Graph Sketches

Technical Note

James Abello* Irene Finocchi† Jeffrey Korn*

* Information Visualization Research
Shannon Laboratories, AT&T Labs-Research
{abello,jlk}@research.att.com

† Computer Science Department
University of Rome "La Sapienza"
finocchi@dsi.uniroma1.it

Abstract

We introduce the notion of *Graph Sketches*. They can be thought of as visual indices that guide the navigation of a multi-graph too large to fit on the available display. We adhere to the Visual Information-Seeking Mantra: Overview first, zoom and filter, then details on demand. Graph Sketches are incorporated into *MGV*, an integrated visualization and exploration system for massive multi-digraph navigation. We highlight the main algorithmic and visualization tasks behind the computation of Graph Sketches and illustrate several application scenarios. Graph Sketches will be used to guide the navigation of multi-digraphs defined on vertex sets with sizes ranging from 100 to 250 million vertices.

Keywords: visualization, massive data sets, graphs, hierarchies.

1 Introduction

One of the great visualization challenges today is the representation and fluid navigation of complex systems. We concentrate on very large multi-digraphs of sparse density and low diameter. Geographic information systems, telecommunications traffic [4], World-Wide Web [2] and Internet data [3] are prime examples of the type of graphs whose navigation can be guided by our Graph Sketches approach.

1.1 The bottlenecks

When visualizing massive data, two of the most fundamental issues are those associated with the I/O and screen bottlenecks. The I/O bottleneck is caused by the substantial difference between CPU speeds and external memories [7]. The screen bottleneck [4] is caused by the simple fact that the amount of information that can be displayed at once is ultimately limited by the number of available pixels and the speed at which the information is digested by a user. Even though a large number of pixels diminishes the screen bottleneck, it does not help the user's visual processing abstraction unless the display metaphor incorporates some global data set semantics. We propose mechanisms to alleviate the screen bottleneck.

1.2 Approach

Our approach is based on the hierarchical surfaces metaphor presented in [6]. Its effectiveness on very large graphs depends on a good recursive clustering that can be mapped to a partition of the screen space. Each such mapping is what we call a *Graph Sketch*. Graph Sketches should offer simple overviews of a very large graph macro-structure (Figure 1).

These views are zoom-able and are parameterized by user specified subgraph thresholds. When the obtained subgraph is small

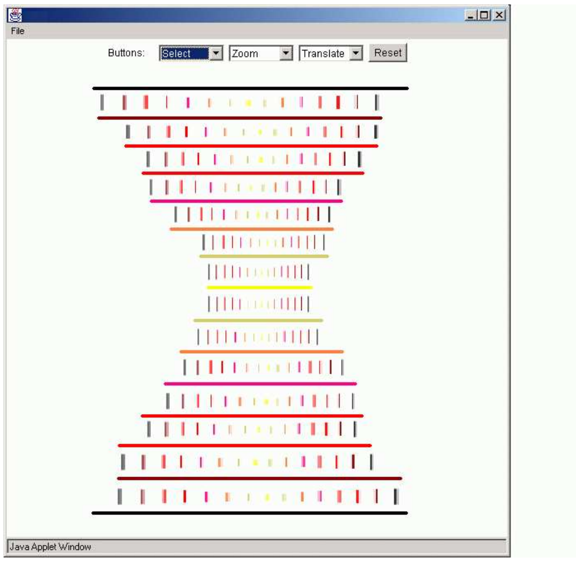

Figure 1: Orthogonal Bars Sketch

enough to fit on the available screen, the graph representation and its processing can be varied.

The concentration of this paper is on large graphs. With this in mind, we consider a multi-digraph *large* if its number of vertices is greater than $d * log(d)$ where d is the number of available display pixels.

Graph Sketches provide 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 to search for graph representations that encapsulate the essential features of either a clustering algorithm or a typical subspace that contains a feasible answer.

This paper present 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 a multi-digraph hierarchy with very little extra effort. *Graph Sketches* are amenable to distributed visual exploration.

1.3 Related Work

Multi-level graph views offer the possibility of drawing large graphs at different levels of abstraction. The higher the level of

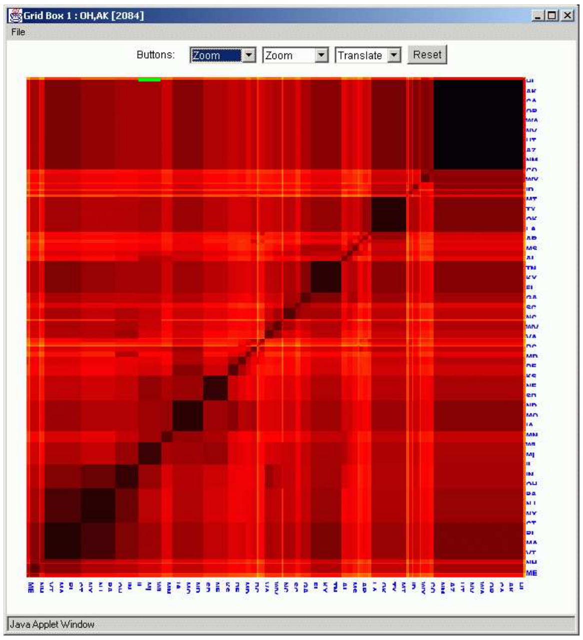

Figure 2: Location-based Graph Sketch of call detail traffic among US states.

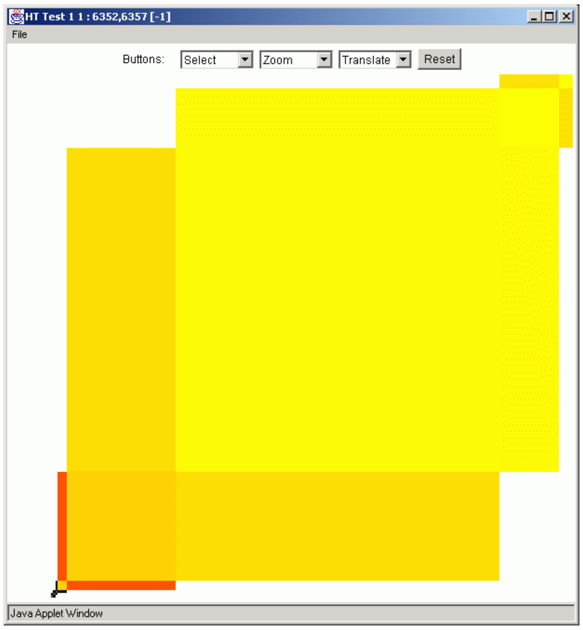

Figure 3: BFS Sketch. Each diagonal box represents a level of BFS. The remaining boxes represent the subgraphs induced by adjacent levels. Their edge density is color mapped.

abstraction, the coarser the provided graph view. Compound and clustered graphs have been considered in [8, 9]. In [10], some of the limitations of force-directed based methods for drawing large graphs are addressed. A central idea is to produce graph embeddings on Euclidean spaces of high dimensions and then projecting them into a two or three dimensional subspace. The method is based on a maximal independent set filtration of the vertexes of the graph and it is not apparent how to obtain such a filtration in the case of external memory graphs.

The primary difficulty with the surfaces approached described in [6] is that 2D surfaces are not easy to refine locally. We concentrate here on methods of computing, from the input graph, hierarchy trees that can be turned into efficient *Graph Sketches*. *Graph Sketches* can be viewed as a formulation that provides a uniform overall view of massive graph data together with scalable, efficient and flexible visual navigation tools.

The layout of the paper is as follows: In Section 2, we introduce graph sketches; the main elements of the computational engine, and its fundamental operations and I/O performance are covered in section 3. In the same section we briefly review the main components of the C-Java visualizer (MGV) that manipulates *Graph Sketches*. Section 4 points out some future research directions.

2 Graph Sketches

A *Graph Sketch* is a screen zoom-able macro-view of a very large graph. The goal is to use the *sketch* to guide the search for "interesting" subgraphs. The *sketch* should be tailored to the task at hand. For example, if the goal is to find dense subgraphs, the *sketch* needs to incorporate some notion of distance. This in turn, affects the type of recursive clustering that must be used to define the sketch. In general, a good deal of ingenuity will be necessary to design *sketches* that become effective visual navigation aids. With this framework in mind, designing a good navigation sketch for a particular problem becomes the central algorithmic question that needs to be resolved before a useful interactive visualization can be proposed. In this context, visualization is no longer just a presentation aid; it becomes part of the computational process.

A *sketch* for a graph G is a multi-digraph defined on a partition V_0,V_1,... V_k of $V(G)$ that is embbedable on the available pixel array. A multi-edge from V_i to V_j represents the set of edges in G that run from vertices in V_i to vertices in V_j. The multiplicity is just the number of such edges. We refer to this multi-graph as a $k - view$ of G. For a given graph problem P, if a solution on G can be obtained from solutions to P on the V_i's, then in principle one can use divide and conquer to search for a solution to P. This is the case for certain graph problems when the k-view is planar.

Given an algorithm that computes a *sketch* for a graph G, it can be used recursively to generate a tree T, such that $Leaves(T)$ represent a refinement of the original partition defining the *sketch*. This hierarchy tree T determines a hierarchical partition of $E(G)$. This means that a detailed view of an sketch multi-edge can be obtained by zooming into it. In other words,from an initial planar embedding of the sketch one can zoom in locally into any of the edges. This locality provided by the planar clustering allows the user to explore the multi-digraph edge hierarchy in a fluid manner. Of course, all of this is possible if the detailed view of a macro-edge can be computed efficiently.

2.1 Constructing a $Sketch(G, T)$

Given a procedure **Construct-View(G)** that produces a d-*view* for G with partition V_0,V_1,...,V_d, **Construct-View** is invoked for each i on the subgraph induced by V_i. It is important to notice that all these invocations are independent of each other and that by the end of the computation of the d-*sketch*, the only references that are kept are those from each obtained multi-edge to the actual input data that it represents. Only the subgraph to be expanded needs to reside in memory. Care needs to be taken to carry with each call a mapping from the current vertex names to the local ones. The depth of the recursion is controlled by the number of available pixels d, a time or space budget, and problem defined parameters. When the recursion is finished a data structure representing the obtained hierarchy tree and a mapping from the tree leaves to the partition of $V(G)$ that they represent is produced. This data structure may reside in memory or on disk depending on the amount of available RAM. Notice that the complexity of constructing a $Sketch(G,T)$ depends strictly on the complexity of the procedure **Construct-View** and on the quality of the obtained partition.

2.2 Sample Sketches

The main task to design a *good sketch* for a problem P is to devise a partitioning scheme for the input graph that guarantees that the space of solutions for P can be obtained by a suitable combination of the space of solutions of P restricted to the subgraphs induced by each set in the partition. Of course, this may not be the case for all problems and we know of no easily computable criteria to classify a problem as partitionable (in the sense described here). Nevertheless, we provide concrete examples of sketches for some NP-Hard problems.

- The most direct example of a *sketch* comes from graphs whose vertices have associated a geographic location. A classical example is the graph whose vertices are telephone numbers and the edges consists of phone calls among them. In this case, the hierarchy T on the vertex set is pre-established and consists of the subdivision of the earth in continents, countries, states, counties, towns, etc. An embedding of the hierarchy is provided by a cartographic map. A so called *star map* drawing was proposed in [4] to place the underlying graph on top of the map embedding. An alternative view can now be provided by using a matrix based sketch. The rows and columns of the matrix are ordered according to a Peano-Hilbert ordering determined by the geographic position of the vertices (see Figure 2). The edge density of the corresponding subgraphs is represented by a suitable color map.

- Consider the problem of finding a largest cardinality clique in an arbitrary connected graph G. A *B*readth *F*irst *S*earch tree of G determines a partition of $V(G)$ defined by distances from the BFS root. The corresponding multi-graph is planar (in fact, ignoring directions, it is simply a path) and the number of sets in the partition is just the depth of the BFS tree. So the only condition that could fail for this multi-graph to be considered a *sketch* is that the depth of the BFS tree is larger than $sqrt(d)$ where d is the number of available pixels. In this case, successive folding of the path can be done until it fits on the available screen. More generally, any planar k-view can be transformed into a related planar d-view where $1 < d < k$. The assertion that a BFS based partition of $V(G)$ is a "$good$" *d-sketch*, for the maximum clique problem, follows from the observation that cliques of G are by definition induced subgraphs where all the vertices are at distance exactly 1. Therefore, cliques can span at most two consecutive levels of any BFS tree. A screen embedding is obtained by mapping each vertex of the hierarchy, to a box placed diagonally inside its parent's box with the side lengths of the two boxes being in the same proportion as the ratio of the cardinalities of their corresponding sets of descendant leaves. Because, the sketch is based on a BFS view of G, the subgraph consisting of the edges between consecutive levels, gets naturally assigned to the only adjacent boxes that are determined by consecutive boxes on the diagonal. Each box is painted according to a density based color map. When zooming on a box, its interior is partitioned according to its children and the color map is recomputed according to its children densities. The diagonal boxes corresponding to the leaves of the hierarchy tree can be though of as a coordinatization of the visual space (see Figure 3). If more detailed connectivity is desired a conventional drawing representation can be invoked. An overview representation is always mantained on an auxiliary window with an indication of the hierarchy tree level at which the exploration is taking place. Experimentally we have been able to detect, in call detail graphs, that the largest clique also have logarithmic size.

- A more economical sketch can be obtained by mapping each node of the hierarchy tree to a colored bar where the length is proportional to the size of its set of descendants leaves and where the color again encodes a map density. The collection of bars representing the set of children of a pair of bars are placed parallel to each other and in the order of their BFS levels. In the case of zooming into the children of just one bar its children are placed inside a zoomed version of the bar in a direction orthogonal to that of the parent bar. Initially, the root bar gets assigned a fixed but arbitrary direction. We refer to this BFS sketch embedding as the *orthogonal bars* $sketch$, (see Figure 1).

- Consider now the problem of computing an edge minimum k-*view* of an arbitrary graph G with the added restriction that every set in the partition must be smaller than an input value s. This problem is also NP-complete. However a *D*epth *F*irst *S*earch spanning forest F of G that satisfies the property that if a vertex v belongs to a short cycle, then F contains a path that goes around some short cycle that contains v, can be used to obtain a sketch of G. In the case that several short cycles contain v, priority is given to the unique cycle that is determined by the most recently added edge to the spanning tree. This is a two pass algorithm. In the first one the spanning tree is constructed. In the second pass, a bottom up procedure that contracts pendant leaves and interior vertices with exactly two leaves as children produces a partition of the vertex set. It is not hard to show that this procedure computes a k-*view* of a graph G in space $O(|E(G)|)$ and time $O(|E(G)| * deg * log(k))$ where deg is the maximum degree of a vertex in G. It is not clear that this *sketch* is a good one for this constrained k-view problem.

- The Network Decomposition Problem presented in [12] consists of finding a coloring of $V(G)$ with a distance parameter l such that each color class is partitioned into an arbitrary number of disjoint clusters, the shortest path distance between any pair of nodes in a cluster is at most l and clusters of the same color are at least distance 2 apart. The goal is to find such a decomposition of a network where both the number of color classes and the distance parameter l are both $O(log(n))$ where n is the number of vertices in G.

  Despite the apparent similarity between this problem and the previous two such a decomposition can be found in optimal time $O(|E| + n)$ by a simple greedy construction. This decomposition can be used as a base for a *sketch* but it is not clear for what class of graph problems this is a *"good"* sketch.

3 Implementation

3.1 Sketch maintenance

In order to effectively use $sketches$, the following pre-processing steps are necessary.

- Compute an external memory BFS. This can be done in $O((|V| + |E|/B) * log(|V|/B) + sort(|E|))$ *I/O*'s by using a modification of a data structure originally proposed by [13]. B is the size of the disk block.

- Build an in-core index to a disk resident data structure that contains for each level of the BFS its induced subgraph and for each pair of adjacent levels the subgraph consisting of all the edges going from one level to the other in both directions. The in-core index will only keep a reference to the disk location, the associated density function value and a few book keeping items. With this information, the corresponding screen embedding is computed as depicted in Figure 3 or Figure 1. The corresponding portion of the current hierarchy tree T is also stored in memory. Now, for those vertices of the hierarchy tree whose associated induced subgraph fits in main memory the corresponding full hierarchy subtree is computed, using an internal memory implementation. Notice that all these computations can be made independently. For those vertices of the hierarchy tree whose vertex set fits in main memory but not its edge set, a semi-external version of BFS is invoked [5]. Those vertices of T whose associated vertex set does not fit in memory are processed again by a fully external BFS algorithm. Notice that all these computations are amenable to parallelization since they are independent. At the end of these steps, we have a disk resident representation of the hierarchy tree T and a mapping from its leaves to the actual vertices that they represent in the input graph.

3.2 Navigation

From any given layer the user can move to any of the adjacent layers by partial aggregation or by refinement of some sets in the corresponding partition. Namely, from any given multi-edge e in a current sketch the user can zoom into e's corresponding detailed view or he/she can also zoom out into the subgraph that generated e.

The main navigational operation used by the computational engine is:

- **Edge zoom:** Given an edge (u, v), *zoom*$((u, v))$ is defined as follows: {delete the edge (u, v); $replace(u)$; $replace(v)$; add all the edges in the next Sketch layer that run from the children of u to the children of v}.

The user can customize provided color maps to refine the search at different levels of granularity. An assorted set of multilinked views and labels are at his/her disposition to identify the actual values being represented by the color maps.

Graph Sketches are incorporated into MGV (a *Massive Graph Visualizer*) whose more salient features are:

- It handles hierarchical views of massive multi-digraphs.
- It consists of a C-computational engine (server) and a Java-3D visualizer (client), which may reside on separate machines. In fact, the visualizer can run on multiple desktops allowing different users to navigate a massive data set independently.
- It provides a drill-down zoom-able interface together with a collection of multi-linked views.
- Context is maintained by using multiple cameras. One provides an overview and the others trail each other depending of a user specified zooming interval. A persistent history of previous navigations of the hierarchy is maintained.
- Users can plug-in alternative *Sketch* visualizations, and can apply their own filters to the subgraphs accessible from the sketch.

Adding a new sketch to MGV requires implementing a small set of functions that handle rendering, zooming, labeling and linking. The sample sketches described in this paper were each on the order of between 500 and 1000 lines of Java.

4 Conclusions

Graph Sketches offer a unified view of computation and visualization on very large graphs. Very large graph visualizations need to be aware of the intrinsic algorithmic question that needs to be solved in order to provide interactive navigation that can guide a user towards the discovery of interesting graph sub-structures. Tailoring a graph decomposition to an exploration task appears to be an interesting angle that deserves further study. Devising useful 3D sketches is a tantalizing area of research. A question that comes to mind is: Are there any other interesting graph problems for which the BFS based sketches introduced here are beneficial?

References

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

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

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

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

[5] 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.

[6] 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.

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

Lecture Notes in Computer Science, 1547:111-124, 1998.

[8] 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.

[9] K. Sugiyama, K. Misue. Visualization of structural information: Automatic drawing of compound digraphs. In *IEEE TRansactions on Systems, Man and Cybernetics*, Vol. 21, N o 4, pages 876-892, 1991.

1999.

[10] 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.

[11] D. Karabeg. Parallel Algorithm Graph Reduction. *TR No. CS88-120*, University of California, San Diego, March 1988.

[12] L. Cowen. A linear time algorithm for network decomposition. *Dimacs TR series*, No 94-56, December 1994.

[13] V. Kumar, E. J. Schwabe. Improved algorithms and data structures for solving graph problems in external memory. *Proc. 8th IEEE SPDP*, pages 169-176, 1996.