Matrix Zoom: A Visual Interface to Semi-external Graphs

James Abello*
DIMACS, Rutgers University
Piscataway, NJ
USA

Frank van Ham†
Department of Mathematics and Computer Science
Technische Universiteit Eindhoven
The Netherlands

ABSTRACT

In web data, telecommunications traffic and in epidemiological studies, dense subgraphs correspond to subsets of subjects (i.e. users, patients) that share a collection of attributes values (i.e. accessed web pages, email-calling patterns or disease diagnostic profiles). Visual and computational identification of these "clusters" becomes useful when domain experts desire to determine those factors of major influence in the formation of access and communication clusters or in the detection and contention of disease spread. With the current increases in graphic hardware capabilities and RAM sizes, it is more useful to relate graph sizes to the available screen real estate S and the amount of available RAM M, instead of the number of edges or nodes in the graph. We offer a visual interface that is parameterized by M and S and is particularly suited for navigation tasks that require the identification of subgraphs whose edge density is above certain threshold. This is achieved by providing a zoomable matrix view of the underlying data. This view is strongly coupled to a hierarchical view of the essential information elements present in the data domain. We illustrate the applicability of this work to the visual navigation of cancer incidence data and to an aggregated sample of phone call traffic.

CR Categories: H.5.2 [Information Interfaces and Presentation]: User Interfaces—Graphical User Interfaces; H.2.8 [Database Management]: Database Applications—Data Mining I.5.5 [Pattern Recognition]: Clustering—Algorithms

Keywords: Graph Visualization, Hierarchy Trees, Clustering, External Memory Algorithms, Cancer Data, Phone Traffic

1 INTRODUCTION

A variety of data sets can be interpreted as a relation A between a set of subjects K (i.e. users, patients, entities, etc) and a set of attribute values L (i.e. web pages accessed, diagnosis characteristics, entities compounds, etc). A can then be viewed as a $|K|$ by $|L|$ matrix where A[x, y] = 1 if and only if subject x has attribute value y, i.e. A becomes the adjacency matrix of a graph G with vertex set $V(G) = K \cup L$. In many applications, the set of subjects and the set of attributes are disjoint (i.e. the graph is bipartite), however this is not a restriction since any graph has a unique adjacency matrix representation (modulo columns and row permutations). This matrix view of a graph opened the very successful use of spectral methods to elucidate inherent graph clustering structure of several sorts [13]. This line of research has been used to produce aesthetically pleasing node-link graph layouts [11]. A natural way to proceed would then be to obtain a hierarchical clustering on the layout and choose from it a coarse enough level that can be then embedded on the screen. This begs the question of how then one can zoom into the current node-link layout to obtain the next node-link level of granularity. Zooming into a vertex offers no major problem since this amounts to allocating enough screen space around the vertex to locally embed the lower level subgraph that it represents [3].

However, zooming into an aggregated edge leaves us with the unpleasant task of embedding locally the collection of edges it represents with the added constraint that their endpoints already have predetermined positions. The problem becomes aggravated when the collection of edges at a lower level is so large that their local embedding clutters the available screen space. This is in our view one of the major difficulties in using node-link diagrams to visually navigate large graphs even if a hierarchical clustering has been computed. On the other hand, if a clustered graph is visually embedded as an adjacency matrix there is no difference between zooming into edges or vertices since in both cases there is a well defined local area of the screen into which we can zoom in. This is the main justification to use matrix-based representations of large clustered graphs if the user task at hand requires navigation at different levels of granularity.

Another reason is that, very often, higher levels of clustering tend to produce very dense graphs and in this case nodelink diagrams are again not very well suited. In summary, in this work we concentrate on the problem of navigating hierarchically clustered graphs and we assume that such a clustering is provided to us as a hierarchy tree on the vertex set of an input graph (semi-external). We illustrate the use of the proposed interface to navigate data provided by the SEER program of the US National Cancer Institute and we also use our interface to navigate a hierarchical view of a phone traffic data set for which not even the vertex set fits in RAM (i.e. a fully external memory graph). In the next section we discuss related work and present an overview of our ideas. Section 3 describes the framework that can be used to efficiently navigate semi-external graphs. Section 4 gives an overview of the visual interface and its main features. Section 5 illustrates navigation results on cancer incidence data and on aggregated sample of phone traffic. Finally, we present conclusions and further work in section 6.

2 RELATED WORK AND PAPER OVERVIEW

The incidence matrix of a graph allows to some extent visualization of a clustering. This amounts to relabelling the nodes of the graph such that nodes in the same cluster have consecutive labels. If there are enough edges within a cluster (i.e. if its density is high) and few edges going out of it then the resulting permuted matrix exhibits a diagonal block structure with the remaining "few" entries scattered in the remaining portions of the matrix. Different reordering algorithms [6] produce then potentially different clusters, however standardized comparison of cluster methods is virtually non-existent. Usually, the solutions quality is measured in terms of a cost function on the produced clusters, but devising such cost functions is non-trivial [5].

Having a clustering, each of the clusters gets collapsed into a macro vertex and the edges between clusters are aggregated into macro edges providing a coarse view of the input graph. Iteration of this process is what is referred in the literature as hierarchical graph

*e-mail: abello@dimacs.rutgers.edu
†e-mail: fvham@win.tue.nl

IEEE Symposium on Information Visualization 2004
October 10-12, Austin, Texas, USA
0-7803-8779-1/04/$20.00 ©2004 IEEE

clustering [10]. The required data structure to maintain the obtained clusters is a Hierarchy Tree. It is simply a rooted tree T whose set of leaves is in one to one correspondence with the vertices of the input graph. Each internal tree node represents a cluster obtained by collapsing its set of descendant leaves (see figure 1). A view of the input graph consists then of a careful selection of a set of nodes in the hierarchy tree that corresponds to a partition of the graph vertex set and its induced macro view. Navigation from one macro view to another corresponds to refinement or aggregation of selected subsets in the partition. We tune this navigation by using memory, screen, time and graph size parameters as upper bounds on the available resources.

Navigation tuning becomes essential especially when the input graph is too large to fit on the available RAM (the I/O bottleneck). Even if the graph fits in RAM it is necessary to control the screen space usage since the number of pixels available is often scarce (the screen bottleneck) compared to usual graph sizes. In order to offer predefined levels of interactivity it is necessary to provide efficient access mechanisms to the raw graph data and fast layout algorithms for selected subgraph slices. We address all these issues in a unified manner by building on the work of [2, 3, 14], paying special attention to improving the interface. We offer quite reasonable levels of user interactivity by a combination of fast data access, selective use of both matrix and node link diagrams, and fast semantic and geometric zooming. Needless to say that the fundamental data structure behind the scenes is an enhanced hierarchy tree. More succinctly, our contributions are:

- a. A matrix driven interface to browse hierarchically clustered graphs whose vertex set fits in RAM but whose edge set does not. It is worth to point out that the use of a matrix representation does not preclude the use of node link representations at lower levels of granularity. In fact, the navigation interface is equipped with parameterized controls that allow the use of both representations at the user's discretion.

- b. The incorporation of screen, graph and RAM parameters into each node of the hierarchy tree to empower the navigation algorithm to make decisions about the type of graph representation that is most suitable at each level of the hierarchy (section 3.3)

- c. The use of a variation of a kd-index to access subgraph slices when the input graph is too large to fit on the available RAM (section 3.2)

- d. A strong visual coupling of the hierarchy tree with the canvas display (Figure 3). This provides in effect a uniform and permanent hierarchical view of the entire input graph. Navigation from one macro view to another is explicitly presented.

- e. Mechanisms to provide a smooth transition from graph matrix views into node link representations. At this stage, vertices are visually ranked according to a local peeling decomposition.

- f. Illustration of the applicability of the methods to the fluid navigation of US SEER Cancer data [8] when viewed as a bipartite graph with about 6 million edges and a high level view of a fully external graph with about 250 million vertices (section 5).

3 Memory and Screen Bounded Graph Macro-views

We assume all through out that $|M|$ is the size (in words) of random access memory and that $|S|$ is the size of the available screen. We reserve the term nodes to refer to vertices of a tree.

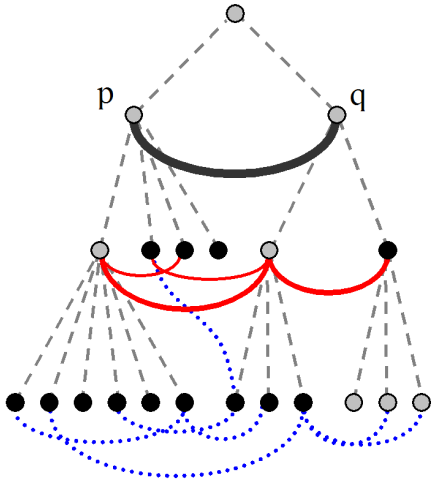


Figure 1: Schematic illustration of graph theoretic concepts. The set of dark nodes is a sample of a maximal antichain for this tree. The operation details(p,q) returns the subgraph consisting of all leaf nodes and the dotted edges(painted blue), while expansion(p,q) yields the subgraph consisting of edges with both endpoints at depth 2(painted red).

3.1 Definitions

- A **multi-digraph** is a triplet $G = (V, E, m)$ where V is the vertex set, E a subset of $V \times V$ is the set of edges and $m : V \times V \rightarrow R+$ 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 (assume that $m(x,y)$ assigns the value 0 to non-edges of G). When in need of emphasizing those edges that have multiplicities we use the term multi-edges. The weighted adjacency matrix of G is a matrix $A(G)$ where the entry (x,y) contains the value $m(x,y)$. G is called a semi-external graph if the vertex set fits in RAM but not the edge set (i.e. $|V(G)| < |M|$ but $2 \times |E(G)| > |M|$ assuming that a vertex and and an edge fit in one and two memory words respectively [1]). When both $|V(G)|$ and $|E(G)|$ do not fit in RAM , G is called fully external.

- **Antichain.** For a rooted tree T, and a node p in T let T_p denote the subtree of T rooted at p and let Leaves(T) denote the set of leaves of T. Nodes p and q of T are called incomparable in T if neither p nor q is an ancestor of the other. A set of pairwise incomparable nodes in T is called an **antichain**. Any maximal antichain in a tree T corresponds to a partition of $Leaves(T)$. $Leaves(T)$, $Root(T)$ and any maximal set of tree nodes at the same distance from the root are the most simple examples of antichains.

- **Hierarchy Tree.** A rooted tree T is called a **hierarchy tree** for a graph G if Leaves(T) = $V(G)$. We assume that from each leaf of T we have random access to the in-degree and out-degree of its corresponding vertex in $V(G)$. This is a reasonable assumption since this information can be obtained in one pass over $E(G)$ provided that the vertex set fits in RAM (i.e. G is semi-external).

3.2 Hierarchical Coordinates for the Adjacency Matrix of G

Starting from the root, a depth first search (dfs) traversal of the hierarchy tree T, for a multi-digraph G, provides a hierarchical set of

coordinates for G's adjacency matrix $A(G)$ as follows. Associate with every node p in T, the interval of dfs numbers going from $dfs(p)$ to the maximum dfs number in the subtree T_p; call such an interval I(p). This collection of intervals provides a hierarchical view of A(G) by mapping the entire matrix to the square with coordinates [0,0], [0, maxdfs of T], [maxdfs of T, 0] and [maxdfs of T, maxdfs of T]. This process is iterated recursively. We obtain in this fashion a hierarchical subdivision of space where each incomparable ordered pair of nodes (p,q) in T is associated with the submatrix cell $I(p) \times I(q)$.

- The cell $I(p) \times I(q)$ corresponds to a subgraph of G that we call **details**(p,q) whose vertex set and edge set are defined as follows: V(**details**(p,q)) = Leaves(T_p) $\cup$ Leaves(T_q) and E(**details**(p,q)) = $\{(u,v) \in E(G)$ such that $u \in$ Leaves(T_p) and $v \in$ Leaves(T_q)$\}$.

The number of non-zero entries in the submatrix cell $I(p) \times I(q)$ is precisely |E(details(p,q))| and the density of this subgraph is $|E(\text{details}(p,q))| \div |I(p) \times I(q)|$. We are interested in detecting subgraphs details(p,q) with density above certain threshold. Since the central statistic that is carried all trough out in a hierarchical manner is precisely $|E(\text{details}(p,q))|$ we record its definition for future reference.

- The multiplicity of an ordered pair of nodes p and q in a hierarchy tree T is

$$m(p,q) = \sum_{(u,v) \in E(G)} m(u,v)$$

where u $\in$ Leaves(T_p) and v $\in$ Leaves (T_q)

The ordered pair (p,q) is called a virtual multi-edge if $m(p,q)$ is non zero. Notice that in general a virtual multi-edge (p,p) represents the subgraph of G induced by Leaves(T_p) and $m(p,p)$ is its aggregate multiplicity. These types of multi-edges correspond to local interactions. Aggregate interactions at the same level of the hierarchy correspond to multi-edges that are at the same distance from the root (horizontal edges). In the case of phone call traffic, they represent traffic between regions, states, counties, towns, etc.

- For a virtual multi-edge (p,q), expansion(p,q) is the multi-digraph with vertex set equal to children$(p) \cup$ children(q) and all the multi-edges running between children(p) and children(q).

A good mental picture is that each virtual multi-edge (p,q) has its own hierarchy of edge slices where each level represents an aggregation of previous levels and where the bottom most level is the subgraph of G that we call details(p,q) consisting of the directed edges running from Leaves(T_p) to Leaves(T_q). In order to have fast disk I/O access to these subgraphs we use a greedy variation of a kd tree index called the gkd index as proposed in [4]. During data loading, besides constructing the gkd tree we also build a redundant R*-tree that indexes the leaf pages of the gkd tree. In this way, fast construction and balanced lookups are possible because the redundant R* tree can be built without scanning the gkd leaves themselves. The details of how this is achieved can be found in [4].

3.3 Screen Bounded Hierarchy Trees

Since the number of screen pixels available is the ultimate constraint for a visualization we introduce now a process called hierarchy tree regularization. Its purpose is to embed a given hierarchy tree T into a possible larger hierarchy tree RT such that the number of children of each node is not more than a specified integer value $|S|$ which corresponds to the available screen space. Notice that RT must have the same set of leaves as T. Starting at the root (in breadth first search order), if the number of children of a node p is more than $|S|$, they get grouped into groups of at most $|S|$ nodes each. Each group of nodes is assigned to a common parent and these artificial parent nodes become the new children of p. To make the subdivisions as meaningful as possible without exceeding our computation time budget, we first check wether there are any childnodes that have no incident edges. This can be done efficiently since we have random access to the degree for each node. Next, we check if there are any leafnodes mixed in with non-leafnodes and aggregate both in a separate group. If both of these tests fail we aggregate nodes into S groups by the order in which they are processed. Note that, time and memory space permitted, we can use any kind of clustering criteria at this point. We could even compute an explicit clustering on the child nodes and the edges among them, but this would take in general an excessive amount of time.The artificial nodes introduced by this process are colored lightly in the interface to indicate that they were introduced for the purposes of visual navigation only, i.e. they do not necessarily convey application domain semantics (see Figure 3).

function RegularizeTree (Tree T, Screensize S)

- **Input:** An enhanced hierarchy tree T for a multi digraph G and a parameter $|S|$ corresponding to the number of available screen pixels.

- **Output:** A hierarchy tree RT for G such that each node in RT has at most $|S|$ children.

```
begin
  for each node n in T in BFS order do
    RegularizeNode(n,S);
end;

function RegularizeNode(Node p, Size S)
begin
  if nrchildren(p) > S then
    if nrisolatedvertices(children(p)) > 0 then
      group all isolated vertices under new parent
    else
      if nrleafchld(p) > 0 and nrleafchld(p) < nrchld(p) then
        group all leafnodes under new parentnode
      else
        subdivide children under |S| new parentnodes;
end;
```

3.4 RAM bounded Macro Views of a Semi External Graph

In this section we show how we can use the concepts of antichains and the regularized hierarchy tree of the previous section to generate a RAM resident macro view of any graph.

- For a multi-digraph G with an enhanced hierarchy tree T as above, a maximal antichain in T is called a T-cover of G. Notice that Leaves(T) is a T-cover of G and that substituting in a T-cover C a set of nodes by their common parent produces another T-cover C'. In this case we say that C' is **smaller** than C. T-covers provide layered views that facilitate localized navigation of the data. Their cardinality can be as small as 1 (i.e. the root of the tree) or as large as the number of leaves in T.

- A T-view of G is the multi digraph with vertex set the nodes of a T-cover and all the virtual multi-edges running among them.

Since G is assumed not to fit in RAM (which we assume to be larger than the available screen S) we need to compute a T-view whose number of edges fits in memory(M). We show next how to obtain such RAM bounded T-views for an external memory graph (see the pseudo code for the function *Cover*). The idea is to descend from the root in the hierarchy tree by *expanding* (see definition in section 3.2) only those nodes in the current antichain with the greatest degree provided that their substitution by their children still fits in RAM (i.e. the obtained antichain size is less than $|M|$).

Function Cover (Size $|M|$, Tree RT)

- **Input:** An upper bound $|M|$, an enhanced and regularized hierarchy tree RT for a multidigraph G.

- **Output:** A greedy RT-Cover C with no more than $|M|$ elements such that Maximum outdegree(C), indegree(C) is minimum.

begin
C = root(RT);
currentCoverSize = 1;
degree = outdegree(C) = indegree(C) = $|E(G)|$;
C = sort C by non-decreasing order of out degree;
notFinished = true;
while (notFinished) do
for each p in C do
if (|children(p)| + currentCoverSize - 1 $<$ $|M|$)
then
newC = newC $\cup$ children(p) - p;
currentCoversSize = $|newC|$
if ($|newC| = |C|$) **then** notFinished = false;
C = newC;
end

3.5 Antichain Driven Navigation

Using the combination of hierarchy tree regularization, memory bounded T-covers and a fast index to retrieve graph slices one can process in principle any secondary storage multi-digraph defined on millions of vertices provided that an explicit enhanced hierarchy tree T is provided. In order to provide reasonable levels of interactivity we focus next on tuning the navigation process . This is achieved by allowing the interface to jump directly from a virtual multi-edge (p,q) to a view $T_{p,q}$ whose number of multi-edges is not more than $|S|$. Such view is obtained by invoking the function Cover twice with input T_p and T_q and screen parameter $\sqrt{|S|}$ and taking the bottom up aggregation of $details(p,q)$ determined by the two obtained local antichains (i.e. local tree covers, see figure 2). We call the obtained view $T_{p,q}$ the local cross product view of the multi-edge (p,q). Its number of multi-edges is certainly not more than $|S|$ (the specified screen budget). Using this process we can quickly navigate the entire tree without having to click through each individual level.

3.6 Overall Computational Flow

We present next the overall view of the main computations involved in preparing a data set for interface navigation.

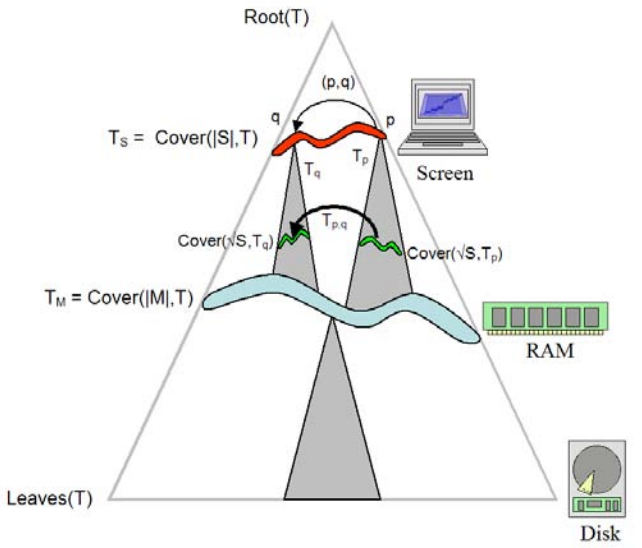


Figure 2: Schematic depiction of antichain driven navigation. The antichain T_M is a cover of an external tree T, indicating the part of the tree that fits in RAM. The antichain T_S is a smaller cover of T that fits on screen. When quick-zooming into and edge (p,q) we compute new antichains on T_p and T_q and display the crossproduct of both antichains $T_{p,q}$.

- **Input:** Memory and Screen Parameters, $|M|$ and $|S|$. A disk resident multi-digraph $G = (V,E,m : V \times V \rightarrow R+)$ with the in and out degree of each vertex. A memory resident hierarchy tree T for G with a domain specific label associated to each node of T.

- **Output:** Two macro views T_M and T_S of G with the first one bounded by $|M|$ and the second one bounded by $|S|$.

I. **Screen Regularization of** T
RT = *Regularize*(T, $|S|$)

II.**Enhance the hierarchy tree RT**
Traverse RT in dfs (depth first search) order from the root assigning to every node $p \in RT$, the min and max dfs values in the set Leaves(RT_p). The interval of dfs numbers between these two values is a subset of the interval I(p) defined in section. We refer to this subinterval as $LI(p)$.

Store with node p, the values in(p) and out(p), of the sum of all the indegrees and outdegrees of its descendant leaves(i.e. the nodes in Leaves(RT_p)).

III. **Build a gkd index to access E(G) according to RT** [4]

IV. **Compute a T_M view of G using the subintervals LI(p) defined above**
Compute C_M = *Cover* ($|M|$, RT) and use the mapping that associates with every p in C_M the interval of dfs numbers $LI(p)$ to externally sort $E(G)$ and aggregate the results according to the cover C_M. The obtained multi digraph is a T_M view of G. Based on edge density greedily select a subset of virtual multi-edges in T_M and use the gkd index to cache their corresponding subgraphs.

V. **Compute from T_M a bottom up T_S view**

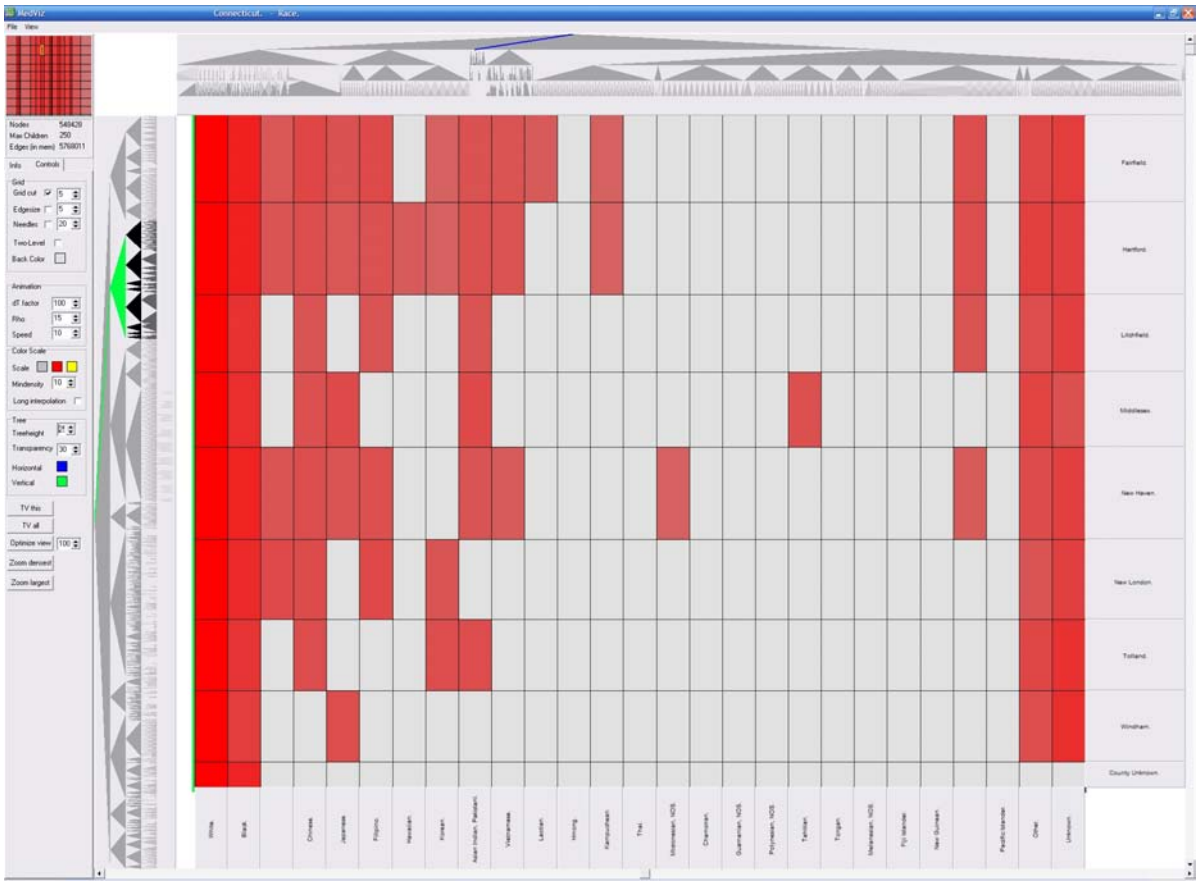


Figure 3: A screenshot of the interface, showing racial distribution for colon cancer cases in Connecticut counties. The overview window is in the top left, hierarchy trees are on the top and left of the matrix, node labels are on the bottom and right. Three counties (rows 1,2 and 5) show more diversified racial distributions.

In order to guarantee that the cover T_S is smaller than T_M we perform an upward aggregation of nodes in T_M. Let Temp denote the subtree of RT consisting of its root and all the paths from it to the elements of C_M, i.e. Leaves(Temp) = C_M. Consider C'_M to be the antichain consisting of all the parent nodes of elements in C_M. A node p in C'_M can be added to C_M and its children can be deleted from C_M if the sum of the degrees of p's children is minimum. The process is iterated until one antichain is found whose cardinality is not more than $|S|$. Call this antichain C_S. Aggregation of T_M according to C_S provides a view T_S of T_M (which in turn is a view of G) that can be embedded on the available screen.

VI. **Prepare for antichains driven navigation**
For every node p in C_S compute Cover($|S|$,RT_p). For specially selected virtual multi-edges (p,q) where p and q are in C_S, compute in memory all the multi-edges between Cover($|S|$,RT_p) and Cover($|S|$, RT_q). This is done by aggregating the in memory resident T_M view of G. Those multi-edges for which this T_p, q view is not precomputed are computed on demand from the interface.

The I/O complexity of the overall computation is dominated by the gkd index construction and by the number of virtual edges (p,q) that we want to cache. This is necessary in order to offer a reasonable level of interactivity (Steps III and IV above). The internal computation is dominated by the time taken to aggregate the in memory resident T_M view of G.

In the next section we discuss how these notions translate into visual navigation of a semi-external memory graph.

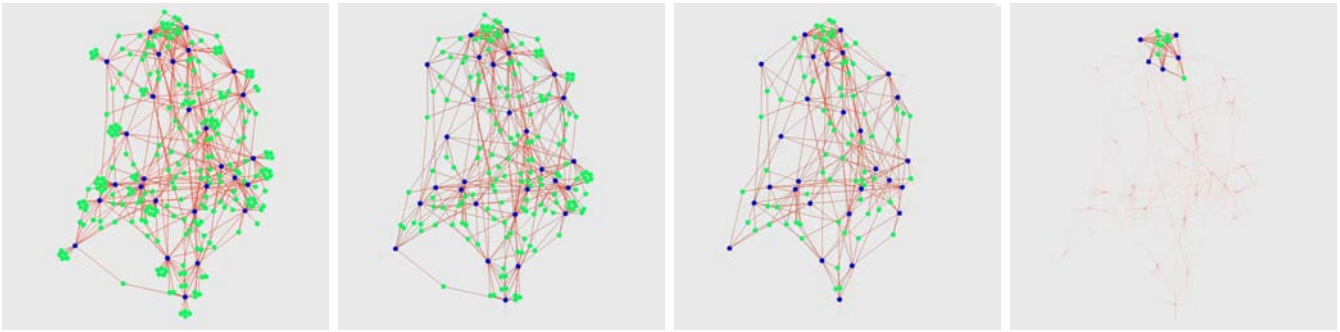
4 Matrix Zoom: A Matrix Driven Interface

Providing visual access to detailed information within a global context is certainly one of the fundamental problems in Information Visualization. We provide views of semi-external memory graphs at different levels of abstraction by using the graph's adjacency matrix as a unifying mechanism when visually navigating "large" hierarchical clusterings(see Figure 3).

4.1 Detail and Context

The interface (Figure 3) is designed around a canvas that displays a visual representation of an aggregate matrix view of the data. The labels, appearing on the bottom row and rightmost column, index the attributes being considered. The color of the rectangles corresponds to the proportion of data records that share the corresponding attributes. The color scale is tuned depending on data characteristics. For cancer data and phone traffic we found that a logarithmic color map performs better because of the wide range of the density parameter, ranging from 1 to 10^{-6}.

Using the terminology of the previous section, the canvas displays a color mapped representation of a $T_{(p,q)}$ view of a data sub-



Whole subgraph | Nodes with peel > 1 | Nodes with peel > 2 | Nodes with peel > 3

Figure 4: Node link view of part of the SEER data set. Blue nodes are properties, green nodes are patients. Consecutive peeling reveals a small clique of patients (green) that have at least four properties (blue) in common.

graph. The subtrees involved (i.e. T_p and T_q) are color marked in two special screen regions (next to the "axes") reserved for a sketch representation of the hierarchy tree. The cross product of $Leaves(T_p)$ and $Leaves(T_q)$ corresponds to the canvas region. The relative position of this region within the entire data set is indicated in the global overview which resides at the North-West corner of the interface. The explicit and fixed representation of the hierarchy tree next to the canvas region is a feature that to our knowledge had not been addressed properly in the past. This is a very effective way to keep a uniform and global view of the data without cluttering the display. It is flexible in the sense that its representation can be altered to suit specific data domains. For example, if a subtree of the hierarchy tree represents a collection of geographical attributes the representation can be changed to that of a physical map with the corresponding subdivisions. In case of communications traffic on a geographical space the canvas corresponds to the "cross product" of the two geographical maps. In the case of cancer incidence data, the data can be viewed as a subset of the cross product of two hierarchies. One being the geography where the patients population resides and the other being a cancer specific hierarchy that incorporates oncological information.

Another way to provide context when navigating the matrix is by rendering the parent matrix $T_{parent(p),parent(q)}$, along with an indication of the area the user is currently browsing in a fixed section of the interface (visible in the top left of figure 3). This helps the user to mentally maintain the navigation history, reducing the chance that he or she loses the global context when browsing the matrix.

4.2 Switching to node link diagrams

As discussed in the introduction node link diagrams are not well suited if one needs to navigate hierarchical graph views at different levels of granularity. However, if a sparse graph fits on the screen it makes sense to draw its corresponding node link diagram. This corresponds simply to switching the canvas display to a node link diagram if the corresponding subgraph is at the lowest desired level of granularity, if its number of edges is not more than $|S|$ and if its density is below certain threshold (i.e. it is sparse). Because of its flexibility we have chosen a spring embedder algorithm [9, 7] to generate a layout. According to our experience, it is also desirable to attach some ranking to the nodes depending on the application domain. For example, in the case of SEER cancer data, since the vertices are patients and attribute values, assigning some measure of relevance to the attributes is one important epidemiological question. We rank the graph vertices according to their "peeling numbers" [12].

Peeling is a time ordered process that permutes the vertices of a graph by visiting, in non decreasing order of degrees, the adjacency list of each vertex exactly once and reducing the degree of each of its highest degree neighbors by one. The order in which the vertices adjacency lists get visited is the peeling order. The current degree of a vertex when its adjacency list is visited is its peeling number. It is apparent that the peeling number of a vertex must be less than or equal to its degree. Vertices with the highest "peel" number are in some sense the most relevant in the data set, because they indicate a highly connected subset. If for example a large number of patients with a disease all share the same properties there might be a correlation between this disease and the properties.

The interface provides user interaction, with screen bounded node link layouts, by providing a color map slider that "peels" the layout according to the vertices peeling values. The edges are colored according to a refined version of the density color map used in the matrix view (see Figure 4). The node link layout and the corresponding matrix view are linked. This feature is helpful in assessing the type of information that is more easily discernible from the matrix view versus the corresponding node-link diagram.

4.3 Interaction

The user is allowed to zoom into any subcell currently visible on the screen. Mathematically this corresponds to selecting two nodes T_p and T_q in the current antichains and displaying $expansion(T_p, T_q)$ on the screen. Zooming out of $expansion(p,q)$ corresponds to displaying expansion(parent(p),parent(q)). To help the user maintain context we use smooth interpolation between the two views, similar to the one used in [14]. A major improvement is that we no longer require both nodes to be at the same level in the tree. Since we are retrieving edges on the fly, arbitrary sets of rows and columns can be collapsed or expanded by clicking on their respective labels. This corresponds to selecting one node p in an antichain and replacing it with $children(T_p)$. Instead of zooming manually, the user can also opt to jump directly to a farther out antichain computed by the visualization by shift clicking on a specific cell.

We adjust the aspect ratio of the newly displayed matrix to match the screen aspect ratio. This helps us to make more effective use of screen space and avoids displaying matrices with a very unfavorable aspect ratio. Furthermore, the user can hide any row or column or have the system hide automatically any rows and columns that do not contain any data. This allows us to display sparse matrices much more efficiently. To facilitate interaction with large matrices we also implemented a geometrical zoom which allows the user to select any area of the canvas and view it in closeup. Since many of the subcells in the matrix have widely differing sizes, using a linear size mapping would make some subcells inaccessible because their relative size is simply too small. We therefore applied a logarithmic screen mapping that preserves size order but still keeps the smallest

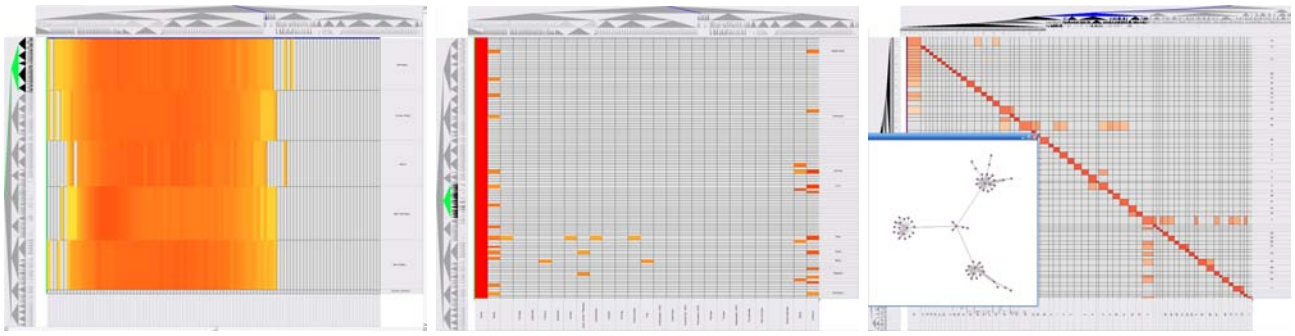


Figure 5: **(a)** Distribution of skin cancer versus age in the San Francisco - Oakland register, showing a peak at earlier ages in San Francisco (4th row), **(b)** Distribution of skin cancer versus race in Iowa's counties and **(c)** high level view of a spanning tree of a 260 million node graph, detailing phone traffic.

subcells visible at all times. Semantics feedback is provided by static and dynamic labels. Static column and row labels describe the nodes in the graph. Dynamic labels provide information on the matrix cells (i.e. multi edges) currently displayed.

5 Two sample data sets

We have navigated SEER cancer incidence and survival data for US patients from 1973 up to 2003. We have also used our interface to depict the type of hierarchy obtained by a recursive algorithm that produces connectivity based clusters.

5.1 Navigating SEER Cancer Data

SEER [8] stands for the Surveillance, Epidemiology and End Results program of the US National Cancer Institute. It is an authoritative source of data on cancer incidence and survival. Each record consists of 72 items that provide specific cancer diagnosis and treatment information, patient demographics and geographical and temporal data. SEER data is widely utilized to identify geographic and population differences in cancer patterns, to investigate environmental factors that influence cancer incidence and survival, and to study cancer treatment outcomes. Standard epidemiological techniques include regression methods and comparison of a variety of rates and ratios. A central question of interest is to identify "clusters" or other "useful" structure in the underlying data. The approach is to use this structural information as a trigger to stimulate further processing and analysis.

5.1.1 Data Model

We view the data as a bipartite graph from the set of patients to the set of attribute values. The set of patients is endowed with a geographical hierarchy with 20 subtrees corresponding to the 20 SEER registries (visible on the left hand side of the canvas, see Figure 3.5). Following epidemiologists advise, we extracted 17 attributes and build a hierarchy for them (visible at the top of Figure 3.5). This hierarchy consists of separate subtrees for cancer specific information such as primary cancer site and tumor size; treatment type (surgery, amount of radiation); temporal information (diagnosis date, age, etc) and patient demographics. This ad-hoc separation is based on the arguable assumption that geographical, temporal and demographic information may shed light on factors contributing to cancer detection and treatment.

From our point of view, this is the place where strong interaction with domain experts is necessary in order to obtain specific application semantics that can be incorporated in a "useful" manner into any interface. Our approach is then to provide the interface with mechanisms that allow the user to plug hierarchy subtrees that are in his/her view more appropriate for an specific task and application.

5.1.2 Colon Rectal, Prostate and Skin Cancer Data Sets

After the SEER data for Colon Rectal, Prostate and Skin cancer were transformed into graphs they had 437738, 358783 , 142406 vertices and 5182643, 5526803 , 1866531 edges respectively. As a byproduct of this transformation we automatically identified all those patients with missing data and all those attribute values that are complementary (i.e. Male, Female) or extremely rare (Data anomalies). So the computational infrastructure did clean the data in several ways not anticipated by us. Among the interesting typical findings we mention the following:

- The skin cancer age distribution in San Francisco county in the San Francisco - Oakland registry shows a peak (dark red area) at around 35-36 years of age. This peak is not present in other counties in the same registry. Note that the cause of this might be anything from more sun(bathing), higher skin cancer awareness in San Francisco county or simply a younger overall population, so it's very hard to draw immediate conclusions from this observation. Nevertheless, it is interesting enough to investigate further. (See figure 5a);

- Skin cancer race distribution in Iowa is severely tilted to whites. This might again be a consequence of the racial distribution in Iowa (likely) or it might be that whites are more susceptible to skin cancer (also likely). To answer these questions the data must be compared to the population distribution. (See figure 5b).

We want to point out that our knowledge of epidemiology is very cursory and our intention is to show that with some guidance from domain experts one can enhance the interface to find "automatically" patterns like the ones described above. The reason is that these findings did not require a priori information that we could use to direct our search. A pundit could say that all of this could have been done by just using a data base query system. The point is well taken if the user knows precisely what he is looking for. However, we work under the premise that very often this is not the case. What we offer is a tool that can help domain experts formulate quickly a set of hypothesis for further analysis. It is a visual and intuitive exploratory data analysis tool that may help the user get a quick glimpse over the entire data space in a hierarchical manner.

5.2 Navigating aggregate US phone traffic

The work described in [3] reports on the use of an external algorithm that produces a hierarchy for a phone traffic graph with over 260 million vertices (US telephone numbers) and over 4.3 billion edges (phone calls for a period of over 20 days). We did wonder if we could use our interface to navigate a restricted macro-view of the obtained hierarchy (since we have no access to the corresponding lower level data).

In this case, the tool was used to navigate a chopped hierarchy of maximum spanning forests. Contrary to the SEER data set both axes of the matrix display the same hierarchy. A subcell of the canvas depicts a maximum spanning subtree. It connects two other maximum spanning subtrees of previously connected subgraphs that were formed during the clustering algorithm recursion (See figure 5c). In other words, the depicted hierarchy trees together with the canvas show an anatomical description of a giant semi-external graph which the first author had not seen before even though he had previously worked with the corresponding data set.

Previous approaches [3] to produce a global image of such monster graphs were handicapped by the need to zoom into multi-edges which, as we alluded in the introduction, is the fundamental barrier to semantic zooming on node link diagrams. In summary, our interface has moved us closer to the challenging task of producing visual graph representations that can be navigated efficiently when just the vertex set of the graph fits in RAM.

6 Conclusions and Further Work

We have presented a framework for the interactive navigation of (semi-)external graphs. The method assumes a given hierarchy and uses it to construct a hierarchy of adjacency matrices. This collection of matrices can then be navigated interactively. More concretely our major contributions are:

- Parametrization and scalability: Our framework is parameterized in terms of memory and screen size and is able to deal with semi-external graphs. Most other visualizations don't even deal with this case in which the graph does not fit in RAM. Parametrization also allows us to decide on the fly when we can switch from matrix views to node link diagrams or other suitable representations.

- Efficient navigation: by using antichains for navigation the user does not have to click through multiple (possibly sparse) layers in the hierarchy to view a subgraph. Instead we compute generalized views that just fit the available screen space.

- Flexibility: The ability to plug arbitrary hierarchy trees into the visualization allows us to navigate any kind of relational data no matter its size. Subcell coloring can be done on arbitrary measures, depending on the task the user wants to perform. This means that Matrix Zoom is applicable to a wide range of data domains.

- Strong visual coupling between graph hierarchies and the display: this is in our view a major improvement over previous attempts to semi-external graph navigation.

We have applied our method to semi-external graphs (with more than 5 million edges) resulting in a number of interesting patterns. Because neither of the authors has an epidemiological background their real life value will be evaluated by domain experts, but they clearly show the serendipitous nature of the provided interface. Future work will focus on enhancements to the interface that facilitate user's queries. Concretely we will provide partially automated navigation to help the user search for dense subgraphs.

As an extension on the data side, we will add features to navigate fully external graphs. This can be achieved by moving the part of the graph below the antichain T_M to a server, which has its own gkd-index to answer user queries that fall below T_M.

An important improvement on the visualization side is to incorporate small glyphs (such as barcharts or starglyphs) on top of the matrix view, to display a set of density values instead of a single density value per cell. This will allow us to compare related or time dependent parameters visually. For example, how does colon cancer compare with prostate, and skin cancer in the US? Note that such queries are not easily answered using node link diagrams. The resulting differences in layouts make it hard to compare different visualizations. Visualization of time dependent relational data by using adjacency matrices will be an interesting research path to explore in the future.

Acknowledgements

The authors thank Fred Roberts for his encouragement and acknowledge the support provided by DIMACS and the US national Science Foundation under grants NSF CCR 00-87022 and NSF EIA 02-05116, as well as the Netherlands Organisation for Scientific Research (NWO) under grant 612.000.101. Many thanks to Professor Dona Schneider, and especially to Dave Millman for the many hours devoted to the preprocessing of SEER cancer data. Stephen Kobourov and Sandra Sudarsky entertained some discussions related to graph peeling.

References

[1] J. Abello, A. Buchsbaum, and J. Westbrook. A functional approach to external graph algorithms. *Algorithmica*, 32(3):437–458, 1998.

[2] J. Abello and J. Korn. Mgv: a system for visualizing massive multidigraphs. In *IEEE Transactions on Visualization and Computer Graphics*, volume 8, no. 1, pages 21–38, 2002.

[3] J. Abello, J. Korn, and M. Kreuseler. Navigating giga-graphs. *ACM Proceedings of Advanced Visualization Interfaces(AVI)*, pages 290–299, 2002.

[4] J. Abello and Y. Kotidis. Hierarchical graph indexing. *ACM Conference on Information and Knowledge Management(CIKM)*, pages 453–460, 2003.

[5] Charles J. Alpert and Andrew B. Kahng. Recent directions in netlist partitioning: A survey. Technical report, Computer Science Department, University of California at Los Angeles, 1995.

[6] Vladimir Batagelj and Anuska Ferligoj. Clustering relational data. In Gaul, Opitz, and Schader, editors, *Data Analysis*, pages 3–15. Springer, Berlin, 2000.

[7] Peter Eades. A heuristic for graph drawing. In *Congressus Numerantium*, volume 42, pages 149–160, 1984.

[8] L.A.G. Ries et al. (eds). Seer cancer statistics review, 1975-2000. 2003.

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

[10] Istvan Jonyer, Lawrence B. Holder, and Diane J. Cook. Graph-based hierarchical conceptual clustering. *International Journal on Artificial Intelligence Tools*, 10(1-2):107–135, 2001.

[11] Yehuda Koren. On spectral graph drawing. *Proceedings of The Ninth International Computing and Combinatorics Conference (COCOON'03)*, pages 496–508, 2002.

[12] S.B. Seidman. Network structure and minimum degree. *Social Networks 5*, pages 269–287, 1983.

[13] D. Spielman and S. Teng. Spectral partitioning works: Planar graphs and finite element meshes. *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, pages 96–105, 1996.

[14] Frank van Ham. Using multilevel call matrices in large software projects. *Proc. of IEEE Symp. on Information Visualization 2003*, pages 227–232, 2003.