Exploring Webgraph Compression with SUBDUE

Peter Webb
Ryan Wisnesky
CS 222

January 2007

Introduction

Storing a copy of the world wide web is beyond the capability of most organizations. However, it is often the case that when we are using the web, we are interested only in some particular subset of the information it contains. If we could capture only the information we are genuinely interested in, it might be feasible to store a copy of the web. Of course, different uses of the web call for different kinds of information to be stored, but regardless of the application, we are, in essence, *compressing* the web. Our paper presents a framework wherein different uses of compression can be viewed as variations on a single, generic technique, and demonstrates how a particular technology – SUBDUE – can be used to efficiently implement this technique. It is our hope that by capturing various uses of webgraph compression within a single framework, someone who wants to capture and compress specific information from the web need do no more than state what he wants to capture and push a button.

Our approach to webgraph compression is based on removing nodes and redistributing the information contained in the nodes to the nodes that remain in the compressed graph. By judiciously choosing which nodes to remove and what to do with the removed information, many different compression schemes can be achieved that are suitable for a diverse array of applications. For instance,

- A mathematician might want to run PageRank experiments on compressed webgraphs that have similar topological properties to the original web.
- An archivist might want to use compressed webgraphs that deliver the same top N search results across some set of queries to understand what the web looked like in the past.
- A lawyer might want to examine compressed webgraphs that only contain websites with copyrighted words to settle a copyright dispute.

In all of these scenarios, it is impractical to store a complete snapshot of the web. But in all cases, the use of the compressed webgraph is defined by two properties: what nodes to remove and what to do with the information that is removed. The mathematician would want to remove nodes with low PageRank and distribute the removed rank so as to avoid needing to recalculate the PageRank of the new graph; the archivist would want to remove nodes that never appear in a search and simply discard information contained in those nodes; the lawyer would want the compressed graph to contain one node per company that contains all uses of copyrighted words. In all cases, these uses of compression are defined by what to remove and what to do with what is removed.

In section 2, we investigate how to use a tool called SUBDUE to define sets of nodes to remove based on the concept of *labeling*, which we believe is useful for many compression scenarios. In section 3, we investigate a particular approach to manipulating the information being removed that allows for scalable compression. We have created a software framework which allows us to run experiments across a large range of compression scenarios; this work is described in section 4. Finally, we have examined how well our framework performs on a single application: the archival scenario described above. This work is described in section 5.

Related Work

Our work can be viewed as an application of SUBDUE to webgraph compression, an area where related tools have had some success. SUBDUE is based on node labeling schemes and pattern matching based on graph grammars; while graph compression of this kind is an established research area, it is only within the last five years that various groups have attempted to apply these techniques to webgraphs.

SUBDUE itself has been used to examine various structural properties of the web [9] and these properties have been used to implement higher-order web queries [4]. Graph grammars similar to those SUBDUE can produce have been used to model webgraph evolution [10]. Perhaps most importantly, compression of network graphs using labeling information in a way similar to SUBDUE has recently been demonstrated [6].

1 Webgraphs, Matching, and Compression

We define a *webgraph* to be a tuple (N, E) where N is a set of *nodes* and E is a set of *edges*, where a node is an ordered triple with three components: the node's *PageRank*, a real number $0 < r \leq 1$, the node's *identifier*, a set of strings, and the node's *wordlist*, also a set of strings. For example, if `www.tiny.com` has PageRank .1 and contains only two words, "cat" and "mouse", then this node could be represented as (.1, { `www.tiny.com` }, { `cat`, `mouse` }). An edge is simply a directed association between two nodes. Our particular definition of webgraph is designed to be an intermediate form between a fully general account of graphs

and graphs of the kind that the SUBDUE tool uses, with an eye toward being useful in our software framework and for our experiments. For different applications, identifiers, wordlists, and even PageRank itself could be replaced by other concepts, but the overall framework for compression would remain essentially the same.

Our definition of webgraphs suggests a number of degrees of freedom which may vary independently by desired application of compression:

- What nodes should be removed? Call this set r. (given by the *matching function*)
- What happens to the edges attached the nodes in r? (given by the *edge function*)
- What happens to the PageRank of the nodes in r? (given by the *rank function*)
- What happens to the wordlists of the nodes in r? (given by the *list function*)
- What happens to the identifiers of the nodes in r? (given by the *identifier function*)

By choosing appropriate functions, we may defined suitable compression functions for a variety of applications. For instance, if we would like to compress the web so that it contains only "important" websites, we can choose these functions as follows:

- matching function: remove all nodes with PageRank in the lower half of the PageRank distribution
- edge function: simply remove all connections from the nodes that are removed
- rank function: distribute the removed PageRank uniformly across the graph (alternatively, use a strategy given in [2])
- list function: the wordlists of the removed nodes should be discarded
- identifier function: the identifiers of the removed nodes should be discarded

The result of compression will be a webgraph that contains only important websites and whose PageRank distribution matches approximately what would be obtained by running PageRank on the newly compressed graph.

2 SUBDUE as a Matching Function

To compress a webgraph, we need to know what set of nodes should be removed; determining this set is the focus of this section. What to do once we've found that set is discussed in the next section.

Because the set of nodes to remove is completely application specific, we would like to have a general purpose mechanism for searching for nodes to remove that satisfy a certain

property. That way, we can set some parameters in a general function and ask the function to search the webgraph. Without this generic ability, we are forced to write specific matchers for every application. In general, it is non trivial to write a matching function to find sets satisfying arbitrary properties.

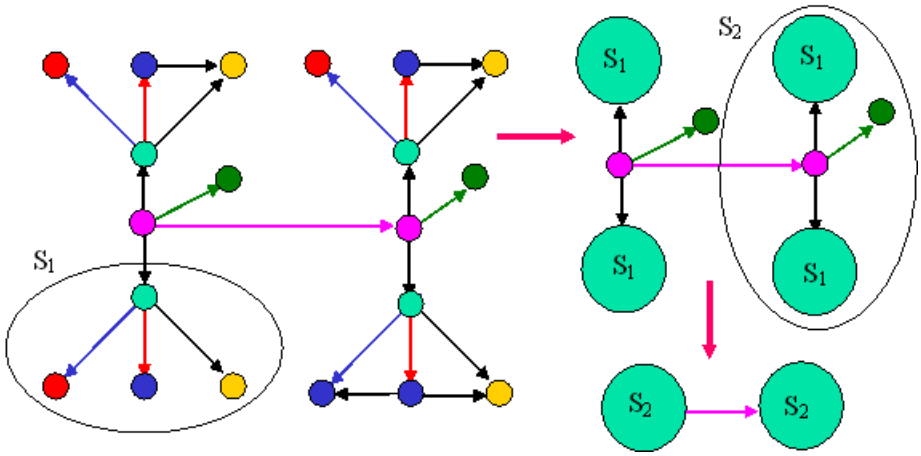
SUBDUE is a program that searches for isomorphic (or near isomorphic) subgraphs in a given graph by utilizing a user defined label for each node [3]. Associated with each node given to SUBDUE is a *label*, which is simply an element in an arbitrary domain that admits an equality comparison. SUBDUE efficiently searches for recurring patterns among nodes by examining the relationship of node and label configurations to each other and attempts to identify patterns of nodes that would minimize the amount of information needed to describe the graph that would result if the patterns are all removed. (The particular process by which SUBDUE operates is known as *minimum description length* matching.) By choosing an appropriate labeling for our graph, we may simply call on SUBDUE to find suitable sets of nodes to remove. For instance,

- a mathematician interested in preserving the pagerank distribution among highly ranked nodes could give every node with a low pagerank label α and every node with a high pagerank label β. SUBDUE would then search for patterns among and between these labels in an attempt to extract the most common recurring subgraph that it can identify; these subgraphs can then potentially be compressed. Alternatively, one could easily use SUBDUE to identify fully connected subgraphs, a potentially interesting application from a topological perspective.

- a data mining researcher could label each every node with the word that causes that page to rank most highly in a query. This would have the effect of compressing large clusters of websites related to a certain theme into one node representing that theme. In fact, this kind of knowledge discovery has been one of the most useful applications of SUBDUE [5].

- a lawyer could label each node with its domain name, causing the compressed graph to have nodes representing corporate entities.

An illustration of the general technique appears is show below; note that in this case, SUBDUE has been configured to identify "almost" isomorphic subgraphs, and to use labels on edges.



It is important to note that SUBDUE does more than scan the graph and return any set of nodes that matches the specified criteria. Rather, it attempts to return sets of nodes whose structure commonly occurs in the graph. It is this ability to find substructure that allows for semantically interesting compression, as evidenced most clearly in the second scenario above.

We exploit the connected nature of these identified substructures to compress webgraphs in a local, scalable way, as described in the next section.

3 Local Compression Functions

Our definitions of rank, edge, identifier and list functions are purposefully vague. In particular, very esoteric and computationally complex functions fit the definition. By requiring the matching function and compression functions to work together, in our case by using SUBDUE as a matching function, we can come up with a very natural set of functions that are scalable and useful for many of the applications we are considering. Below, let G be the domain of webgraphs.

Definition. A *substructure* is a tuple (c, R) where c is a node and R is a set of nodes directly connected (either by in edge or out edge) to c. c is called the *center* of the substructure and the nodes in R are called the *spokes*. A substructure can be considered to be a graph; we will typically treat substructures as graphs without explicitly noting so. Denote the set of all substructures in a graph g as $S(g)$.

Our definition of substructure is essentially a restriction of the definition of a connected graph, the restriction being that a substructure must have one center node and all other nodes must be immediate neighbors of the center. We require this restriction simply because this definition is easier to use in our software framework, as described in the next section. The use of substructures may be replaced with connected subgraph in the analysis that follows with some slight modifications.

In essence, SUBDUE identifies connected subgraphs, and we are trying to use this fact to ensure that our compression operations are local to these identified subgraphs and their immediate surroundings, which is required for compression to scale well. This notion of locality is captured by *local compression functions*:

Definition. A *local compression function* is a function $f(g) : (G, S(g)) \rightarrow G$ that returns a result graph with the following properties:

1. the PageRanks for all nodes that are not connected to the substructure are unchanged
2. the wordlists for all the nodes that are not the center are unchanged
3. the identifiers for all the nodes that are not the center are unchanged
4. the sets of in edges and out edges for all nodes that are not connected to the substructure are unchanged
5. the spoke nodes of the substructure do not appear in the resulting graph and wherever there was an edge to or from a spoke node, there is now an edge to or from the center node.

Intuitively, a compressed graph C compressed by a local compression function can be expressed as a *fold* over the original graph g; that is, $C = f(f(f(g, sub_1), sub_2), sub_3) \ldots$ for some set of substructures sub_n. Importantly, this property means that we can compress a graph by iteration by repeatedly searching for substructures that match our requirements, and that at each iteration, we only need to modify a small part of the graph.

A local compression function is thus defined by three properties: the new wordlist for the center node, the new identifier for the center node, and a strategy for determining PageRank. By using a local compression function in concert with a sequential matching function like SUBDUE that determines substructures, we have a scalable compression algorithm. It is exactly this kind of compression that we have implemented in our framework.

4 Software Framework

In order to evaluate our ideas and provide a tool allowing the types of compression described above, we've implemented a Java application with the following features:

1. A pluggable interface for each degree of freedom described in section 1. That is, when a user loads and compresses a graph, he can independently choose the matching, edge, rank, list and identifier functions that should be used.
2. The ability to load and save graphs in SUBDUE format, enabling quick access to a large amount of pre-existing data sets, including webgraphs, molecular bond graphs, and other highly-patterned graph data.

3. Automated interaction with the SUBDUE application for the purpose of computing SUBDUE-based matching functions. This allows for unlabeled, and more interestingly, labeled pattern-matching using SUBDUE.

4. A simple Naive matcher which randomly identifies potential substructures without any regard to relevance. This is intended to be used as an experimental baseline.

5. The ability to generate random graphs and graph element content according to several useful parameters.

6. The ability to calculate PageRank on arbitrary directed graphs.

7. A variety of strategies for redistributing PageRank from deleted nodes to the rest of the graph.

8. Multiple evaluation functions, including top-N keyword queries for evaluating query performance, graph size for determining how compressed a graph is, and PageRank distance for estimating how far a graph is from its settled PageRank distribution.

9. A simple graph visualization tool for examining the effect of different operations on reasonably sized graphs.

The application uses a Java-based graph packaged called JUNG [1], which enables it to efficiently carry out operations on large graphs. In general, the input graphs are assumed to be webgraphs, so a sparse graph representation is used to minimize memory consumption.

Figure 1 shows the main application screen and Figure 2 demonstrates graph visualization with labels and PageRank calculations.

5 Experiments

We have conducted a series of experiments evaluating the suitability of SUBDUE-based compression to the archivist's scenario described above. The archivist wants to preserve a copy of the web which maintains the quality of top-N search results. The goal of our experiment was therefore to maintain the top-N results on a set of search terms under the chosen compression operations.

Our experiments made use of real-world webgraph data captured by a web crawl. These data sets describe the topological arrangement of several hundred closely related web pages. In order to hold topology constant under changes in content, we assigned content to each page by associating with it a list of about 50 randomly-chosen keywords. The keywords were drawn from a candidate pool of 1000 words. An affinity parameter was used to control the degree to which nearby pages are related to one-another. With affinity set to 100%, all of a page's keywords are guaranteed to appear in at least one of its neighbors as well. With

affinity set to 0%, the random assignment makes no guarantees about word overlap between neighboring pages.

In this setting we would like to create clusters of pages with the same general topic. We therefore labeled each node with the word for which it has the highest query ranking, on the theory that this is likely to constitute a good approximation of the page's topic. The labeled SUBDUE matcher was then used to search for and combine clusters of related pages.

The most intuitive way to combine page information is to compute the union of the word lists of the removed nodes and add these keywords to the center node. This is the approach we used. The same union operation was used to combine the list of site names so that they could all be returned as search results. Ideally, the compressed graphs would consist of clusters of related pages. There is then a trade off between the amount of compression desired and search precision. Compression is achieved because the total number of nodes, links, and word sets is reduced.

If this approach had worked, we would have compared the search performance at a given level of graph compression with that of the Naive algorithm. Unfortunately, even under the most favorable conditions we were able to create with these parameters, it seems that SUBDUE is rarely able to discover useful substructures in this scenario. The structures it returns are generally only one or two nodes in size, indicating a lack of organized structure in the labeled graph. The only conclusion we can draw is that either our experimental setup did not adequately capture relevant properties that may be present in real webgraphs or that this approach is not generally useful for web graph compression of the type we were attempting.

6 Future Directions

Our framework may be extended in many ways; for instance, by adding non-local efficient pagerank distribution functions like those in [2], or by generalizing our local compression functions to use arbitrary connected graphs rather than substructures. One particularly speculative idea we have been thinking about is whether or not it is possible to use SUBDUE to guarantee that a compressed graph will match the top N search results on the original webgraph for some given set of queries. This idea has not been fully explored but our thoughts are presented here to give a flavor for the potentially powerful applications that labeling enables and the interesting new questions it raises.

Definition. A *stratification* is a set of real numbers $0 < r_0 < \ldots r_n < 1$. Each r_n is called a *rank*.

Let S be a stratification and g a webgraph. Our idea is to label each node in g with where that node's PageRank falls in the stratification S. Iteration one of compression would look for common substructures among center nodes with rank r_1 and spoke nodes with rank r_0. Iteration two would merge labels for r_0 and r_1 and then look for substructures among center nodes with rank r_2, and so forth.

We would like to choose the stratification S such that during compression, when PageRank is redistributed, no node ever jumps from rank r_n to rank r_{n+1}. If this restriction holds, the resulting graphs should have the property that when a query is executed, all "pages" with original ranks r_n will always appear below all "pages" with original ranks r_{n+1}, giving a graph in which "pages" are ordered correctly between ranks, if not ordered correctly inside of a rank. If we increase the amount of stratification so that every node has its own label, then we arrive at an algorithm that completely respects PageRank and does not compress at all. (We write "pages" because each node during compression actually represents a set of pages.)

Unfortunately, we are unsure of how to choose a stratification that has this property; ideally, one could find the stratification simply by looking at the distribution of the original graph, or, less ideally, by looking at the topology of the graph. Regardless, it seems like a stratification with this property in some sense represents the variability of PageRank within the graph, which we think is an interesting topic to think about. Of course, because labels are taking PageRank values, this algorithm would undoubtedly require modification before it could be used in, say, the archiving scenario described above, but the algorithm raises interesting questions in its own right.

7 Conclusion

Webgraph compression is a relatively new topic; even the notion of what compression means in this setting is still unsettled. We have chosen to apply the SUBDUE tool to an area that it what not specifically designed for, but an area where it shows some promise. Although our general framework allows for a wide variety of compression scenarios to be expressed, it is less clear that the use of labeling with SUBDUE is necessarily the most natural way to express the properties required of compressed webgraphs. This is most clearly seen in our experiments with the archival scenario, where what seemed like a plausible use of SUBDUE did not lead to good results, at least in our admittedly small experimental setup. Overall, given the success of SUBDUE in related areas and the growing need to compress – or mine – information from the web, it seems likely that SUBDUE can be applied in some manner to this area. Only time will tell.

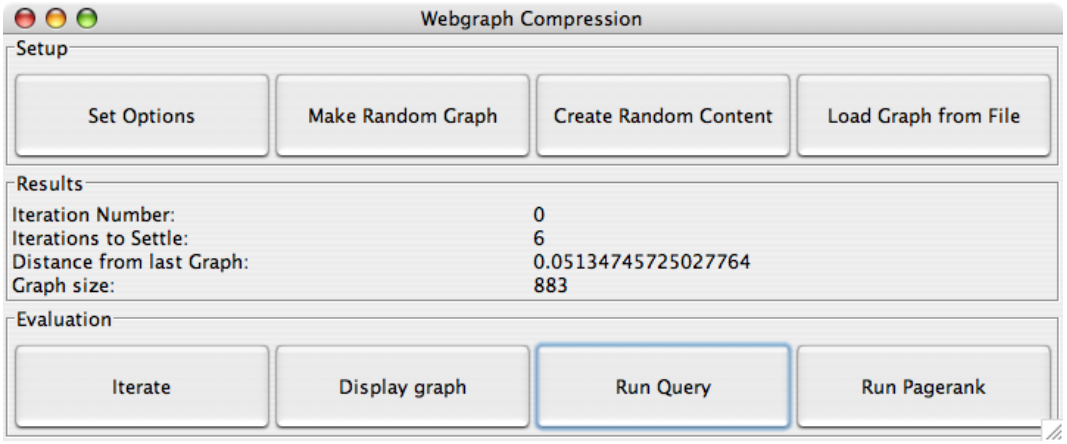


Figure 1: The main application screen

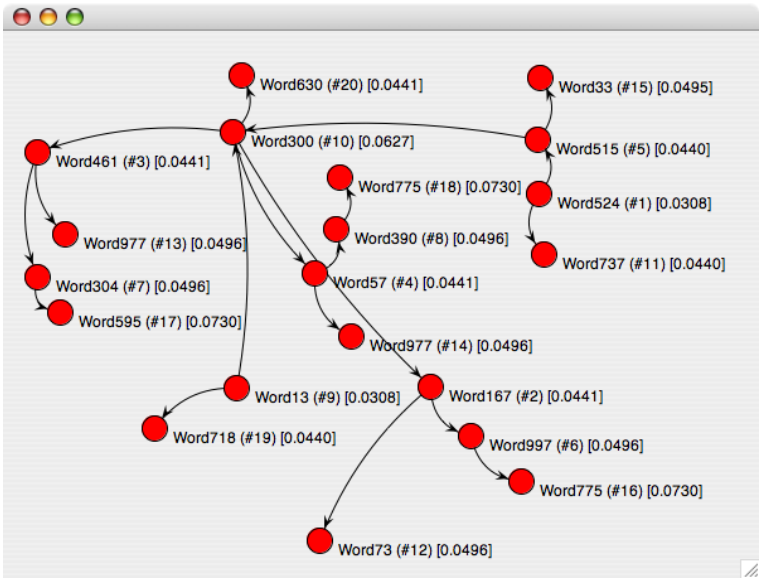


Figure 2: A sample graph visualization with labels and PageRanks

References

[1] Java Universal Network/Graph Framework. [updated 20 October 2006; cited 14 January 2007]. Available from http://jung.sourceforge.net/.

[2] *Towards Exploiting Link Evolution.* S. Chien, C. Dwork, R. Kumar, D Simon, and D. Sivakumar, Workshop on Algorithms for the Web, November 2002.

[3] *Substructure Discovery in the SUBDUE System.* D. Cook et al, AAAI Workshop on Knowledge Discovery in Databases p169-180, 1994.

[4] *Structural Web Search Using a Graph-Based Discovery System.* D. Cook et al, Florida Artificial Intelligence Symposium, 2001.

[5] *Graph-based Data Mining.* D. Cook and L. Holder, Encyclopedia of Data Warehousing and Mining, Idea Group Publishing, J. Wang (ed.), 2005.

[6] *Compressing network graphs.* C. Gilbert and K. Levchenko, LinkKDD workshop at the 10th ACM Conference on KDD, 2004.

[7] *Deeper Inside PageRank.* A. Langville and C. Meyer, Journal of Internet Mathematics Vol 1 No 3 p335-380, 2004.

[8] *Inferring the Structure of Graph Grammars from Data.* T. Oates et al, International Conference on Knowledge Based Computer Systems, 2002.

[9] *Structural Analysis of the Web.* P. Patnaik and S. Sanyal, IADIS International Conference, 2006.

[10] *Modeling Webgraph Evolution with Graph Grammars.* L. Ribeiro et al, 3rd International Conference on Graph Transformation, 2006.