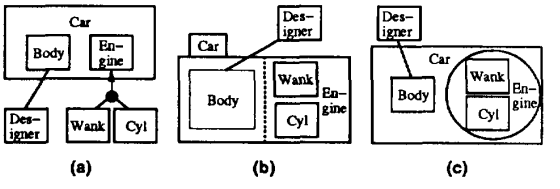

Figure 1: Three kinds of onion notation

For rest of the paper we will concentrate on single onion graphs, which is the least ambitious and thus most natural starting point for discussing aesthetics and layout algorithms. Within the NSR project[2]

*Direct responses on this article to Guttorm Sindre, UNIT-NTH/IDT, N-7034 Trondheim, NORWAY. Phone: +47 7 593469, Fax: +47 7 594466, Email: guttorm@idt.unit.no

[1] called "subgraphs" in [6]

[2] Norsk System Rammeverk, a technology transfer project between the Norwegian Institute of Technology and the three companies Metis, Sysdeco, and Taskon.

a prototype drawing editor has been developed for object-oriented data modelling [3], which is the basis for this investigation of aesthetics and layout. The rest of the paper is structured as follows: Section 2 discusses why the graph/onion notation has special needs when it comes to tool support, and in particular for automatic layout. To support automatic layout for combined onion graphs, some aesthetic criteria are needed, and these are discussed in section 3. Then, section 4 outlines some algorithms for automatic layout. Finally, section 5 gives some concluding remarks.

2 The need for automatic layout

In addition to improving the expressive economy and readability by reducing the number of edge symbols in a diagram [2], onion graphs have several other advantages over pure graphs in their possibilities for sophisticated tool support:

- *information hiding:* nodes that become too small to be visible, or too big to fit into the picture, will implicitly be hidden from the diagram. The issue of minimal display size has also been discussed for pure graphs [7], the information hiding effect obtained by this called "automatic holophrasting". Onion graphs provide a more natural framework for this kind of functionality: you *know* where to zoom in (i.e. into nodes).
- *manipulation:* merely placing and moving nodes may assert and change hierarchical relationships. This makes possible an appealing drag-and-drop of functionality to edit hierarchical structures.

These advantages result because the onion notation, unlike the graph notation, puts meaning into a nodes location. However, this also results in some problems. If you want to insert a node in a hierarchical structure, *it has to fit in.* If there is not enough room inside the intended super-node (e.g. because there are already lots of other nodes inside it), the diagram has to be rearranged before the new node can be correctly coupled to the model. As an example of this, consider fig. 2. Imagine here that the node **E**, which we are either creating or moving from somewhere else, is supposed to go into the node **A**. However, there is no room in **A**, and additionally, there is not enough room around **A** to extend it without first moving something else. In a large diagram, the simple move or insert of a node might induce the need for a cascade of diagrammatic rearrangements.

Of course, similar problems might occur for pure graphs — you may want the node **E** to be close to **A**, but there is no room in the vicinity of **A**. However, in a pure graph it is always possible to place the node further away and connect it to **A** with an edge. Although the layout will not be the best, you are at least able to build the wanted conceptual structure. With onion graphs, on the other hand, layout problems may make it impossible to get the modeling done at all. For large models, then, there is a danger that the user will have to spend a prohibitive amount of time rearranging the diagram, even to make possible minor conceptual changes. For many applications, models are built iteratively, with frequent changes, and then, such inefficiency in modeling because of an inferior drawing tool will clearly be unacceptable.

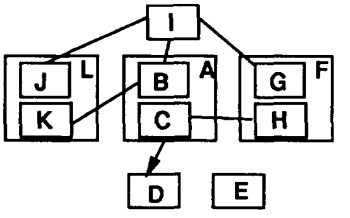

Figure 2: The space problem

Hence, it seems that support for automatic layout, which is important also for pure graphs, is even more important for onion graphs. Whereas much work has been done on establishing aesthetics and automatic layout algorithms for pure graphs (e.g. [1, 10, 11]), hardly anything has been done for onion graphs. The only published work we have been able to track down is [6], which touches upon the problem in the discussion about "subgraph abstractions". However, the work on subgraph abstractions does not elaborate on the case that there may be non-hierarchical relations as well, connected to sub-nodes of an onion. There also exists an editor for SDL diagrams with optional automatic layout facilities [8], but here only two levels of the onion graph are shown at any time.

Since the need for automatic layout in an onion graph can result from small and frequent conceptual changes, *incremental layout* becomes particularly important — if the diagram is totally rearranged each time, the user will easily lose his orientation in a large model. The incremental layout support should typically perform the necessary expansion of a node when something is to be inserted where there is not enough room, and similarly contraction when we get surplus

AREA	minimize the area occupied by the drawing.
BALAN	balance the diagram with respect to the vertical or horizontal axis.
BENDS	minimize the number of bends along the edges.
CONVEX	maximize the number of faces drawn as convex polygons.
CROSS	minimize the number of crossings between edges.
DEGREE	place nodes with high degree in the center of the drawing.
DIM	minimize differences among nodes' dimensions.
LENGTH	minimize the global length of edges.
MAXCON	minimize length of the longest edge.
SYMM	symmetry of sons in hierarchies.
UNIDEN	uniform density of nodes in the drawing.
VERT	verticality of hierarchical structures.

Table 1: A taxonomy of pure graph aesthetics

space from removing something. Although automatic contraction might not seem as pressing as expansion, there should be support for both — if only expansion is supported, the diagram size will grow from modifications even if the complexity of the model is fairly constant.

3 Aesthetic considerations for onion graphs

3.1 Aesthetics

In this section we will first review the aesthetics presented for pure graphs, and then discuss how these can be extended to deal properly with onion graphs. An extensive list of pure graph aesthetics is presented in [11], reproduced in table 1.

With respect to this list, we make the following observations: **AREA**, **BALAN**, **BENDS**, **CROSS**, and **LENGTH** can have the same meaning for an onion graph as for a pure graph, only with the extension that they can also be applied recursively to subgraphs (i.e. the graph located within one particular node).

DIM is clearly useless for onion graphs. **CONVEX** and **UNIDEN** are not directly applicable either, but could possibly be used level-wise within subgraphs, yielding two new candidates for aesthetics:

S-CONVEX within all subgraphs maximize the number of convex faces.

S-UNIDEN all subgraphs should have a uniform density of nodes.

DEGREE could still be applicable to an onion graph. But there are now several alternative ways to compute the degree. One could count only edges, include the number of direct sub-nodes, or the total number of sub-nodes down to the atomic level. The latter seems particularly interesting because the most heavily decomposed nodes are likely to be the most frequently visited for navigation. Moreover, heavily decomposed nodes are likely to be the biggest, and thus the diagram may be most pleasing to look at with these in a rather central position. An alternative to degree could even be to place the largest node in the middle, regardless of degree.

SYMM and **VERT** deal with hierarchies — but with respect to a pure graph notation. If there are hierarchical relations depicted in a graph style (for instance, if we have used onions for generalization and are showing aggregation by trees), **SYMM** and **VERT** may be interesting for these structures.

In addition to the aesthetics listed above, it could also be interesting to include some additional aesthetics specifically for the onion notation:

DISTANCE: Also in pure graphs, it might be requested that nodes should not be too close to each other. In an onion graph, however, such a requirement must always be taken relative to a node's size in a particular picture.

VISIBILITY: Nodes outside a certain size range (either below or above) should not be depicted in the current view. The visibility aesthetic could also be defined with respect to how many levels of an onion it is useful to depict at the same time, or possibly apply both kinds of considerations.

As observed in [11], real life applications will usually have to consider more than one aesthetic, and drawings which are optimal with respect to one aesthetic are generally not optimal with respect to another one. Thus, an algorithm should be able to balance various aesthetics according to user priorities.

3.2 Constraints

In addition to the aesthetics presented in table 1, [11] also lists some constraints which it can be interesting to apply to a layout, reproduced in table 2.

All these constraints might be interesting in various cases — these will be input by the person drawing a diagram, and we will not discuss them in more detail.

CENTER	place a given set of nodes in the center of the drawing.
DIMENS	assign the dimension of the symbols representing specified nodes.
EXTERN	place specified nodes on the external boundary of the drawing.
NEIGH	place a group of nodes close together.
SHAPE	draw a part of the graph with a pre-specified shape.
STREAM	place a sequence of nodes along a straight line.

Table 2: A taxonomy of pure graph constraints

If anything they will result in a less "ideal" drawing from a purely aesthetic point of view, but really simplify the work for the layout algorithm — since the solution space is reduced. Thus, we will not discuss these constraints in any more detail.

The most interesting constraint for the incremental layout of onion graphs would be

STABIL: maintain as much as possible of the diagram topology in the light of changes.

Since the goal of this constraint is to help the user maintain his orientation in the diagram, it seems most important to maintain the relative direction between nodes, followed by the relative distance, whereas the relative size of the nodes does not seem to be so important, i.e. the priorities are *1) direction, 2) distance, 3) size.*

4 Outline of algorithms

4.1 Fresh layout

We will now discuss how existing algorithms for pure graphs can be used iteratively/recursively (up and down the onion structures) for onion graphs. First of all, it can be observed that

- many layout algorithms for general graphs — at least those which yield nice results — are of order n^3 or even $n!$, and thus rather costly for large diagrams.
- thus, a common goal in layout algorithms is to partition the problem in smaller subproblems — to reduce n. This is discussed for purely hierarchical graphs in [5]. One notable advantage of onion graphs in this respect is that they provide a natural partitioning of the problem through the decomposition structure.

```
iterate down:
    for level := top+1 to bottom
        for all subgraphs at level
            apply ordinary layout to subgraph,
            taking into consideration constraints
            from the layout at the level above;
iterate back up:
    for level := bottom-1 to top
        for all subgraphs at level
            try to adapt to change requests
            from the level below,
            as well as optimizing own layout;
            identify all "intolerable"
            constraints from above;
total layout:
    repeat C times
        iterate down;
        iterate back up;
    [iterate down]; { may possibly
      want to go down the last time}
```

Figure 3: Iterative algorithm for fresh layout of onion graphs

Considering one particular subgraph, nodes as black boxes, layout can be computed using algorithms for general graphs. What we need to do in addition is to iterate down and up the hierarchical onion structure, in a way similar to what is suggested in [10] for levels in a pure hierarchical graph. An algorithm for the layout of the whole graph can then be outlined as shown in figure 3.

For a graph with N nodes, the layout for a pure graph would be $O(n^3)$. For an onion graph we will have M subgraphs, each with an average of N/M nodes, and we iterate C times, roughly resulting in $C \cdot M \cdot (N/M)^3$, which is somewhat optimistic since nodes are generally not evenly distributed on subgraphs. Usually, one will not iterate more than 2–3 times, since the improvement in the layout on subsequent iterations tends to be marginal. As an example, let's assume we have a model with 100 nodes and 20 subgraphs, which gives an average of 5 nodes in each subgraph.[3] If we iterate, say, 3 times, the onion graph layout requires approximately $3 \cdot 20 \cdot 5^3 = 7,500$ executions, which — although a little optimistic — is a major improvement on the $100^3 = 1,000,000$ executions required by the corresponding, un-partitioned layout of a pure graph.

[3] In many cases this seems to be a reasonable average, for instance, a usual guideline for DFDs is that every decomposition of a process should introduce 4–7 subprocesses.

4.2 Incremental layout

As observed earlier in the paper, the need for automatic layout will often occur due to minor logical modifications of a model, and thus, support for incremental layout is essential. As opposed to fresh layout, adhering to the original topology then becomes a central priority. The following simple method takes well care of this:

- *Method 1:* if there is insufficient space for an insertion within a node, all sub-nodes of that node (possibly including the one to be inserted) are decreased by the scale needed to create the necessary space.

This algorithm is simple and maintains a stable diagram, but it wastes a lot of space. Scaling down all the sub-nodes of one particular node is diagrammatically equivalent to scaling up the whole rest of the diagram. To waste less space we might want to preserve the original topology only in terms of the relative *position* of the top level nodes in the diagram:

- *Method 2:* If there is insufficient space for an insertion inside some node N, this N will be expanded by scaling up[4]. The siblings of at all sides[5] are pushed away so that their position relative to N's boundary is the same as before.[6]

This method is better than the previous one in most cases, but includes heavier computation. A more detailed discussion of algorithms for incremental layout is beyond the scope of this paper, and is a topic for further work.

5 Conclusions

It has been pointed out that onion graphs have many useful features for visualizing complex models with hierarchical relations, but that such graphs require sophisticated tool support for modeling to be effective, particularly in terms of automatic layout support. Whereas much work has been done for the layout of pure graphs, almost nothing has been published on onion graphs. This paper has discussed aesthetics and layout for onion graphs, elaborating on previous work on ordinary graphs. Incremental layout is seen to be particularly important since the need for rearrangements will often occur after minor conceptual changes. Some of the functionality discussed in this article has been implemented in a limited diagramming prototype in the SEROO project [3], but the most sophisticated features still remain to be realized.

[4] or if possible by stretching.

[5] or on the stretched side

[6] To be precise, the computation should be based on the point of N's boundary which intersects the straight line from the center of the sibling to the center of N.

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, Providence, RI 02912, October 1989.

[2] D. Harel. On visual formalisms. *Communications of the ACM*, 31(5):514–530, May 1988.

[3] H. G. Jokstad. Diagramverktøy for SEROO. Technical report, Faculty of Electrical Engineering and Computer Science, University of Trondheim, December 1992. REBOOT-STP TR 91.

[4] P. McBrien, A. H. Seltveit, and B. Wangler. An entity-relationship model extended to describe historical information. In *Proc. CISMOD'92*, July 1992. Bangalore, India.

[5] E. B. Messinger. *Automatic Layout of Large Directed Graphs*. PhD thesis, University of Washington, Department of Computer Sciences, July 1989. TR Number 88-07-08.

[6] F. Newbery Paulisch and W. F. Tichy. Edge: An extendible graph editor. *Software—Practice and Experience*, 20(S1):63–88, June 1990.

[7] S. P. Reiss and J. N. Pato. Displaying program and data structures. In *Proceedings of the Twentieth Annual Hawaii International Conference on System Sciences*, pages 391–401, January 1987.

[8] *SDL'91 Evolving Methods*, Amsterdam, 1991. North Holland. Proceedings of CCITT SDL Forum, Glasgow, Scotland, Sept. 29–Oct. 4.

[9] G. Sindre. *HICONS: A General Diagrammatic Framework for Hierarchical Modelling*. PhD thesis, Faculty of Electrical Engineering and Computer Science, University of Trondheim, 1990. NTH 1990:44, IDT 1990:31.

[10] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-11(2):109–125, February 1981.

[11] R. Tamassia, G. Di Battista, and C. Batini. Automatic graph drawing and readability of diagrams. *IEEE Transactions on Systems, Man and Cybernetics*, 18(1):61–79, January 1988.