Automatic Graph Clustering (System Demonstration)

Reinhard Sablowski[1], Arne Frick[2] *

[1] Universität Karlsruhe, Institut für Programmstrukturen und Datenorganisation, EMail: RSablowski@aol.com

[2] Tom Sawyer Software, 804 Hearst Avenue, Berkeley CA 94710, EMail: africk@tomsawyer.com

Abstract. We present a new, easy to understand algorithm and programming environment allowing for the interactive or automatic clustering of graphs according to several heuristics.
Our approach is based on graph structure only and can be implemented to run efficiently with a personal computer. It is capable of efficiently clustering graphs with > 3000 vertices. We shall demonstrate the interactive user environment for automatic clustering. As an application, we consider the clustering of large WWW connectivity graphs.

1 Introduction

Clustering is the process of grouping information to achieve a more recognizable presentation of source data. The computation of a clustering generally requires a metric on the data to determine the closeness of data points. The clustering of graphs can be based on either graph structure, or on some semantic properties of the application domain. In this paper, we do not make any assumptions about the "meaning" of vertices and edges, but focus solely on the structure. In this context, the only available metric is the *graph-theoretic distance* of vertices, which is defined as the length of the shortest path joining them, if one exists. If a drawing of a graph is known, the Euclidean distance between the vertex positions can be used instead of their graph-theoretic distance.

1.1 Criteria for a good Clustering

Given these assumptions, what are criteria for good clustering? We argue that the importance of a *cluster vertex*, i.e. a vertex in the resulting clustering that represents a group of vertices in the original graph, should be a function of its *weight*, i.e. the number of vertices and edges represented by it. The cluster weights should come close to a user-defined value to achieve a clustering of desired granularity.

$$GA = \sum_{i=1}^{\|v\|} (g_i - G_{\text{predef}})^2 \tag{1}$$

* This research was performed while the author was working at Universität Karlsruhe, Institut für Programmstrukturen und Datenorganisation.

Also, the cluster sizes should be approximately equal. In other terms, the standard deviation of cluster sizes should be minimal.

$$\sigma = \sqrt{\frac{1}{\|v\|}\sum_{i=1}^{\|v\|} g_i - \overline{g}} \tag{2}$$

Connectivity within a cluster (*intra-connectivity*) should generally be stronger than the connectivity between clusters (*inter-connectivity*). Therefore, the total number of inter-cluster edges should be minimized:

$$EH = \|\{e \in E | \text{e is inter-cluster edge}\}\| \tag{3}$$

Realizing that these criteria may conflict each other, a global optimization strategy should be used, combining these criteria by assigning weights to each of them. This results in a global energy function

$$E = w_1 GA + w_2 \sigma + w_3 EH \tag{4}$$

that is supposed to represent the overall quality of a clustering.

2 Related work

The problem of clustering graphs is closely related to that of partitioning graphs, for which there is a variety of literature [2]. Little appears to be known about the problem of how to efficiently cluster large graphs in an interactive environment, although the problem is closely related to the task-scheduling problem for parallel processors. The requirement there is to have partitions of approximately equal size and sparse interconnectivity, which is similar to our notion of a well-clustered graph.

The well-known Basic-ISODATA algorithm for arbitrary clustering problems in Euclidean spaces is quite fast in practice, but it requires *a priori* knowledge of the number of clusters to partition into. Other general strategies for cluster analysis in Euclidean spaces are discussed in [1].

The idea of visualizing WWW graphs seems to have appeared first in [6]. However, this approach is based on semantic information.

3 Cluster Drawings

In order to reveal information about the graph using clusterings, we need to find ways to communicate the clustering. An intuitive technique is to visualize these clusters using graph drawing techniques.

In addition to the well-known global aesthetics criteria employed by spring-embedder algorithms, drawings of clusterings should display vertex and edge weights, and hide intra-cluster edges.

4 Idea

4.1 Basics

Our approach for clustering large graphs is based on the successive identification of structural elements (*patterns*) in the graph. About the simplest pattern occurring in a graph are *leaves*, i.e. nodes with in-degree 1 and out-degree 0. Other suitable patterns include *paths* (nodes with in-degree=out-degree=1), triangles and similar simple structures.

The existence of such groups of vertices indicates a close relationship between their constituent nodes and should therefore be clustered. It is important that patterns can be identified efficiently.

A clustering step consists of clustering of all vertices satisfying one or more patterns. This process can be iterated under the user's control to yield a hierarchical clustering of desired weight and density.

This approach is similar to a stepwise parsing of the graph according to a graph grammar [3, 7], where patterns are *non-terminals* and vertices are *terminals*.

4.2 Data Structures

Although the basic idea is straightforward, it remains a challenge to implement efficiently. A data structure suited for the hierarchical clustering of graphs should support the following basic operations efficiently: Insertion and deletion of nodes and edges as well as recursive clustering and unclustering nodes.

The need for efficient data structures is motivated by the following scenario. Assume first that a dense graph has to be grouped into two clusters. After forming the first cluster, all edges from a node outside the cluster that are connected to two or more nodes within the cluster need to be drawn as a single new, heavier edge. After the clustering step is completed, there will be only a single, very heavy edge connecting both clusters. In this situation, unclustering the first group has to reveal all edges to vertices in the second cluster, and this should be efficient.

Our implementation of a suitable data structure is based on an object-oriented design that defines ClusterVertex as a subtype of Vertex and therefore allows for the polymorphic use of ClusterVertex, wherever the abstract data type Graph requires a Vertex. Similarly, ClusterEdge is a subtype of ClusterEdge. The type ClusterEdge is based on two ClusterVertex stacks. Whenever a vertex v is inserted into a cluster c, c is pushed upon the corresponding edge stacks of all incident edges, thus providing a fast and information-preserving method for retrieval of cluster information.

5 Result characteristics

Our pattern-based approach does not create a predefined number of clusters. Instead, the process is under the user's control, who chooses graph patterns from an extensible set of standard patterns such as leaves, paths, triangles etc.

By iteratively reapplying this pattern-based compression, the resulting graphs may contain original nodes as well as small and large clusters. The number of resulting nodes

depends heavily on the input graph. We have found that in general, the final node count represents a compression by a factor of 20–80 for *WWW-Graphs* (see Sect. 6).

One possible technique to visualize clusterings is to apply a spring-embedder algorithm [4]. This is actually a quite natural way of drawing clusterings, as one of the main aesthetics criteria employed by spring-embedder algorithms is that vertices connected by an edge should be close together. This paradigm coincides with our criteria for good clusterings, that would require the nodes within a cluster to be drawn close together, while clusters themselves should be separated in a drawing. The drawings resulting from applying spring-embedder a algorithm generally reveal a lot of the original graph's structure (cf. Fig. 3).

Conversely, spring-embedder algorithms may be helpful to *compute* clusterings in the first place, too. As spring-embedder drawings tend to have related nodes close to each other, geometric proximity and statistical techniques [5] may be applied to compute clusterings from spring-embedder drawings as well.

6 An Application: WWW graphs

As an application domain in which very large real-life graphs occur naturally, we have chosen to consider *WWW graphs*, whose vertices are defined by the URL's of WWW documents, and whose edges are defined by the hyper-links between them. A so-called web robot was used to automatically extract graphs of this nature, starting from a single URL.

One particular graph gathered this way has about 3000 nodes and 3500 edges. Fig. 1 shows this graph in its raw, unclustered state. This example can be clustered well, revealing interesting structural properties (cf. Fig. 2), with little interaction in under two minutes of runtime on a regular IBM-compatible personal computer (based on Intel's Pentium P60 processor). The final result is shown in Fig. 3.

References

1. M. R. Anderberg. *Cluster Analysis for applications*. Academic Press, 1973.
2. Bosak. *Graph Partitioning*. Kluwer, 1990.
3. F. J. Brandenburg. Designing graph drawings by layout graph grammars. In Roberto Tamassia and Ioannis Tollis, editors, *Proceedings of Graph Drawing'94*, volume 894 of *Lecture Notes in Computer Science*, pages 416–427. DIMACS Workshop on Graph Drawing, Springer Verlag, 1995.
4. P. Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.
5. J. Hartigan. *Clustering Algorithms*. J. Wiley and Sons, 1975.
6. S. Mukherjea, J. D. Foley, and S. Hudson. Visualizing complex hypermedia networks through multiple hierarchical views. Technical Report 95-08, Georgia Institute of Technology, Graphics, Visualization and Usability Center, College of Computing, Atlanta, GA 30332-0280, 1995. Also appeared in the Proceedings of the ACM SIGCHI CHI'95, May 1995, Denver, Colorado.
7. G. Zinssmeister and C. McCreary. Drawing graphs with attribute graph grammars. In *Graph Grammar Workshop*, pages 355–360, Williamsburg, 1994.

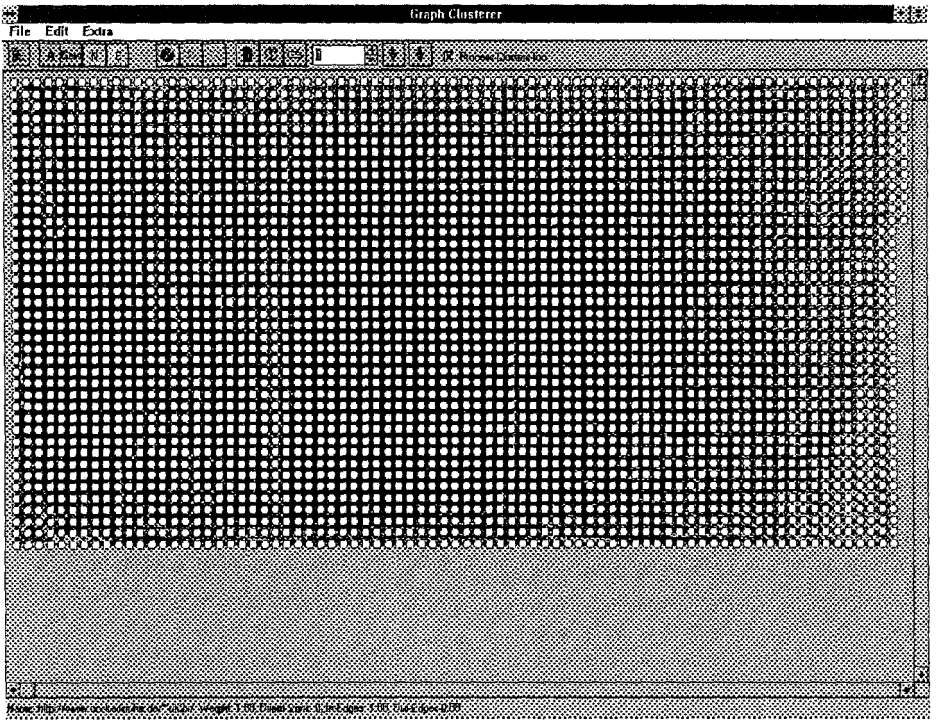

Fig. 1. Original, unclustered graph. No structure is recognizable at all.

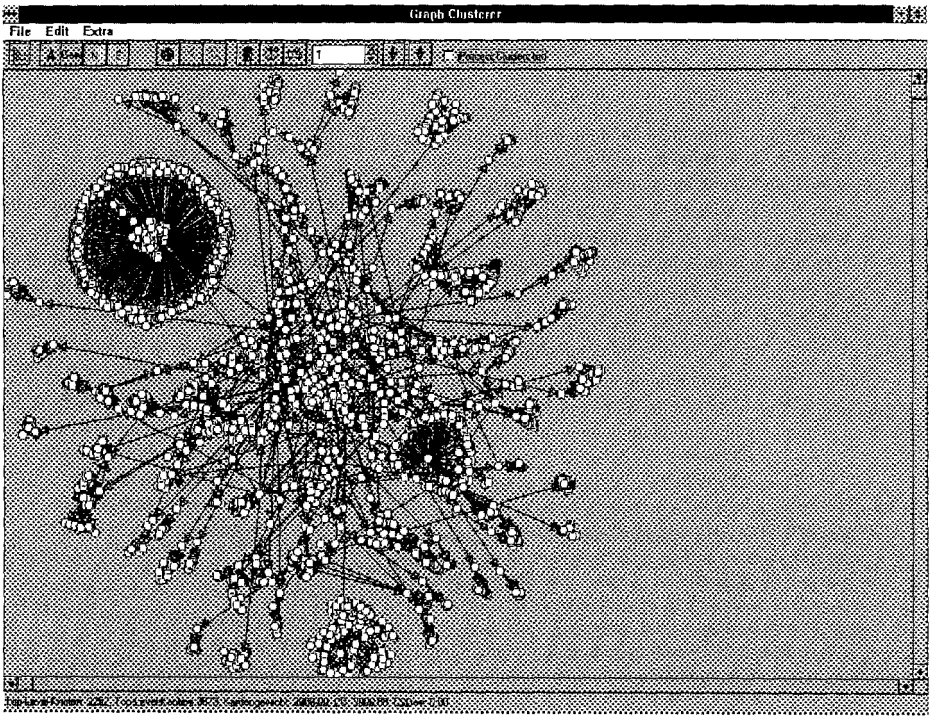

Fig. 2. The process of clustering may reveal rich internal structure.

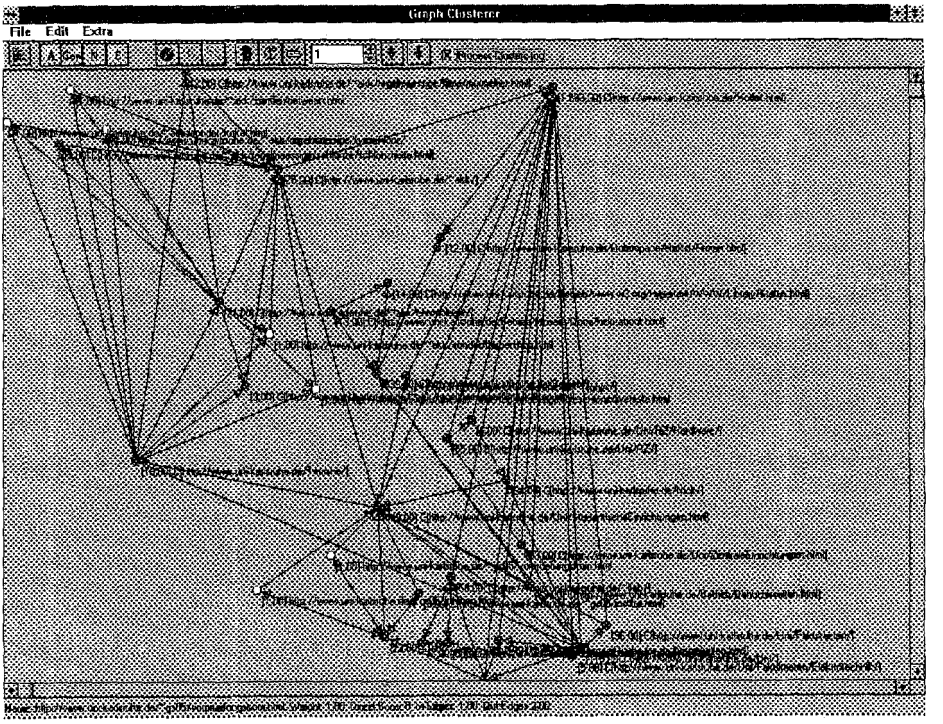

Fig. 3. Final clustering.