Abstract

Data in the world, and more specifically on the Internet is growing to massive sizes. In order to make this information more useful, it must first be more accessible. The INEX Initiative competition is aimed with the goal of identifying and comparing methodologies for categorizing information into clusters. The competition will be run on 60 gigabytes of data from Wikipedia, with the ultimate goal of accurately categorizing and clustering in order to reduce search time through this data.

In this work, we construct a clustering algorithm based of the link structure of a subset of underlying pages. The resulting webgraph is pruned using a max flow min cut algorithm[10, 2, 8] which is initially seeded using different heuristics. We compare search space reduction results and construct a visualization of the clustered documents. We were able to generate clusters on the INEX data set as well as visualization of clustered data on several different datasets.

1 Introduction

The ultimate goal of the competition is to optimize search times based off of search keywords by grouping pages into some form of categorical clusters. However, an important side-note is that the contest guidelines also clearly stipulate that in addition to looking for a best-off solution, they’re also interested in a comparison between solutions that are derived from standard evaluation criteria such as entropy, F-score, or Normalized Mutual Information[1]. Unfortunately, as the competition finished as of the time this work is completed, these evaluation criteria were not available to us. As such, we proceed to evaluate our created clusters through other methods such as search space reduction, and utilization of a graph visualization process, Walrus[3].

2 Background

The general focus of the INEX competition is to take a huge, unwieldy data set and pare it down so that the most likely pieces to have the sought information will be searched first. Effectively, the goal is to create a method that will decrease the required search time on a given data set. With this in mind, our approach first and foremost considers the attributes of the data set.

Each node in our data set is representative of a Wikipedia page. Through general perusal of Wikipedia, it becomes clear that in general, the pages are fairly well connected throughout the site. As such, many (if not most) related topics have multiple paths to each other through the related web structure.
Taking this into consideration, we attempt to look at that underlying link structure, and create our clusters of information based off these attributes.

3 Addressing the Problem

The main focus of our solution is on the task of providing an effective clustering methodology for the INEX data set of Wikipedia pages. While much information for all of the Wikipedia documents in the dataset was supplied, we will be relying solely on the graph represented by the underlying link structure of the documents. We intended to discover sub-communities within the graph through the use of Max-Flow Community Discovery techniques. In short, given a sample of seed documents, for which we believe a community is centered on, we attempt to identify all other documents connected to the seeds such that we maximize the notion of information flow through the subgraph, or community. In order to determine our community seeds we will make use of couple heuristics to provide us with rough estimate of existing document clusters. Ultimately we were able to achieve results using just the Max-Flow technique. In the following section we describe the concepts of Max Flow, Min Cut and Community Discovery.

3.1 Max Flow, Min Cut

In graph theory, the Max Flow, Min Cut problem refers to an analysis of a graph and it’s capacities along different paths [10, 2]. For our purposes, a general understanding of the terminology becomes necessary to explain the utility of using a Max Flow or Min Cut algorithm:

Edge Capacity - This would be similar to an edge weight in a weighted, directed graph. In this context however, it represents the “amount” of information that can be pushed across a node. Think of this as the bandwidth across a given edge, or the amount of data that can be pushed across the edge at the same time.

Flow - The sum of the minimum capacity of all paths from one node to another node.

Using these attributes of a graph, we can cut the graph into two disjoint sections via a Max-Flow Min-Cut algorithm.

According to the Max-flow Min-cut theorem [10], in order to separate two nodes from each other, the minimum capacity that needs to be removed from the network is equivalent to the maximum amount of flow passing from the source to the sink.

Why is this useful? If we can identify two nodes in the graph as belonging to different clusters, then we hypothesize that the underlying link structure of the graph will closely resemble these clusters. That is, given a set of Wikipedia pages in the form of a directed graph where the nodes are individual web pages, the directed edges are links from one page to another, and where the capacity of each link is equivalent, that the flow between two nodes in the same cluster should be larger than the flow between two nodes in separate clusters.

Using the Ford-Fulkerson algorithm [8] on a specific source and sink node within a graph, the result is the separating of the graph into two sections while guaranteeing that the link removal in the graph corresponds to the minimal possible capacity removal. If the supposition that the link structure accurately corresponds to logical clusters is correct, then since application of the Ford-Fulkerson algorithm to our graph would effectively separate two nodes with minimal effect on the link structure (as shown in Fig-
Figure 1: Graph cut using Ford-Fulkerson Algorithm

ure 1), we can also say that it effectively separates our clusters at an appropriate boundary.

3.2 Max-Flow Community Discovery

Given a particular Graph of linked web pages we define a maximum flow community as a collection of web pages. Additionally, Each page in the community has more links to other pages within the community than it does to pages which are not in the community [9]. Or more precisely, Let:

\[ V = \text{the set of all Web Pages} \]
\[ E = \text{the set of all Hyperlinks,}\{v1, v2\} \]
\[ G = (V, E), \text{the Web Graph} \]

Let Community = \( C \subset V \)
Let \( e_{cc} = \{e \in E \mid c \in C, v1 = c \lor v2 \in C\} \)
Let \( e_{cv} = \{e \in E \mid c \in C, v1 = c \lor v2 \notin C\} \)

\( C \) is a Max-Flow Community IFF
\[ \forall c \subset C, |e_{cc}| > |e_{cv}| \]

The Max-Flow Community discovery algorithm begins with a set of seed pages which we believe to represent a community. These seeds are a subset of the set of all pages. We then go through the graph making each link bidirectional with a capacity equal to the total number of seed pages we are using. Next, an artificial source page is added to the graph, whereby it is unidirectionally linked to all seed pages with an infinite capacity. Similarly, an artificial sink page is added to the graph, whereby it is unidirectionally linked to all non-seed pages with a unit capacity. The heuristic for choosing the capacity of each link ensures that any cuts to the graph will be to edges contained in the original graph [7].

Figure 2: Max Flow Community Discovery

To find the community from the constructed graph, we hand the seeds and the modified webgraph over to a Max Flow, Min Cut algorithm. This algorithm returns a sub-graph, whereby all pages accessible from the source node comprise the community. We can then iteratively recompute this community by extracting out a number of pages to add to set of seed pages. The number of pages and which pages to choose directly affect how the next iteration regenerates the community.

4 Implementation

We implemented our system using Java as the language with Eclipse as the IDE. The implementation is explained in three parts, parsing of the data set, running of the clustering algorithms and finally the output and visualization. The wikipedia data from INEX was comprised of links, tags, trees, entities, bi-
grams and stems. An object Document was created that contained variables for each of the data sections. Figure 3 shows the flow of the program.

4.1 Parsing the Data

We created a class called JJEParse with static methods to parse each section of the data set. While we implemented the parsing ability for each of these data sections, we ended up only needing the parsed links information to run the clustering algorithms.

The links came as newline separated lists of numbers. The first number was the document id and the rest of the numbers on a line were links to other document ids.

**Example of Raw Links** - 165000 8192 21882 455547 590699 34844 23301 30680 8210131

The parsing was accomplished with a standard scanner looping through lines and building a Graph object. The Graph object is a Hashmap with the document ids as the keys, linked to another hashmap with link ids as keys to the capacities of the links. The capacities are all initially set to zero. Once the graph is created, the parsing is done.

While the data was being parsed a links structure was created to keep track how many links each document has. This allows the seeds to be chosen from the documents with the most links and therefore probably the center of a community. The particular structure is a list of a list where each index of the first list returns document id’s that have the index’s number of links.

4.2 Initializing Seeds

After the graph is constructed the seed documents are selected. The link structure allows us to easily take a certain percentage of top nodes with the most links for use as possible seeds. Our hypothesis is that a node with the most links represents a good position for the source of a cluster given that webpages that link together often have similar material. The top linked node is then chosen as the first seed. For increased accuracy from seeds, we also implemented the ability to add seeds based on the links of the top seed to create a community of seeds. Currently we only use the top linked nodes and not their links as seeds. The find cluster algorithm is then run on these top linked seeds.
4.3 Finding a Community

Our implementation of finding a community has two versions. Our initial implementation followed the algorithm set forth by [9]. Unfortunately we ran into problems with this process. On this data, we found that our implementation almost always ended up cutting the source from the rest of the graph. After sitting down and mapping it out we discovered that this appeared to make sense with our understanding of the algorithm. This lead us to the possibilities that either a) our understanding was flawed, or b) the algorithm was a poor set for this type of data. Due to time constraints, we did not investigate this further, and instead abandoned the algorithm for a home grown approach. We begin by discussing our initial implementation of Max Flow Community Discover, and then present our implementation of discovering communities for Hub Nodes.

4.3.1 Max Flow Community Discovery

The Max Flow Community Discovery was implemented as a looping process initialized with a set of community seeds. For each seed the process attempts to iteratively build a community. Once the list of community seeds is exhausted the process exits.

The iterative process of building a community around a community seed has a heuristic associated with it for determining when it is done building a particular community. Given a community seed containing a number of documents it initializes the web graph as outlined previously and hands that graph over to the Max Flow, Min Cut algorithm. What is returned is the documents associated with the community seed. The iterative process then chooses and adds a percent of the documents in the community to the seed community which are most closely linked to the documents in the seed community. This percentage is variable and was tuned by us to determine a value that was affective for community discovery. We settled on a relatively small percentage or five. We then choose to add only documents close to the existing seed community since these better reflect documents in the community than documents on the outskirts. Adding edge documents might result in documents that would normally be cut, being added to the community.

After updating the seed community the process iterates again, constructing the webgraph, finding a community, and updating the seeds. The iteration process continues until either the size of the community has not grown relative to a threshold value. This threshold value is some medium sized percentage, of around 35 percent, which we can tune discover a value that is affective. The idea behind this is that, we can stop recalculating our community once we observe that adding additional seeds is not significantly changing the community. This value tends to be high because, generally, adding documents to the seed community affectively increases the flow being able to be sent across the web graph. This causes a significant amount of documents to be added to the resulting community compared to the small number of documents added to the seed. Thus, the community must drastically grow by adding new seeds to the community in order to continue iterating.

4.3.2 Hub Node Community Discover

As stated previously, our initial implementation was not succesful in finding communities. Hence, we abandoned the original approach in favor of our own. In this approach we use nodes with the highest number of outgoing links as the basis for selecting where communities resided.

Our implementation involves iterating over the list of documents, sorted by their number of outgoing links. A chosen document is used as the source for
the Max Flow Min Cut algorithm. The returned list of documents still attached to the source is saved as a cluster. We would sure to remove from this cluster, any document that appeared in a previous cluster. This process then loops, grabbing subsequent documents from the document list, and using each as the source. The process continues until every document appears in a cluster, or more simply, there is nothing left to cluster.

4.4 Max Flow, Min Cut

Our implementation of the Max-flow, Min-cut algorithm uses the Edmond-Karp implementation of the Ford-Fulkerson max-flow algorithm [8, 6]. This breaks down to utilization of a breadth-first search when determining the flows across a graph. Once the max flow has been determined, we take all edges with maxed capacity, and attempt to find some combination of these edges that satisfies two conditions:

- The sum of the edges’ capacities is equal to the max-flow of the graph
- Upon removal of the edges, there is not longer a path from the source to the sink node

Our algorithm currently searches for edges satisfying this condition using the same breadth-first search algorithm. After finding an appropriate set of edges, we find all nodes still connected to the source, and list that as the cluster.

5 Visualization

In order to better see the overall link structure of the web graph, we make use of an Open Source project called Walrus[3]. Walrus is an interactive 3D visualization tool that works on large directed graphs[3]. In order to visualize the data properly, we created a method for outputing our internal representation of the web graph as well as our cluster. The graph is represented in the file as a collection of directed edges. We made use of Walrus’s attribute specification system to identify cluster by color. In order to tell Walrus which nodes in the graph belong to which cluster, we merely had to specify an extra attribute identifying the color of each node. Thus, nodes of the same color belong to the same cluster. While the implementation of the Visualization output may seem trivial, it greatly helped us in our analysis of
the overall quality of our community identification process.

We used two different datasets to see if how accurate our clusters were, both of which were obtained from Walrus sample visualization files[3]. The first dataset we used was a representation of the directory structure of the source code for the Walrus project. This dataset has an inherently hierarchical structure, as files are clustered into directories, which also contain subdirectories of files, etc. We were pleasantly surprised to see that our clustering approach was able to accurately identify each directory of files as a cluster. As you can clearly see in Figure 4, each differently colored cluster of nodes represent a logical directory of files.

The second file that we used to test our clustering algorithm was a graph designed to look like a palm tree. This graph, while hierarchical in nature, has very few nodes that have a high degree of interlinking and is thus intuitively tough to cluster. As you can see in Figure 5, our algorithm produced very poor clusters on this data, for which most nodes were only linked to a small subset of nodes. This, however, makes sense from our algorithms perspective since we intended to only cluster nodes together which had a high degree of interlinking.

6 Results & Conclusion

Our tests were run on a machine using 3 GB of RAM, with a Dual Core Pentium 2.0 Ghz processor. Running our algorithm on the 54,000 nodes and 15 million links from the INEX data, we found a running time of roughly 6 hours to provide a cluster of between 2,000 - 2,500 nodes. Thus, determining all top level nodes in the INEX nodes requires a running time of roughly a week of computation to determine all top level clusters. Once determined, it would be possible to then sub-cluster the results with slightly faster running times for each level.

From this, along with a comparison of the project being run concurrently [4], we can determine that our results produce clusters of roughly similar cluster sizes. Unfortunately, these results are clearly not optimal with respect to the expectations of INEX.

However, we do believe there has been value added, at the very least in the process followed by this project, as well as in the identification of problems with applying community finding and clustering algorithms to this particular domain.

7 Future Work

The next step in the process is to attempt to matchup the INEX grading script with the current output. In its current form, our code will need some significant tweaking to get the output into a satisfactory form. Once this has completed, the results should be fairly easy to obtain. Given this, a more full evaluation of the current algorithm can be done.

The most immediate place where the actual algorithm can be extended and improved is in the seed selection for both the source and the sink. With our initial attempt at community discovery on top of cluster discovery failing, we had to fall back to a relatively simple heuristic of number of outgoing links and choosing a random sink. More than likely, simply improving the selection process for where we’re attempting to cluster could provide very large increases into the effectiveness of the actual clustering.

Additionally, getting community discovery to work in concert with the clustering could yield positive results. That was our initial impression, but our attempt failed. Picking this thread up and attempting to more clearly identify why it failed, and remedy the problem could prove very beneficial to the algorithm.

Finally, one of the most limiting aspects of our algorithm is the time constraint it imposes. As our re-
sults discussed, we find a running time of roughly 6 hours to find a single cluster of around 2,000 - 2,500 nodes out of a data set of 54,000 nodes with 15 million links. Running this on the INEX data to convergence, with full clustering would require roughly a week of computation. It may be possible to optimize our clustering algorithm to reduce this time period.

References


