GraphClust: a Method for Clustering Database of Graphs

D. Reforgiato Recupero[1], D. Shasha[2]

[1]Dipartimento di Matematica e Informatica
Università degli Studi di Catania
e-mail: diegoref@dmi.unict.it
[2]Computer Science Department
New York University
e-mail: shasha@cs.nyu.edu

Abstract

Any application that represents data as graphs may be interested in finding patterns in those graphs. To do this in an unsupervised fashion requires the ability to find subgraphs that are similar to one another. That is the purpose of GraphClust. GraphClust is an algorithm and software that clusters directed and undirected labelled graphs. The algorithm proceeds in two phases: it first finds highly connected substructures in each graph and then it uses those substructures to represent each graph as a feature vector. Clustering itself can be done using the *k*-means or the Antipole method [7], though other methods are of course possible. We validate the cluster quality by using the silhouette method [5]. Moreover, SVD decomposition leads to the computation of highly co-occurring substructures.

Index terms - Text clustering, Document vectors, Graphs clustering, Graphs substructure.

1 Related Work

In the last few years, developing algorithms for clustering data represented by graphs has been recognized as a problem in the pattern recognition community [6]. Nevertheless, graph clustering is still an open problem for two reasons. First, many interesting exact graph matching problems, i.e. subgraph isomorphism, maximum common subgraph, etc., are NP-complete. So exact graph clustering algorithms using graph matching are extremely time consuming. Second, the proper distance metric between graphs is a matter of debate.

Spectral methods try to represent the most interesting properties of the input graphs using vectors, thus reducing the graph clustering problem to a problem in a vector space [3, 4, 14]. This allows a new *spectral* method, closely related to latent semantic indexing, to be used.

Text retrieval (see [16, 8, 2, 15]) has focussed on the need to locate textual information efficiently. A classical text searching method (see [8]) involves modelling a text collection in document-term matrix, and evaluating a document's relevance to a query using a linear algebraic dot product. In a term-document matrix A, $A[i, j]$ gives the number of occurrences of term j in document i. Queries are normally represented as a bit vector over the same set of terms. The similarity between document vectors (the rows of document-term matrices) can be found by their inner product. This corresponds to determining the number of term matches (weighted by frequency) in the respective documents. Another commonly used similarity measure is the cosine of the angle between the document vectors. This can be achieved computationally by first normalizing (to 1) the rows of the document-term matrices before computing inner products. Singular Value Decomposition (SVD) has been shown to work well for text retrieval over the last fifteen years [9, 12]. The motivation is simple: large document-by-term matrices have a significant amount of redundant data. Removing this information allows a more precise and efficient search. Singular Value Decomposition achieves rank reduction by breaking the matrix A in the product of 3 matrices T, S, D^T which are truncated to r dimensions.

Latent Semantic Indexing (LSI, [13]) attempts to project term and document vectors into a lower dimensional space spanned by the true "factors" of the collection. This uses a truncated Singular Value Decomposition (SVD) of the term-document matrix.

Subdue is another method to capture essential structure information from graphs. The Subdue substructure discovery system ([1]) discovers repetitive subgraphs in a labelled graph representation by using the minimum description length principle. Experiments show Subdue's applicability to several domains, such as molecular biology, image analysis and computer-aided design.

In this paper, GraphClust, a new algorithm for clustering labelled graphs, will be presented. The problem of mapping the graphs as feature vectors is solved by creating some substructures such that the frequency of substructure j in the graph i is stored at $A[i, j]$. After this, the rows of the matrix A are finally clustered. A list of highly correlated substructures is also created by using the SVD reduction.

2 Design

GraphClust assumes that each node of the database graphs has a unique identification number and a label. Edges are unlabeled (for purpose of this paper).

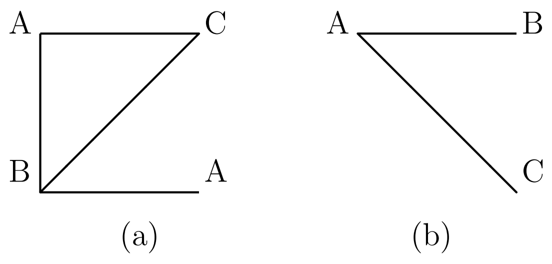


Figure 1: Dataset of two graphs.

Graph (a)		Graph (b)	
Initial Node	Substructures generated	Initial Node	Substructures generated
top-left A	{A,AC,AB,ABA}	A	{A,AB,AC}
B	{B,BC,BA,BA}	B	{B,BA,BAC}
bottom-right A	{A,AB,ABC,ABA}	C	{C,CA,CAB}
C	{C,CA,CB,CBA}		

Table 1: Patterns generated from the dataset of Fig. 1 using AllPairShortestPath with $l_p = 3$.

	C	CA	CB	CBA	A	AB	ABA	B	BAC
graph (a)	1	2	2	2	2	4	2	1	0
graph (b)	1	2	0	0	1	2	0	1	2

Table 2: Matrix A generated from the patterns of Table 1.

GraphClust deals with either directed or undirected graphs. The substructures can be discovered in two ways:

- by using the AllPairShortestPath algorithm; in this case, for each graph of the dataset and for each vertex v, all the shortest paths of length 1 up to a small constant l_p are generated from v. Each path is represented by the sequence of node labels in that path.
- by using the Subdue substructure discovery system ([1]); in this case, for each graph g of the dataset, Subdue finds common or approximately common substructures of g.

A matrix having a number of columns equal to the number of found substructures and a number of rows equal to the number of the graphs in the dataset is created. Each entry $A[i, j]$ represents the number of times in which the substructure j is contained in the graph i.

Subdue is more suitable when the graphs in the database have few labels compared to the number of nodes. In that case, there is a high likelihood of finding common substructures. If AllPairShortestPath is run, it finds for each graph all the label sequences corresponding to paths of length 1 up to a small constant l_p and therefore it creates more columns in the matrix A than Subdue. However, if too many substructures are found with either AllPairShortestPath or Subdue, GraphClust considers only the *max_sub* most frequent, where *max_sub* is a constant of the system. For example, on chemical compounds, where usually there are not so many nodes and edges, Subdue provides a better solution. For graphs with many edges, AllPairShortestPath should be used because it takes less time.

Once that the matrix A is completed, we cluster its rows. There are two possible clustering algorithms to use: one is the k-means algorithm in which the user chooses the number of clusters k to create; the other is the Antipole Clustering [7] in which the user chooses a “tightness” measure (an integer value in the range 1 to 4) where the higher the measure the smaller the cluster radius and hence the larger the number of generated clusters. Antipole Clustering [7] is much faster than k-means even if it is not possible to know a-priori the number of clusters that will be created. The metric distance used in both clustering algorithms just described can be either Euclidean distance or inner product distance. Euclidean distance is appealing for applications having a natural geometry. Inner product is better for non-spatial applications such as text-similarity.

In table 2 a matrix obtained from the patterns of table 1 generated by applying the AllPairShortestPath algorithm with $l_p = 3$ to the dataset in Fig. 1 is shown.

Another operation we perform when the matrix A is complete, is the creation of correlated substructures. By using the SVD method, the substructure-graph matrix A^T is broken apart into the product of 3 matrices T, S and D^T. Table 3 shows a SVD of the matrix A in table 2. These matrices are truncated to r dimensions with r chosen by the user. Dimensionality reduction reduces the noise present in the substructure-substructure matrix revealing a more robust relationship between the substructures. The substructure-substructure correlation matrix X_r is then computed by multiplying $T_r \times S_r \times (T_r \times S_r)^T$. Table 4 shows the substructure-substructure correlation matrix X_r computed with the matrices T_r, S_r of table 3 reduced for $r = 1$. When the number of substructures is too large and then it would be too expensive to compute the singular value decomposition, GraphClust considers only the substructures more interesting (with larger support in the graphs data).

The three steps of the basic GraphClust algorithm are shown in Fig. 2.

$$\begin{pmatrix} -0.20 & -0.17 \\ -0.40 & -0.33 \\ -0.26 & 0.35 \\ -0.26 & 0.35 \\ -0.33 & 0.01 \\ -0.66 & 0.02 \\ -0.26 & 0.35 \\ -0.20 & -0.17 \\ -0.13 & -0.68 \end{pmatrix} \times \begin{pmatrix} 6.8 & 0 \\ 0 & 2.61 \end{pmatrix} \times \begin{pmatrix} -0.89 & 0.46 \\ -0.46 & -0.89 \end{pmatrix}$$

$$T \qquad \times \qquad S \qquad \times \qquad D^T$$

Table 3: Matrices T, S, D^T generated by the Singular Value Decomposition of A^T of Table 2.

3 Algorithms

It turns out that GraphClust consists of 16 different algorithms broken down along the four binary dimensions described in the section 2. The main concept of GraphClust is the mapping of the data graphs into k-dimensional vectors. To perform this step we have introduced the concept of substructures and the methods used to find these substructures.

In this section, the algorithms used by GraphClust in the three steps of its main procedure will be discussed.

Subdue discovers interesting and repetitive subgraphs in a labelled graph representation using the minimum description length principle; Subdue discovers substructures that compress the original data and represent structural concepts in the data. By replacing previously-discovered substructures in the data, multiple passes of Subdue produce a hierarchical description of the structural regularities in the data. Subdue uses a computationally-bounded inexact graph match that identifies similar, but not identical, instances of a substructure and finds an approximate measure of closeness of two substructures when under computational constraints. In addition to the minimum description length principle, other background knowledge can be used by Sub-

$$\begin{pmatrix} -0.20 \\ -0.40 \\ -0.26 \\ -0.26 \\ -0.33 \\ -0.66 \\ -0.26 \\ -0.20 \\ -0.13 \end{pmatrix} \times \begin{pmatrix} 6.8 & 0 \end{pmatrix} \times \left(\begin{pmatrix} -0.20 \\ -0.40 \\ -0.26 \\ -0.26 \\ -0.33 \\ -0.66 \\ -0.26 \\ -0.20 \\ -0.13 \end{pmatrix} \times \begin{pmatrix} 6.8 & 0 \end{pmatrix} \right)^T =$$

	C	CA	CB	CBA	A	AB	ABA	B	BAC
C	2	4	2	2	3	6	2	2	2
CA	4	8	4	4	6	12	4	4	4
CB	2	4	4	4	4	8	4	2	0
CBA	2	4	4	4	4	8	4	2	0
A	3	6	4	4	5	10	4	3	2
AB	6	12	8	8	10	20	8	6	4
ABA	2	4	4	4	4	8	4	2	0
B	2	4	2	2	3	6	2	2	2
BAC	2	4	0	0	2	4	0	2	4

Table 4: Reduced correlation substructure-substructure matrix $X_r = T_r \times S_r \times (T_r \times S_r)^T$ for $r = 1$.

```
Basic Algorithm

GraphClust_Basic($DataBase, r$)
    - - - Start step 1 - - -
1   Creates substructures of the data graphs;
    - - - End - - -

    - - - Start step 2 - - -
2   Creates a matrix $A$ having as number of rows, the number
    of data graphs, and as number of columns, the number of
    substructures;
3   for each graph $i$
4       Fills the entry $A[i, j]$ with the number of occurrences
        of substructure $j$ in the graph $i$;
5   end for each;
    - - - End - - -

    - - - Start step 3 - - -
6   Clusters the rows of $A$ and
    Create the highly co-occurring substructures
      pairs by using $r$ in the SVD process;
    - - - End - - -

7   end GraphClust_Basic.
```

Figure 2: GraphClust: the three steps of the basic algorithm.

due to guide the search towards more appropriate substructures. Once the substructures and the matrix A have been created, the clustering is performed by the k-means or Antipole [7] clustering method. In our implementation of k-means, initial k centroids $q_1^1, q_2^1, \ldots, q_k^1$ are computed by using the Gonzalez (see [11]) algorithm; then, the rest of the objects are assigned to a class according to the relation $x_l \in C_j^t$ iff $d(x_l, q_j^t) \leq d(x_l, q_i^t),\ 1 \leq j,\ i \leq k,\ i \neq j$. Next, new centroids are computed in such a way that the performance index, $\gamma_i = \sum_{x \in C_i^t} |x - q_i^t|^2, i = 1, 2, 3, \ldots, k$, is minimized. This is achieved making $q_i^{t+1} = \frac{1}{n_i^t} \sum_{\forall x \in C_i^t} x$. If $q_i^{t+1} = q_i^t$, the process finishes, otherwise, the objects are grouped again.

The Antipole Clustering (see [7]) algorithm of bounded radius is performed by a top-down procedure starting from a given finite set of points S which checks if a given splitting condition is satisfied. This condition asks for two points whose distance is greater than the radius. If there are no two such points, then splitting is not performed and the given subset is a cluster on which an approximate centroid is then found. Otherwise, a suitable pair of points (A, B) of S called Antipoles is generated and the set is partitioned by assigning each point of the splitting subset to the closest endpoint of the Antipole (A, B). As seen in [7] the randomized algorithms used by Antipole clustering makes its construction much faster than k-means's.

4 Complexity

Here is a description of the worst case complexity for the three steps of GraphClust. Let $|D|$ be the number of graphs in a database D. The first and second steps of the algorithm depend on which algorithm is used to create the patterns. If AllPairShortestPath is used, then the complexity of the first step is $\mathcal{O}(\sum_i^{|D|}(n_i{}^3))$, where n_i are the number of nodes of the graph i; in this case the complexity of the second step is $\mathcal{O}(|D||pat|\sum_i^{|D|}(n_i m_i^{l_p}))$, where $|pat|$ is the total number of patterns generated and m_i is the number of patterns starting from n_i. If the Subdue algorithm is used, then the complexity of the first step becomes $\mathcal{O}(\sum_i^{|D|}(\sum_{j=1}^{nsubs}(ninst_j \times gm_j)))$, where $ninst_i$ is the maximum possible number of non-overlapping instances for substructure j and gm_j is the user-defined maximum number of partial mappings that are considered during a graph match between substructure definition j and a potential instance of the substructure. In this case the complexity of the second step is $\mathcal{O}\ (|D||pat|\sum_i^{|D|}(\sum_{j=1}^{nsubs}(ninst_j \times gm_j)))$, where $ninst_i$ and gm_j have already been described above. Details of the Subdue complexity analysis can be found in [17].

For the third step we will consider for first the clustering process and then the SVD computation. The clustering process complexity depends on which algorithm is used. K-means takes time $\mathcal{O}(tk|D|)$, where $|D|$ is the number of objects (graphs), k is the number of clusters, and t is the number of iterations. Normally, $k, t << |D|$. The Antipole algorithm [7] has a worst-case complexity of $\frac{\tau(\tau-1)}{2}|D| + o(|D|)$ in the input size $|D|$, where τ is the bounded radius (see [7] for further details).

Hence, given the higher complexity of Subdue, GraphClust by using Subdue for finding the substructures should be used with small datasets having sparse graphs with no more than 100 edges.

As mentioned above, for example, Subdue is more suitable than AllPairShortestPath when GraphClust is used to cluster a dataset of chemical compounds.

Conversely, when GraphClust is used to cluster big datasets having large numbers of edges, then the speed of AllPairShortestPath makes it preferable to Subdue.

Finally, the SVD process complexity takes $\mathcal{O}\ (|pat|^2 \cdot |D| + |pat| \cdot |D|^2)$ so it would not be pratical for huge datasets. To make the process effective also with large datasets, the SVD considers only the *max_sub* substructures more interesting (more present in the database) where *max_sub* is a constant of the algorithm and it is fixed at 30.

5 Performance Studies

We start with a systematic evaluation of the quality of the clusters. The Silhouette method [5] is used to show how good the clustering obtained is. Moreover, we want to show how well the final clustering performed by GraphClust captures the graphs present in different categories known *a priori*. For this, we have used an artificial graph generator [10] to create a database containing five different categories of undirected graphs. The five categories includes randomly graphs, regular 2D-meshes, regular 3D-meshes, irregular 2D-meshes, irregular 3D-meshes. For each category we have generated 1000 graphs with 30 nodes and 1000 graphs with 80 nodes. The number of edges vary from 50 to 200. Each group of 1000 graphs differs for the 20% of the edges. Thus, our artificial dataset contains 10000 graphs. An optimal clustering creates 10 clusters, each one containing one structural group of graphs. Tables 5 and 6 depict the global silhouette values, GSu, for each partition, and the silhouette values, S_i, for each number of clusters c, for $c = 10$ to 15 obtained by GraphClust. Clustering in table 5 is based on GraphClust with the Antipole Tree data structure [7] whereas clustering in table 6 uses GraphClust with the k-means algorithm. For both clustering algorithms, the substructures have been discovered by using the AllPairShortestPath fixing the constant $l_p = 4$. In both tables $c = 10$ is suggested as the best clustering configuration for the examined data set and this is also the optimal number of clusters known for when the data set has been created.

For the experiments we used a Mobile Intel Pentium Processor 2.30GHz and 512MB of RAM with Linux operating system. The algorithm performs well and gives high quality for both artificial and real data sets. For the syntethic experiments it run in 10 minutes while for the real dataset it run in 30 minutes.

c	GSu	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	S_{11}	S_{12}	S_{13}	S_{14}	S_{15}
10	0.712	1	0.86	1	0.82	0.05	0.71	1	0.88	0.12	0.66					
11	0.665	1	0.86	1	0.82	0.03	0.81	0.11	1	0.88	0.12	0.66				
12	0.693	1	0.86	1	0.82	0.03	0.81	0.11	1	1	0.88	0.12	0.66			
13	0.607	1	0.86	1	0.82	0.03	0.81	0.11	1	1	0.21	0.24	0.12	0.66		
14	0.518	1	0.12	0.10	1	0.82	0.03	0.81	0.11	1	1	0.21	0.24	0.12	0.66	
15	0.489	1	0.05	0.13	0.12	1	0.82	0.03	0.81	0.11	1	1	0.21	0.24	0.12	0.66

Table 5: Global Silhouette values for clustering obtained by using GraphClust with Antipole Tree data structure.

Now, we have to show that this clustering is also coherent with the *a priori* classification of the data set. Recall that in the optimal clustering each cluster contains a single structural group where in each group the graphs differ by 20% of their edges. Table 7 shows the similarity in percent between the best clustering obtained in table 5 and 6 for $c = 10$ and the optimal clustering. A value $x\%$ for the cluster C_i obtained with GraphClust means that C_i is equal to $x\%$ of the optimal cluster C_i. To measure the robustness of the clustering obtained, a pair of graphs g_1 and g_2 are considered to be consistent in the two clusterings if they are in the same cluster in both cases or in different clusters in both cases. Otherwise they are inconsistent. In table 8, for

c	GSu	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	S_{11}	S_{12}	S_{13}	S_{14}	S_{15}
10	0.886	1	1	0.66	0.88	1	0.86	0.82	0.81	0.81	1					
11	0.830	1	1	0.66	0.89	1	0.86	0.82	0.81	0.81	1	0.25				
12	0.714	1	1	0.66	0.20	1	0.87	0.82	0.81	0.81	1	0.25	0.12			
13	0.678	1	1	0.66	0.20	1	0.11	0.82	0.81	0.81	1	0.25	0.12	1		
14	0.643	1	1	0.66	0.20	1	0.14	0.82	0.81	0.81	1	0.25	0.16	1	0.10	
15	0.660	1	1	0.66	0.20	1	0.09	0.82	0.81	0.81	1	0.25	0.11	1	0.10	1

Table 6: Global Silhouette values for clustering obtained by using GraphClust with k-means algorithm.

$c = 10$, the output clustering generated by GraphClust has been compared with the optimal clustering. The value *consistent_value* shows the number of graphs pairs that are consistent divided by the total number of graphs pairs for the output obtained.

Clustering Algorithm	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}
Antipole Tree	100%	100%	100%	100%	66.0%	95.14%	100%	100%	28.9%	50%
K-means	100%	54.4%	50%	100%	100%	100%	100%	100%	100%	45.6%

Table 7: Similarity in percentual between the best clustering found ($c = 10$) in Tables 5, 6 and the optimal clustering.

Clustering Algorithm	Number of consistent pairs	Total number of pairs	*consistent_value*
Antipole Tree	48704861	49995000	0.97
K-means	48746936	49995000	0.97

Table 8: Robustness between the best clustering found in Tables 5, 6 and the optimal clustering.

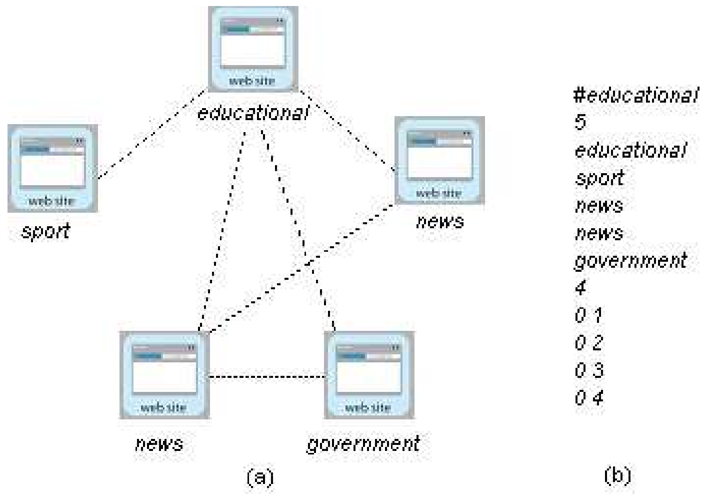
6 Experiments on real data

In the previous section we showed the performance of GraphClust clustering on syntethic datasets. Now we will show some clustering results on graphs representing the connections of internet web-sites. We considered as data sources 4 internet domains : Spanish (es), Chinese (cn), Egyptian (eg) and Chile (cl). We classified every web-site in these domains according to the following 9 categories: *news*, *sports*, *educational*, *research*, *government*, *industry*, *media*, *tourism*, *banking*. We represented web-sites as graph nodes and their links as graph edges.

We built 4 datasets in the following way: first of all, for each domain D, we labelled each web-site with one of the above 9 categories. Then we considered each graph out up to one edge. Fig. 3 shows a small data source and the LNE representation of one of its graphs. The 4 datasets thus obtained have a number of graphs varying from 3000 to 5000 graphs. GraphClust was executed in order to group in same clusters the web-sites with the similar structure. GraphClust was run by using AllPairShortestPath to find the web-sites substructures and the k-means algorithm for clustering with $k = 9$ in order to obtain 9 clusters. Table 9 shows the results of GraphClust on the 4 datasets. Each row contains the statistics of the clustering for each dataset. A percentual value

$x\%$ means that for the underlined dataset, one of the obtained clusters contains the $x\%$ of web-sites belonging to the category specified in the column.

Note that the high percentual values prove that GraphClust was able to perform a very good clustering for all the tested datasets.



```
#educational
5
educational
sport
news
news
government
4
0 1
0 2
0 3
0 4
```

Figure 3: (a) Very small data source with 5 web-sites. (b) The graph for the *educational* web-site in LNE format.

domain	*news*	*sports*	*educational*	*research*	*government*	*industry*	*media*	*tourism*	*banking*
Spanish	65.4%	59.8%	51%	50.3%	61%	53.5%	54%	51%	52.3%
Chinese	60.2%	51.1%	55.3%	56.7%	66.7%	63.2%	50.4%	50.6%	53%
Egyptian	55.4%	52.4%	55%	55.3%	68.2%	60.1%	52%	55%	58.2%
Chile	58%	59.3%	54.9%	50%	68.9%	52.8%	50%	51.2%	53%

Table 9: Clustering results on 4 internet domains.

7 Conclusion and Future work

We have proposed a new method for clustering labelled graphs that entails (i) identifying interesting substructures, (ii) clustering the graphs by their substructures and (iii) finding highly co-occurring substructures pairs. The algorithm performs well and gives high quality for both artificial and real data sets. Graphclust is implemented in ANSI C and a software implementation is freely available at www.cs.nyu.edu/shasha/papers/graphclust/.

We are extending GraphClust to deal with nearest and range query search. Also, to reduce the space complexity, we will make use of the Berkeley Database to store all the computed paths. Moreover, new techniques for finding common substructures and for classifying the most important ones are under study.

Acknowledgement

We thank Professor Davood Rafiei from the Department of Computing Science of University of Alberta for having provided us the internet data graphs of Spanish, Chinese, Egyptian and Chile domains.

References

[1] http://cygnus.uta.edu/subdue, The SUBDUE Knowledge Discovery System.

[2] M.W. Berry, Z. Drmac, and E.R. Jessup. Matrices, vector spaces, and information retrieval. *SIAM Review*, 41(2):335–362, 2003.

[3] B.Luo, R.C.Wilson, and E.R.Hancock. Spectral feature vectors for graph clustering. *Proceedings of joint Syntactical and Structural Pattern Recognition and Statistical Pattern Recognition*, 2396:83–93, 2002.

[4] B.Luo, R.C.Wilson, and E.R.Hancock. Spectral clustering of graphs. *Proceedings of 4th IAPR-TC15 Graph based Representations in Pattern Recognition*, pages 190–201, 2003.

[5] N. Bolshakova and F. Azuaje. Improving expression data mining through cluster validation. *Information Technology Applications in Biomedicine, 2003*, pages 19–22, 2003.

[6] H. Bunke. Graph-based tools for data mining and machine learning. *Proceedings of Machine Learning and Data Mining in Pattern Recognition*, pages 7–19, 2003.

[7] D. Cantone, A. Ferro, A. Pulvirenti, D. Reforgiato, and D. Shasha. Antipole tree indexing to support range search and k-nearest-neighbor search in metric spaces. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 17(4):535–550, 2004.

[8] D.R. Cutting, D.R. Karger, J.O. Pedersen, and J.W. Tukey. Scatter / gather: A cluster-based approach to browsing large document collections. *Proc. ACM SIGIR 92*, pages 318–329, 1992.

[9] S. Deerwester, S.T. Dumais, T.K. Landauer, G.W. Furnas, and R.A. Harshman. Indexing by latent semantic analysis. *Journal of the Society for Information Science*, 41(6):391–407, 1990.

[10] A. Ferro, R. Giugno, A. Pulvirenti, D. Reforgiato Recupero, and D. Shasha. Blastgen, a graphs generator for graph matching benchmarking. *Preprint*, 2005.

[11] T.F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

[12] David Hull. Improving text retrieval for the routing problem using latent semantic indexing. *In Proceedings of the 17th ACM/SIGIR Conference*, pages 282–290, 1994.

[13] P. Husbands, H. Simon, and C. Ding. On the use of singular value decomposition for text retrieval. *Proc. of SIAM Comp. Info. Retrieval Workshop*, pages 145–156, 2001.

[14] S. Kosinov and T. Caelli. Inexact multisubgraph matching using graph eigenspace and clustering models. *Proceedings of joint Syntactical and Structural Pattern Recognition and Statistical Pattern Recognition*, 2002.

[15] G. Kowalski. *Information retrieval systems: Theory and implementation.* Boston: Kluwer Academic Publishers, 1997.

[16] Steinbach M., Karypis G., and Kumar V. A comparison of document clustering techniques. *Proc. Text Mining Workshop, KDD 2000*, pages 1–11, 2000.

[17] S. Rajappap. Interactive biasing in graph-based data mining. *Master Thesis in Computer Science and Engineering*, 2003.