Partitioning CiteSeer's Citation Graph *Revised Version*

Grégory Mermoud, Marc A. Schaub, and Grégory Théoduloz

School of Computer and Communication Sciences,
Ecole Polytechnique Fédérale de Lausanne (EPFL),
1015 Lausanne, Switzerland

Abstract. CiteSeer is a freely available, online citation database. In this paper, we explore the citation graph of CiteSeer. First, we characterize that graph as given by CiteSeer. Second, considering a set of anchor papers, we give heuristics to find a partition of the graph whose cut is minimal. Finally, we evaluate our solution using different metrics.

1 Introduction

CiteSeer is an automatically-built, freely available, online database of publications in the field of computer science. It indexes most publications and in particular allows the user to browse through the net of published paper along citations and references. Therefore, it captures the relation among papers. The idea of this project is to use the citation graph that can be built from CiteSeer to a find a partition of the entries by topics. Partitioning that kind of data can be done in essentially two ways. One can either take a semantical approach or a graph-theory-based approach. We opt for a graph-theory-based approach. Therefore, we formulate the problem informally as follows :

Given the citation graph, consider K anchor papers. Find a partition of the citation graph in K partitions such that there is exactly one anchor per partition and such that the number of edges between nodes in different partitions is minimal.

The anchors that we consider are 16 highly cited papers from clearly distinct domains. We hope that such a minimal cut will have some semantically interesting properties, grouping in a same cluster papers on close topics.

Three problems are to be addressed and each of them is detailed in the following sections.

The first problem to face is to get a reasonably reliable citation graph and to characterize it so as to have a good knowledge of the data on which we are working.

The second problem is to evaluate the quality of a solution by using an appropriate metric.

The third problem is to find a tractable algorithm to solve the problem considering our very graph. Even though mathematicians have already studied this problem – known as *multiterminal cut* – there exists no algorithms which can be directly reused and which work as-is on a graph as large as ours.

Finally the best solution is evaluated using both the values of the metrics and its semantical soundness.

2 Data Set

2.1 Retrieving and Processing the Data

CiteSeer provides three different sources of information:

– the HTML-based web site tailored for human use,
– the XML-based OAI interface tailored for harvesting,
– an archive of a snapshot of the OAI records.

The different data sources differ on their reliability and completeness. We evaluate it with 30 papers from 3 distinct areas. The most accurate one – though not harvestable – is the HTML version. At the time of the analysis, the data within the archive and on OAI were similar. Nevertheless those data are not immediately usable as our experience showed. Details on how we build the graph using them is given in Section 2.2.

As aforementioned, the citation graph on which we are working is the one built from the archive of records from OAI. A Perl script preprocesses the data in order to extract the information required for building our graph. It takes as input a set of XML records and gives as output a simple text file specifying for every record: (1) an identifier, (2) the document's publication date, (3) the identifiers of the references and (4) the identifiers of the citations ("is referenced by"). The C++ code builds the graph from the generated text file.

2.2 Building a Correct Graph

The web version of CiteSeer contains two kinds of elements: *documents*, which have been extensively processed and for which detailed information is available, and *contexts*, which are elements that appear in documents, mostly as citations, but for which the actual resource has not been processed. Books are mainly represented as contexts. Beside the title, date and author information, contexts only provide a list of documents in which they are referenced.

The record of a paper A in the OAI data represents connections between this one and other papers B_i in two different ways: *References* and *IsReferencedBy* links. A careful verification of these links based on the references actually mentioned in the electronic version of several papers shows that the relation *References* link represent a correct subset of the papers cited by paper A, but that an *IsReferencedBy* links between A and B_i does not mean B_i *cites* A. After comparing the reference information in the OAI data with the CiteSeer web page for the corresponding papers and contexts in more than 20 cases, we propose the following explanation for the *IsReferencedBy* links:

- The internal CiteSeer database represents documents and contexts separately. Each document has a *documentID* and each context has a *contextID*. The ranges of these identifiers overlap. The fact that a document has the same numerical identifier than a context is solely due to this overlap and does not imply any relationship between them.
- In the OAI data, a document and the context with the same numerical identifier are inadvertently mixed together.
- A *References* link from document A to a document B means that document A cites document B.
- An *IsReferencedBy* link from document A to document B means that document B cites a context C such that the *contextID* of C is equal to the *documentID* of A.

We build a representation $G = (V, E)$ of this corrected graph such that node $u \in V$ is either a document or a context and has an initial weight $w_N(u) = 1$ and every *References* and *IsReferencedBy* link is represented by an undirected edge $(u, v) \in E$ with an initial weight $w_E(u, v) = 1$. The weights on the edges are necessary to be able to collapse edges that are either cut together or not cut at all and the weights on the nodes, to allow grouping of nodes that are necessarily part of the same cluster.

2.3 Graph Simplification

We perform simplification in order to reduce the size of the graph while keeping an exact solution that will have the same minimal cut as the original graph. Doing so improves the running time of the algorithms that we use to find a min-cut.

We first note that a node which has only one connection will be part of the same cluster as its only direct neighbor and the connecting edge will not be part of the min-cut. Ignoring the node and the edge will therefore not change the result of the algorithm. Similarly, for a node which has exactly two connections, at most one of its adjacent edges will be in the min-cut and it will necessarily be the one with lower weight. We can therefore add an edge of this weight between the two nodes to which the node with degree two is connected and remove that node and the two connecting edges. It can be shown that exact simplifications of this kind are not possible for nodes of higher degree. We apply these simplification steps repeatedly to each non-anchor node of the graph until there are no more nodes of degree one or two left in the graph.

Formally, for a given node u of the graph:

- if $degree(u) = 1$, $\exists! v | (u, v) \in V$:
  - Remove the edge (u, v).
  - Set $w_N(v) = w_N(v) + w_N(u)$.
  - u will be in the same cluster as v.
- if $degree(u) = 2$, $\exists! v \exists! v' | (u, v) \in V, (u, v') \in V, v \neq v', w_E(u, v) \leq w_E(u, v')$:
  - Remove the edges (u, v) and (u, v').
  - Set $w_N(v) = w_N(v) + w_N(u)$.
  - If $\exists (v, v') \in V$: Set $w_E(v, v') = w_E(v, v') + w_E(u, v)$, else : add the edge (v, v') with $w_E(v, v') = w_E(u, v)$.
  - u will be in the same cluster as v'.

Starting from an initial graph with 1,151,200 nodes and by applying these simplification steps on our graph, we remove a total of 344,328 nodes. Furthermore, in the original graph 211,643 nodes have degree 0 and can therefore be ignored for our computation. In the simplified graph, all anchors are connected together in one component of size 593,931.

2.4 Graph Characterization

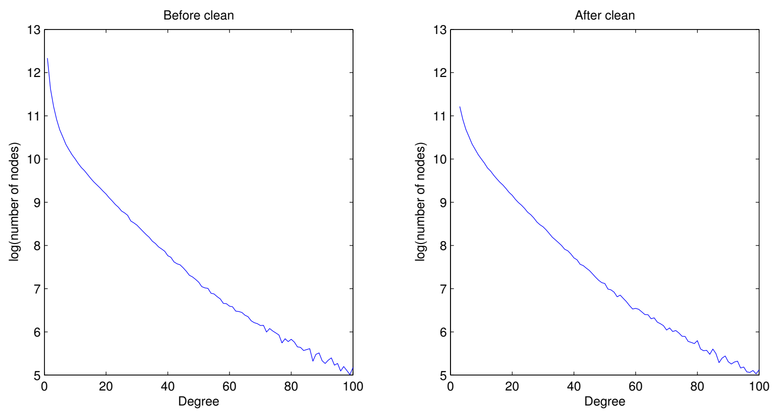

Fig. 1. Distribution of nodes per degree before and after having applied the cleanup algorithm explained in Section 2.3

It is interesting to know the characteristics of the citation graph before choosing an algorithm or a heuristic. One of the important property is density. In particular, it is interesting to study the distribution of node's degree. This kind of study has already been done for similar graph. For example, [1] studies the citation graph of the computer science literature. It claims that the degree are distributed following a power-law distribution. Figure 1 shows that in our case, the distribution is not a power-law (if it were, the log of the number of nodes with a given degree would have followed a straight line); the number of nodes with lower degree is larger than if it were following a power-law distribution.

3 Metrics

A K-terminal cut ($K = 16$) is a partition $\{V_1, ..., V_K\}$ of V such that each V_i, $i \in \{1, ..., K\}$ contains one anchor. The unnormalized cut is defined as:

$$c_p^u = \sum_{\substack{(u,v)\in E \\ u\in V_i, v\in V_j \\ i\neq j}} 1 \tag{1}$$

A *trivial solution* to this problem is to consider each anchor as being a cluster of its own with the exception of the anchor with highest degree, which is clustered with all other nodes. This solution has an unnormalized cost of 3,898. While this solution has a very low cost, it is also totally meaningless from a semantics point of view. For this reason we normalize this cut by dividing by the geometric mean of all clusters:

$$c_p^n = \sum_{\substack{(u,v)\in E \\ u\in V_i, v\in V_j \\ i\neq j}} \frac{1}{\sqrt[K]{|V_1||V_2|...|V_K|}} \tag{2}$$

This metric however doesn't take into account the fact that a significant number of nodes have degree zero and can therefore be part of any cluster. A strategy which places these nodes into the different clusters in a way that maximises the geometric mean will lead to significantly better measures even though such an optimization is totally wrong from a semantic point of view. We decided to cluster all nodes which are not part of a connected component containing at least an anchor into a separate cluster $V_\varnothing$. The metric we used in our optimization is therefore:

$$c_p^n = \sum_{\substack{(u,v)\in E \\ u\in V_i, v\in V_j \\ i\neq j}} \frac{1}{\sqrt[K+1]{|V_1||V_2|...|V_K||V_\varnothing|}} \tag{3}$$

Using this metric, the trivial solution has a cost 296.239.

4 Algorithms

4.1 Different Approaches

To solve this problem, we can choose between essentially three different approaches: (1) exact algorithms, (2) randomized algorithms and (3) heuristics. To make our choice, we have to take into account the two following challenges:

- *The problem is known to be NP-hard.* [2] proves that for a fixed number of anchors of three and more, the problem is generally NP-hard. If the graph is planar, there exists a polynomial algorithm to solve the problem for a fixed number of anchors. Unfortunately the citation graph is far from being planar.
- *The graph is very large.* With approximatively $700,000$ nodes and $5,000,000$ edges, we cannot use exponential algorithms. Even quadratic algorithms would take too long to execute on such a large graph. Therefore, we are looking for algorithms with a complexity as close as possible to linear.

Having considered these two points, the use of exact algorithms can definitely be ruled out because of a complexity that is exponential (because of the NP-hardness of the problem). Nevertheless, algorithms exist for slightly different problems or for graphs with different properties (e.g. according [3], the multi-terminal cut problem is polynomial on acyclic digraphs, undirected trees and rooted trees). One possible approach would have been to use one of the suggested algorithm by transforming our graph so as to have the property of interest and use an exact algorithm to solve the problem. The result is not guaranteed to be optimal because the original citation graph does not have the required property. Nevertheless, it would have been an interesting approach to pursue. Finally, linear program formulation exists (e.g. [4]) but they are generally zero-one linear programs that are hard to solve and in our settings, the number of variables and constraints is so large that, without further optimizations, the problem is not tractable.

Consequently, we choose between randomized algorithm and heuristics.

The advantage of a well-characterized randomized algorithm is that we know that it finds the optimal solution with a given probability P_{opt}. If the algorithm is run sufficiently often (a number of times of the order of $1/P_{opt}$) it has a high probability of finding an optimal solution. The problem with them is to find such a characterization. Moreover, since the problem is NP-hard it would not be surprising if the number of times we would need to run the algorithm is exponential itself.

The last option is heuristics. They can be cheap to run but the main issue is that we generally do not know how well they perform. Therefore, they usually have to be run several times but if their running time is small enough, it is not an issue. Moreover, their accuracy really depends on the data. Possible approaches include linear programming based heuristics [5] and energy-based layout [6]. We decide to try a local optimization algorithm, inspired by Kernighan-Lin [7].

Consequently, considering the too high cost of exact solution and the difficulty of characterizing randomized algorithm, we focus on finding appropriate heuristics.

4.2 Local Search Heuristic

This heuristic iteratively minimizes the local number of cuts for every node by putting the node into the cluster in which most of its direct neighbors are. Each run starts from a random partition and stops once no more nodes need to be moved to another partition. Such a solution is however likely to be a local minimum, and in order to escape from it, we shuffle the data by randomly assigning a cluster to a significant number of nodes (we use a value of 150,000) and then start the local optimizations again. The normalized cost is recomputed after every iteration, and the solution with the best normalized cost will be considered as best solution of the run. While the shuffle initially has a very negative impact on the cost of the min-cut, we observe (see Figure 2) that a few iterations of the the local optimization over all nodes are sufficient to obtain a better cost than before the shuffle. This can be explained by the fact that the local optimization algorithm has a high inertia: if a group of nodes belonging to the same cluster are tightly connected together, a shuffled node of that group will be moved back into the cluster at the next iteration of the local optimization. On the other hand, in a case where a node has approximately the same number of neighbors in two clusters, a shuffle might be significant to lead to a sticky change, or even to a change that will propagate further. The shuffle is therefore both conservative enough to avoid jeopardizing a good solution and provides enough noise to escape local minimums.

Starting from random partitions has the advantage of providing approximately equal clusters, which is useful for minimizing the normalized metric. We also tried to start from a more elaborate initial solution which clusters a node with the closest anchor. The distance from every node to every anchor can be found in linear time using a breadth-first search. The heuristic however converges significantly slower than when starting from a random partition.

This algorithm could be further extended to become a true genetic algorithm: while it already includes a powerful local search and a random shuffle similar to point mutations, it doesn't merge different solutions as in crossover. Due to lack of time, we were unable to pursue further into this direction, but given the performance of this simple heuristic and the fact that minimizing the normalized metric means balancing two goals, minimizing the number of cuts and keeping cluster sizes similar, we think that genetic algorithm have the potential to lead to very good solutions.

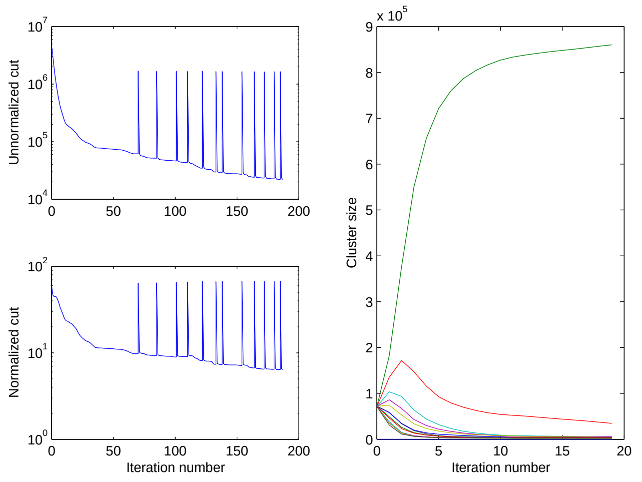

Fig. 2. Evolution of the cost of solutions obtained by iteratively improving the local min-cut for every node (left). The peaks illustrate shuffles when the local optimization reaches a minimum. Evolution of the cluster size during the first iterations of the local search (right). The *Theory* cluster starts dominating early in the run.

4.3 Contract Algorithm

Contraction-based algorithms are widely used for finding minimum cuts in undirected graphs [8]. This approach is based on *contracting* graph edges. Given an undirected graph $G = (V, E)$, a weight function $weight : E \to \mathbb{N}$ and an edge $(u, v) \in E$, the contraction of (u, v) results in a contracted graph G' that is identical to G except that the vertices u and v in G are now neighbors of the new node uv in G' ; the weight of the edge (uv, z) in G' is the sum of the weights of the edges (u, z) and (v, z) in G (if any).

We use as basis a randomized algorithm for the min-cut problem that we modified to take into account our problem constraints. Indeed, the basic algorithm produces a contracted graph G' containing only two nodes that represent an unconstrained bipartitioning. In our case, we want the algorithm to result in a graph G' containing m vertices – one per anchor – in which all the others are merged. Thus, the algorithm should not contract an edge linking two anchors. Similarly, a node in which an anchor is merged should become an anchor itself. Of course, the ending condition becomes $|V| = n$. Formally, the algorithm is as follows:

```
while V_card > n do
   pick at random an edge (u,v)
   if isAnchor(u) and isAnchor(v) then
```

```
      continue
   else
      contract(u,v)
   end if
end while
```

The main issue with such an algorithm is that straightforward implementations are dramatically slow, but we precisely need a low running time in order to perform numerous runs and compensate for their highly uneven results. Indeed, in our particular case, the number of runs should be at very least n^2 where n is the number of vertices. To get acceptable running times, this algorithm should be implemented by using adjacency matrices and optimized operations on them.

5 Results

5.1 Local Search

We were able to make 50 distinct runs of the local search algorithm in which every node was optimized 250 times and 150,000 nodes where shuffled each time the algorithm converged to a local minimum. The best solution has a normalized cost of 5.77519 (20.58 if we normalize ignoring the nodes with degree 0) for a total of 17,065 cut edges. The resulting partition has one major cluster of size 915,874 with anchor *Theory.* All other partitions have a size of approximately 1,000 papers. While the partitioning is still very uneven, this is still significantly better than the naive solution since the size of the smaller clusters has been increased by three orders of magnitude.

The list of document in each cluster can be found on our additional materials page[1].

One interesting point to notice is that the random partition , which cuts approximately 4,500,000 edges on average, has an average normalized cost of approximately 60, which is already significantly better than the naive solution. The fact that the local search algorithm is able to very significantly lower the number of cut edges while optimizing the normalized metric shows that this algorithm seems well suited for this problem.

Given its core position in Computer Science, Theory is the most plausible candidate for forming a big cluster, even though our result tends to show that even the normalized metric is too conservative in the sense that it groups papers that are not really close to another anchor into the big cluster.

[1] http://mtcserver.epfl.ch/∼theodulo/partition/

5.2 Contract Algorithm

Due to lack of time, we have not been able to perform even a single run of the contraction-based algorithm. We did not choose the most efficient implementation as described in 4.3 and it resulted in dramatically long running time.

6 Possible extensions

Altough in this work we consider the graph as a single fixed entity, its structure has actually been evolving over time. It would be interesting to take this component into consideration either by analyzing the graph as it was earlier in the history of computer science, or to build it in a directional way with respect to time. The first idea would provide an interesting insight into the evolution of domains separations whereas the second one would give more structural importance to the earlier days of Computer Science in the structure of the current graph.

We think that studying the evolution of this graph over time could lead to interesting insights about, for example, the evolution of domains, the emergence of new domains, the importance of papers in the graph over time and could possibly allow to pinpoint events or papers that had a major, transformative, impact on Computer Science.

7 Conclusion

The heuristic that we propose provides good results even though it is particularly simple. In particular, local optimization combined with the random partial shuffle provide interesting results. The current partial shuffle algorithm allow to escape from many local minima and we are confident that genetic algorithms can be applied to further optimized the solution.

To get better results, taking into account some semantical information would be reasonable. For this very particular graph, information like publication date or common co-authors are probably useful to get a more sensible partition and can be used to improve heuristics.

References

1. Y. An, J. Janssen, and E. Milios, “Characterizing and mining the citation graph of the computer science literature,” Tech. Rep. CS-2001-02, Faculty of Computer Science, Dalhousie University, 2001.
2. E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis, “The complexity of multiterminal cuts.,” *SIAM J. Comput.*, vol. 23, no. 4, pp. 864–894, 1994.

3. M.-C. Costa, L. Létocart, and F. Roupin, "Minimal multicut and maximal integer multiflow: a survey," *EJOR Eur. J. On Oper. Res.*, vol. 162, no. 4, pp. 55–69, 2005.
4. N. Garg, V. V. Vazirani, and M. Yannakakis, "Approximate max-flow min-(multi)cut theorems and their applications," in *STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, (New York, NY, USA), pp. 698–707, ACM Press, 1993.
5. G. Calinescu, H. Karloff, and Y. Rabani, "An improved approximation algorithm for multiway cut," in *STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing*, (New York, NY, USA), pp. 48–52, ACM Press, 1998.
6. A. Noack, "An energy model for visual graph clustering," in *Graph Drawing, 11th International Symposium*, Lecture Notes in Computer Science, pp. 425–436, Springer, 2003.
7. B. Kernighan and S. Lin, "An efficient heuristic procedure for partitionaning graphs," *Bell System Technical Journal*, vol. 49, no. 2, pp. 291–308, 1970.
8. D. R. Karger, "An $o(n^2)$ algorithm for minimum cuts," *Theory of Computing, Proc. 25th ACM Symposium on*, pp. 757–765, 1993.