Multilevel Refinement for Hierarchical Clustering*

George Karypis, Eui-Hong (Sam) Han, and Vipin Kumar

Department of Computer Science & Engineering
Army HPC Research Center
University of Minnesota, Minneapolis, MN 55455
Technical Report # 99-020

{karypis, han, kumar}@cs.umn.edu

Abstract

Hierarchical methods are well known clustering technique that can be potentially very useful for various data mining tasks. A hierarchical clustering scheme produces a sequence of clusterings in which each clustering is nested into the next clustering in the sequence. Since hierarchical clustering is a greedy search algorithm based on a local search, the merging decision made early in the agglomerative process are not necessarily the right ones. One possible solution to this problem is to refine a clustering produced by the agglomerative hierarchical algorithm to potentially correct the mistakes made early in the agglomerative process. The problem of refining a clustering has many similarities with that of refining a min-cut k-way partitioning of a graph. In this paper, we explore multilevel refinement schemes for refining and improving the clusterings produced by hierarchical agglomerative clustering. This algorithm combines traditional hierarchical clustering with multilevel refinement that has been found to be very effective for computing min-cut k-way partitioning of graphs. We consider several clustering objective functions for the proposed refinement step and investigate the usefulness of these objective functions. Our experimental results demonstrate that this algorithm produces clustering solutions that are consistently and significantly better than those produced by hierarchical clustering algorithms alone. Furthermore, our algorithm has the additional advantage of being extremely fast, as it operates on a sparse similarity matrix. The amount of time required by our algorithm ranged from two second for a data set with 358 items, to 80 seconds for a data set with 9133 items on a Pentium II PC.

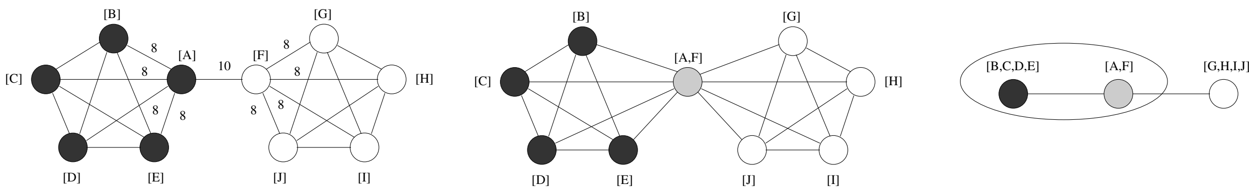
1 Introduction

Hierarchical methods are commonly used for clustering in Data Mining [9, 17, 2]. A hierarchical clustering scheme produces a sequence of clusterings in which each clustering is nested into the next clustering in the sequence. An

*This work was supported the Army Research Office contract DA/DAAG55-98-1-0441, the NSF CCR-9423082, and the Army High Performance Computing Research Center under the auspices of the Department of the Army, Army Research Laboratory cooperative agreement number DAAH04-95-2-0003/contract number DAAH04-95-C-0008, the content of which does not necessarily reflect the position or the policy of the government, and no official endorsement should be inferred. This work was also supported by IBM Partnership Award. Access to computing facilities was provided by AHPCRC and the Minnesota Supercomputer Institute. Related papers are available via WWW at URL: **http://www.cs.umn.edu/˜karypis**

agglomerative algorithm for hierarchical clustering starts with n points, and at each step it merges the two most similar points [9]. Different measures have been proposed for computing similarities [6, 7, 11]. In some of these schemes, a model of the cluster connectivity is used to compute similarities [9, 3, 7, 11].

Hierarchical clustering is a greedy search algorithm based on a local search. Hence the merging decision made early in the agglomerative process are not necessarily the right ones. For example, for the data set in Figure 1 (a), the similarity between data points A and F is 10, which is larger than the similarity among all other pairs shown in the figure. Hence, hierarchical scheme will merge A and F first in the agglomerative process (Figure 1 (b)). Eventually, both A and F will become part of one of the two clusters, even though they really belong to different clusters.



(a) (b) (c)

Figure 1: An example of data set in which the greedy merging decision leads to a wrong clustering solution.

One possible solution to this problem is to refine a clustering produced by the agglomerative hierarchical algorithm to potentially correct the mistakes made early in the agglomerative process. For example, given the clusters in Figure 1 (c), the refinement could break up $\{A, F\}$ and move F to the cluster containing $\{G, H, I, J\}$, thus correct the mistake made in the step of Figure 1 (b). In general, individual data points or collections of them could be moved from one cluster to another to optimize some cluster quality objective. Many such cluster quality objectives have been investigated [9, 7], and can potentially be used for such refinement.

The problem of refining a clustering has many similarities with that of refining a min-cut k-way partitioning of a graph. Given a graph that has been partitioned into k parts, the refinement of this k-way partitioning moves nodes across partitions to minimize the weighted sum of the edges straddling the partition boundaries [15]. The problem of refinement of k-way partitioning of a graph has been studied extensively in the context of graph partitioning, and efficient multi-level algorithms are available to solve this problem [15]. In multilevel graph partitioning algorithms, a sequence of coarser graphs is constructed and a k-way partitioning of the coarsest graph is computed. In each ensuing uncoarsening step, the k-way partitioning is successively refined using variations of the Kernighan-Lin (KL) [18] refinement heuristic. Since this refinement is performed at many levels, even simple variations of KL become very powerful [12].

Similar multi-level refinement schemes can be used to refine a clustering and potentially correct mistakes made early. In fact, the agglomerative hierarchical clustering schemes have a great deal of similarity with the coarsening phase of multilevel algorithms for finding a min-cut partitioning of graphs [8, 13]. Both schemes obtain successively coarser representations of their original data sets and both schemes use a locally greedy approach to construct these representations. However, conventional hierarchical clustering algorithms lack the refinement phase of the partitioning algorithm. As the extensive experience with multilevel graph partitioning has shown, the refinement phase (especially when applied in a multilevel fashion) is capable of significantly improving the overall quality of the solution [12]. A similar multi-level refinement algorithm holds the potential to improve upon the locally greedy decisions used in the agglomerative hierarchical scheme.

In this paper, we explore multilevel refinement schemes for refining and improving the clustering produced by hierarchical agglomerative methods. We consider several clustering objective functions for the proposed refinement step, and investiage the usefulness of these functions in the context of variety of data sets.

The rest of this paper is organized as follows. We first discuss agglomerative hierarchical clustering in Section 2. In Section 3, we present our clustering refinement algorithm. Section 4 provides an experimental evaluation of our clustering algorithm and compares it with other hierarchical algorithms. Finally, we provide summary in Section 5.

2 Review of Agglomerative Hierarchical Clustering Algorithms

There are many different variations of agglomerative hierarchical algorithms [9]. These algorithms primarily differ in how they update the similarity between existing clusters and the merged clusters. In some methods [9], each cluster is represented by a centroid of the points contained in the cluster, and the similarity between two clusters is measured by the similarity between the centroids of the clusters. These methods tend to fail on clusters of arbitrary shapes and different sizes. Recently proposed algorithm CURE [6] remedies some of these drawbacks by representing each cluster with a collection of representative centroids.

In many cases, pair-wise similarity is the only information available, making the use of centroid based hierarchical scheme difficult or impossible. Many agglomerative schemes can work for data for which only pair-wise similarity is available. In the single link method [9], each cluster is represented by all the data points in the cluster. The similarity between two clusters is measured by the similarity of the closest pair of data points belonging to different clusters. Unlike the centroid/medoid based methods, this method can find clusters of arbitrary shape and different sizes. However, this method is highly susceptible to noise, outliers, and artifacts.

In some agglomerative hierarchical algorithms, the similarity between two clusters is captured by the aggregate of the similarities (i.e., interconnectivity) among pairs of items belonging to different clusters. The rationale for this approach is that subclusters belonging to the same cluster will tend to have high interconnectivity. But the aggregate inter-connectivity between two clusters is a function of the size of the clusters involved, and in general, pairs of larger clusters will have higher inter-connectivity. Hence, many such schemes normalize the aggregate similarity between a pair of clusters with respect to the expected inter-connectivity of the clusters involved.

One often used method assumes that a cluster of size n contains n^θ edges, where θ is between 1 and 2. Let A and B be two clusters of size n and m, respectively. If A and B belong to the same natural cluster, then there will be a total of $(n+m)^\theta$ edges in the cluster. Of these, n^θ and m^θ edges will be internal to clusters A and B, and $(n+m)^\theta - n^\theta - m^\theta$ edges will be across clusters A and B. Hence the similarity between A and B is computed as the ratio

$$\frac{\text{Aggregate similarity between items A and B}}{(n+m)^\theta - n^\theta - m^\theta}$$

This is essentially the model described in [7]. For $\theta = 2$, this also becomes essentially equal to the group average model [9]. We will refer to this as the generalized group average model.

Graph Sparsification Most of the algorithms discussed above work implicitly or explicitly with the $n \times n$ similarity matrix such that (i, j) element of the matrix represents the similarity between i^{th} and j^{th} data items. Some algorithms derive a new similarity matrix using the original matrix [10, 5, 9, 7], and then apply one of the existing techniques on this derived similarity matrix. In many cases, the new derived similarity matrix is just a sparsified version of this original similarity matrix from which certain entries (e.g., those whose value is below a threshold) have been deleted. In other cases, the derived similarity matrix has entirely different values [10, 5, 7]. The sparsified/derived matrix can help eliminate/reduce noise from the data, and substantially reduce the execution time of many algorithms. In some cases, it can also provide a better model of similarities for the problem domain. For example, mutual shared method presented in [10] helps remove noise and outliers and is shown to provide a better model to capture similarities among transactions in [7]. Two most common techniques for sparsifying dense graphs are k-nearest graph [10, 5, 9], shared nearest neighbor [10, 7], and their variations [5, 7]. In some cases, these sparsification techniques are so effective that the fragment the graph according to cluster boundaries. But in presence of outliers and noise, finding good clusters of the resulting sparse graph is still quite challenging for most problems..

3 Cluster Refinement Algorithm

In order to refine a clustering solution, we must develop two major components. First, we need to develop schemes that can capture the overall goodness of a clustering in the form of a clustering *objective function* such that the optimization of this function translates to an improvement of the overall clustering solution. Second, we need to develop effective algorithms to find groups of items whose movement to different clusters will optimize the objective function.

In the remaining of this section we present two different ways of defining the clustering objective function and present a cluster refinement algorithm that is based on the multilevel refinement paradigm.

3.1 Refinement Objective Functions

There is no single canonical method to capture/describe the clustering objective. Recently, Guha, Restogi, and Shim [7] have proposed a function for measuring the goodness of a clustering solution. Given a similarity matrix S, and a p-way clustering, they define the goodness of the clustering as

$$E_p = \sum_{i=1}^{p} n_i \frac{\sum_{v,u \in C_i} S[v,u]}{n_i^{\theta}}, \tag{1}$$

where C_i denotes the ith cluster of size n_i. The rational of the above goodness function is as follows [7]. For a particular cluster C_i, the quantity $\sum_{v,u \in C_i} S[v,u]$ measures the degree of connectivity of the nodes in the cluster. Since a good clustering should maximize the degree of connectivity for each cluster, a metric like $\sum_{i=1}^{p} \sum_{v,u \in C_i} S[v,u]$ that simply sums up the amount of similarity between nodes in the same clusters should have been sufficient. However, such a metric gives the highest goodness to a clustering that puts all the data items into a single cluster. To address this problem, Equation 1 divides the degree of connectivity among the nodes of each cluster with the quantity n_i^{θ}, that represents the expected degree of connectivity among the items of cluster C_i. The value of parameter θ depends on the data set as well as the kind of the clusters we are interested in. Finally, the goodness measure for each cluster is weighted by the size of the cluster, so that large clusters contribute more to the overall goodness function.

To see the utility of the cluster objective function, consider the simple hierarchical clustering instance shown in Figure 2 that shows a data set with six items. As easily seen, this example has two natural clusters, one containing the set of items $\{A, B, C\}$ and the other containing the set $\{D, E, F\}$. However, the hierarchical algorithm, starts by combining items C and D together. Even though this is the best available choice in the beginning, it does not lead to the optimal solution. Since the algorithm does not revisit these choices later, it can never correct its mistake. This mistake can potentially be corrected if the final clustering of Figure 2 (e) is refined in the context of some objective function.

If the objective function of Equation 1 is used to evaluate the two clusters found by the hierarchical algorithm on the example of Figure 2, then the cluster consisting of $\{A, B, C, D\}$ has a goodness of $70/4^2 = 4.375$, whereas the cluster $\{E, F\}$ has a goodness of $18/2^2 = 4.5$, leading to an overall goodness value of of $4 * 4.375 + 2 * 4.5 = 26.5$. If D is moved to the cluster containing $\{E, F\}$, then the resulting clusters $\{A, B, C\}$ and $\{D, E, F\}$ have an overall goodness value of $3 * (50/3^2) + 3 * (50/3^2) = 33.3$, which as expected, is better than that of the previous clustering.

There are number of potential problems with the above objective function. First, this function can lead to incorrect information if the internal connectivity of clusters is sufficiently different than the one assumed. Given that cluster

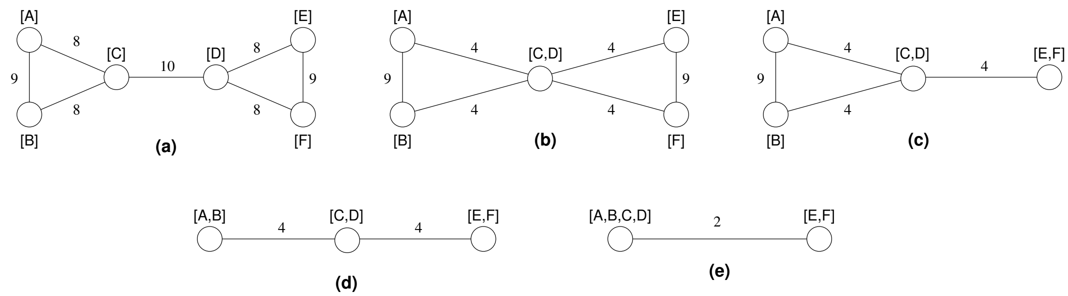


Figure 2: An example of the hierarchical clustering algorithm. Note that the locally greedy decisions performed by the algorithm can lead to sub-optimal clustering solutions.

connectivity is often unknown, a wrong choice of θ can lead to incorrect clustering even if the objective function is maximized. Note that this becomes a serious problem when the data set contains clusters that have different levels of interconnectivity, as no single level of θ is valid for all clusters. Another problem with this scheme is that many agglomerative hierarchical schemes work with sparsified graph. In fact, the multi-level refinement scheme presented in this paper assumes the similarity graph to be sparse. For such sparse graphs, the value of θ becomes quite sensitive to cluster sizes.

Another possible objective is to minimize the external connectivity of the clusters. The external connectivity is essentially minimized by min-cut k-way partitioning of the similarity graph. A key problem with the min-cut objective is that it often gets optimized when there are $k - 1$ clusters containing just one data point each, and one large cluster containing the remaining points. For example, for the data set shown in Figure 3 (a), the min-cut objective for three cluster is optimized when one cluster contains A, the second cluster contains B, and the third cluster contains the remaining data points (as shown in Figure 3 (b)).

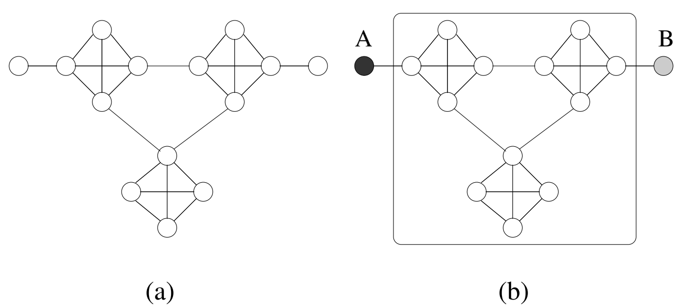


Figure 3: An example of clusters in which min-cut objective leads to a wrong clustering solution.

This problem can be corrected if the edge weights are scaled according to the ratio-cut heuristic. That is, weight w of an edge between two clusters is scaled by a factor of $\frac{1}{|A| \times |B|}$ (and is thus replaced by $\frac{w}{|A| \times |B|}$), and the objective becomes one of minimizing this scaled weighted sum. For example, Figure 4 shows the new scaled edge weight when the ratio-cut heuristic is used. The new scaled weights of the edges connecting the clusters in Figure 4 (a) is larger than those of the edges connecting the clusters in Figure 4 (b). Hence, the clustering solution shown in Figure 4 (b) is preferred over the clustering solution shown in Figure 4 (a).

This objective is very similar to the objective that drives the agglomeration step in the group averaging scheme, as it effectively assumes full inter-connectivity between all pairs of data points across two clusters, and uses the expected connectivity ($|A| \times |B|$) to scale down the edge weight. A more general formulation will scale each edge between A and B by $(|A| + |B|)^\theta - |A|^\theta - |B|^\theta$. This is similar to the agglomeration method used in ROCK. In this case, the expected inter-connectivity is controlled by parameter θ.

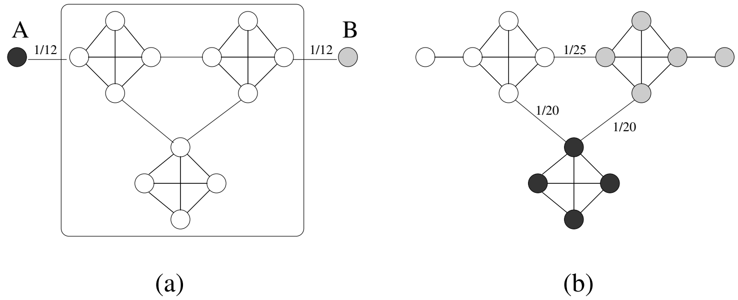


Figure 4: An example of clusters in which ratio-cut heuristic is useful.

This objective function is able to correctly refine the clustering of Figure 2. But this clustering objective function can also be misleading in certain situations. Specifically, if the data contains clusters of widely different sizes, this objective will tend to break larger clusters. For example, consider the clusters shown in Figure 5. With $\theta = 2$, the scaling factor in the objective function is $2 \times |A| \times |B|$ for two clusters A and B. Hence, the ratio cut of the cut X in Figure 5 is $\frac{2}{2\times1\times80} = 0.013$, whereas the ratio cut of the cut Y is $\frac{25}{2\times41\times40} = 0.008$. Hence the cut Y is chosen and the wrong clustering based on this cut is produced.

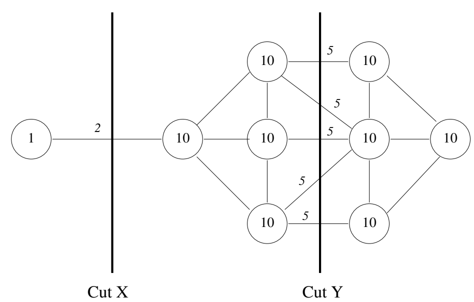


Figure 5: An example of clusters in which ratio-cut objective leads to a wrong clustering solution. Note that numbers inside circle represents number of data points in the subcluster and the numbers on the edge indicate the total number of edges between two subclusters.

3.2 Refinement Algorithm

As discussed earlier, the goal of the cluster refinement algorithm is to find groups of items, such that by moving them to different clusters it optimizes a certain objective function. One way of finding the desired groups, is to find them a single item at a time, using a greedy scheme. That is, we can repeatedly move the item that will lead to the greatest improvement of the objective function. Unfortunately, a scheme like that can easily be trapped into local minima. This is because quite often, in order to substantially improve the objective function, we may have to move entire sets of items between clusters. However, if we start moving these *desired* sets of items single item at a time, the objective function may initially become worse before it gets better. One way of addressing this problem is to use much more sophisticated refinement algorithms that are capable of climbing out of local minima (*e.g.*, simulated annealing). However these type of algorithms can significantly increase the amount of time required to perform the cluster refinement.

Recently, a new class of refinement algorithms have been developed in the context of graph partitioning, that have small computational requirements and they are very effective in climbing out of local minima [16]. These ***multilevel refinement algorithms*** work as follows. Given a graph $G = (V, E)$, and a partitioning vector P, they first obtain a sequence of successively *coarser* graphs $G_1, G_2, \ldots, G_k$. Graph G_1 is obtained from the original graph G, by finding a maximal matching of the vertices of G subject to the constraint that each pair of matched vertices belongs to the same partition, and collapsing the matched vertices together to form the vertices of G_1. Each successive graph G_{i+1} is obtained from the previous coarse graph G_i in a similar fashion. Note that since each successive coarse graph does not combine vertices that belong to different partitions, the original partitioning is preserved in the entire sequence of graphs. Once this sequence of graphs has been constructed, then a greedy refinement algorithm is obtained to improve the quality of the partitioning at the coarsest graph G_k. The new partitioning is then projected to the next level finer graph G_{k-1}, and it is further refined using a greedy algorithm. This process of projecting and refining the partitioning at each successive finer graph is performed until the original graph has been reached. Multilevel refinement algorithms are very effective in climbing out of local minima, because they operate at different representation scales; thus, they can easily identify groups of items to be moved together.

Our cluster refinement algorithm is based on this multilevel refinement paradigm. The input to our multilevel cluster refinement algorithm is the sparse similarity graph $G = (V, E)$ used by the clustering algorithm to represent the similarity relations among the data items, and a p-way clustering vector P produced by the clustering algorithm, such that $P[i]$ is the cluster that the ith item is assigned to. Starting from the original graph, our algorithm constructs a sequence of successively coarser graphs $G_1, \ldots, G_k$, until we obtain a graph that has exactly p vertices (one for each cluster), and then applies a simple randomized greedy refinement algorithm at each successive finer graph. This randomized greedy refinement algorithm consists of a number of passes. During each pass of the algorithm, the various vertices in the graph (*i.e.*, sub-clusters) are visited in a random order. For each vertex v, it computes the improvement of the value of the objective function obtained if v was to move from the cluster that it currently belongs to, to one of the other clusters that v is connected to. If some of these moves improve the objective function, then the one that leads to the highest improvement is selected and v is assigned to this cluster. If all the moves worsen the value of the objective function, then v is not moved. The refinement algorithm stops either when after an entire pass not a single vertex was moved to another cluster, or when a predetermined number of passes has been performed. In our experiments, we allowed the refinement algorithm to perform a maximum of five passes. In our extensive experience with multi-level partitioning of very large graphs (over 1 million nodes), we have found this limit to be quite sufficient, as much of the improvement occurs in just a few iterations.

From the above description of the multilevel refinement algorithm and the clustering objectives discussed in Section 3.1 we can define three distinct clustering refinement algorithms. The first algorithm tries to increase the internal inter-connectivity of the items in the clusters by maximizing Equation 1. The second algorithm tries to reduce the inter-connectivity between clusters by minimizing the ratio cut of the resulting p-way clustering. Finally, the third algorithm tries to achieve both by selecting to move items if such moves improve both Equation 1 as well as the ratio cut; that is, a move is performed if the resulting clustering has a higher quality as measured by Equation 1 and at the same time it has a smaller ratio cut.

The multilevel cluster refinement algorithm can be used in many different ways. One possible approach is to use it to refine the final clustering solution produced by the hierarchical clustering algorithm, as it is done in our experiments. However, an alternate approach is to use it to periodically refine the current clustering solution as it is being computed.

3.3 Computational Complexity

The overall complexity of the multilevel clustering refinement algorithm depends on the rate in which the size of successively coarser graphs is decreasing. In particular, if the size of successively coarse graphs decreases by a constant factor, then the complexity of the algorithm is linear on the number of items and the number of edges in the sparse similarity graph [14, 16]. Since successive coarse graphs are constructed by computing a maximal matching of the vertices and collapsing together the vertices that were matched, the number of vertices of successive coarse graphs tends to decrease by a factor of two. In this case, if the sparse similarity graph was obtained using a k-nearest neighbor approach [9], then the overall complexity is linear on the number of items (as the total number of edges is linear on the number of nodes). In the worst case, when the size of successively coarse graphs decreases by just a few vertices at a time, then the complexity of the refinement algorithm will be quadratic on the number of items.

In general, the overall complexity of a clustering algorithm that uses the multilevel refinement algorithm depends on the amount of time required to compute the initial similarity matrix and the amount of time required by the clustering algorithm. For most problems, the dense similarity matrix can be computed in time that is quadratic on the number of items, and the initial clustering solution can be obtained in a similar amount of time.

4 Experimental Results

To evaluate the ability of our multilevel refinement algorithms to further improve the quality of a clustering, we used them to refine the solutions produced by the hierarchical clustering algorithm described in Section 2 that is based on the generalized group average model. We used five data sets to perform these comparisons. Two of these data sets consist of points in two dimensions and were synthetically generated, one was obtained from Reuters newswire and the other two were obtained from the TREC collection of documents [22].

For each one of the data sets we constructed an $n \times n$ similarity matrix, using techniques that are appropriate for the nature of each particular data set. Details on how the similarity matrices were constructed are presented along with the experimental results in the following sections. From each one of these five similarity matrices, a sparse graph representation was obtained by using the k-nearest neighbor graph approach [10, 5]. In all the experiments presented in Section 4.2 we selected k to be equal to 10, and we studied the effectiveness of our refinement algorithms for different values of k in Section 4.3.

Recall from Sections 2 and 3, that both the hierarchical clustering algorithm that was used to obtained the clustering solutions as well as the clustering objective functions that are used by our refinement algorithms, require that we specify the value of the parameter θ that models the degree of inter-connectivity between the items in a cluster. In all the experiments presented in Section 4.2 we kept θ to be equal to 1.8 for both the clustering as well as the clustering refinement algorithms. The sensitivity of our refinement algorithms to different values of θ is studied in Section 4.3 in which we present an extensive experimental evaluation of our algorithms for different values of θ.

In addition to the hierarchical clustering algorithm presented in Section 2 that operates on the sparse k-nearest neighbor graph, we also compare the quality of the clusterings produced by our algorithms against two other algorithms. In the case of point data sets, we compare our results against CURE [6], and in the case of the document data sets, we compare our results against the standard hierarchical algorithm based on the group average method [9] that operates on the dense similarity matrix.

For the rest of this section, we will use DH to denote the traditional hierarchical clustering algorithm that operates on the dense similarity matrix, SH to denote the hierarchical clustering algorithm described in Section 2 that operates on the k-nearest neighbor graph, rSH-RC to denote our multilevel refinement algorithm that uses the ratio-cut objective, rSH-IRC to denote our multilevel refinement algorithm that optimizes both the internal connectivity as well as the ratio cut, and rSH-I to denote our multilevel refinement algorithm that optimizes the internal connectivity. Finally, all the experiments were performed on a 300Mhz Pentium II workstation.

4.1 Cluster Evaluation

One of the hardest problems in comparing different clustering algorithms is finding an algorithm-independent measure to evaluate the quality of the clusters. This is especially true for data sets with many different categories. In general, if a cluster contains items that belong to only a single category, then it is a good cluster. However, evaluating clusters that contain items from different categories is less clear.

We use entropy as a measure of quality of the clusters (with the caveat that the best entropy is obtained when each cluster contains exactly one data point). Let CS be a clustering solution with m clusters. For each cluster, the class distribution of data is calculated first. Then using this class distribution, the entropy of each cluster j is calculated using the formula $E_j = -\sum_{C_i} p_{C_i} \log p_{C_i}$, where p_{C_i} is the fraction of data within the cluster with the class label C_i, and the sum is taken over all classes, $C_1, C_2, \ldots, C_k$. When a cluster contains data from one class only, the entropy value is 0.0 for the cluster and when a cluster contains data from many different categories, then entropy of the cluster is higher. The total entropy is calculated as the sum of entropies of the clusters weighted by the size of each cluster: $E_{CS} = \sum_j \frac{E_j n_j}{n}$, where n_j is the size of cluster j and n is the total number of data points. We compare the E_{CS} to that of a random clustering solution with the same number of clusters. We use the entropy gain of the clustering solution over the random clustering as the final goodness measure. In other words, the goodness of a clustering solution CS is defined as $E_{RS} - E_{CS}$, where RS is a random clustering solution with m clusters. Hence, the goodness measure is high for a good clustering solution and low for a bad clustering solution.

4.2 Qualitative Comparisons

Two-Dimensional Data Sets Our first data set consists of two two-dimensional point data sets DS1 and DS2 shown in Figures 6(a) and (d), respectively. DS1 contains nine circular clusters arranged in a 3×3 grid, and DS2 contains 10 ring-shaped clusters, pairs of which are concentric, with a line of outlier points cutting through all the 10 clusters and with some random outlier points. The DS1 data set contains 6,000 points, whereas the DS2 data set contains 8,000 points. The similarity between two points was computed as the inverse of their Euclidean distance.

Figure 6(b) shows the nine clusters obtained by the SH algorithm with $\theta = 1.8$ for DS1. The points in the different clusters are represented using a combination of different colors and different glyphs. (Furthermore, we have drawn an outline around each cluster to make it easier for people that do not have access to a color printout). As we can see from the figure, even though SH is able to correctly cluster most of the data points, it does make a number of mistakes. In particular, the middle cluster of the second row, contains points from both the left and the right clusters. Similarly, the cluster middle cluster of the top row contains a good fraction of the points of the left top-row cluster, whereas the right cluster at the top row contains some nodes from its cluster to the left and its cluster to the bottom. Figure 6(c) shows the nine clusters obtained by applying our multilevel clustering refinement algorithm rSH-RC, on the output of the SH algorithm with the same value of θ (similar results were also obtained for rSH-IRC). As we can see from the resulting clustering, our clustering refinement algorithm was capable to correct most of the mistakes made by SH. In fact, six clusters are perfect, whereas the remaining three clusters contain a small number of errors.

Figure 6(e) shows the 10 clusters obtained by the SH algorithm with $\theta = 1.8$ for DS2. The SH algorithm was able to find most of the clusters, but it made three mistakes. As Figure 6(e) illustrates, it merged the inner-ring with portion of the outer-ring of the second and last pair of clusters (from left to right), and it merged portions of the outer rings of the last two pairs of clusters. Figure 6(f) shows the 10 clusters obtained by applying rSH-IRC on the clustering solution obtained by SH with the same value of θ (similar results were also obtained for rSH-RC). As we can see from this figure, rSH-IRC was able to correct the three mistakes made by SH and obtain a perfect clustering solution. Finally, Figure 6(g) shows the 10 clusters obtained by another hierarchical clustering algorithm, CURE [6], that is especially suited for this type of data sets. As we can see from this figure, CURE was not able to find any of the 10 clusters.

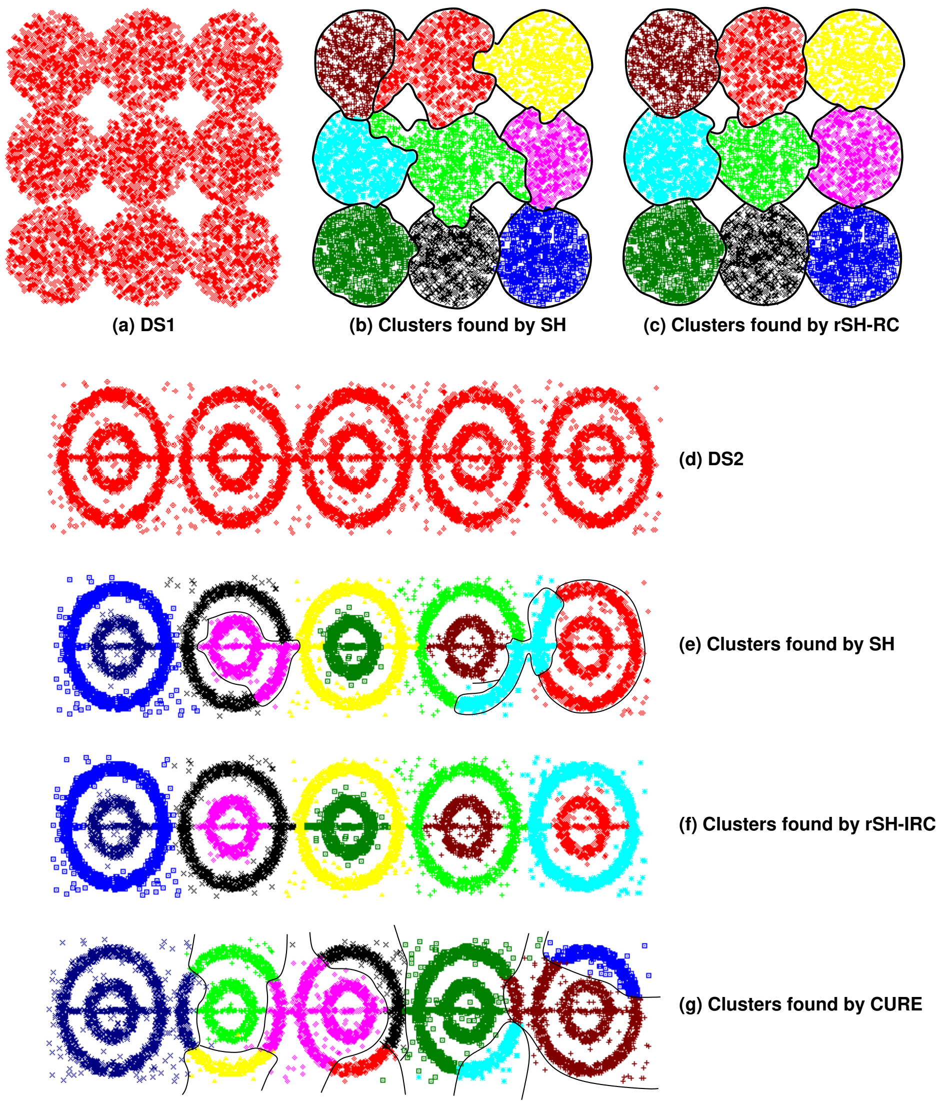


Figure 6: The two point data sets and the clusters produced by SH, rSH-RC, rSH-IRC, and CURE algorithms.

Los Angeles Times Data Sets The collection of the Los Angeles Times articles are part of the TREC 5 data set [22]. The Los Angeles Times data set consists of two sets of documents LA1, and LA2 that were created by selecting the articles published over two separate months (January and February of 1989) under certain sections of the newspaper. We used the section name of the article as the category for this data set. The category distribution is shown in Table 1.

We filtered the words using stop words and Porter's suffix-stripping algorithm [20]. Even after this prunning, the number of words in these documents was more than 31,000. In calculating similarity of two documents, we used the cosine similarity measure after scaling the documents with TFIDF [21]. The cosine similarity between two document X and Y is defined as

$$cos(X, Y) = \frac{\sum_{w \in W} x_w \times y_w}{\sqrt{\sum_{w \in W} x_w^2} \times \sqrt{\sum_{w \in W} y_w^2}}$$

where W is the collection of words appearing in the whole document set and x_w is the TFIDF weighted value of word

Category	No. of items – LA1	No. of items – LA2
Financial	555	487
Foreign	341	301
National	273	248
Metro	943	905
Sports	738	759
Entertainment	354	375
Total	3204	3075

Table 1: The various categories of the LA1 and LA2 data set and the distribution of records to each category.

w in document X. Note that this cosine measure gives value between 0.0 and 1.0, and 0.0 means that two documents do not match any word and 1.0 means that these two documents match perfectly.

Table 2 shows the quality of the clusters (as measured by the entropy of the clustering solution) produced by DH, SH, rSH-RC, and rSH-IRC for a 10-, 20-, 40-, and 80-way clustering. Comparing the various algorithms, we see that both the rSH-RC and rSH-IRC algorithms were able to improve the clustering solutions produced by the SH algorithm, and achieve the overall best results. The clusters produced by the hierarchical scheme on the dense graph are the worst for all cases. For the 10-way clustering, rSH-RC and rSH-IRC perform about 36% better than SH for both LA1 and LA2, for the 20-way clustering, rSH-RC and rSH-IRC perform 20% and 19% better than SH for LA1 and LA2, respectively, for the 40-way clustering, rSH-RC and rSH-IRC perform 17% better than SH for both LA1 and LA2, and for the 80-way clustering, rSH-RC and rSH-IRC perform 14% and 11% better than SH for LA1 and LA2, respectively. Also note that there is little variation between the two different refinement objectives used in the rSH-RC and rSH-IRC algorithms.

	LA1				LA2			
No. Clusters	10	20	40	80	10	20	40	80
Algorithm								
DH	.01	.25	.30	.38	.01	.20	.36	.41
SH	.33	.39	.43	.46	.37	.42	.45	.49
rSH-RC	.45	.46	.51	.53	.51	.50	.53	.54
rSH-IRC	.45	.48	.50	.52	.50	.50	.52	.55

Table 2: The quality of the clustering solution produced by DH, SH, rSH-RC, and rSH-IRC algorithms, for clustering the LA1 and LA2 data sets for 10, 20, 40, and 80 clusters.

An interesting trend that can be observed from Table 2 (and also holds for other data sets) is that the relative improvement achieved by our multilevel cluster refinement algorithms over SH as well as DH increases as the number of clusters decreases. In other words, as the clustering problem requires the algorithm to correctly cluster the data set using fewer clusters (and thus becomes harder), the multilevel refinement results in a greater degree of improvement in the overall clustering solution. On the other hand, as the number of desired clusters increases, the clustering problem becomes somewhat easier, which limits ability of our refinement algorithms to significantly improve upon an already good solution. For example, for the 10-way clustering, rSH-IRC obtain a clustering solution that is 36% better than SH, whereas for the 80-way clustering, rSH-IRC produces a clustering solution that is only 11% better.

Reuters Data Set The Reuters data set is from Reuters-21578 text categorization test collection Distribution 1.0 [19]. This data set contains 21,578 documents and each document is labeled with none, one, or many categories.

From these 21,578 documents we selected only the ones that belonged to a single category. This resulted in 9133 documents, whose category distribution is shown in Table 3. The similarity between the various documents were computed using the cosine measure after applying the same preprocessing steps as those used in the Los Angeles Times data sets.

Category	No. of items	Category	No. of items	Category	No. of items
earn	3923	acq	2292	commodity	535
economic index	892	energy	473	interest	271
metal	296	money	307	ship	144

Table 3: The various categories of the Reuter data set and the distribution of records to each category.

Table 4 shows the quality of the clusters produced by the four clustering algorithms for a 10-, 20-, and 40-way clustering. As with other data sets, we see that the algorithms that use multilevel refinement produce the best clustering solutions, whereas DH performs the worse. Compared to SH, we can see that rSH-RC and rSH-IRC perform 14%, 13%, and 11% better for the 10-, 20-, and 40-way clusterings, respectively.

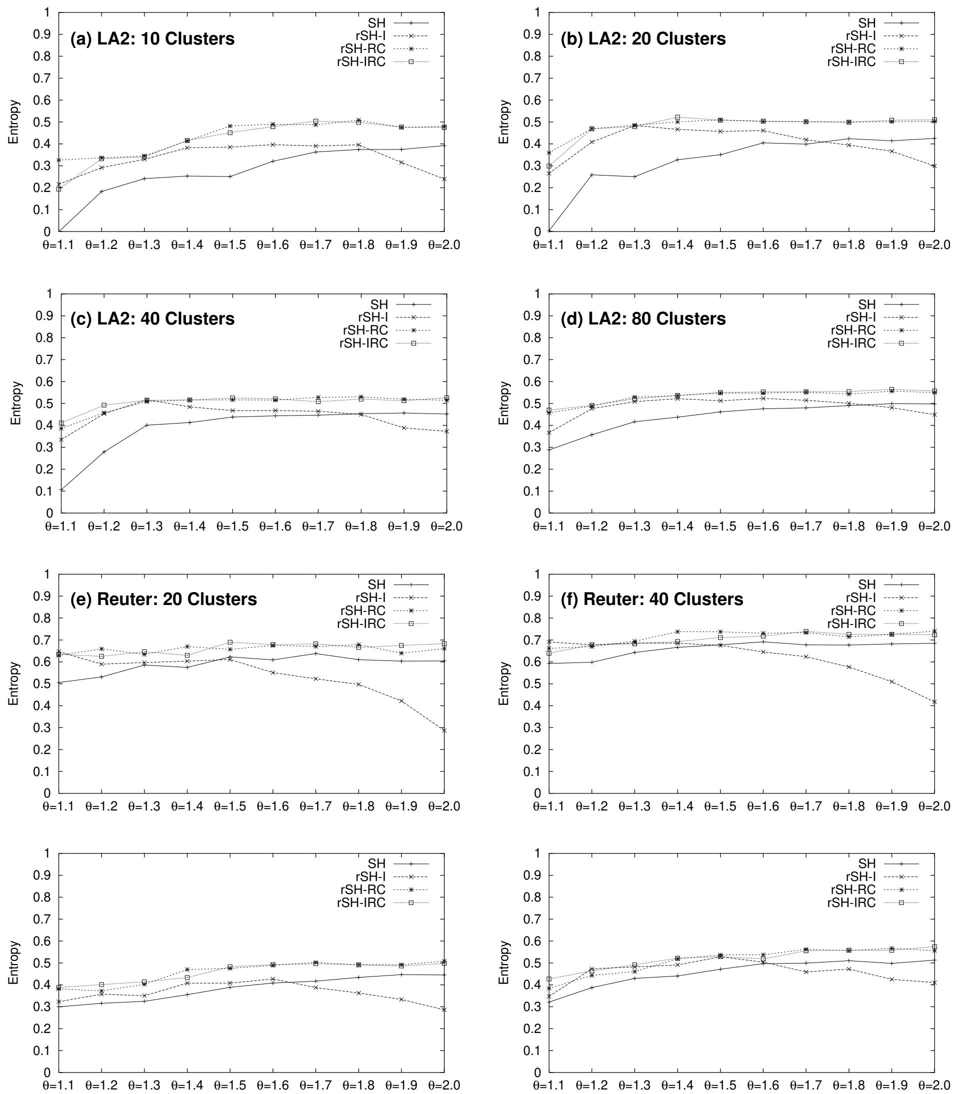
No. Clusters	10	20	40
Algorithm			
DH	.03	.27	.43
SH	.50	.59	.65
rSH-RC	.56	.68	.71
rSH-IRC	.58	.67	.73

Table 4: The quality of the clustering solution produced by the DH, SH, and mlSH algorithms, for clustering the Reuter data sets for 60 and 120 clusters.

4.3 Parameter Study

To study the sensitivity of our multilevel clustering refinement algorithms for different values of θ, we performed a sequence of experiments in which we varied θ from 1.1 up to 2.0 in increments of .1. Figure 7(a)–(f) shows the quality of the produced clusterings for the LA2 and Reuters data sets for different number of clusterings and different values of θ. Each plot of Figure 7 shows the quality of the clusterings produced by four schemes for different values of θ. The schemes shown are the following: (i) the SH algorithm which performs no refinement, (ii) the rSH-RC algorithm that refines the solution of SH using the ratio-cut objective function, (iii) the rSH-IRC algorithm that refines the solution of SH using the combination of the inward looking objective and the ratio-cut, and (iv) the rSH-I algorithm that refines the solution of SH using only the inward looking objective function.

A number of interesting observations can be made by looking at the various plots of Figure 7(a)–(f). First, irrespective of the value of θ and the number of clusters, the rSH-RC and rSH-IRC algorithms are able to further improve the quality of the clustering produced by SH. Moreover, compared to SH, the quality of the clustering solution produced by the rSH-RC and rSH-IRC algorithms are less sensitive to the value of θ (their quality lines are flatter). For example, looking at Figure 7(c), we can see that for $\theta < 1.3$, SH produces clustering solutions that have poor quality. However, the rSH-RC and rSH-IRC algorithms are able to substantially improve these clustering solutions. Also, similarly to the results presented in Section 4.2, the rSH-RC and rSH-IRC schemes perform very similarly (at least for these data sets).



(g) LA2: 20 Clusters (20 neighbors) (h) LA2: 40 Clusters (20 neighbors)

Figure 7: The quality of the clusters for different values of θ and different sparse graphs.

Figure 7 also compares the sensitivity of our multilevel clustering refinement algorithm on the degree of sparsification. In particular, Figures 7(b)–(c) and 7(g)–(h) shows the quality of a 20- and 40-way clusterings of LA2 for two different sparse graphs. The first set of results were obtained using the 10-nearest-neighbor graph (that was used throughout our experimental evaluation) whereas the second set of results were obtained using the 20-nearest-neighbor graph. Comparing these two sets of results, we can see that our refinement algorithms are equally effective in further refining the quality of the clusterings produced by SH.

Figure 7 also shows the sensitivity of the rSH-I refinement algorithm on the value of θ discussed in Section 3.1. As we can see from the various experiments, rSH-I performs equally well to both rSH-RC and rSH-IRC for small values of θ; however, as θ increases to a range above that of the underlying connectivity of the data set, the objective function used by rSH-I becomes unstable and leads to poor clusters. Also note that as the graph becomes somewhat denser (which is the case with the 20-nearest neighbor graphs shown in Figures 7(g) and (h)) rSH-I performs reasonably well for larger values of θ.

5 Conclusions and Directions of Future Research

In this paper we presented a new multilevel hierarchical clustering algorithm that builds upon recent advances in clustering and graph partitioning. As our experimental results have demonstrated, our algorithm combines traditional hierarchical clustering with multilevel refinement to produce clustering solutions that are consistently and significantly better than those produced by hierarchical clustering algorithms alone. Furthermore, our algorithm has the additional advantage of being extremely fast, as it operates on a sparse similarity matrix. The amount of time required by our algorithm ranged from two second for a data set with 358 items, to 80 seconds for a data set with 9133 items on a Pentium II PC.

Our work has demonstrated the value of refining the clustering solution in the multilevel context. Our current algorithm uses a variation of the standard hierarchical clustering algorithm to obtain an initial clustering. However, the multilevel refinement paradigm is independent of the particular choice of hierarchical algorithm, and other hierarchical clustering algorithms (*e.g.*, ROCK [7], CURE [6]) can also be used to obtain the initial clustering solution.

We believe that this paper represents the first attempt to develop a robust framework for refining clustering solutions in a multilevel setting. However, a number of key questions remain to be addressed. In particular, the choice of proper objective function is essential for the overall success of the multilevel refinement framework [6]. As our experiments seem to indicate, the objective function based upon the ratio cut is quite robust for a wide range of values of θ. Nevertheless, it is important to determine the domains in which it has limited applicability or fails out-right.

Finally, our experimental results (as well as those of other researchers [10, 5, 9]) have indicated that the sparsification of the similarity matrix often leads to better clustering performance. In the context of multilevel refinement, sparsification of the similarity matrix is even more important, as the complexity of each refinement step is proportional to the number of non-zeros in the similarity matrix.

In this paper, we ignored the issue of scaling to large data sets that cannot fit in the main memory. These issues are orthogonal to the ones discussed here and are covered in [23, 1, 6, 4].

References

[1] P.S. Bradley, U. Fayyad, and C. Reina. Scaling clustering algorithms to large databases. In *Proc. of the Fourth Int'l Conference on Knowledge Discovery and Data Mining*, 1998.

[2] M.S. Chen, J. Han, and P.S. Yu. Data mining: An overview from database perspective. *IEEE Transactions on Knowledge and Data Eng.*, 8(6):866–883, December 1996.

[3] R. Dubes and A.K. Jain. *Algorithms for Clustering Data*. Prentice Hall, 1988.

[4] V. Ganti, R. Ramakrishnan, J. Gehrke, A. Powell, and J. French. Clustering large datasets in arbitrary metric spaces. In *Proc. of the 15th Int'l Conf. on Data Eng.*, 1999.

[5] K.C. Gowda and G. Krishna. Agglomerative clustering using the concept of mutual nearest neighborhood. *Pattern Recognition*, 10:105–112, 1978.

[6] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. CURE: An efficient clustering algorithm for large databases. In *Proc. of 1998 ACM-SIGMOD Int. Conf. on Management of Data*, 1998.

[7] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. ROCK: a robust clustering algorithm for categorical attributes. In *Proc. of the 15th Int'l Conf. on Data Eng.*, 1999.

[8] Bruce Hendrickson and Robert Leland. A multilevel algorithm for partitioning graphs. Technical Report SAND93-1301, Sandia National Laboratories, 1993.

[9] A.K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988.

[10] R.A. Jarvis and E.A. Patrick. Clustering using a similarity measure based on shared nearest neighbors. *IEEE Transactions on Computers*, C-22(11), November 1973.

[11] G. Karypis, E.H. Han, and V. Kumar. Chameleon: A hierarchical clustering algorithm using dynamic modeling. Technical Report TR-99-???, Department of Computer Science, University of Minnesota, Minneapolis, 1999.

[12] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48(1):96–129, 1998. Also available on WWW at URL http://www.cs.umn.edu/~karypis.

[13] G. Karypis and V. Kumar. A fast and highly quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 1998 (to appear). Also available on WWW at URL http://www.cs.umn.edu/˜karypis. A short version appears in Intl. Conf. on Parallel Processing 1995.

[14] G. Karypis and V. Kumar. A fast and highly quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1), 1999. Also available on WWW at URL http://www.cs.umn.edu/˜karypis. A short version appears in Intl. Conf. on Parallel Processing 1995.

[15] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, Accepted for publication, 1997. Also available on WWW at URL http://www.cs.umn.edu/˜karypis.

[16] George Karypis, Rajat Aggarwal, Vipin Kumar, and Shashi Shekhar. Multilevel hypergraph partitioning: Application in vlsi domain. 1998 (to appear). A short version appears in the proceedings of DAC 1997.

[17] L. Kaufman and P.J. Rousseeuw. *Finding Groups in Data: an Introduction to Cluster Analysis*. John Wiley & Sons, 1990.

[18] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2):291–307, 1970.

[19] D. D. Lewis. Reuters-21578 text categorization test collection distribution 1.0. *http://www.research.att.com/ lewis*, 1999.

[20] M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[21] G. Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley, 1989.

[22] TREC. Text REtrieval conference. *http://trec.nist.gov*, 1999.

[23] T. Zhang, R. Ramakrishnan, and M. Linvy. Birch: an efficient data clustering method for large databases. In *Proc. of 1996 ACM-SIGMOD Int. Conf. on Management of Data*, Montreal, Quebec, 1996.