Approximate Maximum Weight Branchings

Amitabha Bagchi* Ankur Bhargava† Torsten Suel‡

September 21, 2005

Abstract

We consider a special subgraph of a weighted directed graph: one comprising only the k heaviest edges incoming to each vertex. We show that the maximum weight branching in this subgraph closely approximates the maximum weight branching in the original graph. Specifically, it is within a factor of $\frac{k}{k+1}$. Our interest in finding branchings in this subgraph is motivated by a data compression application in which calculating edge weights is expensive but estimating which are the heaviest k incoming edges is easy. An additional benefit is that since algorithms for finding branchings run in time linear in the number of edges our results imply faster algorithms although we sacrifice optimality by a small factor. We also extend our results to the case of edge-disjoint branching of maximum weight and to maximum weight spanning forests.

Keywords. Analysis of algorithms; Graph algorithms.

1 Introduction

Given a graph $G = (V, E)$, (V, B) is a *branching* if B is a subset of E such that each vertex in (V, B) has in-degree at most one and there are no cycles. Branchings are basic graph structures which have found applications in various fields of computer science. Motivated by a data compression problem [12] we prove the following general theorem about weighted branchings:

> If $w(G_k)$ is the weight of a maximum weight branching on a subgraph of G where each node only retains its k heaviest incoming edges, and $w(G)$ is the weight of a maximum weight branching on the entire graph, then
>
> $$\frac{w(G_k)}{w(G)} \geq 1 - \frac{1}{k+1}.$$

Thus, we can a compute a branching with a weight almost as large as the maximum possible weight on a dense graph by only considering a few incoming edges for each vertex. Since algorithms for computing maximum weight branchings [15, 3] depend at least linearly on the number of edges in the graph, this implies faster algorithms for approximate maximum weight branching after appropriate preprocessing of the dense graph. More importantly, in many scenarios that can be modeled as graph problems, the main cost is in computing appropriate edge weights for the input graph rather than in the actual graph computation; our result implies that for branching problems it suffices to exactly compute only the weights of the heaviest edges in the reduction.

*Dept. of Computer Science and Engg., Indian Institute of Technology, Hauz Khas, New Delhi 110016, India. bagchi@cse.iitd.ernet.in

†Google, 1600 Amphitheatre Parkway, Mountain View, CA 94043, USA. ankur@cs.jhu.edu

‡Dept. of Computer and Information Science, Polytechnic University, 6 Metrotech Center, Brooklyn, NY 11201, USA. suel@poly.edu

1.1 A Simple Application of Maximum Weight Branching

Consider the following simple application in the context of data compression [12], which provided the initial motivation for our work. We are interested in compressing a collection of files where there is a significant degree of similarity (or redundancy) between many of the files. For example, web pages from the same site frequently share certain elements in their page layout and menu structure, and may also contain some similar and repeated content. This inter-file redundancy can be exploited to achieve better overall compression of the collection.

The process of compressing one file (the *target file*) with respect to another file (the *reference file*) is called *delta compression* or *differential compression* [7, 9, 17, 10]. But given a collection of n files, in what order should we apply delta compression between pairs of files to minimize the overall size? There is an exponential number of possible orderings of the pairwise compression steps. Obviously, it is beneficial to compress each file with respect to another very similar file. However, we have to avoid cycles, such as A being compressed with respect to B and B being compressed with respect to A, since it would be impossible to uncompress the resulting data.

This scenario [12], as well as related problems in the compression of web graphs [1] and multispectral images [16], can be modeled as a maximum weight branching problem on a directed graph. Finding an optimal set of delta compression steps is equivalent to finding a maximum weight branching on a complete directed graph with one node v_A for each file A in the collection, where edge (v_A, v_B) has a weight equal to the savings in bits obtained by delta compressing B with respect to A instead of compressing B by itself [12].

One problem with this reduction is that the resulting graph has n^2 edges, slowing down the maximum weight branching computation. However, in practice a much more significant challenge is the computation of the edge weights: The only way to determine the precise weight of an edge is to actually run the delta compressor on the two files involved. Of course, the final branching usually contains mostly fairly heavy edges, and thus it would be highly desirable to compute for each node only the weights of these most promising edges, instead of materializing the entire graph. Fortunately, in our scenario efficient techniques are known for estimating the similarities among pairs of files and for finding for each file the k most similar other files (or k-nearest neighbors) under various definitions of file similarity [11, 2, 6, 8].

Thus, a promising approach would be to compute for each node only the weights of the incoming edges from the k most similar files, and then compute the maximum weight branching on this subgraph. But can we show that this results in a compression scheme that is almost as good as using the complete graph? There are two issues here. First, the techniques for finding k-nearest neighbors in [11, 2, 6, 8] assume certain formal similarity measures between files that do not precisely model the benefit obtained by an actual delta compressor that uses a combination of various state-of-the-art compression techniques to minimize size. However, this issue is unlikely to be completely resolved, and in practice the known techniques seem to be able to identify promising references files for each file. The second question is, assuming that we have correctly identified for each file the k best references files, if we run a branching computation on this subgraph are we guaranteed to get a compression scheme whose benefit approximates that of a scheme based on the complete graph?

This question lead us to the following very simple and natural conjecture about maximum weight branchings: If $w(G_k)$ is the weight of a maximum weight branching on a subgraph of G where each node only retains its k heaviest incoming edges, and $w(G)$ the weight of a maximum weight branching on the entire graph, we conjecture that $\frac{w(G_k)}{w(G)} \geq 1 - \frac{1}{k+1}$.

In this paper we settle the above conjecture in the affirmative. We also show that this result can be extended in a natural way to c edge disjoint branchings [4] of maximum total weight, and to maximum weight spanning forests in undirected graphs.

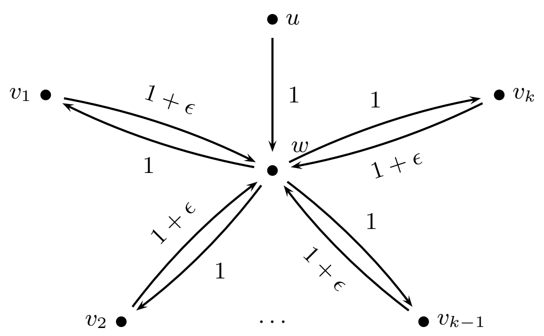

Figure 1: Tightness. Let the graph be G. All other edges in this graph have 0 weight. The weight of the maximum weight branching in G is $k+1$ while the maximum weight branching in G_k is $k+\epsilon$. As $\epsilon \longrightarrow 0$ we get a $\frac{k}{k+1}$ factor.

2 Maximum Weight Branchings

We consider a directed graph $G = (V, E)$ with an edge weight function $w : E \rightarrow \mathbb{R}^+$. A *branching* (V, B) is a subgraph of G with an edge set $B \subseteq E$ such that (V, B) is acyclic and the in-degree of any vertex of (V, B) is at most 1. Note that in general, a branching forms a forest of rooted directed trees. The weight of a branching B is $w(B) = \sum_{e \in B} w(e)$. A *maximum weight branching* is a branching with weight at least that of any other branching.

We define the *k-heavy subgraph* of G, denoted G_k, as the subgraph that contains only the k heaviest edges incoming to each vertex. If the in-degree of a vertex is less than k then we assume edges of weight zero. Ties are broken arbitrarily.

We prove the following theorem relating the weights of the maximum weight branchings in G and G_k. The theorem follows as a corollary of the more general Theorem 2.2 below, and thus we defer the proof.

Theorem 2.1 *Let G be a weighted directed graph and an integer $0 < k < n-1$. The weight of the maximum weight branching in G_k is at least $1 - \frac{1}{k+1}$ times the weight of the maximum weight branching in G.*

2.1 Tightness of Approximation

We first prove that the above bound of $1 - \frac{1}{k+1}$ is in fact tight. Figure 1 gives an example of a simple graph that shows that the bound is the best possible in the worst case.

2.2 Maximum Weight c-Edge Disjoint Branchings

We define a c-EDB (a set of c edge disjoint branchings) in $G = (V, E)$ as a tuple $(V, B_1, B_2, \ldots, B_c)$ such that for all $1 \le i \le c$, (V, B_i) is a branching in G and the edge sets $B_1, \ldots, B_c$ are disjoint. The weight of a c-EDB is the sum of the weights of all the c branchings. A maximum weight c-EDB is a c-EDB on a weighted directed graph such that its weight is at least the weight of any other c-EDB. Efficient algorithms are known for computing edge-disjoint branchings in graphs. Edmonds inaugurated the area by giving conditions for existence of edge-disjoint branchings [4]. Tarjan [13, 14] gave efficient algorithms for computing edge-disjoint branchings which were improved and used in the context of determining connectivity by Gabow [5].

We present a generalization of theorem 2.1. For $c = 1$ we get the statement of theorem 2.1.

Theorem 2.2 *Let G be a weighted directed graph, k be a natural number for which G_k is well defined and c be a natural number such that $c \leq k$. The weight of the maximum weight c-EDB in G_k is at least $1 - \frac{1}{k-c+2}$ times the weight of the maximum weight c-EDB in G.*

Proof. We prove the theorem by constructing a c-EDB in G_k from a maximum weight c-EDB in G such that its weight is at least a $1 - \frac{1}{k-c+2}$ fraction of the weight of the maximum weight c-EDB in G. The maximum weight c-EDB in G is a set of c branchings: $B_1, B_2, \ldots, B_c$, such that no two branchings share a common edge. The edges that these branchings are composed of, are not necessarily amongst the k heaviest incoming edges at each node. We transform each of these branchings into branchings that use only the k heaviest incoming edges at each node. There is some loss in weight in the course of this transformation.

We start with branching B_1. There is no specific order required of the branchings, so B_1 could be chosen arbitrarily. In addition the same transformation procedure is applied to each of the branchings, therefore we restrict the discussion to B_1. Let E_k be the k-heavy edges in G. Consider that there is an edge $e = (u, v)$ in B_1 such that, $e \notin E_k$ and every descendent edge in B_1 is in E_k. Let $e_1, e_2, \ldots, e_k$ be the k-heavy edges incident on v. None of these k edges are part of B_1. For all $1 \leq i \leq k$,

- e_i is in exactly any one of the branchings $B_2, B_3, \ldots, B_c$; or
- e_i results in a cycle in B_1.

Note that since there are $c-1$ other edge disjoint branchings there can be at most $c-1$ edges of the k-heavy edges that may already be in use in other branchings. This leaves $k-c+1$ edges that cannot be part of any branching. We denote this set of $k-c+1$ edges by the set X. We perform one of the following two transformations:

Transformation 1. If any edge $x \in X$ is incident on v from a non-descendent of v in B_1 then simply add x and remove e from B_1. Next we continue the search for an edge in B_1 that is not in E_k.

Transformation 2. If all the edges in X are incident on v from descendents of v then each edge must result in a cycle in B_1. There are $k-c+1$ cycles possible. Each cycle has at least one distinct vertex (the distinct vertices are the sources of the edges in X). This means that each cycle must have at least two distinct edges. At least one of these must already be in the branching. Therefore, there are at least $k-c+1$ distinct edges that are descendents of v in B_1. We denote this set of edges by the set Y. Let y be the least weight edge in Y. Let $x \in X$ be the edge that creates a cycle with y in it. We perform one of the two following edits the choice of which is decided by the weight of y:

2.1 if the weight of y is less than that of x then remove edges y and e from B_1 and add x to B_1; else

2.2 remove e from B_1.

Next we continue the search for an edge in B_1 that is not in E_k.

It is clear that with these two primitive transformations we can convert a c-EDB into a k-heavy c-EDB as long as $k \geq c$. Next we show the accounting to prove that the loss in weight due to the transformations is bounded. Transformation 1 does not require any accounting because the weight of the branching increases after applying it. In the case of transformation 2, there is a net loss of at most $(w(Y)+w(e))/(k-c+2)$. To show this we present the argument in the two different possible outcomes of transformation 2.

In subcase 2.1 of transformation 2 we remove edges e and y and insert edge x. Since x is heavier than y and e, we have effectively removed the smallest edge either e or y from amongst $k-c+2$ edges $(Y \cup \{e\})$. The net loss is $(w(Y)+w(e))/(k-c+2)$ and is charged to $Y \cup \{e\}$.

In subcase 2.2 of transformation 2 we only remove edge e. Since e is the smallest edge amongst $k-c+2$ edges $(Y \cup \{e\})$ the net loss is $(w(Y)+w(e))/(k-c+2)$ and is charged to $Y \cup \{e\}$.

The crucial factor in the accounting is that once transformation 2 is applied edge e is removed and therefore the edges that have been charged can never be charged again. There is a possibility that they may be involved in a type 1 transformation but never a type 2. Therefore, the weight of the transformed structure is at least a $1-\frac{1}{k-c+2}$ multiplicative factor of the weight of the maximum weight c-EDB in G. □

3 Maximum Weight Spanning Forests

A *Forest* of an undirected graph $G=(V,E)$ is a subgraph $H=(V,F)$ such that $F \subseteq E$ and H is acyclic. The *maximum weight spanning forest* of G is a forest H such that its weight is no less than the weight of any forest of G. Note that Prim's or Kruskal's MST algorithm yields a maximum weight spanning forest when run on G. A k-heavy subgraph of an undirected graph, G is a subgraph $G_k=(V,E_k)$ such that $E_k \subseteq E$ and for each $(u,v) \in E_k$, (u,v) is one of the k heaviest edges incident on u or one of the k heaviest edges incident on v. Ties are broken arbitrarily. Note that $|E_k| \leq k|V|$. A c-EDF (a set of c edge disjoint forests) is a set of c forests on an undirected graph such that no edge of the graph is in more than one forest. A maximum weight c-EDF is a c-EDF on a weighted undirected graph such that its weight is at least the weight of any other c-EDF in that graph.

Let G be an undirected graph, k be a natural number for which G_k is well defined and c be a natural number such that $c \leq k$. The following corollaries are a consequence of theorem 2.2.

Corollary 3.1 *The weight of the maximum weight forest in G_k is at least $1-\frac{1}{k+1}$ times the weight of the maximum weight forest in G.*

Corollary 3.2 *The weight of the maximum weight c-EDF in G_k is at least $1-\frac{1}{k-c+2}$ times the weight of the maximum weight c-EDF in G.*

References

[1] M. Adler and M. Mitzenmacher. Towards compressing web graphs. In *Proc. of the IEEE Data Compression Conference (DCC)*, March 2001.

[2] A. Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences (SEQUENCES'97)*, pages 21–29. IEEE Computer Society, 1997.

[3] P. Camerini, L. Fratta, and F. Maffioli. A note on finding optimum branchings. *Networks*, 9:309–312, 1979.

[4] Jack Edmonds. Edge-disjoint branchings. In R. Rustin, editor, *Combinatorial Algorithms*, pages 91–96. Algorithmics Press, 1972. Proceedings of the 9th Courant Computer Science Symposium.

[5] Hal Gabow. A matroid approach to finding edge connectivity and packing arborescences. In *Proc. 23rd Annu. Symp. on Theory of Computing*, pages 112–122, 1991.

[6] T.H. Haveliwala, A. Gionis, and P. Indyk. Scalable techniques for clustering the web. In *Proc. of the WebDB Workshop*, Dallas, TX, May 2000.

[7] J. Hunt, K.-P. Vo, and W. Tichy. Delta algorithms: An empirical analysis. *ACM Transactions on Software Engineering and Methodology*, 7, 1998.

[8] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proc. of the 30th ACM Symp. on Theory of Computing*, pages 604–612, May 1998.

[9] D. Korn and K.-P. Vo. Engineering a differencing and compression data format. In *Proceedings of the Usenix Annual Technical Conference*, pages 219–228, June 2002.

[10] J. MacDonald. File system support for delta compression. MS Thesis, UC Berkeley, May 2000.

[11] U. Manber and S. Wu. GLIMPSE: A tool to search through entire file systems. In *Proc. of the 1994 Winter USENIX Conference*, pages 23–32, January 1994.

[12] Z. Ouyang, N. Memon, T. Suel, and D. Trendafilov. Cluster-based delta compression of a collection of files. In *Third Int. Conf. on Web Information Systems Engineering*, December 2002.

[13] R. E. Tarjan. A good algorithm for edge-disjoint branching. *Inf. Proc. Lett.*, 3(2):51–53, 1974.

[14] R. E. Tarjan. Edge-disjoint spanning trees and depth-first search. *Acta Informatica*, 6:171–185, 1976.

[15] R. E. Tarjan. Finding optimum branchings. *Networks*, 7:25–35, 1977.

[16] S. Tate. Band ordering in lossless compression of multispectral images. *IEEE Transactions on Computers*, 46(45):211–320, 1997.

[17] D. Trendafilov, N. Memon, and T. Suel. zdelta: a simple delta compression tool. Technical Report TR-CIS-2002-02, Polytechnic University, CIS Department, June 2002.