Visualization of Lattice Structure

Shengjie Wang
wangsj@uw.edu

ABSTRACT
Lattice is a special form of graph with a start node and a terminal node, and data are encoded on the paths from the start to the terminal. The key feature of lattice is common edges, which are the edges shared by multiple data-encoding paths. Lattice is widely used in the fields of speech recognition, natural language processing and bioinformatics, as common edges can be applied to compress data, to speed-up algorithms, and to discover common structures within the data. In this paper, we focus on visualization of the lattice structure, with emphasis on the common edges. In particular, we aim to build a system, which takes any lattice as input, displays the lattice structure effectively, and supports various interactions for users to either explore or edit the lattice. With such system, we wish to gain some intuition about how to build the optimal lattice from raw data, which is still an unsolved problem.

Author Keywords
Visualization, Lattice, Graph, Interaction.

ACM Classification Keywords
Visualization [: Graph Visualization and Interaction]

INTRODUCTION
Lattice is a special form of graph with a start node and a terminal node, and data are encoded on the paths from the start to the terminal. The key feature of lattice is common edges, which are the edges shared by multiple data-encoding paths.

Figure 1 shows an example of a simple lattice, where we encode the data of two words, namely, "seattle" and "seahawk", into the paths from the start node "S" to the end node "T". The shared edge, which is colored in red, resembles the shared prefix of the two words: "sea".

Please note that lattice is different from prefix/suffix tree structure, as such common edges can appear anywhere on a path, not necessarily in the front part (prefix) or in the end part (suffix). In general, lattice is a more generalized structure to capture the common parts in the data.

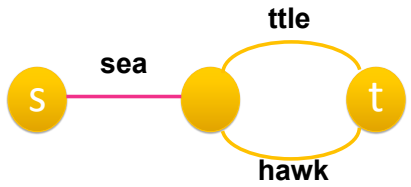

Figure 1. ***Lattice Example*** **– a simple lattice, which encodes the data of two words: "seattle" and "seahawk". The prefix "sea" is a common edge. "S" denotes the start node and "T" denotes the end node.**

Common edges are of strong interests for the following reasons.

1. Common edges indicate the common structures shared by the data.
2. Common edges compress the data, as we encode the duplicated information only once.
3. Common edges can be applied to speeding-up algorithms in a dynamic programming manner, as we do not need to apply our algorithms same edge multiple times.

For the above three characteristics of common edges, lattice is a widely-used structure in the fields of speech recognition, natural language processing, and bioinformatics. It is so strong a structure that we believe visualization of lattice can be extremely useful, both for better understanding of the lattice, and for applications in the fields mentioned above.

In this paper, we will focus on visualization of lattice structure, with emphasis on the common edges. In particular, we aim to build a system, which takes any lattice as input, displays the lattice structure effectively, and supports various interactions for users to either explore or edit the lattice. With such system, we wish to gain some intuition about how to build the optimal lattice from raw data, which is still an unsolved problem.

However, we note that such task is hard to accomplish. Visualization of lattice structure is a challenging problem for the following reasons.

1. Visualization of graph structure in general is a hard problem. It has been a research topic for a long time about how

Submitted for review to CHI 2009.

to construct a neat layout of the structures, how to display the information, and how to allow users to interact with the visualization.

2. Lattice in particular has important properties, such as the common edges, and data on the paths, that have not been addressed by previous graph visualization methods.

3. We need to support various operations for users to edit the lattice structure. To build such interface, which allows intuitive interactions to change the lattice to any possible configuration is challenging.

In this paper, we aim to overcome the above challenges. As you can see in details in the latter part of the paper, we adopt a force-directed layout to visualize the graph structure, and use various visual encoding techniques to display the common edges and data on the paths. We also design four major operations, which allow users to change the lattice structure freely and intuitively.

In the rest of the paper, we will 1) discuss the related work about graph visualization; 2) introduce our methods to visualize lattice structure; 3) our results and working system for lattice visualization; 4) discussions about such system; 5) conclusion and future works.

RELATED WORK

In general, lattice visualization involves the following problems: 1) layout and display of graph structure in general; 2) display graph edges effectively; 3) large scale graph visualization;

Force directed layout is a widely-used layout for graph visualization [2]. Such approach addresses the graph layout problem as a physical simulation of strings, which connect pairs of nodes in the graph. For our approach about lattice visualization, we use such technique to do the graph layout. However, to allow better exploration and flexibility, we also allow users to pin the nodes, so that they are not affected by the simulation anymore.

Holten et al. [3] proposed an edge bundling mechanism to visualize graphs and address the visual clutter. Generally, they applied force-directed simulation on the edges of graphs to make them attracting/distracting each other. Such work is related to ours, as we need to deal with the overwhelming number of edges. Whereas our project is different, as we already have the common edges, and to highlight those edges are of strong interests of our project.

Abello et al. [1] described a large-scale graph visualization system without extra hierarchical information about the graph. Their approach included first building a hierarchical tree to abstract the input graph, and then applying graph-clustering methods on the hierarchical tree. In such a manner, the large-scale graph would be displayed as clusters, and users could drill down each cluster to see the details. Lattice can be potentially huge, yet we have to deal the problem in a different way as the target information are stored on the paths, and clustering on graph nodes will not be meaningful. Rather than that, we decide to adapt to the common edges, which can be viewed as clutter on edges, and let users to drill down to a subgraph of paths, which share the selected common edge.

METHODS

In this section, we will explain the techniques exploited to visualize the lattice structure. In general, we tackle the visualization problem in three perspectives: graph layout, data exploration, and interactions for structure change.

Graph Layout

Lattice is a special form of graph structure. To display the graph structure neatly on a 2d canvas, we adopt the force-directed layout of graph. Such approach addresses the graph layout problem as a physical simulation of strings connecting particles with charges, where strings are the edges and particles are the nodes. To avoid collisions on the nodes, we assign the same charge on the nodes to make them compel each other. To avoid nodes being too apart, we assign the length of strings to be the length of the data encoded. Therefore, users can have a direct sense about how much data is encoded on each other by just observing its length. Such visual encoding also contributes to the judgment about the importance of a common edge. Intuitively, a longer common edge is generally more informative. We note that as the actual length of each edge is determined by the simulation, the length is not expected to be exactly proportional to the length of the data. However, for most of the cases, the more-data-encoding edges are longer than less-data-encoding ones.

Every lattice contains a start node and a terminal node, and we decide to exclude the two nodes from the force simulation in order to make them more observable. In our default layout, we choose to put the start node on the top left corner and the terminal one on the bottom right corner. We select such placement rather than the horizontal/vertical one to gain better space usage. In addition, we allow users to adjust the placement themselves if they think other layouts are more intuitive. For nodes other than the start/terminal node, users can exclude them out of the simulation by dragging and slowly releasing. We provide such functionality as for some cases, the simulation does not yield clean results, so users can adjust the layout for better views.

Force-directed layout handles the layouts of nodes on the graph, and here we discuss how we decide the positions of edges. Lattice is not a simple graph in the sense that there can be multiple edges with the same start node (local start node, not the global one for the entire lattice) and end node. Therefore, straight edges only would not satisfy our requirement. We tackle the problem by drawing every edge as part of a circle, and we adjust the curvature of each edge by changing the radius of the circle. For every two nodes, we have a list of edges connecting them, and we index the edges arbitrarily. The smaller-indexed edges will have a circle with larger radius, and vice versa. Thus, we expect to see the smaller-indexed edges to be closer to a straight line, and the larger ones to be closer to a semicircle (we force it not to go over 180 degrees). Moreover, we also change

the direction of the curvature every other edge. Suppose we have two nodes on a horizontal line, the edges with odd indices will be curved towards top and the even-indexed edges will be curved towards the bottom. As different lattices may have significantly different number of edges, it is hard to decided one maximum radius for all the lattices. Therefore, we also provide a slider for use to choose the maximum radius, which on the observing side, changes the curvature of all the edges.

The key feature of lattice is common edge. To make such edges obvious, we encode the width of each edge proportional to the number of paths through such edge. In another word, the "fatter" the edge, the more it is shared in the data.

Data Exploration

Lattice encodes data on paths from the start node to the terminal node. Along with the layout of the graph, we display the data encoded on each edge as well. The position to show the data is calculated in a similar manner as the position of each edge. Particularly, we find the point on the part of the circle, whose projection will be on the middle of the straight line connecting two nodes.

To show the entire data piece on any path, we design some interactive operations. Users can query about certain edge to see all the data pieces sharing such edge by moving the mouse over. For the feedback, we highlight all the paths on the graph, as well as displaying the detailed data information on a side window.

For data pieces shown on the side window, users can also move mouse over to query about the corresponding path, which conveys such data. Accordingly, the corresponding path will be highlighted on the graph layout.

Furthermore, we allow users to click on certain edges to drill down to the subgraph, which consists of only the paths sharing the clicked edge. We support such functionality as a way to deal with lattice of huge size. In such a way, users can filter out the rest of the graph which is not of interest, and focus on the subgraph with certain selected data.

All the interactions described above are implemented as a depth-first search algorithm. For each edge, we first specify the indices of paths which share such edge. Upon selection, we start from the start node, and search for all the edges which contain the path indices of the selected edge. A mapping from edge to data is also calculated as DFS goes through, for user to query path from data pieces in the side window.

Structure Change

As the target of the lattice visualization, we wish to gain intuition about how to construct the optimal lattice from raw data. Therefore, we support several structure change operations, which can change the lattice to any other one without changing the encoding data instances.

1. Node Insertion:
   We allow users to insert a new node to an existing edge, which also separates the information into two parts. Users can perform such operation by moving mouse horizontally around an edge, and releasing it at the desired position to add a new node. The desired position, which separates the encoding data, is shown in the side window for data display. We support such operation, as potentially the separated edges may have exactly the same data as some other edges. Insertion of a new node allows users to find those edges, and perform further processing on the separated edges.

2. Node Deletion:
   Users can remove an existing node. As a result, all the edges going into the node, and edges going out of the node will be reconstructed into longer edges with out the deleted node as a middle point. Such operation is useful when there are certain dummy nodes, which do not provide useful information. The dummy nodes can result from some other operations, which construct new edges and nodes. Moreover, node deletion can construct longer edges, which can potentially have the same content as other edges.

3. Node/Edge Merging:
   Users can merge two nodes or multiple edges together. Such operation is desired when users find certain edges with the same data encoded, but start/end on different nodes. In such a case, user can merge the nodes first to make the edges have same start/end nodes, and then continue to merge the edges into a common edge. By doing so, we are constructing more common edges, which makes the lattice a more compressed structure.

4. Edge Separation:
   Users can also separate a common edges into multiple edges. Such operation is meaningful as the existing lattice may have some common edges which are not optimal. Particularly, one or more of those edges after separation can exist in some longer data instance, which potentially share the same content with other edges. Moreover, edge separation also allows users to get a more detailed view of the lattice structure, by showing exactly how many paths there are sharing the separated edge.

We also record the history of the structure changes, and allow users to go back to history to undo the operations. To save the space, only the structural information (which nodes /edges are active) is saved in the history, while we use the same objects of nodes and edges through out the operations.

To make the above operations manageable, we create a tool bar of buttons designated for the operations. Buttons are enabled only when the correct inputs are selected. For example, if user selects a node, no edge operation is possible.

RESULTS

In this section, we will mainly show our visualization results, as well as a walk-through about how to utilize our system to build more compressed lattice structure.

To demonstrate the effectiveness of our system, we will use two lattices. The first one is a simple lattice, which encode four English sentences. The second one is a lattice of peptide sequences. We call the first lattice as word lattice, and the second one as peptide lattice for the rest of our discussion. Though we take these two lattices as examples, please note that our system is capable of handling any input lattice.

Results on Graph Layout and Data Exploration

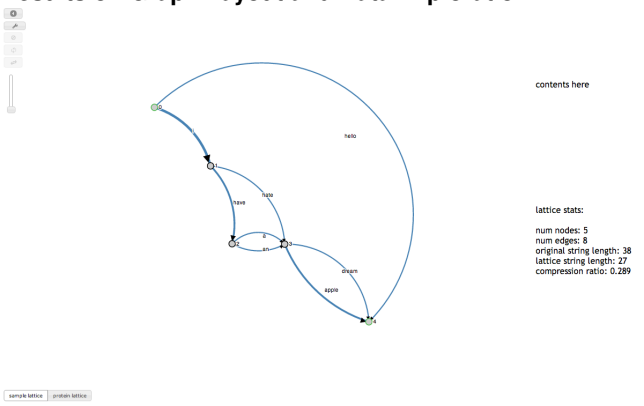

Figure 2. ***Word Lattice Visualization*** **– our system visualizing a lattice with four sentences.**

In Figure 2, we show a picture of our visualization of the sentence lattice. The major middle part of the visualization is the lattice structure. The top left part is a tool bar with buttons for various operations defined in the previous session. On the right side, we have two side windows. The top one is the content window, which displays information about the data information. The bottom window is the statistics window, which shows the statistical information about the current lattice. Such information includes measurements such as the number of nodes and edges, and the rate of compression. The information in the statistics window is updates as we interact with the lattice.

As discussed in the previous session, we apply a force-directed layout, and from Figure 2, we see a clear layout of nodes on the canvas. Common edges are also appropriately displayed, as we clearly see that the edge "i" has a larger width than other edges.

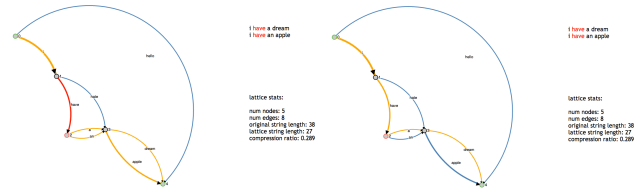

Figure 3. ***Word Lattice Visualization*** **– our system visualizing a lattice with four sentences. Mouse is moved to "have" edge in the left one. Mouse is moved to the data "i have a dream" in the right one.**

In Figure 3, we show how users explore the encoded data on our system. In the left graph, user moves mouse over to edge "have", and all the paths through such edge get highlighted in orange. The corresponding data pieces also get displayed in the content window with the selected word colored in red. In the right graph, user moves mouse over the displayed data: "i have a dream", and the corresponding path gets highlighted in the graph structure.

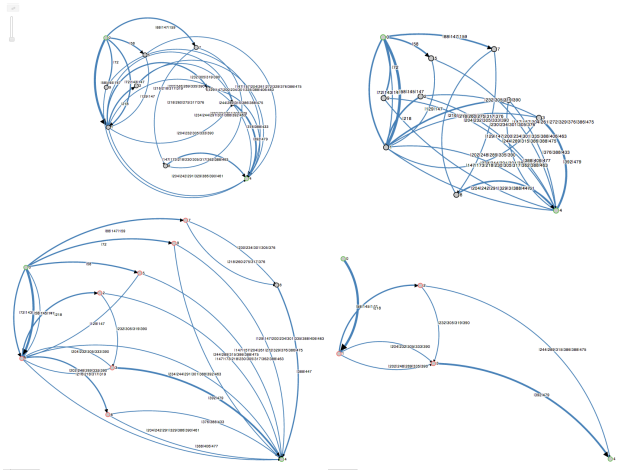

Figure 4. ***Peptide Lattice Visualization*** **– our system visualizing the peptide lattice. The top left graph is the default layout. The top right one is the layout with decreased curvature. The bottom left one is constructed by excluding certain nodes from the force-directed layout. The bottom right one is the subgraph obtained by drilling down on certain common edge.**

Next, we show how to use the layout interactions provided by our system to clean up a messy input lattice. As shown in Figure 4, we have four visualizations of the peptide lattice. The top left one is the default layout of the lattice, where lots of edges are too curved and get cluttered together. We first change the curvature of the graph, which yields out the visualization in the top right. We then modify the layout of nodes, which gives the bottom left one. In the end, we select certain common edge and drill down to a subgraph.

Obviously, by applying the above three operations supported by our system, users can get a much cleaner view of the original lattice.

Results on Structure Change

Here we show a walk-through of the operations supported to demonstrate the powerfulness of the editing functionalities of our system. In particular, we start with the word lattice, and we build a more compressed lattice in the end via four simple operations.

Firstly, we insert a node on the "have" edge between the letter "a" and "v", as shown in the top left picture from Figure 5. We do the same thing to the "hate" edge, where the new node is inserted between letter "a" and "t". Now we have two edges with the same contents, namely "ha". By applying node merging, we can merge the two end nodes of "ha" (top right of Figure 5). Obviously, we can merge the two "ha"s into a new common edge now (bottom left of Figure 5), and we can delete the node separating "i" and "ha" to get a longer common edge "iha" (bottom right of Figure 5).

By performing the above operations, we increase the com-

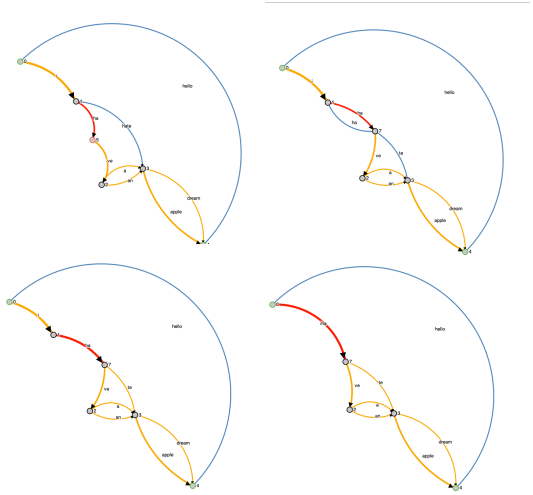

Figure 5. ***Word Lattice Structure Change*** **– using our system to change the lattice structure to gain a more compressed lattice.**

pression ratio 0.289 of the original lattice to 0.342 of the new lattice.

Result on Running Time

The operations supported by our system are all quite efficient and react to people in real time. As shown in Table 1, running time of all the supported operations is around 10 ms.

lattice	insert n.	del n.	merge n.	merge e.	separate e.
word	5.20	6.40	6.00	7.20	5.20
peptide	5.40	9.20	7.60	14.20	5.60

Table 1. ***Running time in ms for supported operations*** **– node insertion, node deletion, node merging, edge merging, and edge separation.**

DISCUSSION

In this section, we will discuss the new insights and practices that our system has brought.

Lattice Construction

To best of our knowledge, no prior work on visualizing a lattice structure has been done. Our system is the first visualization of lattice, which also allows users to explore the encoded data as well as modifying the existing lattice structure to gain a better lattice.

Though no solid user evaluations have been conducted, from our experience of using the system, it is much more efficient for us to find some common structures to make a more compressed lattice using our visualization system, than finding those structures from the texts, which represent the graph in an adjacency list. We believe that such experience transfers to more general cases and to more users, as for our system, it is very easy for users to try certain operations, see the results visually and immediately, and undo the operations if necessary, while for the texts search, users suffer a lot from transferring from textual information to structural information.

Meanwhile, as we design the operations for modifying the lattice, we also make an important observation that for lattices representing the same data, we can always transform one lattice to another by using the operations defined in our system: node insertion, node deletion, node merging, edge merging and edge separation. Therefore, it is possible to search for the optimal lattice by iteratively applying these operations to the existing lattice, which might contribute a new algorithm for lattice construction.

Graph Clustering on Edges

Lattices can be potentially huge graphs, depending on the size of data encoded. Graph clustering is a classic method to deal with visualizing graphs of large scale: we display clusters of subgraphs on the top level, and let users to drill down each cluster to see more details.

Most clustering methods are bounded to nodes. However, for the lattice case, information is stored on the edges, and such information gets lost if we do the node clustering.

In our system, we get around such problem by allowing user to drill down certain paths and filter out the rest of the graph which has no intersection with the selected paths. Note that we do not solve the large-scale problem entirely, as on the top level, we still need to show all the nodes and edges. However, we gain some insights about how to perform the graph clustering on edges.

Intuitively, we can cluster based on the paths that users may click. Specifically, for every edge going out of the start node, we can group them according to the paths. Suppose we let the path to extend to the terminal node, therefore on the top level, we will only have the start node and terminal node, and a limited amount of edge clusters connecting them. Moreover, we do not need to extend to the terminal node, and we can stop on certain node, and start a new clustering from there on, so that we preserve some structural information on the top level. Inside each cluster, we can also apply the clustering technique recursively, so that we get a clean visualization on every level.

Automatic/Manual Layout

Lattice structure can be so complex that the force-directed layout method is not able to yield a clean view of the structure. As shown in the result session about the peptide lattice, the default layout contains some visual clutters. To get around with such problem, we decide to give users more flexibility of changing the layout according to their own interests, rather than fully relying on the automatic layout algorithm (as shown in the result session, users can get clear view of the peptide lattice through some simple operations).

For automatic/manual approach, It is not clear about which one would be better to tackle the layout problem. Ideally, the system should support an automatic layout, which should be as visually clean as possible, as well as some freedom to manage the layout for users. However, we also note that as we put more constraints for the automatic approach, which are often required if we want more clear views, we poten-

tially lose more freedom on the manual approach.

Solid studies on users might be necessary on this point to evaluate how much effort should we put on both sides, so that users feel most comfortable about the layout. We also note that there are a few possible improvements for the current layout of lattice: 1) the edges should be compelling each other so that we can see each path clearly; 2) the shape of each edge is chosen to be a part of semicircle, while some other shapes may utilize the space better, such as bezier curves.

CONCLUSION AND FUTURE WORK

In this paper, we focus on developing a visualization system for the lattice structure, which takes any lattice as input, displays the structure information effectively, and allows users to modify the structure for a better lattice. We design lots of layout, visual encoding and interaction techniques for users to 1) get a clean view of the structure, 2) observe the common edges easily, 3) query data/structure from structure/data, and 4) change structure intuitively.

For future work, we expect the following improvements of our system:

1. Better layout of the edges, so that then chances that edges overlay each other are smaller.
2. Support graph clustering, especially the edge clustering technique proposed in the discussion section, so that we can display lattices of large scales.
3. Support more structure change operations and polish the existing operations more user-friendly.

REFERENCES

1. J. Abello, F. van Ham, and N. Krishnan. Ask-graphview: A large scale graph visualization system. *IEEE Trans. Vis. Comput. Graph.*, 12(5):669–676, 2006.
2. T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Softw., Pract. Exper.*, 21(11):1129–1164, 1991.
3. D. Holten and J. J. van Wijk. Force-directed edge bundling for graph visualization. In *Proceedings of the 11th Eurographics / IEEE - VGTC Conference on Visualization*, EuroVis'09, pages 983–998, Aire-la-Ville, Switzerland, Switzerland, 2009. Eurographics Association.