Navigating Giga-Graphs

James Abello
AT&T Labs-Research
abello@research.att.com

Jeffrey Korn
AT&T Labs-Research
jlk@research.att.com

Matthias Kreuseler
University of Rostock
mkreusel@informatik.uni-rostock.de

ABSTRACT

An effective way to process a graph that does not fit in RAM is to build a hierarchical partition of its set of vertices. This hierarchy induces a partition of the graph edge set. We use this partition to produce a macro view of the graph. A screen embedding of this macro view is a *Graph Sketch*. We describe the use of *Rectangular FishEye Views* to provide drill-down navigation of graph sketches at different levels of detail including the graph edges data. A higher level of detail of a sketch focus area is obtained by distorting the lower detail context. Alternative visual representations can be used at different sketch hierarchy levels. We provide two sketch screen embeddings. One is tree-map based and the other is obtained by a special sequence of graph edge contractions. We demonstrate the application of our current Unix/Windows prototype to telecommunication graphs with edge sets ranging from 100 million to 1 billion edges(*Giga-Graphs*). To our knowledge this is the first time that focus within context techniques have been used successfully for the navigation of external memory graphs.

Categories and Subject Descriptors

H.5.2 [**Information Interfaces and Presentation**]: User Interaces; I.3 [**Computing Methodologies**]: Computer Graphics; G.2.2 [**Discrete Mathematics**]: Graph Theory; H.3 [**Information Systems**]: Information Storage and Retrieval

General Terms

algorithms, experimentation, human factors

Keywords

visualization, massive data sets, hierarchies, graph sketches, Fish-Eye views, external memory algorithms.

1. INTRODUCTION

Telecommunications traffic [6], World-Wide Web [3] and Internet Data [4] are typical sources of graphs with sizes ranging from 1 million to 1 billion edges (i.e. *Mega* and *Giga-Graphs*). These graphs are not only too large to fit on the screen but they are in general also too large to fit in main memory. Therefore the screen and RAM sizes are the two main bottlenecks that we need to face in order to achieve reasonable processing and navigation. We present mechanisms to deal in a unified manner with both bottlenecks. They are based on the notions of *Graph Macro-Views*, *Graph Sketches* and *Rectangular FishEye Views*. Our examples come from AT&T call detail data, where vertices are telephone numbers and edges represent phone calls, weighted by the number of phone calls in the period of approximately 20 days (approximately 260 million vertices and 1.5 billion edges).

1.1 Approach

The central idea is to use a hierarchical decomposition of the graph edge set that is inherited from a hierarchical decomposition of the vertex set. This allows us to compute macro-views of the input graph at different levels of detail. A screen zoom-able embedding of a graph macro-view is called a *Graph Sketch* [5]. Navigation from one level of the edge hierarchy to the next is provided by refinement or partial aggregation of some sets in the corresponding partition. These operations correspond to zooming/unzooming on a local area of the sketch embedding (see Figures 5 and 6). We use *Rectangular FishEye Views*[8] to provide higher visual level of detail on the focus area distorting the lower detail context. In order to provide the user with a uniform mental map at all levels of the navigation we provide two linked views of the macro-graph structure. One is tree-map based (see Figure 4) and the other consists of a circular layout of the higher levels of the hierarchy tree (see Figure 3). The tree-map drives the navigation on the circular view. When a focus area is selected on the tree-map the corresponding subgraph at its next level is depicted. This is achieved by coupling in a very direct manner the focus area of a navigation with a selected node on a special macro-view of the input graph. This macro-view is obtained by recording the sequence of edge contractions executed by a top-down version of Boruvka's minimum spanning tree algorithm [2]. The edge contractions preserve the basic connectivity of the underlying graph and at the same time can be implemented efficiently on an external memory setting [9]. The obtained hierarchy tree provides in this manner enough information to recover the basic anatomy of the underlying minimum spanning tree and this in turn can be used to derive an anatomic macro-view of the entire graph.

Our approach provides a unified view of computation and visualization of very large graphs. Namely, visualizations become the product of graph decompositions that are tailored to a particular very large graph problem of interest. Different graph representations may be necessary for different goal driven navigations. We suggest searching for graph representations that encapsulate the essential features of either a clustering algorithm or a typical sub-

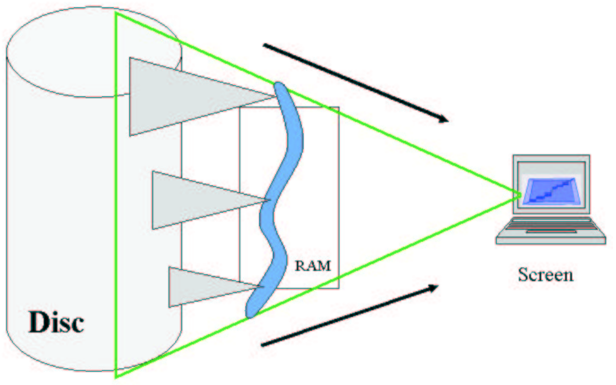

Figure 1: Overall Approach

space that contains a feasible answer. This paper presents techniques that are particularly helpful in guiding the navigation of very large graphs in order for a user to drive the computation towards a set of feasible answers.

It is worth mentioning that the approach advocated here allows the use of a commercial relational database to query the multi-digraph hierarchy with very little extra effort. Also, the techniques that we introduce can be adapted to a distributed computational environment.

1.2 Related Work

Multi-level graph views offer the possibility of drawing large graphs at different levels of abstraction. The higher the level of abstraction, the coarser the provided graph view. Compound and clustered graphs have been considered in [13, 14, 15]. The use of binary space partitions to produce graph clusters was introduced in [12]. However, the quality of the corresponding multi-level drawings depends heavily on the initial embedding of the graph on the plane. In [17], some of the limitations of force-directed based methods for drawing large graphs are addressed. *Graph Sketches* were introduced in [5] and its incorporation into a system called MGV was described in [7]. This work integrates these previous works with *Rectangular FishEye Views* to better navigate very large graphs. This is achieved by coupling in a very direct manner the focus area of a navigation with a selected node on a special macro-view of the input graph obtained via edge contractions.

1.3 Paper Layout

In Section 2, we discuss how graph *Macroviews* and graph *Sketches* are derived from hierarchical partitions of a graph vertex set. The fundamental algorithmic operations and their performance are discussed in Section 3. In Section 4 we present the details of an edge contraction based *sketch* that is modeled after Boruvka's algorithm to compute a Minimum Spanning Forest of a weighted graph [2, 9]. We also introduce there two screen embeddings of the macroviews introduced in Section 4. Implementation issues are covered in Section 5. Section 6 contains concluding remarks and points out some future research directions.

2. MACRO-VIEWS AND GRAPH SKETCHES

A partition of the vertex set of a graph provides a macro view of it by collapsing all the vertices on the same set of the partition into one macro-vertex and representing all the edges that run between two sets of the partition by a macro-edge with an appropriate multiplicity. More formally,

- A multi-digraph is a triplet $G = (V, E, m)$ where V is the vertex set, E a subset of VxV is the set of edges and $m : E \to N$ is a function that assigns to each edge a non-negative multiplicity. We denote by $V(G)$ and $E(G)$ the set of vertices and edges of G respectively. When there is no danger of confusion, we abuse notation and use V and E to refer both to the sets and their cardinalities.

- A multi-digraph $G' = (V', E', m')$ is called a k-*macroview* of a multi-digraph $G = (V, E, m)$ if V' is a partition of V with k subsets, where (u', v') is a macro edge in E' iff there exists u in u' and v in v' such that (u, v) is an edge in E. It is required also that $m'(u', v') = \sum_{(u,v) \in E(G)} m(u, v)$ for $u \in u'$ and $v \in v'$.

  We say in this case that the partition V' determines the macro-view G'.

When the vertex set of G has a hierarchical partition one can derive a special collection of macro-views of $E(G)$ where one macro-view can be obtained from the other by either a partial refinement or aggregation of sets in the partition (see Figure 1). In more formal terms,

- For a *rooted* tree T, let $Leaves(T)$ = set of leaves of T. Vertices p and q of a *rooted* tree T are called *incomparable* in T if neither p nor q is an ancestor of the other.

- Any maximal set C of incomparable nodes in T determines a partition of $V(G)$. The sets in the partition consist of $\{Leaves(q) : q \text{ is in } C\}$. We denote with G_C the macro-view of G determined by the partition associated with C. For every p in C, we add the self-loop $(p, p, m(p, p))$ in order to represent the subgraph of G induced by $Leaves(p)$ with $m(p, p)$ being its aggregated multiplicity.

- A zoom-able screen embedding of G_C is called a *Graph Sketch*.

3. NAVIGATING THE SKETCH HIERARCHY USING FISHEYE VIEWS

Given an embedded sketch edge (u, v) its zoomed version is defined as follows:

- Edge-zoom(u,v) : delete the macro-edge (u, v), replace u and v by their children; add all the macro-edges that run from the children of u to the children of v.

In order to provide efficient use of the screen space we use a FishEye View as follows. First we embed a macro-view of G determined by a complete level of T (i.e. all nodes at some fixed distance from the root). Then we compute a partition of the screen directly from the layout coordinates with the restriction that every embedded point belongs to exactly one region. Each screen region now becomes a potential logical window to which a FishEye View transformation can be applied, i.e. it can be selected as a focus region. This focus region is enlarged and the lower level context is distorted. In the enlarged focus region we display the details associated with the zoomed version of the edge (u, v) as defined above. Different screen partitions based on the layout coordinates can be obtained. In our current prototype we have implemented a simple grid subdivision which allow us to use Rectangular Fish Eye views along the lines presented in [8].

Our overall approach can be summarized as follows.

Overall Algorithm

1. Compute a rooted tree T such that $Leaves(T) = V(G)$ and such that every tree node has fan out not more than $\sqrt{d}$ where d is the number of available display pixels. This is a natural restriction since on a screen with d pixels we can display at most d edges.

2. For every edge (x, y) of G let $lca_T(x, y)$ denote the level of the least common ancestor of x and y in T. Edges (x, y) and (x', y') are labeled $l + 1$ iff $l = lca_T(x, y) = lca_T(x', y')$. Edges with the same label are placed on the same equivalence class for further processing.

3. Choose a level L_r of T such that its number of vertices is not more than $\sqrt{d}$ where d is the number of available display pixels.

   Compute the macro-view $G' = (V', E', m')$ of G where $V' = \{Leaves(u) : u \text{ is in } L_r\}$.

4. Set $Macroview = G'$.

5. Embed $Macroview$ on the screen and compute from its layout coordinates a grid partition of the screen in such a manner that every embedded point belongs exactly to one grid box.

6. Each grid box becomes now a potential logical window to which a Rectangular FishEye view transformation can be applied, i.e. it can be selected as a focus region. A focus region displays at a higher level of resolution the details of a macro-edge (u, v) where u and v are nodes in the current view.

7. Update $Macroview$

The most expensive part of the algorithm described above is the hierarchy computation (step 1). This may be tied to a particular navigational task. [5] introduces several examples of hierarchical graph decompositions. Some of them are Breath First Search driven and others are based on some form of low diameter network decomposition. Hierarchy trees of low diameter are important since the level of screen reuse provided by a focus-within context technique is severely limited if the hierarchy has very high diameter. In the case of telecommunication traffic and Internet and World Wide Web data, it has been observed experimentally that they are not only sparse but also of low diameter. If the obtained hierarchy does not satisfy this requirement it can be transformed to a larger hierarchy with lower diameter as a preprocessing step. This can be done by collapsing the deepest leaves in reverse postorder until a desired diameter is achieved. In case of very large hierarchies, we have taken the approach of precomputing some k levels of the hierarchy and expanding the remaining levels on demand. In the next section we describe a hierarchical decomposition of the vertex set of a graph that is driven by a top down version of Boruvka's minimum spanning tree algorithm. Our choice is based on the fact that contraction preserves connectivity and that Boruvka's algorithm implicitly builds a hierarchy of Minimum Spanning Forests which we can use as the basis of our sketch.

4. A BORUVKA SKETCH

We use an edge contraction based algorithm to compute a minimum spanning forest of a weighted graph as discussed in [9] (see Figure 2). The algorithm can be implemented by using simple techniques like sorting, selection and bucketing.

Namely, Let $G = (V, E)$ be a graph. Let $f(G) \subseteq V \times V$ be the delineated list of edges in a minimum spanning forest (MSF) of G.

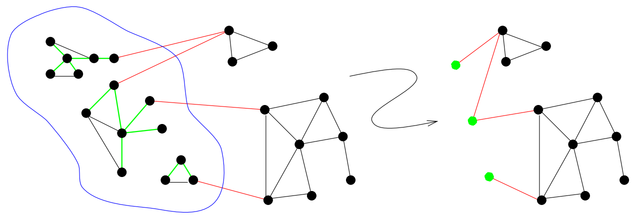

Figure 2: Contraction step in Boruvka's algorithm. Each connected component of the subgraph induced by the vertices inside the glob is contracted to a single vertex.

Consider $E' \subseteq V \times V$, and let $G' = G/E'$ denote the result of contracting all vertex pairs in E'. For any $x \in V$, let $s(x)$ be the super-vertex in G' into which x is contracted. Thus, given procedures to contract a list of components of a graph and to re-expand the result, one obtains the following simple algorithm, to compute $f(G)$.

A version of Boruvka's Algorithm

1. Let E_1 be the lower cost half of the edges of G, when they are sorted by weight, and let $G_1 = (V, E_1)$.

2. Compute $f(G_1)$ recursively.

3. Let $G' = G/f(G_1)$.

4. Compute $f(G')$ recursively.

5. $f(G) = f(G_1) \cup R(f(G'))$, presented as a delineated list, where R is the inverse of the contraction in Step 3: each edge in $f(G')$ is replaced by the corresponding original edge in G.

An essential operation for the efficient implementation of this algorithm is to *Relabel* a set of edges according to a Forest F. This is really a very fundamental form of contraction. We refer the reader to [9] for details.

4.1 The Boruvka Hierarchy Tree

In order to derive a tree T such that $Leaves(T) = V(G)$ all we need to do is to record for each vertex x its associated super-vertex $s(x)$ in G' into which x is contracted during Boruvka's algorithm. Initially, $x = s(x)$. The depth of the obtained tree is logarithmic and we can stop the recursion when a subproblem fits in internal memory. The overall I/O complexity is $O(sort(E)log_2 \frac{E}{M})$ where M is the size of main memory.

Each vertex u on this Boruvka hierarchy tree (with the possible exception of the root) represents the subgraph induced by the set of vertices of G which are descendant leaves of u. We exploit this fact visually by associating with an embedded hierarchy tree node its associated subgraph. The subgraph becomes the higher detail resolution of the hierarchy tree node, i.e. when a node is selected as a focus node its corresponding bounding box is enlarged and the corresponding subgraph can be explored in that region in a similar fashion (see Figures 5 and 6).

In order to provide overall context we select a maximal set C of incomparable nodes in the Boruvka hierarchy tree such that the size of the subtrees rooted at the elements of C is not above a prespecified bound (in general this is a function of the available number of pixels and/or a time-space constraint).

Contracting the Minimum Spanning Forest output by Boruvka's algorithm according to the chosen C give us a Spanning forest of

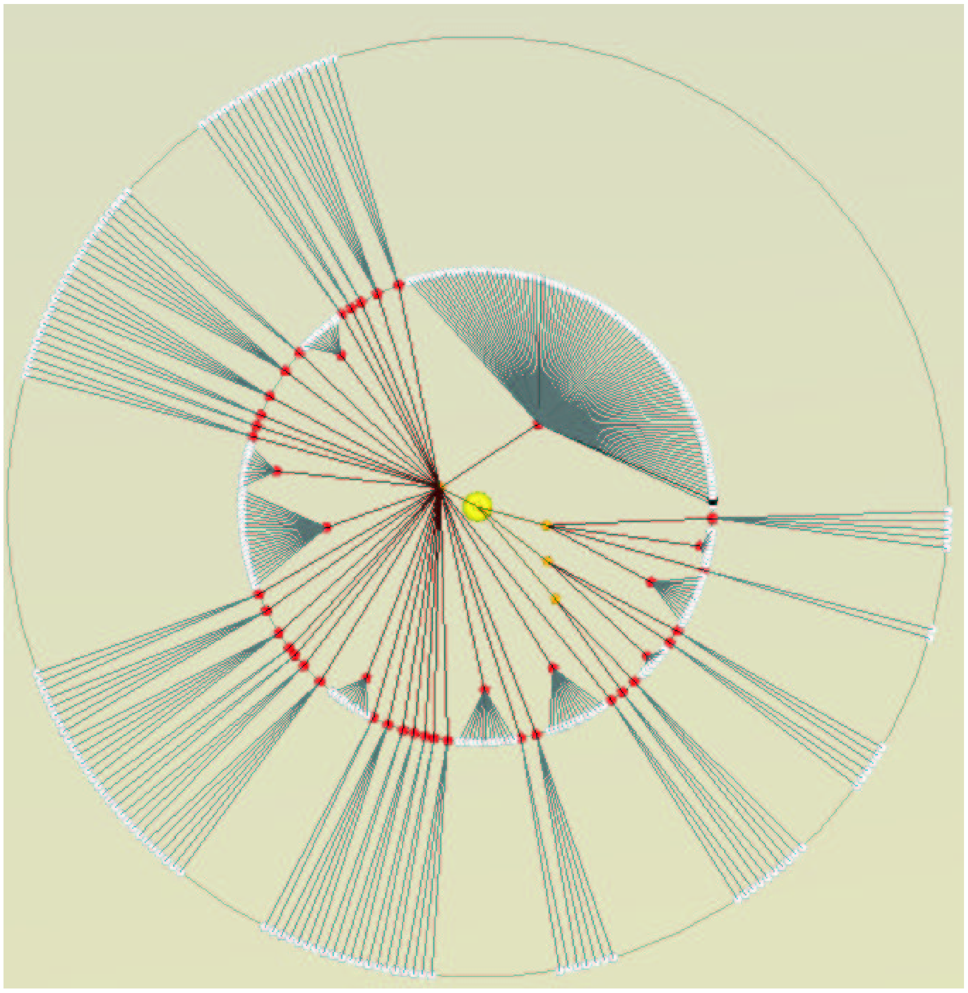

Figure 3: **The leaves of this hierarchy tree represent a partition of 260 million vertices. Interior nodes represent the collapsing of a collection of subgraphs into a node as mandated by Boruvka's algorithm. Each tree vertex represents the subgraph induced by its descendant leaves. The density of this subgraph is encoded by the color of the node. Vertices at the same tree distance from the root appear on the same circumference. Vertices representing larger subgraphs are pushed closer towards their parents. This provides a local macroview of a region in a graph too large to be displayed on screen. Selecting a pair of incomparable vertices in this hierarchy tree corresponds to querying the underlying data for the subgraph whose edges run from the descendant leaves of one node into the descendant leaves of another. The deeper these nodes are in the hierarchy tree, the smaller the corresponding subgraphs are. At these lower levels of granularity more traditional methods of visualization can be used (see Figure 6).**

the sketch G_C. This spanning forest contains the essential macro-connectivity information of the input graph G . We use a specialized circular layout of this spanning forest of the sketch G_C to highlight those areas of the original graph with potentially more "interesting" subgraphs (see Figure 3). Simultaneously we present a tree-map view of the sketch G_C which we use to drive the navigation (see Figure 4). These two views are linked and this facilitates the use of large screens. Namely, the tree-map view can be used on a workstation to drive the navigation of a Graph sketch that is embedded on a large screen.

5. IMPLEMENTATION ISSUES

The current prototype is implemented in C++. It consists of a navigator library and a GUI. The navigator library consists of two separated sets of functions. One of them provides an interface to the data objects and the other builds the corresponding geometric objects. The GUI is in charge of rendering the created geometry. This design allow us to interface with external data processors that produce hierarchy trees using their own algorithms. For example, to incorporate a Boruvka sketch into the visualization, a data processor needs to provide the corresponding initial hierarchy tree and later on must be able to send to the navigator the data associated with a hierarchy tree node on demand. This makes the visualization independent of the particular algorithm used for the clustering of the underlying graph.

The GUI is implemented using Qt [18]. Qt allows development of cross-platform applications that can be run on Windows, Mac and a variety of Unix/Linux platforms. Qt is fully object oriented and easily extensible. The type-safe *signals* and *slot mechanisms* provide powerful capabilities for registering events, message handling and user interactions.

6. CONCLUSIONS

In order to navigate a graph whose size is larger than the screen size a rectangular FishEye View becomes useful if a macroview of the graph can be embedded in a way that is locally zoom-able. This

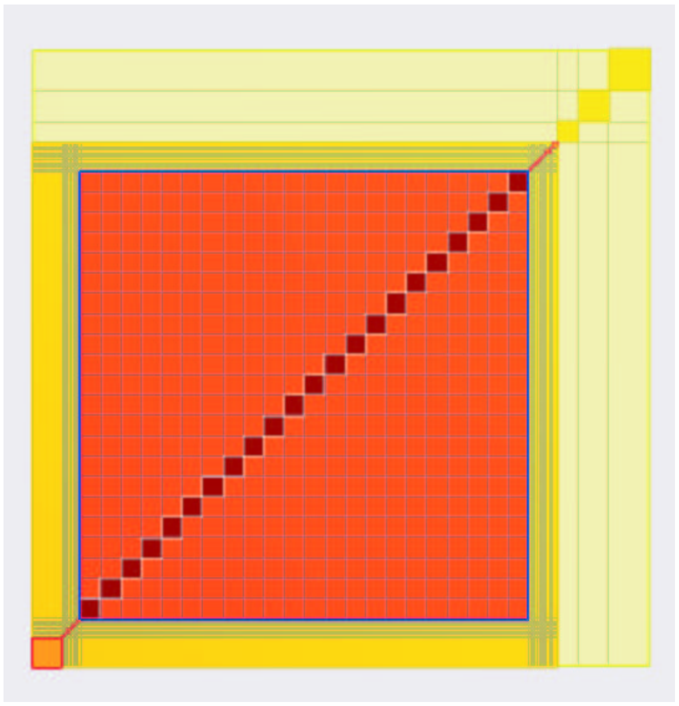
Figure 4: Tree-map Visualization. Each box corresponds to a vertex in the hierarchy tree.

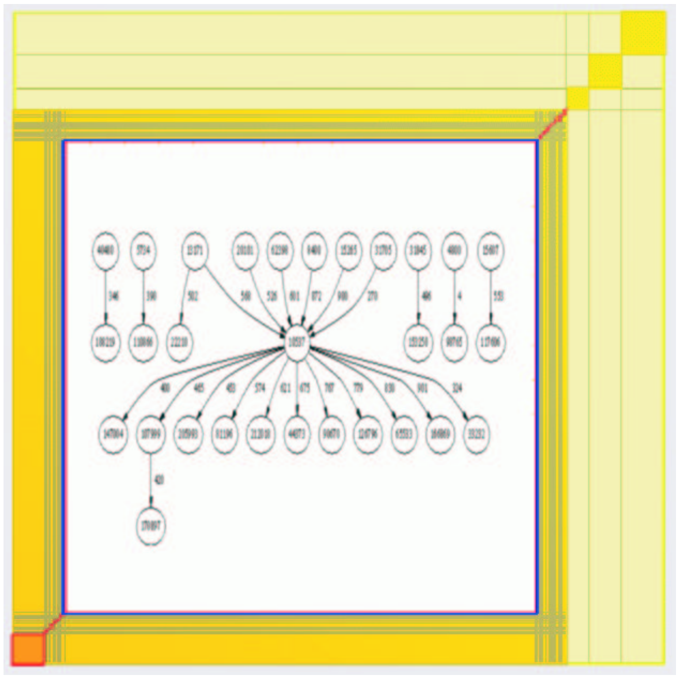
Figure 5: Zooming into a node of the hierarchy. A traditional boxes and arrows display is embedded into the focus area.

corresponds abstractly to the use of a recursive algorithm in the decomposition of the input graph. The unifying concept of these ideas is the notion of a hierarchical decomposition of the edge set of a graph that is derived from a hierarchical decomposition of the vertex set. Extrapolating this idea, in order to navigate a graph whose

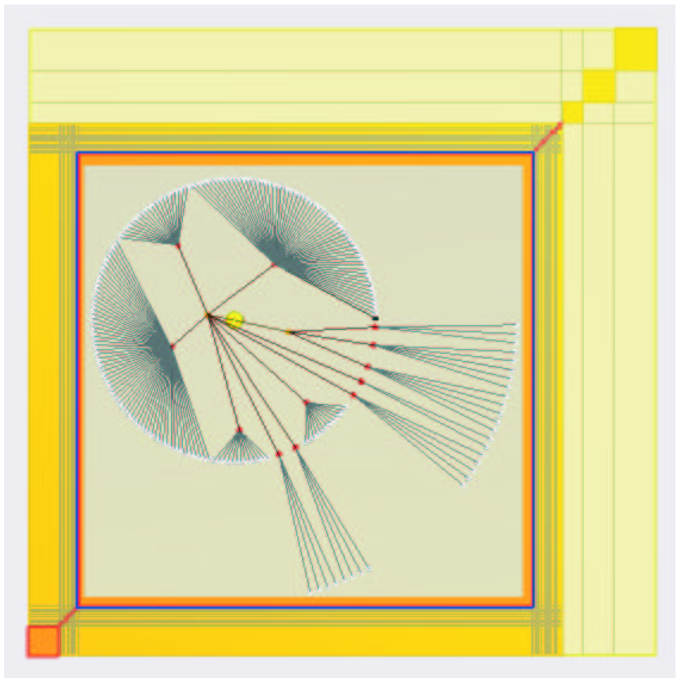
Figure 6: An alternative embedding in the focus area depicting the tree hierarchy below the focused node.

size is larger than the available RAM one needs to build an index in RAM to a disk resident partition of the vertex set. A hierarchy in memory is built whose set of leaves corresponds to the index entries. This hierarchy is mapped to the screen in a locally zoom-able embedding and when navigation asks for details which are not resident in RAM, they get computed on demand. This suggests that an important and useful research direction to pursue is to invent screen embeddings that are recursive in nature but that have certain overall level of uniformity. These type of embeddings coupled with a suitable Focus Within Context technique give a better utilization of the screen space. In fact, it enlarges it. Requiring a global level of uniformity minimizes the user tendency to get disoriented as is the case when navigating with hyperbolic views. Rectangular FishEye Views offer a great level of uniformity and are efficient to compute and maintain. Other subdivisions of the screen space based on Voronoi regions may be worthwhile to consider.

We would like to remark in closing that the use of visualization on the exploration of massive data sets is a very promising area of highly interdisciplinary nature. The work reported here shall be taken as an infant step towards the representation and fluid navigation of complex systems.

7. REFERENCES

[1] B. Shneiderman. Information Visualization: Dynamic queries, starfield displays, and LifeLines. In *www.cs.umd.edu*, 1997.

[2] O. Borůvka. O Jistém Problému Minimálním In *Práce Moravské Přírodovědecké Společnosti v Brně (Acta Societ. Science. Natur. Moravicae*, Vol. 3, pp.37-58, 1926.

[3] A. Broder Graph Structure in the Web. In *Networks*, Vol. 33, pages 309-320, 2000.

[4] M. Faloutsos, P. Faloutsos, C. Faloutsos. On power-law relationships of the Internet topology. In *Comp. Comm. Rev.*, Vol. 29, pages 251-262, 1999.

[5] J. Abello, I. Finocchi, J. Korn. Graph Sketches. In *IEEE Proc. Information Visualization*, pages 67-71, San Diego, Ca, 2001.

[6] J. Abello, J. Korn. Visualizing Massive Multi-Digraphs. In *IEEE Proc. Information Visualization*, pages 39-47, Salk Lake City, 2000.

[7] J. Abello, J. Korn. MGV: A System to Visualize Massive Multi-Digraphs. In *IEEE Transactions on Computer Graphics and Visualization*, Vol. 8, No. 1, pages 21-38, Jan. 2002.

[8] U. Rauschenbach, S. Jeschke, H. Schumann. General Rectangular FishEye Views for 2D Graphics. In *Proc. Intelligent Interactive Assistance and Mobile Computing,*, IMC, 2000.

[9] J. Abello, A. Buchsbaum, and J. Westbrook. A functional approach to external memory graph algorithms. In *European Symposium on Algorithms*, volume 1461 of *Lecture Notes in Computer Science*, pages 332–343. Springer-Verlag, 1998.

[10] J. Abello, S. Krishnan. Navigating Graph Surfaces. In *Approximation and Complexity in Numerical Optimization: Continuous and Discrete Problems*, P. Pardalos(Ed.), pages 1-16. Kluwer Academic Publishers, 1999.

[11] J. Abello, J. Vitter. (Eds) External Memory Algorithms. Volume 50 of the AMS-DIMACS Series on Discrete Mathematics and Theoretical Computer Science, 1999.

[12] C. Duncan, M. Goodrich, S. Kobourov. Balanced Aspect Ratio Trees and Their Use for Drawing Very Large Graphs. Lecture Notes in Computer Science, 1547:111-124, 1998.

[13] P. Eades, Q. W. Feng. Multilevel Visualization of Clustered Graphs. Lecture Notes in Computer Science, 1190:101-112, 1997.

[14] P. Eades, Q. W. Feng, X. Lin. Straight-line drawing algorithms for hierarchical and clustered graphs. In *Proc. 4th Symp. on Graph Drawing*, pages 113-128, 1996.

[15] K. Sugiyama, K. Misue. Visualization of structural information: Automatic drawing of compound digraphs. In *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 21, No 4, pages 876-892, 1991.

[16] D. Harel, Y. Koren. A fast multi-scale method for drawing large graphs. *TR MCS99-21*, The Weizmann Institute of Science, Rehovot, Israel, 1999.

[17] P. Gajer, M. Goodrich, S. Kobourov. *A multidimensional approach to force directed layouts of large graphs.* In *Proc. of Graph Drawing*, Lecture Notes of Computer Science, Springer Verlag, 2000.

[18] QT. `http://www.trolltech.com`