The Natural Communities Algorithm for Tracking the Evolution of Large Networks

John Hopcroft
Omar Khan
Brian Kulis
Bart Selman
Justin Yang

January 19, 2004

Abstract

Tracking the evolution of communities in large databases requires a clustering algorithm that is stable with respect to small changes in the input data. In this paper, we present the natural communities algorithm, a scalable, agglomerative clustering algorithm designed for large linked networks. We discuss how this algorithm has been applied to the NEC CiteSeer citation graph, which contains approximately 250,000 papers and 4.5 million citations. Using the algorithm, we track the evolution of computer science communities, identifying areas of strong growth as well as established communities. In the remainder of the paper, we discuss validation techniques for our natural communities concept, and future work.

1 Introduction

The emergence of many large linked data sources in recent years has resulted in widespread research for analysis of many aspects of such networks. Clustering, a fundamental problem in data mining, has been applied to many of these networks, such as the World Wide Web and citation graphs of scientific literature. Other analysis techniques for large networks include link analysis techniques for web search, including the HITS algorithm and PageRank [4, 6], as well as the analysis of “small-world” properties of the networks and the development of random graph models that attempt to exploit such properties.

One aspect of these networks that has only begun to be studied is their temporal nature. Nearly all large networks continue to evolve over time: pages are constantly added and deleted to the World Wide Web, and new publications cause the citation graph to grow. Most research done on studying large networks has focused only on snapshots of the network at one particular timestep. Clustering, for example, only considers a static view of a network. As a result,
clustering algorithms tend to be quite unstable with respect to changes in data. Even when data does not change, many clustering algorithms can produce vastly different results on the same data. Algorithms that find local optima, such as k-means, suffer from this problem for large networks and high-dimensional data. Agglomerative clustering algorithms are also known to be quite unstable for high-dimensional data; changes tend to propagate up the clustering tree, and we will illustrate this further in a subsequent section.

In order to be able to track how clusters change over time, we will require a stable clustering algorithm. Using the NEC CiteSeer citation graph, we are interested in such issues as determining the emergence of new scientific communities and predicting the final size of existing communities. Current clustering methods make these tasks impossible. At first glance, it may appear that hierarchical agglomerative clustering is a poor choice for tracking changes in networks. After all, given $n$ data points, an agglomerative clustering algorithm produces $n - 1$ clusters, and it is unclear which of these clusters correspond to recognizable topics and which are noise. However, we show that by running several agglomerative clusterings under small perturbations of the data, we are able to extract a subset of clusters that appear frequently in the clusterings. We thus exploit the instability of agglomerative clustering to discover the subset of communities that are the most robust. Furthermore, these communities correspond to coherent topics and can be used to track the growth of the network.

In this paper, we present results of the natural communities algorithm and discuss evidence suggesting that the natural communities concept is useful and valid. We first present related work. Then we introduce the algorithm and discuss the CiteSeer citation graph, the object of our experiments. We then show how the algorithm can be used to track changes in the CiteSeer graph. In the next section, we present our validation techniques. Finally, we provide a conclusion and potential future work.

2 Related Work

A significant portion of this paper comes from two earlier papers [?,?]. The first of these papers focuses on presenting the natural communities algorithm, and showing how it can be used to identify stable communities in large linked networks. The paper also discusses instability issues associated with hierarchical agglomerative clustering. The second of these papers focuses on using the natural communities algorithm to track the evolution of computer science using the CiteSeer database.

A large amount of work has been done prior to these papers for organizing large document collections. The first automated clustering of documents based solely on reference structure was done in [8], for the Science Citation Index. This paper used co-citation analysis, which is discussed further later in the paper. Some work has been done on clustering the CiteSeer database: one paper uses co-citation analysis [7] and another uses network flow [2].

Unlike in these papers, our focus has been on determining how such doc-
ument collections evolve over time. Our contribution in this realm has been the development of a stable clustering algorithm for tracking changes. Stability has received little attention in the data mining community. In [5], the authors discuss the stability properties of the popular link analysis algorithms HITS and PageRank. Using ideas from matrix perturbation theory, they stabilize the HITS algorithm.

NEED MORE HERE...

3 The CiteSeer Citation Graph

For our experiments, we use the NEC CiteSeer citation graph. The data that we used contains the citation information of computer science papers published through October 2001. As CiteSeer is automatically generated using a webcrawl of computer science sites, the database only contains the full text of papers which are available online. As a result, the text of nearly all papers appearing before 1990 do not appear in the database. However, the bibliographies for full-text papers are known, and hence the citation graph also contains all papers appearing in bibliographies of full-text papers. We refer to the papers whose full text is in the database as the core of the database. The papers in the core have references to other papers in the core, as well as non-core papers. There are approximately 250,000 core papers in our snapshot of CiteSeer, and 1.6 million non-core papers.

The structure of the citation graph is a directed network. There is a directed edge from paper \( p \) to paper \( q \) if \( p \) references \( q \). In the CiteSeer graph, there are approximately 4.5 million references total. Non-core papers have an out-degree of zero, since we have no reference information for them, and an in-degree of at least one.

Table 1 contains some statistics on the CiteSeer citation graph, and Figure 1 gives a pictorial representation of the papers in the database.

Due to the automatic generation of papers in the database, there are several errors and inaccuracies. Prior to clustering, we cleaned the graph by filtering out those papers that provide little information about the structure of the network. Specifically, we removed papers referenced by only one paper, and papers that reference fewer than five papers. This reduced the number of nodes in the network, which helped to speed up all computation done with the network.
4 Natural Communities Algorithm

In this section, we discuss the natural communities algorithm. We begin by discussing our measure of distance. We proceed to contrast different types of clustering, then we discuss an efficient implementation of our agglomerative clustering. We show how agglomerative clustering is instable and, finally, we discuss how these instabilities lead to our natural communities algorithm.

4.1 Measuring Distance

The first step in running hierarchical agglomerative clustering is to develop an appropriate distance measure. As our experiments are based on a citation graph, our distance measure will be depend only on the reference structure of the graph.

Two common techniques in measuring similarity between documents in bibliometrics are co-citation analysis and bibliographic coupling. In co-citation analysis, if we are given papers $p$ and $q$, we are interested in the papers that reference both $p$ and $q$, and we can define a distance measure based on the number of such papers. This method is useful after a significant amount of time has passed, and papers have garnered many citations. It is less useful for newer papers and papers that are referenced rarely or never. On the other hand, bibliographic coupling considers the papers that both $p$ and $q$ reference. Since co-citation takes a considerable amount of time to build up, and we are interested in tracking temporal changes in data over small intervals of time, bibliographic coupling makes more sense to use as the basis for a distance measure.

The assumption we have in using bibliographic coupling is that when authors write papers, they tend to reference other papers on a similar topic. In particular, authors tend to cite important papers that characterize the community that their paper is in. We believe that this will be useful in detecting emerging communities, as authors begin to co-cite new community-defining papers.
The next step is to define a distance measure between two papers. We believe that using any reasonable distance measure will lead to the discovery of natural communities, and as a result, we chose a standard distance function based on cosine similarity. Cosine similarity has been used extensively in data mining, especially for high-dimensional text data. For our purposes, we consider each paper \( p \) to be an \( n \)-dimensional vector, where \( n \) is the total number of papers in the citation graph. Entry \( i \) is 1 if \( p \) references paper \( i \), and 0 otherwise. Let us denote this vector as \( r_p \).

Using reference vectors \( r_p \) and \( r_q \), we can then measure the similarity between \( p \) and \( q \) using the cosine similarity between the papers. This is simply:

\[
\cos(r_p, r_q) = \frac{r_p \cdot r_q}{\|r_p\| \|r_q\|}
\]

This can easily be turned into a distance function. For example, we simply say that the distance between \( p \) and \( q \) is simply \( 1 - \cos(r_p, r_q) \).

In our algorithm, however, we need a notion of distance between clusters, which is a bit more complicated. In our scheme, we say that the reference vector of a cluster \( C \) is simply the normalized sum over all papers in \( C \) of the reference vectors (using the 2-norm), i.e.

\[
r_C = \frac{\sum_{p \in C} r_p}{\|\sum_{p \in C} r_p\|_2}
\]

As before, we can use the reference vectors to define the cosine similarity between clusters. We also define \( n_C \) as the number of papers in cluster \( C \). Then we say that the distance between two clusters \( C \) and \( C' \) is simply

\[
\text{dist}(C, C') = \sqrt{\frac{n_C n_{C'}}{n_C + n_{C'}} (1 - \cos(r_C, r_{C'}))}
\]

The square-root term is a scaling term used in the literature [?]. In our case, it leads to a layering effect of merges that promotes a balanced clustering tree. We have also explored other, less standard measures, as in [?]. In both cases, we chose a distance measure that produced a fairly balanced clustering tree. Distance measures such as Euclidean distance tended not to work well for our problem, because a “skinny” tree would be produced. In such a case, a single cluster quickly dominated and grabbed all papers. This was undesirable for producing clusters on single topics.

4.2 Types of Clustering

For the natural communities algorithm, we chose to use agglomerative clustering, though other options were possible. Another potential choice is to use an algorithm such as k-means. In k-means, we start with a random clustering of the data into \( k \) clusters. At each step, we compute the centers of the clusters, then reassign points to their closest centers. We continue this process until
convergence, thus leading to a local optima of the clustering objective function. For more information about k-means and its variants, see [3].

This method has several drawbacks. We do not know what $k$ to choose, and we would have to run many clusterings, with several values of $k$, to obtain a reasonable set of clusters to analyze. Furthermore, the local optima that are found by k-means and related algorithms suggest that it would be much more difficult to find stable clusters from the data. Finally, our experiments have shown that k-means tends to cluster around highly referenced papers and books. This can be problematic, since clusters often form around highly-referenced books on very general topics, which is quite undesirable.

We instead chose to use agglomerative clustering. We begin with each paper as a cluster of size one. At each step, we merge the two nearest clusters, and repeat until only one cluster remains (or, alternatively, until $k$ clusters remain, for some $k$). An advantage to this approach is that we will be able to extract a hierarchy of communities from the algorithm. Agglomerative clustering is by no means a perfect choice, as it can be quite slow. We can, however, speed up the algorithm, as the next section will discuss.

4.3 Efficient Clustering

We will briefly discuss how to exploit the sparse nature of the CiteSeer network in our agglomerative clustering algorithm.

The simplest implementation of agglomerative clustering requires $O(n^3)$ time, where $n$ is the number of papers. This is because, at each step, we would calculate the $O(n^2)$ pairwise distances, and choose the pair of clusters with the smallest distance. As there are $n$ merges total, we get a running time of $O(n^3)$. We speed this simple approach up in two ways.

First, we do not have to look at all pairwise points to determine the nearest neighbor for a given cluster. Suppose we would like to calculate the nearest neighbor to cluster $C$. For this cluster, we have a reference vector $r_C$ associated with it. The non-zero entries of this vector correspond to references made by papers in the cluster. The only potential nearest neighbors are those clusters that share non-zero reference vector entries. More specifically, we look at the papers corresponding to non-zero entries of $r_C$. Let the set of these papers be $P$. We then identify those clusters that reference papers in $P$. We have a data structure that maintains this information for easy access and updating. By identifying only the clusters that are potential nearest neighbors, we save considerably on computation time.

Second, after each merge, we do not have to recalculate all nearest neighbors, but just those that have been affected by the merge. Suppose two communities, $A$ and $B$, have just been merged together. Consider a third community $C$. The only distances that change are the following: $dist(A \cup B, C)$, $dist(A, C)$, and $dist(B, C)$. The second and the third become invalid, and only the first must be computed. Furthermore, we do not always have to recompute the nearest neighbor of $C$ to $A \cup B$. Let the distance from $A \cup B$ to its nearest neighbor be $d_m$. The nearest neighbor of $C$ changes only if its nearest neighbor was $A$ or $B$, and
or if its new nearest neighbor is $A \cup B$. We must only compute the distance of $C$ to $A \cup B$ if its previous distance was greater than $d_m$.

These observations lead to a very efficient implementation of the agglomerative clustering algorithm for the CiteSeer database.

### 4.4 Instabilities in Agglomerative Clustering

Let us briefly discuss the issue of instabilities in agglomerative clustering, as it will suggest a way to find stable clusters using agglomerative clustering.

We will show that, for the CiteSeer database, the level of instability present is quite dramatic. Instability is known to be present in agglomerative clustering, but our experiments show that it is impossible to track the evolution of clusters using only simple agglomerative clustering. We will see how to use instability to pinpoint stable clusters in the next section.

First, we must define an appropriate notion of comparison between two clusterings. Suppose we have two clustering trees $T$ and $T'$, each of which contain several clusters. For a fixed cluster $c$ in $T$, we would like to identify the best match for $c$ to the clusters in $T'$. Intuitively, we seek to obtain the cluster in $T'$ that has the highest overlap to $c$. We express that as follows: for $c \in T$, we define the bestmatch of $c$ to the clusters in tree $T'$ as:

$$
\text{bestmatch}(c, T') = \max_{c' \in T'} \left\{ \frac{|c \cap c'|}{\min(|c|, |c'|)} \right\}
$$

Hence, we compare $c$ to every cluster in $T'$, and determine its overlap. We take the cluster $c' \in T'$ that has the maximum overlap as the best match. The min condition in the equation captures the fact that a high overlap implies that the two clusters are roughly equal in size. For example, the root of $T'$ will contain all or nearly all of the papers in $c$, but the cluster sizes are vastly different, and so the root should not have a high overlap to $c$.

A simple experiment to test instabilities in agglomerative clustering on the CiteSeer database is to run two clusterings of the data, one on the full dataset, and on with a single randomly selected paper removed. For this experiment, the paper chosen was [1], which has 33 references and is cited by 2 papers. When running the clustering algorithm on the perturbed graph, we simply removed the outgoing edges of the removed node (the references of the paper). We kept the two incoming references to minimize the impact of removing the paper. This had the effect of moving the paper from the core to the non-core (i.e., if the full text of the paper had never been crawled).

We ran this experiment, which resulted in two clustering trees. Then, we determined, for each cluster from the full dataset, its best match to the clusters of the perturbed data. The results appear in Table 2. The first column gives the cluster size range; the second gives the number of clusters in each size range; the third column gives the number of clusters for which $\text{bestmatch}(c, T') = 1$; the fourth column gives the average best match value; and the final column gives the standard deviation of the average best match value.
<table>
<thead>
<tr>
<th>Size Range</th>
<th># clust.</th>
<th># full</th>
<th>match</th>
<th>std.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-2</td>
<td>9858</td>
<td>9858</td>
<td>100.0</td>
<td>0.0</td>
</tr>
<tr>
<td>3-3</td>
<td>10025</td>
<td>10025</td>
<td>100.0</td>
<td>0.0</td>
</tr>
<tr>
<td>4-4</td>
<td>9936</td>
<td>9935</td>
<td>100.0</td>
<td>0.5</td>
</tr>
<tr>
<td>5-5</td>
<td>10042</td>
<td>10041</td>
<td>100.0</td>
<td>0.2</td>
</tr>
<tr>
<td>6-12</td>
<td>38459</td>
<td>38543</td>
<td>100.0</td>
<td>0.5</td>
</tr>
<tr>
<td>13-24</td>
<td>14701</td>
<td>14685</td>
<td>99.9</td>
<td>1.1</td>
</tr>
<tr>
<td>25-48</td>
<td>7573</td>
<td>7560</td>
<td>99.9</td>
<td>1.2</td>
</tr>
<tr>
<td>49-99</td>
<td>4015</td>
<td>3993</td>
<td>99.8</td>
<td>2.7</td>
</tr>
<tr>
<td>100-200</td>
<td>1930</td>
<td>1892</td>
<td>99.3</td>
<td>5.9</td>
</tr>
<tr>
<td>201-400</td>
<td>993</td>
<td>956</td>
<td>99.0</td>
<td>5.7</td>
</tr>
<tr>
<td>401-800</td>
<td>466</td>
<td>423</td>
<td>97.5</td>
<td>9.2</td>
</tr>
<tr>
<td>801-1600</td>
<td>260</td>
<td>222</td>
<td>96.5</td>
<td>11.2</td>
</tr>
<tr>
<td>1601-3200</td>
<td>142</td>
<td>94</td>
<td>92.2</td>
<td>14.8</td>
</tr>
<tr>
<td>3201-6400</td>
<td>76</td>
<td>34</td>
<td>89.4</td>
<td>15.9</td>
</tr>
<tr>
<td>6401-12800</td>
<td>40</td>
<td>8</td>
<td>78.5</td>
<td>16.0</td>
</tr>
<tr>
<td>12801-25600</td>
<td>16</td>
<td>0</td>
<td>74.0</td>
<td>13.5</td>
</tr>
<tr>
<td>25601-51200</td>
<td>9</td>
<td>0</td>
<td>74.0</td>
<td>13.5</td>
</tr>
<tr>
<td>51201-102400</td>
<td>5</td>
<td>0</td>
<td>59.8</td>
<td>14.7</td>
</tr>
<tr>
<td>102401-204800</td>
<td>1</td>
<td>0</td>
<td>61.8</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Table 2: Instabilities due to randomly removing a single paper out of 250,000
Table 3: Best Matches over several runs with 5 percent removed

<table>
<thead>
<tr>
<th>Size Range</th>
<th># clust. BT</th>
<th>avg. match</th>
<th>std.</th>
</tr>
</thead>
<tbody>
<tr>
<td>100-400</td>
<td>2812</td>
<td>0.42</td>
<td>0.07</td>
</tr>
<tr>
<td>401-1600</td>
<td>558</td>
<td>0.41</td>
<td>0.07</td>
</tr>
<tr>
<td>1601-6400</td>
<td>149</td>
<td>0.38</td>
<td>0.07</td>
</tr>
<tr>
<td>6401-102400</td>
<td>46</td>
<td>0.40</td>
<td>0.08</td>
</tr>
</tbody>
</table>

As can easily be seen from the table, most small clusters are unaffected by the removal of a paper in the graph. Naturally, the formation of small clusters will be largely independent of the paper that was removed, since most papers do not share a distance with the removed paper. However, as we go up the clustering tree, we see a “cascading effect”, wherein changes propagate higher and higher up the tree. In fact, we see that clusters of size six thousand and larger are changed, on average, twenty percent or more.

It is quite surprising to see such a drastic change resulting from a small change to the graph. In fact, a similar, though not as pronounced, effect occurs when only removing a single edge from the graph. When removing a larger fraction of the graph - say, 5 percent of the core nodes - the trees are vastly different.

In a related experiment, we considered removing 5 percent of the core nodes and running 45 clusterings. We fix a base tree $T_1$, then determine the best match for each cluster in $T_1$ to all the clusters in trees $T_2$ to $T_{45}$.

Table 3 displays the results of this experiment. The second column gives the number of clusters in the base tree for each size range, the third column gives the average best match value over all 44 remaining trees, and the final column gives the standard deviation of the average best match value. As can easily be seen, with five percent of the papers removed, most clusters have a fairly low best match value between trees.

We now show how to exploit this instability to locate stable communities.

4.5 The Algorithm

The approach for our algorithm will be based on pinpointing the stable communities amid the majority of unstable communities. When running several clusterings of the CiteSeer graph, each of which has a small percentage of papers removed, we find that a few of the communities appear over and over again, largely unchanged. These are the communities which we seek to uncover.

Let us fix a perturbation value of 5 percent. We then generate a series of subgraphs

$$\{G_1, G_2, ..., G_n\}$$

where each $G_i$ is the subgraph induced by a random 95 percent subset of the core vertices in $G$. We then run our agglomerative clustering algorithm on these
graphs, producing a set of trees

\[ \mathbb{T} = \{T_1, T_2, ..., T_n\} \]

We say that our base tree is \( T_1 \). Then, we define a natural community as follows:

**Definition 1** A community \( c \) in base tree \( T_1 \) is natural iff in a fraction \( f \) of the clustering trees in \( \{T_2, ..., T_n\} \), the best match of \( c \) has a value greater than \( p \).

The values \( f \) and \( p \) are adjustable parameters, and depend on the application. For particularly robust communities, both \( p \) and \( f \) would be set close to 1.

Let us also define the strength of a community:

**Definition 2** The strength of a community \( c \) from base tree \( T_1 \) is the fraction of the clustering trees in \( \{T_2, ..., T_n\} \) for which \( c \) has a best match above \( p \).

Using this definition of strength, an equivalent definition for natural community would be a community which has a strength of at least \( f \). We will use this notion of strength in the validation section.

As an example, we ran an experiment with \( n = 45 \). We set \( f = 0.6 \) and we set \( p = 0.7 \) for smaller communities (those under size 1000) and \( p = 0.5 \) for larger communities. The change in \( p \) was due to the fact that larger communities tend to be less well-defined. In [?], we used a slightly different distance measure than the one discussed above, and we were able to identify larger natural communities without changing the \( p \) value. We considered communities greater than size 100.

See Table 4. This table gives the number of natural communities for different size ranges, along with average best match values, and standard deviation of the best match values. Out of a potential 3500 communities greater than size 100 in the base tree, the algorithm identified 170 natural communities that covered areas of computer science as well as math and physics. Verification techniques using journal data and keyword data indicated that these communities were quite coherent and - in the case of smaller communities - were on a single topic.

We can reduce the number of natural communities by merging equivalent natural communities. This is done in the following way: if two natural communities exist, and their overlap is greater than seventy percent, then we remove

<table>
<thead>
<tr>
<th>size range</th>
<th># natural</th>
<th>avg. bestmatch</th>
<th>std.</th>
</tr>
</thead>
<tbody>
<tr>
<td>100-400</td>
<td>116</td>
<td>0.74</td>
<td>0.05</td>
</tr>
<tr>
<td>401-1600</td>
<td>32</td>
<td>0.62</td>
<td>0.07</td>
</tr>
<tr>
<td>1601-6400</td>
<td>17</td>
<td>0.60</td>
<td>0.06</td>
</tr>
<tr>
<td>6401-102400</td>
<td>5</td>
<td>0.60</td>
<td>0.10</td>
</tr>
</tbody>
</table>

Table 4: Natural Communities in CiteSeer
the smaller of the two communities.

The choice of base tree clearly affects the resulting natural communities. A sharper view of the true natural communities in the data could be obtained by running this experiment with several different base trees, and unioning the sets of natural communities outputted by each run of the algorithm. If we assume that $f = 0.6$, then we may potentially miss 40 percent of the natural communities in the data using a single base tree. If instead we use five base trees, and merge new communities into our set of natural communities, the probability of missing a natural community is quite low: $A^5$, which is approximately one percent. In this scheme, we also must be sure to merge equivalent communities. This technique is discussed further in the validation section.

As another note, we ran this experiment with standard k-means to determine if natural communities would emerge when using k-means instead of agglomerative. To that end, we chose $k = 70$ to produce clusters of size 2000 - 3000. Our agglomerative algorithm discovered several natural communities in this range. However, when comparing two runs of k-means, we found that only 3 of 70 communities had an overlap above 50 percent, with only one of these having an overlap above 60 percent. It was clear that standard k-means would not produce natural communities, thus giving another reason for using agglomerative clustering to find stable communities.

5 Tracking Growth

We now focus on using the natural communities algorithm to track the evolution of the CiteSeer database. Our hope is that when comparing the CiteSeer database at different timesteps, the natural communities do not change dramatically. Moreover, we hope that the change in natural communities over time will give us insight into the growth of communities in computer science. We will show in this section that the natural community notion satisfies both criteria, and is therefore a good choice for tracking the evolution of large networks.

5.1 Method

To track changes in computer science, we used two subsets of the CiteSeer database. The first subset - which we will refer to as the 1998 subset - contains the core papers from CiteSeer from 1990 to 1998, along with all reference information (i.e. the non-core papers are unaffected). The second subset - which we will refer to as the 2001 subset - contains all core papers from CiteSeer from 1990 to 2001, with all associated reference information.

We ran the natural communities algorithm on the 2001 data set. The result was approximately 100 natural communities of size 100-350. We chose a subset of 18 of these communities (or, about 20 percent of the communities) for careful analysis. This chosen subset was representative of the rest of the 100 natural communities in terms of size and year distribution. There were 3200 papers total from these 18 communities.
Using these papers, which we denote as $P_{01}$, we create a citation subgraph using the reference information of the papers in $P_{01}$; hence, the citation graph contained the papers in $P_{01}$ as well as the papers referenced by the papers in $P_{01}$. To remove noise, we then eliminated core papers in the citation graph that referenced fewer than five papers, and any non-core papers referenced only once.

This experiment was repeated on the set of paper $P_{98}$, which is the subset of papers in $P_{01}$ that were published before 1999. Once again, we created a citation graph and eliminated noise from it.

### 5.2 Tracking Results

Using these two citation graphs, we determined the natural communities in each of the graphs. This was done by considering $n = 10$ runs for each, with $f = p = 0.8$. We considered natural communities greater than size 75 for analysis.

Once this had been done, we compared the natural communities from the 1998 data to the natural communities from the 2001 data. We found that most communities in the 1998 data had a strong match with a corresponding community in the 2001 data, and vice versa.

Based on the growth of the natural communities between 1998 and 2001, we classified each of the natural communities as being either established or emerging.

Table 5 gives a summary for the established natural communities. The community names were determined by the most frequently-occurring words in the titles of the papers in the community. The third and fourth column give the size from the 1998 data and the 2001 data, respectively. The final column gives the percentage of papers from the 2001 community that were published after 1998 (the growth rate of the community). The communities presented in the table are sorted by their growth rate. It is easy to see that there is significant variety in the growth rates of the communities: active communities

<table>
<thead>
<tr>
<th>Topic</th>
<th>Size '98</th>
<th>Size '01</th>
<th>% '01</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digital Watermarking</td>
<td>97</td>
<td>172</td>
<td>35.5</td>
</tr>
<tr>
<td>Data Mining and Association Rules</td>
<td>78</td>
<td>128</td>
<td>25.0</td>
</tr>
<tr>
<td>Game Search Trees and AI</td>
<td>161</td>
<td>172</td>
<td>8.7</td>
</tr>
<tr>
<td>Network Traffic Control</td>
<td>237</td>
<td>258</td>
<td>8.5</td>
</tr>
<tr>
<td>Crash Recovery for Distributed Systems</td>
<td>139</td>
<td>151</td>
<td>7.3</td>
</tr>
<tr>
<td>Asynchronous Circuit Designed Verification</td>
<td>231</td>
<td>244</td>
<td>6.6</td>
</tr>
<tr>
<td>Synchronous and Asynchronous Systems</td>
<td>203</td>
<td>219</td>
<td>6.4</td>
</tr>
<tr>
<td>Complexity Theory: Enumerability and Querying</td>
<td>78</td>
<td>84</td>
<td>6.0</td>
</tr>
<tr>
<td>Query Optimization for Parallel DBs</td>
<td>119</td>
<td>125</td>
<td>4.0</td>
</tr>
<tr>
<td>Fractal Image Coding and Compression</td>
<td>86</td>
<td>89</td>
<td>2.2</td>
</tr>
</tbody>
</table>

Table 5: Established Natural Communities
such as digital watermarking and data mining have very large growth rates, while communities such as complexity theory and fractal image coding have very small growth rates.

The information in Table 6 summarizes the emerging communities that we found. These communities, which are the quantum complexity community and the wireless network community, are discussed in more detail in [?], where a careful analysis of the emerging communities was performed.

### Table 6: Emerging Natural Communities

<table>
<thead>
<tr>
<th>Community 1998</th>
<th>Size</th>
<th>Community 2001</th>
<th>Size</th>
<th>% 2001</th>
</tr>
</thead>
<tbody>
<tr>
<td>Networking</td>
<td>237 + 130</td>
<td>Ad hoc / Wireless Networks</td>
<td>130</td>
<td>49.2</td>
</tr>
<tr>
<td>Quantum Complexity</td>
<td>96</td>
<td>Quantum Computation</td>
<td>140</td>
<td>30.0</td>
</tr>
<tr>
<td>Subcomm: Quantum Complexity</td>
<td>82</td>
<td>Subcomm: Quantum Algs. and Comm.</td>
<td>38</td>
<td>76.3</td>
</tr>
</tbody>
</table>

6 Validation of the Algorithm

In this section, we provide additional evidence of the validity of our natural community algorithm, through data and experiments that were run to determine the robustness of the natural community concept.

#### 6.1 Best Matches

For the following experiments, $f$ was set to be .70, and $p$ was set to be .70 for communities under size 1000, and .60 for larger communities.

In the natural communities algorithm, the parameter $f$ determines in which fraction of the trees $T_i \in T$ a cluster in the base tree must have a best match above $p$. It is natural to ask the following question: for a natural community $c$, how high are the best matches to those trees in which $c$ has a best match below $p$? We should hope that the best matches are reasonably close to $p$. In fact, after running this experiment, we find that the average value of these best matches is approximately .64. This is significantly higher than average best matches (see Table 3 as a comparison). Similarly, only 0.3 percent of the best matches fall below .40. In general, we find that natural communities appear in all trees, and usually are close to the $p$ threshold. [MORE DATA? TABLE OR GRAPH PERHAPS?]

A related question concerns how well non-base tree communities overlap. Consider a natural community $c$. If $c$ has a best match $c_i$ above $p$ in some tree $T_i$ and a best match $c_j$ above $p$ in some tree $T_j$, how well does the community $c_i$ match $c_j$? If the natural community notion is strong, we expect that these communities will match up quite well, though we cannot be guaranteed that they will.
We examined this question using the $p$ and $f$ values from above. After running the experiment several times, we found that in approximately 85 percent of the $\{c_i, c_j\}$ pairs, $c_i$ matched $c_j$ by at least our threshold $p$ (note that the pair $\{c_i, c_j\}$ is different from the pair $\{c_j, c_i\}$, as order matters for matching). The remaining 15 percent fall short of the $p$ threshold value, though only by 4 percent on average. [MORE DATA?]

### 6.2 Multiple Runs of the Algorithm

As randomness is involved in the natural community algorithm, we would hope that multiple runs of the algorithm would produce nearly the same final natural communities.

To do this, we ran the natural communities algorithm twice, each with 100 trees using the $p$ and $f$ values from the previous section. We say that clusters $c_i$ and $c_j$, where $c_i$ is a natural community from the first run and $c_j$ is a natural community from the second run, form a two-way match if the best match of $c_i$ is $c_j$ and the best match of $c_j$ is $c_i$. From this experiment, it was observed that approximately 80 percent of the natural communities formed a two-way match. We then sought to determine why the remaining 20 percent did not form two-way matches.

To do this, we determined the average strength of the natural communities that had no two-way matches. In our experiment, $f$ was set to be .70, and we found that the average strength of the communities in question was .74. Furthermore, the strength of the best matches of these natural communities averaged .63. Hence, we observe that these communities are close to the filter value for being called natural. Such threshold communities are to be expected, in cases where communities' strength hover around $f$. [WHAT IS THE AVERAGE STRENGTH OF THE COMMUNITIES THAT HAVE A TWO-WAY MATCH??]

Table 7 provides additional data regarding the strength of the communities from the two natural community runs. The total number of natural communities from the first run was 158, while the total number of natural communities from the second run was 150. Though the distribution of strength is quite even across the two experiments, a natural community in one set of trees, and its corresponding natural community from another set of trees may not have the

<table>
<thead>
<tr>
<th>Strength</th>
<th># Comms, Exp. 1</th>
<th># Comms, Exp. 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>70 - 75</td>
<td>55</td>
<td>47</td>
</tr>
<tr>
<td>76 - 80</td>
<td>35</td>
<td>35</td>
</tr>
<tr>
<td>81 - 85</td>
<td>17</td>
<td>21</td>
</tr>
<tr>
<td>86 - 90</td>
<td>22</td>
<td>16</td>
</tr>
<tr>
<td>91 - 100</td>
<td>29</td>
<td>31</td>
</tr>
</tbody>
</table>

Table 7: Strength of Natural Communities over Two Runs
same strength. We observed that some natural communities’ strength differ by quite a large margin.

We postulate that this is due to the fact that, in these experiments, a single base tree was used. Naturally, when we use just one base tree, the strengths of natural communities found are bound to differ from the strengths of the natural communities found by another base tree. [REPEAT EXPERIMENT WITH MULTIPLE BASE TREES]

6.3 Core Communities

Given a natural community, it is an interesting question to ask whether there is a core of papers that appears in the corresponding communities across each of tree in \{T_2,...,T_n\}. Given that each tree was generated by a random 95 percent subset of all the papers, if we have one hundred trees, then the probability that any given paper even exists in all hundred subsets is low: 0.95^{100} = .0059, or under one percent.

A different approach to discovering the core papers in a community is to identify papers that appear in high percentage of the clusters. For our experiment, we sought to identify papers that appeared in 70 percent of the corresponding clusters. We note that the probability that a paper will appear in 70 percent of the subsets is about 96 percent, when using 100 trees. Thus, if it appears in 70 percent of the communities, it appears nearly each time it is present in the subset of papers.

We expect that identifying a core set of papers from a natural community will eliminate noise from a cluster, and give us a more accurate representation of the community. We also expect that the cores from multiple runs will match better than just the natural communities across multiple runs.

After identifying the core communities from the sets of natural communities from two experimental runs, we analyzed the matches of these communities. We found the matches to be consistently high at an average of 93.6 percent. This is significantly higher (more than twenty percent) than the match of corresponding communities using only the standard natural communities. Hence, the concept of a core appears to be a useful one.

[ADDITIONAL CORE VALIDTION: STRENGTH OF THE CORE SECTION (.DOC Pg 8-9)]

We see that the natural community algorithm, along with the notion of a core community, allows us to identify particularly robust communities from the CiteSeer network. As CiteSeer contains significant noise, it is an impressive achievement to be able to extract such a strong set of communities.

6.4 Temporal Analysis

TO BE DONE...

OTHER: WHAT ABOUT VALIDATION OF COHERENCE USING JOURNAL OR KEYWORD DATA? WE DO NOT DISCUSS THIS IN THE PAPER AT ALL.
7 Conclusion

In this paper, we have presented an algorithm for finding natural communities in large linked networks. This algorithm uses the instability present in hierarchical agglomerative clustering to identify clusters that consistently appear despite perturbations in the data. These clusters eliminate the noise present in huge networks to reveal the true structure of the data.

We discussed various aspects of the algorithm, such as the choice in distance function, methods for implementing efficient agglomerative clustering for networks, and experimental results of the algorithm. We also gave a comparison to iterative clustering algorithms such as k-means, explaining why such algorithms are unsuitable for our task.

We have shown how these natural communities can be used to track the growth of networks over time. Specifically, we were able to track the evolution of some computer science communities using the NEC CiteSeer database. The natural communities algorithm was able to reveal emerging communities, as well as the growth of various established communities within computer science.

Finally, we have attempted to provide a strong verification of the natural community concept through a series of careful experiments regarding several aspects of the natural communities algorithm. These experiments proved to strengthen the notion of natural community. Future work includes testing the algorithm on other networks, as well as incorporating more complex networks, such as those with edge weights, into the algorithm.

References


