A Treemap Based Method for Rapid Layout of Large Graphs

Chris Muelder* Kwan-Liu Ma†
University of California, Davis

ABSTRACT

Abstract graphs or networks are a commonly recurring data type in many fields. In order to visualize such graphs effectively, the graph must be laid out on the screen coherently. Many algorithms exist to do this, but many of these algorithms tend to be very slow when the input graph is large. This paper presents a new approach to the large graph layout problem, which quickly generates an effective layout. This new method proceeds by generating a clustering hierarchy for the graph, applying a treemap to this hierarchy, and finally placing the graph vertices in their associated regions in the treemap. It is ideal for interactive systems where operations such as semantic zooming are to be performed, since most of the work is done in the initial hierarchy calculation, and it takes very little work to recalculate the layout. This method is also valuable in that the resulting layout can be used as the input to an iterative algorithm (e.g., a force directed method), which greatly reduces the number of iterations required to converge to a near optimal layout.

Index Terms: I.3.6 [Computing Methodologies]: Computer Graphics—Methodologies and Techniques;

1 INTRODUCTION

Applications in many fields employ graph visualization to present data to the user. For example, document visualizations [18] can represent documents and their citation network with a graph. Similarly, social network visualizations [14] can represent people as nodes in a graph. These visualizations are frequently used to show inherent patterns in the data. In order to emphasize these patterns, many algorithms have been developed that determine how best to place the nodes on the screen. While there are special cases such as trees or directed graphs for which shortcuts can often be used or directional constraints taken into consideration, probably the most commonly used graph layout algorithms are force-directed layouts [21, 11, 17]. These layouts are versatile because they can be applied to any general graph, they can take edge weights into consideration, and they tend to make very aesthetically pleasing layouts. They are good at showing patterns such as clusters since strongly connected vertices in the graph get pulled together by the layout algorithm. Although rapid force-directed layouts have been recently improved[13], they are still generally computationally intensive and do not scale well as the size of the graph increases. Also, force-directed layouts have a problem with local minima in the energy function where the graph can get stuck in a non-optimal layout. As a result, it is not uncommon for the layout algorithm to take many extra iterations to separate clusters from each other, or for a node in a cluster to be separated from the rest of its cluster and have its edges stretching across another cluster of nodes. There are also algebraic layout algorithms that quickly generate a layout by directly working with the matrix of edges between nodes [13]. However, these approaches often have the problem of mapping many nodes to the same locations, so that parts of the graph are obscured.

*muelder@cs.ucdavis.edu
†ma@cs.ucdavis.edu

The treemap based layout algorithm presented here avoids these common drawbacks of force-directed approaches while attaining the speed of algebraic approaches. Since this layout uses operations that are relatively easy to calculate, it is not as computationally expensive as force-directed layouts. Because of this property, and because the treemap layout does not take multiple iterations, it provides results more quickly than force-directed approaches. It also avoids the problem of nodes being separated from their clusters since the treemap forces clusters into separate regions of the screen. Since the treemap based layout is good at avoiding some of the difficulties inherent to force-directed layouts, it also works well as a preprocessing step to a force-directed layout calculation. That is, the output of the treemap based layout algorithm is a good initial set of positions on which to run a force-directed algorithm. The force-directed layout will then not have to separate clusters that are mixed together, since the treemap layout has done this already. In fact, in experimental tests, using such a layout as a preprocessing step provides an improvement of an order of magnitude over randomized initial positions. Thus, the algorithms can complement each other such that the end result is achieved much faster when they are combined.

2 RELATED WORK

This work draws upon several existing techniques in both the fields of graph visualization and treemap visualization. Many graph layout algorithms have been developed, and there are several variations of and extensions to treemaps. The concept of using the two together has also been previously introduced.

2.1 Existing Graph Layout Techniques

Sometimes a graph has an intuitive layout where the vertices contain positional information that can be used, such as geographical locations. However, most graphs do not have such information, thereby requiring that the positions of vertices be derived. For special cases, such as trees or directional graphs, algorithms are used that exploit certain properties of the graphs. But for general graphs, more flexible algorithms must be used. The two major classes of algorithms used to layout general graphs are as follows:

- *Force-Directed Layouts* position graphs using a physical model. They usually work by iteratively refining the positions of vertices in order to incrementally reduce an energy function. This energy function varies between algorithms, but generally has the property that it is a function of the distances between nodes and the weights of the edges between them. That is, if the edge between two vertices is strong then the two vertices will be strongly pulled together, and if the edge is weak or nonexistent, then the vertices will be repelled. Fruchtermen-Reingold [11], LinLog [21], and Kamada-Kawai[17] are common examples of algorithms that fall into this category. There are several variations of these algorithms that reduce the amount of computation heuristically by limiting the number of calculations per iteration [6] and several of those surveyed by Hachul and Jünger [13]. While these variations are faster, the heuristics can lead to suboptimal layouts or require exorbitant amounts of memory. Other variants accelerate the process through parallel hardware, such as the recent GPU based layout [10].

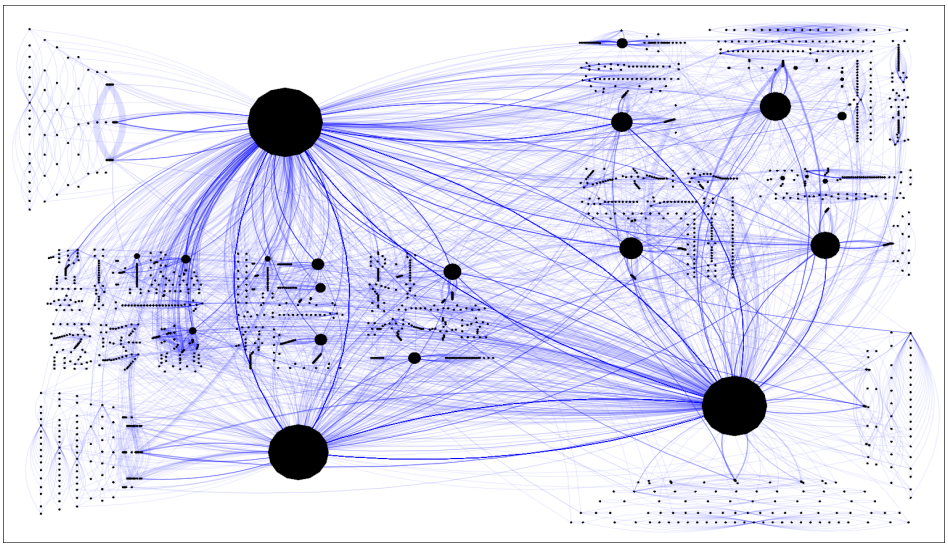


Figure 1: *A graph laid out using our treemap based approach.* This graph portrays the links between websites that came from a search on the word "California" [7]. Nodes are clustered into a hierarchy, and laid out by applying a treemap to this hierarchy. Levels of the hierarchy below a threshold are clustered together into larger nodes. It can very easily be seen that there are three primary groups of websites that link to each other, and a plethora of others that are not as tightly linked.

- *Algebraic Layouts* There are also several layout algorithms that are based on linear algebra. These algorithms work by directly manipulating the adjacency matrix with linear algebra techniques in order to produce an effective layout. While not very intuitive, these algorithms can quickly produce layouts that are similar to the force-directed layouts. However, these algorithms can fail to produce a good layout in some cases. For instance, in some examples in the survey paper by Hachul and Jünger [13], the algebraic layouts place multiple nodes at the same coordinates, which obscures the structure of the graph. Two such algorithms are described in their survey paper [13]: The Algebraic Multigrid Method ACE and High-Dimensional Embedding.

2.2 Clustering

Frequently, graph data is clustered in order to create an overview or allow interactions such as semantic zooming. In order to use a treemap for layout, the graph must be clustered with a hierarchical method, such as agglomerative or divisive clustering [16]. In agglomerative clustering, each node starts as a separate cluster. The clusters are then progressively merged together until there is only one cluster. One example of such an algorithm is single linkage, where clusters are sequentially merged according to the strongest remaining edge. In divisive clustering, such as spectral clustering, there is initially one cluster consisting of the entire graph. Then the cluster is recursively split into smaller clusters until it is either no longer beneficial or no longer possible to split it further.

2.3 Treemaps

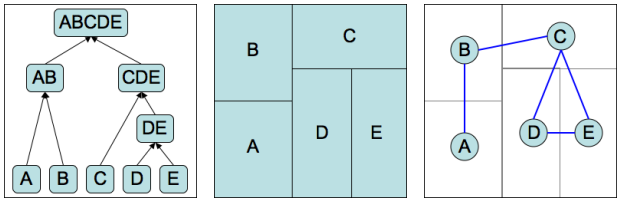
Originally proposed by Ben Shneiderman [24], treemaps have become a common method for representing hierarchical data. Many variations and extensions have since been explored, such as the Squarified algorithm [4], which alters how the screen is divided, and the Voronoi treemap algorithm [3], which removes the rectangular limitation that treemaps usually have. The majority of these works use a treemap to directly show a hierarchy rather than using that representation of the hierarchy to accomplish something else.

2.4 Combining Treemaps and Graphs

In several works, treemaps and graphs have been used together. In the work of Zhao et al. [28], both graph diagrams and treemaps are used interchangeably to represent parts of a tree more efficiently. A node-link diagram is overlaid on a treemap in order to represent the same tree in the paper by Nguyen et al. [20]. However, these only work with trees, and not general graphs. In the work of Abello et al. [2], a treemap is used to interact with the clustering hierarchy of a graph. But this treemap is only used to control the level semantic zooming of the graph, and not the layout. Finally, the work of Fekete et al. [9] divides a graph into a spanning tree and a set of edges, then uses a treemap to represent the tree part of the graph and displays the remainder as nodes and links overlaid on the treemap. This is useful when the graph has an inherent or intuitive tree decomposition, such as web sites, but most general graphs do not have such a nice decomposition. The work presented here is substantially different from these. We use a treemap to layout a general graph, and address several of the accompanying issues.

3 A New Layout Method

This paper presents a novel method of laying out a graph by generating a clustering hierarchy, applying a treemap to the clustering hierarchy in order to allocate regions of the screen that are associated with individual vertices in the graph, and finally placing each node inside its region of the screen. An example of a graph laid out this way is presented in Figure 1. This particular graph is a non-weighted graph of links between search results for the word "California" (also in Figures 6, 8, and 9, $|\mathbf{V}| = 6107$, $|\mathbf{E}| = 15160$, [7]). The other graphs used in this paper are: a graph of network scans, which is a complete graph with edge weights between 0 and 1, but for clarity, edges with weights less than a certain threshold are not shown (Figures 3 and 4, $|\mathbf{V}| = 878$, $|\mathbf{E}| = 385003$, [19]), a small artificial graph of a grid topology (Figure 5, $|\mathbf{V}| = 16$, $|\mathbf{E}| = 24$), a large graph of streets in the San Francisco Bay Area (Figure 7, $|V| = 321,270, |E| = 800,172$, [8]). and an artificial randomly generated complete graph consisting of seven strongly connected clusters with weak inter-cluster edges (Figure 10, $|\mathbf{V}| = 1024$, $|\mathbf{E}| = 523776$). The overall process is shown in Figure 2.



(a) Hierarchically cluster the nodes. (b) Apply a treemap to the clustering hierarchy. (c) Place the nodes in their treemap regions.

Figure 2: *The overall method.* Divide screenspace into regions by applying a treemap to a hierarchical clustering of a graph, then place each node in its associated region.

3.1 Hierarchical Clustering

As shown in Figure 2(a), the first step is to hierarchically cluster the nodes. There are many hierarchical clustering algorithms. Some work well with weighted graphs, while others work well with unweighted graphs. For this paper, single linkage [16] was selected to work with weighted graphs because it is simple and quickly calculated. For unweighted graphs, Clauset, Newman, and Moore's "Fast Modularity" community structure inference algorithm [5] was chosen, since it is very fast, generates a binary hierarchy, and is good at clustering small world networks. While a treemap based layout can use any hierarchical clustering, the ones used here make a binary tree, which keeps the treemap process simple.

3.2 Treemap Generation

Treemaps work by assigning each node in the hierarchy a region of space, then subdividing this space among that node's children. So, initially, the root node of the hierarchy is assigned the entire screen space, and then it is split up according to its children. Then each of the children's areas are split up according to their children, and so on down the hierarchy. One advantage to this representation is that it is space filling so that no screen space is wasted. Applying this concept to the hierarchal clustering of a graph generates a treemap such as the one in Figure 2(b). In this image, each region of space corresponds to a node in the graph, and regions corresponding to nodes that are tightly clustered together end up near each other. While many advanced treemap algorithms exist, such as Squarified treemaps [4], due to the choice of a binary clustering algorithm for this paper, we are constrained to a slice and dice approach.

3.3 Adding the Graph

Once a treemap has been created, the graph can be laid out by placing each vertex in its associated region of the screen. The simplest way to do this is to simply place the node in the center of its region, as is shown in Figure 2(c). Since the treemap represents the clustering hierarchy, each vertex in the graph will be assigned a region in space that is near to regions of space corresponding to other vertices in its cluster. This results in a layout that satisfies several of the properties that are desirable in a graph layout. Namely, the nodes are placed such that strongly weighted edges are kept relatively short compared to weakly weighted ones, connected nodes are generally placed closer than disconnected ones, and important features such as clusters and outliers are readily discernible. Also, this has an added advantage over most other layouts in that it efficiently fills the whole screen, so that more nodes can be displayed simultaneously without obstructing each other. Figure 3 shows the result of applying this approach directly to a real graph.

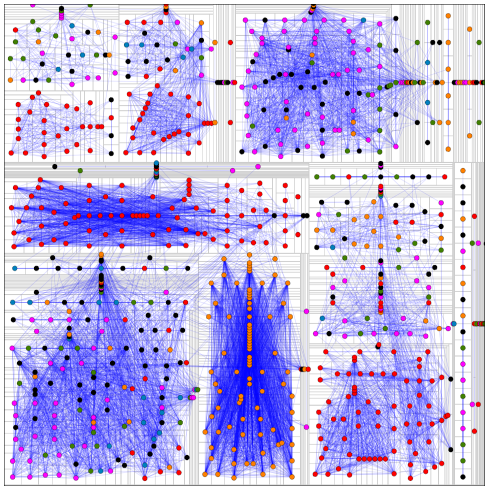


Figure 3: *A graph laid out with our technique.* The dataset shown is a graph of similarity between network scans [19].

4 Evaluation and Enhancements

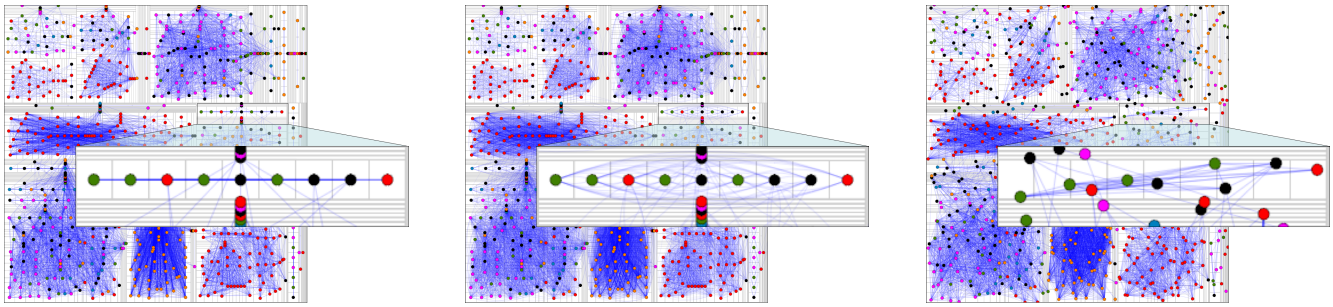
The approach presented in this paper has several advantages when compared to existing algorithms, but it also has some limitations. It performs very well in terms of speed, and it does not take multiple iterations the way force-directed layout algorithms do. Furthermore, the space filling property of the treemaps means that the graph takes up more of the screen space than many other layouts, such as force-directed ones, allowing more space to be used to show detail inside clusters. But, the treemap layout is dependent on the hierarchy, so the choice of clustering algorithm can have a large effect on the results, and an unbalanced hierarchy can lead to poor results. Also, the use of treemaps directly introduces some common treemap-related issues. But these issues can often be alleviated by making simple changes to the algorithm.

4.1 Collinear Vertices

Due to regularity of treemaps, this approach tends to place many identical regions in a row. When this happens, all the nodes for those regions end up being collinear if they are all placed in the middle of their regions (shown in Figure 4(a)). The problem with this is that edges between such nodes will not only cross but completely overlap so that they become indistinguishable from each other. There are a couple of solutions to this problem. One solution is to curve the edges with splines so that they do not overlap, as in Figure 4(b). In the case of a directed graph, this splining is also nice to do in order to simultaneously show direction in the graph, as in the work of Fekete et al. [9]. Or an edge bundling kind of splining could be effective, as described by Holten [1]. Another solution is to simply randomize the placement of nodes in their regions, as is shown in Figure 4(c). While this sacrifices some deterministic properties of the layout, the probability of any three vertices being collinear is small enough to effectively solve this problem.

4.2 Narrow Regions

A common issue with treemaps is that regions can end up being very narrow. When this happens in the treemap graph layout, several nodes in the graph end up in a line so close together that they frequently overlap. In particular, this usually occurs with nodes that are outliers; many outliers get placed closer to each other than tightly clustered nodes do. There are several possible solutions to this issue. Just as in the collinear vertices issue, randomizing the placement of the nodes will alleviate this problem, as can be seen

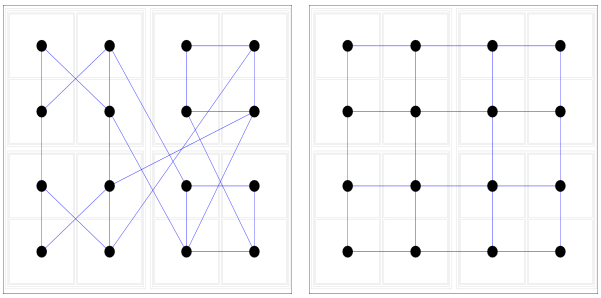


(a) The simple treemap algorithm used causes many vertices to be collinear, and some regions to be very thin.

(b) Curving the edges with splines prevents ones that go between collinear points from overlapping.

(c) Randomizing the placement of vertices within their regions makes them be non-collinear and spreads them out over narrow regions.

Figure 4: *Enhancements to the approach.* Difficulties due to the use of treemaps (a), and two techniques to alleviate problems due to collinear vertices: splining (b) and randomization (c). Randomization also helps with narrow regions of the treemap.



(a) Naive splitting

(b) Directed splitting

Figure 5: *Improved treemap splitting.* Edges can sometimes stretch across clusters. By taking these edges into account during the treemap subdivision process, quality can be improved at the cost of complexity.

in Figure 4(c). Since the narrow regions are longer in one direction, the nodes on average get spread out more over the longer direction, and so are unlikely to overlap. Another solution to the narrow region problem can be found by adjusting the clustering algorithm. Since the treemap is a direct result of the hierarchy generated by the clustering algorithm, if the clustering algorithm makes a tree that is somewhat balanced, the treemap will also be somewhat balanced and the narrow regions problem should be alleviated. Finally, as this is a common treemap problem, it might also be correctable through the use of a more complicated treemap algorithm. For example, the Squarified algorithm was designed to solve this exact issue. However, using a Squarified treemap necessitates a non-binary hierarchy, and would add computation overhead.

4.3 Directed Region Splitting

When an edge goes between two nodes that are in different clusters, it is possible for these nodes to be placed within their clusters such that they are far apart. The problem with this is that the edge between these two nodes now stretches completely across one or both of the involved clusters or even over the entire graph. This is the case in Figure 5(a). When possible, it would be better if these nodes could be placed within their clusters such that they end up near each other, so the cross cluster edge is not crossing over as many of the other edges in the clusters. This can be done by arranging the children during the splitting process according to an energy function of the edges into and out of the child regions. The result of applying this is shown in Figure 5(b) Unlike the energy function in most force-directed algorithms, this energy function does not need to calculate distances between every node for repulsion forces; it only needs to calculate attractive forces of edges. Also, when using a binary hierarchy, there are only two ways to split each region, so the energy calculation only needs to consider which of these is lower energy. Thus, even though this substantially increases the computational cost, since it has to consider each edge, it is still a large improvement over force-directed layouts.

4.4 Cluster Seperation

Randomizing the placing of the nodes in their regions has the effect of making the nodes have an overall uniform distribution over the entire screen. While this is efficient in terms of screen space, it is not always the most aesthetic view. Clusters abut each other, with no white space separating them. This problem can be trivially solved by nesting the treemap regions. In order to have greater separation between clusters than within clusters, the nesting is done proportionately to the size of the region. Also, since the clustering hierarchy can be quite deep, this nesting is only applied to the first n levels of the treemap, where n is user defined. Since the layout process is fast, the user can easily alter the nesting parameters interactively till an aesthetic view is found. Figure 6 shows the results of applying this to the graph of search results for "California" [7].

4.5 Complexity and Scalability

One of the biggest advantages of a treemap layout technique is that it is very fast. In fact, the most time consuming step is not even laying out the nodes, it is generating the hierarchy. A single linkage clustering hierarchy is essentially equivalent to a spanning tree, and can be generated in as little as $O(|\mathbf{E}|+|\mathbf{V}|log|\mathbf{V}|)$ time. The community clustering hierarchy [5] takes $O(|\mathbf{E}|*d*log|\mathbf{V}|)$ time, where d is the depth of the hierarchy. These hierarchy costs dominate the initial layout cost, but they only get calculated once, and thus can actually be done as a preprocess step. The actual node layout only takes $\Theta(|\mathbf{V}|)$ time to do simply, since the hierarchy tree is at least $|\mathbf{V}|$ and at most $2*|\mathbf{V}|$ nodes large. Also, at each recursion, the amount of computation is very small since it only has to divide a region in two. Accounting for edges during the splitting process increases the complexity to $O(|\mathbf{E}|+|\mathbf{V}|)$ since the edges must be considered, but as with the clustering, this can be done once and then reused. When compared to the $O(|\mathbf{V}|^2)$ somewhat computationally intensive calculations that a traditional force-directed layout takes per iteration, the treemap algorithm is much faster. In fact, when

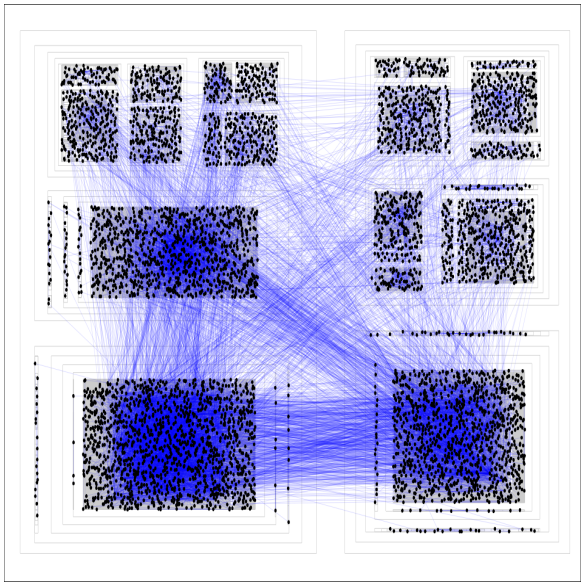


Figure 6: *Cluster seperation.* Nesting the first n levels of the treemap separates clusters, which better shows inter-cluster relationships.

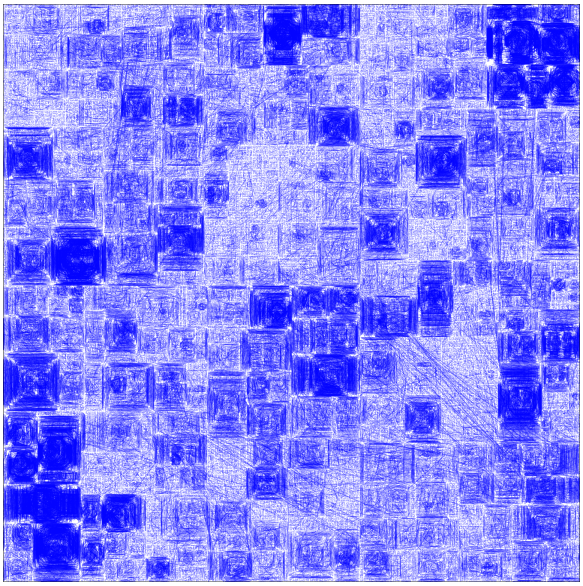


Figure 7: *Scalability.* Our approach can scale to very large networks while still maintaining interactivity. $|V| = 321,270, |E| = 800,172$

$|\mathbf{E}| \geq |\mathbf{V}|log|\mathbf{V}|$, the total worst case complexity is comparable to that of the algebraic layout algorithms. But even when this is not the case, the higher initial complexity can be marginalized by the lower cost of laying out nodes when the nodes need to be laid out repeatedly, as in an interactive system.

Due to the rapid speed of our approach, one of the biggest advantages to our approach is scalability. It can easily scale up to hundreds of thousands of nodes, as is demonstrated in Figure 7. This graph of city streets in the San Francisco Bay Area is quite large ($|V| = 321,270, |E| = 800,172$). Generating the hierarchy with the fast community clustering algorithm takes about 80.65 seconds. However, generating the actual layout only takes 1.34 seconds with directed region splitting, and only .125 seconds with naive splitting. Thus, the layout can be generated at interactive speeds.

4.6 Effectiveness

It does not matter how fast a graph laid out if the resulting layout is not useful. For instance, a randomized layout can be generated very fast, but the resulting layout will rarely be useful. Force-directed layouts such as LinLog generally produce quite good results, but take a long time to do so. The treemap based layout presented here though is both fast and effective. The clustering basis makes it so nodes are near nodes they are tightly clustered to. The directed splitting gives the layout several of the properties that make force-directed layouts good - namely shorter edges and fewer edge crossings. And finally, the screen constraints induced by the treemaps guarantee that the nodes are spread out, so that patterns in the edges can be seen. In order to demonstrate the quality of a treemap based layout, a comparison between it and some other layouts is given in Figure 6, where they are applied to the "California" graph. Timing results were generated by running the programs on one core of a 2.66GHz Intel Xeon Mac Pro with 8Gb of RAM.

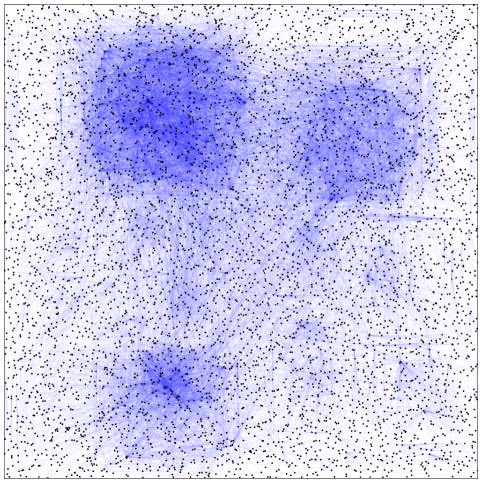
The treemap based layout (Figure 8(a)) was made using the "Fast Modularity" algorithm [5], directed splitting, and randomized node placement within their regions. The whole layout took only one second to compute. As can easily be seen, the most costly operation was the clustering, which took over 90% of the time. Since this is only done once on load, the actual layout can be run at about 13 frames per second, which is easily fast enough for an interactive system. This could be made even faster by only applying the directed splitting once and recalling the ordering in subsequent layouts. The resulting layout very clearly shows three large clusters of nodes, with many internal nodes, and a large number of other nodes distributed around the screen. Within these clusters, interesting features such as nodes of high degree are easily visible. Similar features are also visible in the rest of the graph. The LinLog based layout, shown in Figure 8(b), was easily the slowest, taking over 10,000 seconds to compute 200 iterations. In this layout, it is clear that there is a very tightly connected group of nodes, and many weakly connected or disconnected subgraphs. While it can be seen that there are actually three clusters in the center of the graph, the internals of these clusters can not be seen, since the nodes are so close together. The Grid-Variant Algorithm (GVA) [13], shown in Figure 8(c), is a heuristically accelerated layout based on the Fruchterman-Reingold layout algorithm [11]. As can clearly be seen, this algorithm is much faster than LinLog, taking only 45 seconds to do 400 iterations. However, the results are not as good as LinLog, since the three clusters in the middle are not distinguishable from each other. It also induces a grid-like arrangement of many of the nodes due to the heuristic which is not related to the graph. While the force-directed graph layouts may be more intuitive and perhaps more aesthetically pleasing, neither of them show the internal structures of the clusters that are easily visible in the treemap based layout.

5 Applications

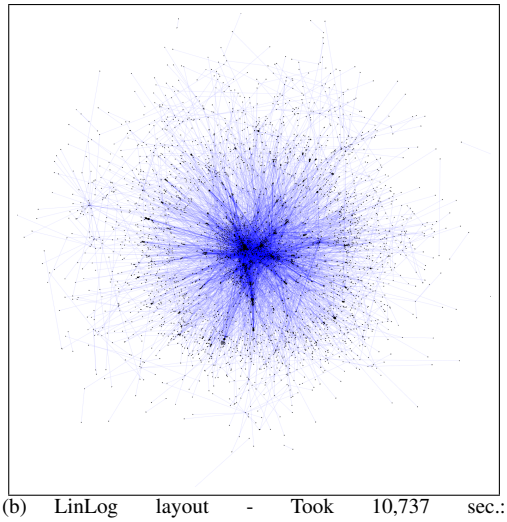
Similar to force-directed layouts, the treemap layout algorithm is applicable to any general graph. It is only dependent on the clustering algorithm chosen. However, there are some applications for which a treemap based layout is particularly effective.

5.1 Semantic Zoom

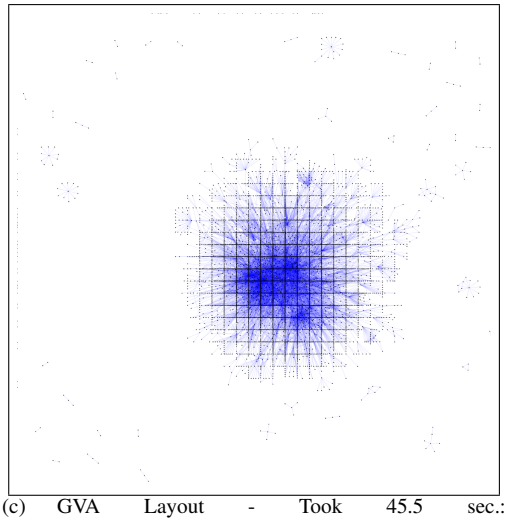
When dealing with large graphs, often times a clustering is generated in order to produce a high level abstraction of the graph. This abstraction can then be used as an overview to the graph, where each node in the abstracted version represents a whole cluster in the detailed graph. However, when granting the user access to the



(a) Treemap based layout - Took 1.00 sec.: 0.923 sec. for clustering, 0.076 sec. for treemap



(b) LinLog layout - Took 10,737 sec.: 200 iterations, 53.7 sec. per iteration



(c) GVA Layout - Took 45.5 sec.: 400 iterations, .113 sec. per iteration

Figure 8: *Treemap Layout versus force-directed layouts (LinLog [21] and the Grid-Variant Algorithm (GVA) [13] based on Fruchterman-Reingold).* The treemap based layout takes far less time to produce, and details such as the interior of the clusters can be clearly seen.

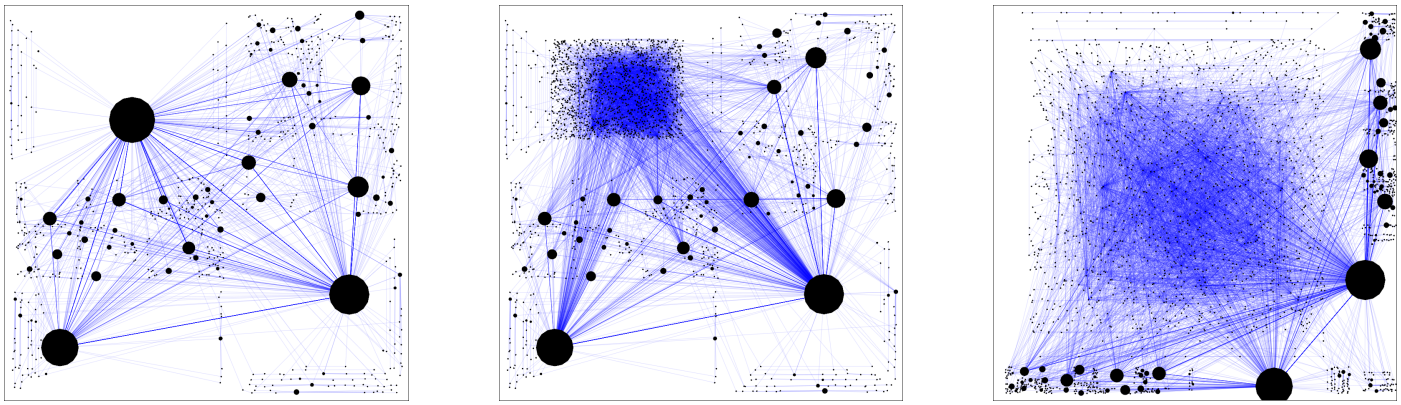
detailed version of the graph, it is beneficial to only show detail for a small portion of the graph rather than the whole. It is also beneficial to maintain the context of the overview so that the user can more easily associate the section of the overview with its detail view. Thus, a "focus plus context" paradigm is often employed, where only the small region of the overview is expanded to the detail view and is shown in place with the rest overview. The treemap layout was designed with this goal in mind, and therefore is very efficient at doing so as is demonstrated in Figure 9. Figure 9(a) shows an overview of the "California" graph, which was generated by only descending some user defined number of levels down the hierarchy. Figure 9(b) shows the result of expanding one clustered node into its detailed graph. Then Figure 9(c) shows this subgraph expanded by distorting the treemap with a technique similar to that used by Shi et al. [23]. This process can be done exceedingly quickly, since almost all the layout work is already done by the hierarchical clustering. The end result of this is that even very large graphs can be rapidly explored interactively and viewed in high detail when desired. There is a minor issue of instability, since the aspect ratio of the distorted areas are different. This could be handled by forcing the splitting process to use the same splits as the undistorted treemap, but this would cause very poor aspect ratios, and so was not done for this example. Another potential solution to this problem would be to use animation, but this is left as future work.

5.2 Preprocessing for Force-Directed Layout

Force-directed layouts have neither the problem of having nodes too close together due to narrow regions in the treemap nor the problem of collinear points, so they often produce results that are considered more aesthetically pleasing. However, they are generally slow and have difficulty in separating out clusters that are mixed together. As was done in several other works [22, 26, 25], spatial decomposition can be used to accelerate the force-directed layout process. That is, if the input points have already been separated into regions of space by cluster, then the number of iterations it takes to calculate the force-directed layout can be greatly reduced. Since this is what the treemap layout approach does, it makes an effective set of initial points for a force-directed layout to start from. Figure 10 shows the results of applying the LinLog [21] force-directed algorithm to both a randomized initial input and an initial input generated with the treemap layout. In both cases, the force-directed layout was terminated when the generated layouts reached a point where all clusters were distinct and looked comparable in both layouts. The force-directed layout would continue to slowly adjust both layouts for many more iterations. However, these iterations would be essentially the same for both inputs, and so are not pictured here; they did not change the topology of the layout, merely the relative positions of the clusters. As can be seen, the number of iterations required to generate a good force-directed layout was an order of magnitude less when the nodes were initially placed with the treemap based layout. In this example, the treemap layout was approximately equivalent to about 125 iterations of the force-directed layout for this data. On the system that performed this computation, each iteration took about two seconds. By applying a treemap based layout to the graph before using the force-directed layout on it, the amount of time to render the graph was reduced from about four and a half minutes to about twenty seconds, which is much more reasonable for an interactive program.

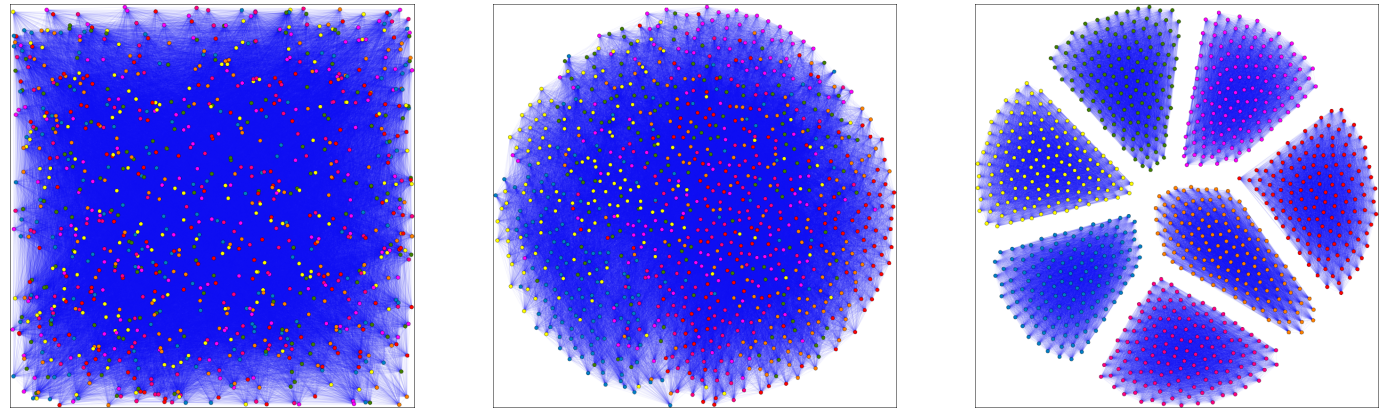
6 Future Work

While the results of this approach have proven themselves useful, there are still several things that can be done to improve upon them. The clustering algorithms we use tend to make unbalanced hierarchies, and the treemap algorithm is quite simplistic. Both of these could be replaced with alternate algorithms to improve the results.



(a) Overview (b) Semantic zoom (c) Semantic zoom + distortion

Figure 9: A treemap based layout can be used to show focus plus context easily through means of semantic zooming and distortion. (a) shows a high level abstraction of the "California" graph [7]. In (b) one of the clusters has been expanded with a semantic zoom, and in (c) the graph has been distorted to show the expanded cluster in more detail.



(a) Initial random placement for LinLog layout energy=896804.375000

(b) LinLog with random init. - 60 iterations energy=705586.312500

(c) LinLog with random init. - 135 iterations energy=681244.625000

(d) Initial treemap based placement energy=888501.625000

(e) LinLog with treemap init. - 5 iterations energy=687810.562500

(f) LinLog with treemap init. - 10 iterations energy=682494.437500

Figure 10: A treemap based layout can also be used as a preprocessing step for a force-directed layout. After only 10 iterations of the force-directed layout starting with the treemap layout the results are comparable to 135 iterations starting from randomized positions. The value of the energy function being minimized by LinLog is given for each iteration.

6.1 Alternate Clustering Techniques

The current clustering algorithm we use is a binary merging clustering technique. The hierarchy that it generates has a tendency to be a very unbalanced tree. For the weighted graphs, complete linkage or average linkage [16] would likely be better in this respect. Ward's method for generating hierarchical clusters [27] is generally considered to be good at generating balanced hierarchies, and so it would also perform very well in this situation. For non-weighted graphs, a better algorithm would be one such as the hierarchical clustering method employed by Koren and North [12], which would generate a fairly balanced hierarchy since it forces the clustering to work on only one semantic level at a time. Also, recent work has been done to improve on the modularity clustering we use to make it more balanced [15]. It is also quite possible that a new more specialized clustering technique could be made with the treemap in mind so that the generated hierarchy tree would be more balanced and thus the areas in the treemap would be kept more regular. That is, such an algorithm would prevent the problem of narrow regions in the treemap, and thus probably generate a better layout overall even if the clusters are not quite optimal mathematically.

6.2 Alternate Treemap Algorithms

Currently, the treemap layout algorithm is just using the slice and dice treemap algorithm. Since the hierarchy is a binary hierarchy, the options are somewhat limited. For example, the standard Squarified algorithm [4] cannot be applied directly to binary hierarchies very easily, since it works by arranging n child nodes at a time. However, it could probably be applied if each level in the treemap considered more than one level of the hierarchy. That is, it could consider the next k levels of the hierarchy to lay out up to 2^k nodes. Also, if the hierarchical algorithm employed creates a non-binary hierarchy, then the Squarified algorithm could be used.

7 Conclusions

Laying a graph out efficiently and coherently is a difficult task. Many times, a layout algorithm that is good for one graph does not work very well for another graph. Thus, there are almost as many different layout algorithms as there are types of graphs to visualize. There are some algorithms that work fairly well for most graphs, such as force-directed layout algorithms, but these algorithms in general take a long time to run. The concept of a treemap based layout is very flexible, since there are as many variations as there are clustering algorithms. That is, if there exists a clustering algorithm that is particularly effective on some set of graphs, a treemap layout based on this clustering algorithm should also be quite effective for visualizing those graphs. In applications where a hierarchical clustering is already being calculated, for multiresolution purposes for instance, the advantage is even greater since the actual treemap calculation takes even less time than it takes to render the graph.

8 Acknowledgments

This research was supported in part by the U.S. National Science Foundation through grants CCF-0634913, IIS-0552334, CNS-0551727, and OCI-0325934, and the U.S. Department of Energy through the SciDAC program with Agreement No. DE-FC02-06ER25777.

References

[1] Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):741–748, 2006. Danny Holten.

[2] J. Abello, S. G. Kobourov, and R. Yusufov. Visualizing large graphs with compound-fisheye views and treemaps. In *Graph Drawing*, pages 431–441, 2004.

[3] M. Balzer and O. Deussen. Voronoi treemaps. In *IEEE Symposium on Information Visualization 2005*, 2005.

[4] M. Bruls, K. Huizing, and J. van Wijk. Squarified treemaps, 2000.

[5] A. Clauset, M. E. J. Newman, and C. Moore. Finding community structure in very large networks. *Physical Review E*, 70:066111, 2004.

[6] J. D. Cohen. Drawing graphs to convey proximity: An incremental arrangement method. *ACM Transactions On Computer-Human Interaction*, 4(3):197–229, 1997.

[7] Data. 'California' search results graph, `http://www.cs.cornell.edu/Courses/cs685/2002fa/`, accessed 12/10/07.

[8] Data. Graph of San Francisco Bay Area streets from the 9th dimacs implementation challenge, `http://www.dis.uniroma1.it/~challenge9/download.shtml`, accessed 12/10/07.

[9] J.-D. Fekete, D. Wang, N. Dang, A. Aris, and C. Plaisant. Overlaying graph links on treemaps. In *InfoVis03, Poster Compendium (Aug. 2003)*, page 8283, 2003.

[10] Y. Frishman and A. Tal. Multi-level graph layout on the gpu. *IEEE Trans. Vis. Comput. Graph.*, 13(6):1310–1319, 2007.

[11] T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software - Practice and Experience*, 21(11):1129–1164, 1991.

[12] E. R. Gansner. Topological fisheye views for visualizing large graphs. *IEEE Transactions on Visualization and Computer Graphics*, 11(4):457–468, July 2005.

[13] S. Hachul and M. Jünger. An experimental comparison of fast algorithms for drawing general large graphs. In *Graph Drawing*, pages 235–250, 2005.

[14] J. Heer and D. Boyd. Vizster: Visualizing online social networks. In *IEEE Symposium on Information Visualization 2005*, 2005.

[15] M. L. Huang and Q. V. Nguyen. A fast algorithm for balanced graph clustering. In *IV*, pages 46–52. IEEE Computer Society, 2007.

[16] S. C. Johnson. Hierarchical clustering schemes. In *Psychometrika*, pages 2:241–254, 1967.

[17] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Inf. Process. Lett.*, 31(1):7–15, 1989.

[18] W. Ke, K. Borner, and L. Viswanath. Major information visualization authors, papers and topics in the acm library. In *InfoVis04 Contest*, 2004.

[19] C. Muelder, K.-L. Ma, and T. Bartoletti. A visualization methodology for characterization of network scans. In *ACM VizSEC 2005 Workshop*, pages 29–38, 2005.

[20] Q. V. Nguyen and M. L. Huang. Enccon: an approach to constructing interactive visualization of large hierarchical data. *Palgrave Macmillan (online)*, 2005.

[21] A. Noack. An energy model for visual graph clustering. *Lecture Notes in Computer Science*, 2912:425–436, Mar. 2004.

[22] A. Quigley and P. Eades. Fade: Graph drawing, clustering, and visual abstraction. In *GD '00: Proceedings of the 8th International Symposium on Graph Drawing*, pages 197–210, London, UK, 2001. Springer-Verlag.

[23] K. Shi, P. Irani, and B. Li. An evaluation of content browsing techniques for hierarchical space-filling visualizations. In *IEEE Symposium on Information Visualization 2005*, 2005.

[24] B. Shneiderman. Tree visualization with tree-maps: 2-d space-filling approach. *ACM Trans. Graph.*, 11(1):92–99, 1992.

[25] C. Walshaw. A multilevel algorithm for force-directed graph drawing. In J. Marks, editor, *Proc. 8th Int. Symp. Graph Drawing, GD*, volume 1984, pages 171–182. Springer-Verlag, 20–23 2000.

[26] X. Wang and I. Miyamoto. Generating customized layouts. In F. J. Brandenburg, editor, *Graph Drawing, Passau, Germany, September 20-22, 1995*, pages pp. 504–515. Springer, 1996.

[27] J. H. Ward. Hierarchical grouping to optimize an objective function. *Journal of American Statistical Association*, 58(301):236–244, 1963.

[28] S. Zhao, M. J. McGuffin, and M. H. Chignell. Elastic hierarchies: Combining treemaps and node-link diagrams. In *IEEE Symposium on Information Visualization 2005*, 2005.