March 2020

Graph Visualization by Organizing Connections in Collapsible Hierarchical Graphs

David Souther

Alan Samanta

Mateusz Jedruch

Matthew Tschiegg

Szymon Barglowski

Graph Visualization by Organizing Connections in Collapsible Hierarchical Graphs

ABSTRACT

Network graphs in certain applications, e.g., cloud-network graphs, have connections in multiple dimensions. At present, it is difficult or inconvenient for a user to visualize such graphs at varying levels of granularity or hierarchy. Per the techniques of this disclosure, a hull is defined as a node with descendants, and a segment is defined as a bundle of edges between descendants below a pair of nodes. By enabling a user to expand or collapse a hull, and by routing edges via segments connecting parent nodes, the described techniques enable a high-level visualization of large graph networks that can be quickly refocused into low-level pictures.

KEYWORDS

- Graph visualization
- Hierarchical graph
- Multi-layer graph
- Collapsible graph
- Multi-dimensional graph
- Graphical representation

BACKGROUND

Network graphs in certain applications have connections in multiple dimensions. For example, cloud-network graphs have connections in two dimensions, e.g., communication and configuration. The communication dimension is represented by a low-connectivity (sparse), directed graph structure. The configuration dimension is represented by an aggregation of resource types under a certain parent type. This creates a compound graph, with every node possibly having a complete graph below it. Additionally, there can be communication between

child nodes of two separate parents. At present, it is difficult and inconvenient for a user to visualize such graphs at varying levels of granularity or hierarchy. Displaying this multi-layer graphical information in a clear and concise way is a problem that, if resolved, can improve user efficiency.

Existing layout algorithms treat the hierarchical compound grouping as a hinting constraint, applying the layout algorithm on all nodes with no children. This results in a single, static image of considerable complexity. Expanding nodes (adding additional child nodes later) has the chance to drastically change the results of the layout algorithm, which can cause user frustration.

DESCRIPTION

This disclosure describes techniques of expandable tree representations of network-configuration hierarchies. Rather than laying out the lowest level of nodes using parent/child relationships as grouping hints, layout proceeds on the top-most level of the compound graph using the expandable tree representations, known as hulls. Expanding nodes only changes the size of the node, which does not affect its position in the graph. Edges are not drawn directly between child-nodes; instead, they are routed up the parent tree to the top level before crossing once to the destination's top-most parent and down the side of the destination tree. A consistent layout of top-level nodes results, which shows edges between low-level nodes running up to the parent, over to a destination parent, and down to their targets. The hulls can be circular or of any other geometric shape.

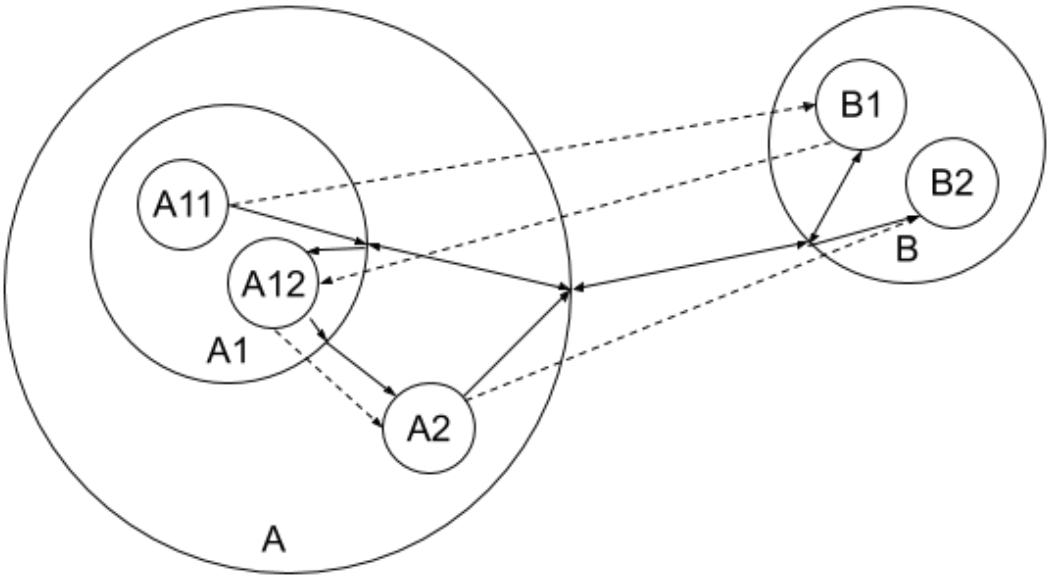

Fig. 1: Elements of a collapsible, hierarchical graph

Fig. 1 is an example that illustrates the elements of a collapsible, hierarchical graph, per techniques of this disclosure. The **nodes** of the graph are A, A1, A11, A12, A2, B, B1, and B2. In multi-layer cloud networks, nodes can represent individual entities such as load balancers, workloads, compute regions, interactive groupings, etc.

Children (or **descendants**) are collections of nodes that are in a hierarchical relationship to a **parent** (or **ancestor**) node. In the example of Fig. 1, the ancestors of node A11 (and of A12) are nodes A1 and A. Similarly, the ancestor of node A1 (and of node A2) is node A, and the ancestor of node B1 (and of node B2) is B. The descendants of A are nodes A1, A2, A11, and A12, and the descendants of node A1 are nodes A11 and A12. The descendants of node B are nodes B1 and B2.

Hulls are containers for children nodes, and may themselves be a node. Hulls can be expanded to view their children and those children's discrete communication segments, or collapsed to reduce the complexity of relatively uninteresting areas of the graph. The hulls of the

graph of Fig. 1 are nodes A (which includes nodes A1 and A2), A1 (which includes nodes A11 and A12), and B (which includes nodes B1 and B2).

Edges are directed connections between nodes of a graph. Edges can be between nodes at any level of the graph child hierarchy. The edges of the graph of Fig. 1 are A11-B1, B1-A12, and A2-B2. **Segments** are bundles of edges between descendants below a pair of nodes, representing the aggregate of edges between a pair of entities in the graph. The segments of the graph of Fig. 1 are as follows:

1. A to B containing A11→B1, B1→A12, A2→B2.
2. A1 to A continuing to B, containing A11→B1, B1→A12.
3. A1 to A2, containing A12→A2.
4. A2 to A continuing to B, containing A2→B2.
5. A11 to A1 continuing to A1 to A continuing to B, containing A11→B1.
6. A12 to A1 continuing to A1 to A continuing to B, containing A11→B1.
7. A12 to A1 continuing to A2, containing A12→A2.
8. B1 to B continuing to A, containing A11→B1, B1→A12.
9. B2 to B continuing to A, containing A2→B2.

```
digraph {
  A { A1 { A11; A12;} A2; }
  B { B1; B2; }

  A11 -> B1;
  B1 -> A12;
  A12 -> A2;
  A2 -> B2;
}
```

Fig. 2: Pseudocode representation of a graph

The graph of Fig. 1 can be represented by the example pseudocode of Fig. 2. Per the

techniques of this disclosure, a graph renderer accepts as input code representations of graphs to produce visualizations of collapsible, hierarchical graphs. The graph renderer can apply sound aesthetic and design choices and provide rich interactive experiences.

Aesthetic and design choices can include overall node layout, edge drawing, label positioning and style, and interaction state styling. Interactive experiences include exploring the graph by expanding into and collapsing nodes that have children, dragging nodes to reposition them, and highlighting or searching nodes as specified by the client. The visual display styles of graph elements are specified by the properties of nodes, hulls, and segments. A client can select, or mark, one or more nodes as points of ongoing interest, hover transiently over an entity to mark it as a point of current interest, etc. These interactions, e.g., expand/collapse, select/unselect, hover/unhover, drag, etc., are exposed as events from the elements comprising the graph.

While performing interactions with the client, the underlying entities transition through combinations of various states, such as:

- **(Soft) Filtered** indicates that an element is relatively uninteresting given the current set of filters applied.
- **Search (Matched)** shows an element as matching a query for a specific item or query.
- **Search (Focused)** shows a single matched element as the primary choice within all matched elements.
- With two sets of data, **Diff Presence** indicates whether the element was in only the left data set, the right data set, or both data sets.
- **Selected** (and **Multi-selected**) indicate one or more entities are currently marked by the operator for an ongoing investigation.
- **Hovered** indicates that the operator has transiently focused on an element or an element

in its nearby neighborhood.

The client code sets these properties in response to operator actions from graph events.

The interplay between the graph and the client sets up a circular data flow. The client provides graph structure and property information. The operator interacts with the graph. The graph generates events. The client responds to those events by updating structure and properties.

Segment tracking

When edges are updated, the ancestor chain of the source and the destination nodes are iterated over. At each level, the segment is created or updated to point to the edge as well as to the next segment in the ancestor chain.

Expansion tracking

Expansion tracking refers to maintaining an account of which nodes are in an expanded state and/or rendered as hulls. In one approach, the client passes to the renderer the set of nodes it would like to have rendered. In this approach, the client creates a copy of its graph data, removes all nodes that are children of unexpanded parents, and keeps the expansion state disjoint from the renderer. This requires the presence of a can-expand property on the node component, e.g., for the renderer to know whether to show an affordance of expansion. In this approach, the tracking of segments is somewhat cumbersome, since there is no requirement to have edges at every level of the hierarchy. For example, for the graph: `digraph { A { a1; a2; } B { b1; b2; } a1→b1; b2→a2; }`, with no nodes expanded, a proposed reduction step must create a synthetic edge between A and B. This edge needs to include the original set of edges, and becomes a complex replica of the segment processing logic. When, for example, A is expanded, synthetic edges for a1→B and B→a2 need to be created.

Per the techniques of this disclosure, the expansion state is tracked with the structure of the graph data itself. This requires no additional logic for managing what segments are to be included or are not to be included in the rendered structure. This moves all visibility decisions to be exclusive at render time, rather than a complex aggregation of underlying primitives. This state can be captured as a property on a node or via a layout manager.

In this manner, the techniques of this disclosure enable high-level visualization of large graph networks that can be quickly refocused into low-level pictures.

CONCLUSION

Network graphs in certain applications, e.g., cloud-network graphs, have connections in multiple dimensions. At present, it is difficult or inconvenient for a user to visualize such graphs at varying levels of granularity or hierarchy. Per the techniques of this disclosure, a hull is defined as a node with descendants, and a segment is defined as a bundle of edges between descendants below a pair of nodes. By enabling a user to expand or collapse a hull, and by routing edges via segments connecting parent nodes, the described techniques enable a high-level visualization of large graph networks that can be quickly refocused into low-level pictures.

REFERENCES

[1] Sander, Georg, “Layout of compound directed graphs,” https://publikationen.sulb.uni-saarland.de/handle/20.500.11880/25862;jsessionid=2F73F6FC224140E0A2B1E1878EDDEA31 accessed on Jan. 19, 2020.

[2] https://www.graphviz.org/