Polytechnic
UNIVERSITY
Brooklyn · Long Island · Westchester

Compressing File Collections with a TSP-Based Approach

Dimitre Trendafilov Nasir Memon Torsten Suel

Department of Computer and Information Science

Technical Report
TR-CIS-2004-02
04/29/2004

Compressing File Collections with a TSP-Based Approach

Dimitre Trendafilov *Nasir Memon* *Torsten Suel*

CIS Department
Polytechnic University
Brooklyn, NY 11201

Abstract

Delta compression techniques solve the problem of encoding a given target file with respect to one or more reference files. Recent work in [15, 12, 7] has demonstrated the benefits of using such techniques in the context of file collection compression. In these scenarios, files are often better compressed by computing deltas with respect to other similar files from the same collection, as opposed to compressing each file by itself. It is known that the optimal set of such delta encodings, assuming that only a single reference file is used for each target file, can be found by computing an optimal branching on a directed graph.

In this paper we propose two techniques for improving the compression of file collections. The first one utilizes deltas computed with respect to more than one file, while the second one improves the compressibility of batched file collections, such as *tar* archives, using standard compression tools. Both techniques are based on a reduction to the *Traveling Sales Person* problem on directed weighted graphs. We present experiments demonstrating the benefits of our methods.

1 Introduction

Collections of files are often compressed by batching files together and then applying a standard compression tool. For example, in the UNIX environment this is most commonly done using *tar* followed by *gzip* or *bzip2*. This approach results in a slightly better compression than if the files are compressed individually by taking advantage of interfile similarities, though only to a limited extend. Delta compression algorithms are concerned with encoding a *target file* in relation to one or more *reference files*; intuitively this can be viewed as compressing only the difference between the files. Recent work [15, 12, 7] has shown that delta compression techniques can improve the compression of file collections by more fully exploiting similarities between different files.

In most of the proposed scenarios, the files in the collection are compressed by performing a number of pairwise delta encodings. There are two fundamental restrictions imposed on the set of these pairwise encodings. First, a file should not be compressed more than once, i.e., a file should exist in only one pair as a target. Second, no cycles should exist in between the pairs - if there are cycles we will not be able to decompress the files. Following these restrictions an optimal set of pairwise encodings can be found by computing a maximum branching on a complete directed weighted graph where for each file there is a corresponding vertex, and the edge weights are given by the benefit obtained from compressing the target vertex with respect to the source vertex rather than on its own (or an estimation of this benefit). Thus, an optimal encoding can be computed in polynomial time using the algorithm for maximum branching in [5, 17].

In this paper, we attempt to find a near-optimal set of delta encodings by a reduction to the problem of finding an optimum tour of a directed weighted graph, commonly known as the *Traveling Sales Person* (TSP) problem. While this problem is NP Complete and thus an optimal algorithm unlikely to be found, there are a number of heuristics that work well in practice. The main advantage of this approach is that it can be easily extended to the case where we compute deltas with respect to more than one reference file, and thus it can lead to better compression than maximum branching-based approaches that are restricted to deltas with respect to a single reference file.

Moreover, the computed tour can be used to improve the compressibility of batched file collections without the use of delta compression. In particular, the popular UNIX tool *tar* is often used to concatenate collection of files into one large file. The resulting *tar* file contains the necessary meta information needed to extract individual files. While the file order in the *tar* file is not important for the correct functioning of *tar*, it can affect the compressibility of the batched file. We present experiments demonstrating that the compressiblity of *tar* files under standard compression tools such as *gzip* and *bzip2* can be improved if the files are placed into the *tar* file in the order computed by our heuristic. Thus, a slight enhancement of the *tar* utility at the sender can result in improved compression while allowing a receiver to decode the file collection using standard *tar* and decompression tools.

1.1 Contributions of this Paper

We study the problem of compressing collections of files by computing a linear file ordering that maximizes the interfile similarities between consecutive files. We propose and evaluate two schemes that can benefit from such file ordering. The first one starts out by delta compressing each file with respect to its immediate predecessor in the ordering. We show that this already results in performance close to that provided by a maximum branching, and that it can be further improved by adding a second reference file. The second scheme batches the files in the order specified by the file ordering, resulting in a file that is better compressible by conventional compressors. Our main contributions are:

- We propose a heuristic for compressing collection of files by using deltas with respect to multiple reference files. The heuristic first orders files based on a TSP heuristic, and then compresses each file with respect to its immediate predecessor and the best of its other predecessors.
- We observe that by concatenating files in a TSP order, we can improve the compressibility of the resulting *tar* file while preserving the logical structure of the collection. The importance of the latter fact is that the batched file will be compatible with the unmodified batch program.
- We implement both heuristics and present experimental results on their performance.

The rest of this paper is organized as follows. The next subsection discusses related work. Section 2 describes the simple TSP heuristic that we used to compute a linear ordering for the files in the collection. Section 3 describes our method for compressing collections of files using deltas computed with respect to two files, while Section 4 describes the approach for improving the compressibility of batched files. Section 5 provides experimental results. Finally, Section 6 provides some open questions and concluding remarks.

1.2 Related Work

For an overview of delta compression techniques and applications, see [16]. Among the main delta compression algorithms in use today are *diff*, *xdelta* [13], and *vdelta* [10] and its newer version *vcdiff* [11]. We use the *zdelta* compressor, which was described and evaluated in [19] and previously used for compressing file collections in [15].

The issue of appropriate distance (or similarity) measures between files and strings has been studied extensively, and many different measures have been proposed. We note that the *diff* tool is based on a symmetric *edit distance* measure, while *vdelta* and other recent Lempel-Ziv type delta compressors such as *xdelta* [13], *vcdiff* [11], and *zdelta* [19] are more related to the *copy distance* between two files. Recent work in [6] also studies another measure called *LZ distance* that models the behavior of Lempel-Ziv type compression schemes. Of course, the most accurate, but also fairly expensive, way to determine the compression benefit of a delta compressor is to run it on the files in question.

However, in many practical scenarios, we would like to quickly estimate the degree of similarity between pairs of files, and in particular to cluster files into groups of fairly similar files, without performing expensive pairwise distance computations between all pairs. A number of efficient sampling-based approaches have been studied for this case [2, 6, 9, 14], often based on similarity measures that are much simpler than edit distance and its variants. In particular, the clustering techniques in [2, 14, 9] that we use later are based on two very simple similarity measures, which we refer to as *shingle intersection* and *shingle containment*. For a file f and an integer q, we define the *shingle set* (or q-gram set) $S(f)$ of f as the multiset of substrings of length q (called shingles) that occur in f. We define the shingle intersection of files f and f' as $I(f,f') = \frac{|S(f) \cap S(f')|}{|S(f) \cup S(f')|}$, and the *shingle containment* of f with respect to f' as $C(f,f') = \frac{|S(f) \cap S(f')|}{|S(f)|}$. (Note that shingle containment is not symmetric.) Our methods can be used with exact values for the graph edges or with rough estimates based on shingle containment or intersection. For a discussion of these measures we refer the reader to [2, 8, 15].

Fast algorithms for the optimum branching problem are described in [5, 17]. Adler and Mitzenmacher [1] have shown that a natural extension of the branching problem to hypergraphs that can be used to model delta compression of collections with two or more reference files is NP Complete. If we restrict delta chains to a maximum length in order to allow for efficient extraction of individual files, as proposed in [7], the resulting optimization problem is also known to be NP Complete [18]. (Such restrictions may also result in reduced compression performance.)

The idea of compressing file collections using pairwise deltas is studied in [15] and [7]. Most closely related is the approach in [15], which performs maximum branching computations on a directed weighted graph that is constructed from the file collection using a variety of clustering and file similarity estimation techniques. We also point out that Manber [14] has previously suggested that similar files could be grouped together in order to improve the compressibility of batched files, but no details and experiments were provided. Finally, graph optimization techniques have also been used to optimize the compression of data sets stored in large tables [3, 4, 20].

2 TSP Greedy Heuristic

The *Traveling Sales Person* problem is one of the most widely known NP Complete problems. We now present the simple greedy heuristic that we use to solve the problem; of course, other known approaches would also work. We observe that in our scenario we have no requirements for particular starting and ending vertices, and that the computed path could be interrupted at some vertices. The basic idea in the heuristic is to always chose greedily the best available edge in the graph. If the selected edge creates a branching or a cycle within the set of edges generated thus far, than it is thrown away, and the next best edge is processed. The algorithm terminates when no more edges can be added to the generated graph tour.

```
TSP-Heuristic(G)

$Q \leftarrow edges(G)$                     Max-priority queue
$n \leftarrow number\_vertices(G)$
$I[n]$                                      Vertices with incoming edges.
$O[n]$                                      Vertices with outgoing edges.
$T \leftarrow \emptyset$                    Computed graph tour.

FOR EACH $v \in vertices(G)$
        $I[v] \leftarrow false$
        $O[v] \leftarrow false$
        Make_Disjoint_Set $DS_v$            Disjoint sets used for cycle prevention.

WHILE $sizeof(T) < n - 1$ AND $Q \neq \emptyset$:
    $e(i, j) = pop(Q)$
    IF NOT $(I[j]$ OR $O[i]$ OR $DS_i = DS_j)$:
        add $e(i, j)$ to $T$
        $I[j] \leftarrow true$
        $O[i] \leftarrow true$
        join$(DS_i, DS_j)$
```

The initialization part of the algorithm instantiates a priority queue containing the input graph edges and initializes a disjoint set data structure for each vertex. This part takes time $O(n + m)$, where m is the number of edges and n the number of vertices. The main loop is executed at most n times, and for each iteration the algorithm removes an edge from the priority queue, compares two disjoint sets, and performs several other constant time operations. The time needed for a single iteration will be at most $log(m) + log(n)$, and thus the overall running time for the algorithm will be $O(n + m + n * (log(m) + log(n)))$. Assuming that m is larger than n, the running time will be $O(m + n * log(m))$. Note also that this method will produce a result even for a disconnected graph, in which case the generated tour will be interrupted at some vertices.

3 Delta Compression Approach

This section presents our algorithms for compressing collections of files using the TSP heuristic in combination with delta compression. We start by building a graph whose edges correspond to the benefit obtained from compressing the target with respect to the reference, or some reasonable approximation of the benefit. The output of the TSP heuristic on such a graph is an ordered set of files. Now we can compress the file collection by doing pairwise delta compression along the generated edges. Assuming that we use exact rather than estimated edge weights, this approach cannot be better than the optimum branching scheme, since a linear ordering is a special case of a branching. Results from our experiments, however, will show that this scheme performs only marginally worse.

A better approach is to compress the collection by using deltas with respect to two or more reference files. We observe that having ordered the files, each file can be safely (i.e., without introducing cycles) compressed with respect to any subset of its predecessors. The obvious choice for the first reference file is the immediate predecessor. A second file can be selected by brute force, by trying all predecessors in the ordering. Based on the analysis in the previous section the time complexity for this algorithm would be $O(n * log(m) + n^2 * D)$, where D is the average time (for the particular file collection) to compute a delta with respect to two files. However, this bound can be improved if we select the second reference file based on approximations of the benefit such as those discussed in Section 1.2. Experiments will show that in all cases this heuristic improves upon the performance of the "optimal" branching approach. For the cost of additional computation, we could compute deltas with respect to more than two reference files. Our experiments showed that this results in only negligible improvements.

To finish the discussion, we need to look at how to handle any jumps existing in our graph tour. We could compress the file we jumped to with respect to the file we had jumped from, but a better approach is to simply concatenate all these files and compress them with a conventional compression tool.

4 Standard Compression Approach

This section presents our method for improving the compression of collections of files without the use of delta compression techniques. As mentioned in the introduction, the most common method for compressing collection of files is to batch the files in the collection and then compress the resulting larger file with a standard compressor. The most popular tool for this in a UNIX/Linux environment is the *tar* utility in conjunction with the *gzip* or *bzip2* compressors. The *tar* program itself does not perform any compression, but simply concatenates the files from the collection into a single file, referred to as an *archive*. In addition it stores some meta information at the beginning of the archive file and in between the concatenated files that allows it to extract files from the archive.

Note that the order in which files are batched is not essential for the *tar* program, and in fact most implementations allow the user to specify an ordering. Theoretically, a good batch compressor should be able to utilize redundancies not only in single files, but across the complete archive. In reality, however, the most commonly used compressors work only with a fixed size window into the input file. For example, *gzip* has a 32KB sliding window, while *bzip2* works with blocks of at most 900KB. Thus, when compressing a particular file inside the archive, a compressor can take advantage of redundancies with other files that are close-by, but not with files that are further away. By placing similar files next to each other in the archive we can improve its overall compressibility. In our implementation we use the linear ordering computed with our TSP heuristic as our file order.

As discussed, the main advantage of this approach is that it does not modify the format of the batch file, and thus the optimized batch file can be uncompressed using standard *tar* and *gzip* or *bzip2*. In fact, our method is implemented with a simple script that first calls a program for computing a good file order, and then supplies the files in the collection to the standard *tar* utility in this order. Thus, we basically augment *tar* with an initial analysis phase that scans the collection to compute the weighted graph, using either precise edge weights or estimates based on sampling as discussed in Section 1.2 for improved efficiency, and then performs the TSP computation. Our experiments will show that in many cases this results in a measurable improvement in compression, and it is thus an attractive option for the efficient distribution of file collections in cases where we do not want to require users to deal with new compressed file formats.

5 Experimental Evaluation

We now present an experimental evaluation of our methods on a set of file collections. In particular, we use six collections of HTML files downloaded in BFS order from six news websites. Each benchmark contains from 99 to 530 files. The average file size in the benchmarks varies from less than 20KB for CBC to above 45KB for USAToday.

We first present results for compressing collections using delta compression techniques as described in Section 3. In Table 1, we compare five different methods for compressing the collections:

- **cat+gzip:** concatenate the files from the collection and compress them with *gzip*
- **cat+bzip2:** concatenate the files from the collection and compress them with *bzip2*
- **ob:** compute an optimal branching and compress using zdelta with a single reference file
- **tsp:** compute a TSP ordering and compress using zdelta with a single reference file
- **tsp2:** compute TSP ordering and compress using zdelta with two reference files

Table 1 contains results based on using exact edge weights, i.e., the weight for edge (r, t) was computed as the benefit of delta compression over *gzip*: $gzip(t) - zdelta(r, t)$. For the results in Table 2 we used sampling to estimate file similarity, as described in [2] and used in [15]. An exception is the column for *tsp2*, where the selection of the second reference file was done by brute force using *zdelta*, although this could also be done based on faster estimation techniques. In all cases, files with no reference files (i.e., jump destinations) were compressed by concatenating them and using *gzip*.

data set	pages	average size	total size	cat+gzip ratio	cat+bzip2 ratio	ob ratio	tsp ratio	tsp2 ratio
CBC	530	19.5 KB	10.6 MB	5.83	9.02	10.16	9.93	10.45
CBSNews	218	37.0 KB	8.3 MB	5.06	14.47	15.79	14.98	16.42
USAToday	344	47.5 KB	16.7 MB	6.30	9.48	9.31	8.86	9.69
CSmonitor	388	41.7 KB	16.6 MB	5.06	15.17	17.84	15.85	19.45
Ebay	100	21.5 KB	2.2 MB	6.78	11.30	11.14	10.77	11.39
Thomas-dist	105	26.5 KB	2.8 MB	6.39	9.90	9.83	9.38	10.10

Table 1: Compression ratios for collections of files using exact edge weights.

data set	pages	average size	total size	cat+gzip ratio	cat+bzip2 ratio	ob ratio	tsp ratio	tsp2 ratio
CBC	530	19.5 KB	10.6 MB	5.83	9.02	9.55	9.77	10.39
CBSNews	218	37.0 KB	8.3 MB	5.06	14.47	15.42	14.74	16.38
USAToday	344	47.5 KB	16.7 MB	6.30	9.48	9.40	9.28	9.98
CSmonitor	388	41.7 KB	16.6 MB	5.06	15.17	16.62	15.37	19.50
Ebay	100	21.5 KB	2.2 MB	6.78	11.30	9.80	10.42	11.27
Thomas-dist	105	26.5 KB	2.8 MB	6.39	9.90	9.40	9.28	9.98

Table 2: Compression ratios for collections of files using estimated edge weights.

We can see that the best overall performing method is the one using two reference files. However, even with a single reference file, the TSP-based method performs almost as well as the optimum branching approach. Finally, we observe that *bzip2* is also a very attractive option for compressing collections of similar files as it is within 15% of the compression ratio of the best method on all data sets and even closer in some cases.

Next we present results for compressing file collections using standard compression tools. Each collection was compressed with the following techniques:

- standard *tar* followed by *gzip*
- standard *tar* followed by *bzip2*
- optimal file ordering followed by *tar* and *gzip*
- optimal file ordering followed by *tar* and *bzip2*

The optimal file ordering was computed using estimated edge weights based on sampling, as described in [2] and used in [15]. We observe that the compressibility of *tar* files is typically improved by about 10%; exceptions are two benchmarks (CBSNews and CSmonitor) with fairly large average file sizes. The reason is that the *gzip* compressor uses a window size of only 32KB, and is therefore unable to fully exploit the similarity between consecutive files. (Similar files typically exhibit the same global structure, i.e., the start of the second file is similar to the start of the first file and so on, so that a 32KB window will not find a lot of redundancy even on files only slightly larger than 32KB.)

data set	pages	average size	total size	tar+gzip ratio	tar+bzip2 ratio	tsp+tar+gzip ratio	tsp+tar+bzip2 ratio
CBC	530	19.5 KB	10.6 MB	5.70	8.86	6.94	9.59
CBSNews	218	37.0 KB	8.3 MB	4.89	13.93	4.94	14.05
USAToday	344	47.5 KB	16.7 MB	6.14	9.30	6.93	9.63
CSmonitor	388	41.7 KB	16.6 MB	4.99	14.94	4.98	14.75
Ebay	100	21.5 KB	2.2 MB	6.46	11.11	7.28	11.34
Thomas-dist	105	26.5 KB	2.8 MB	6.39	9.76	6.78	9.96

Table 3: Compression ratios for batched collections of files.

6 Concluding Remarks

We have proposed an approach for compressing collection of files using deltas computed with respect to two or more reference files. This approach relies on a heuristic for solving the TSP problem on a directed weighted graph and on distance estimation techniques described in [2] and used in [15]. In addition, we have presented a method for improving the compressibility of batched files. As a future improvement, we observe that the brute force mechanism for finding additional reference files could be fairly easily substituted with a more efficient sampling-based algorithm, maybe tuned to the characteristics of a specific compression tool. As noted, such a scheme can be easily combined with a batching tool such as *tar* via a simple script that first performs a quick analysis and ordering of the files in the collection. With appropriate tuning, we believe that this analysis would result in fairly moderate CPU overhead compared to simply running *tar* and *gzip*.

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] A. Buchsbaum, D. Caldwell, K. Church, G. Fowler, and S. Muthukrishnan. Engineering the compression of massive tables. In *Proc. of the 11th Annual ACM-SIAM Symp. on Discrete Algorithms*, pages 175–184, 2000.

[4] A. Buchsbaum, G. Fowler, and R. Giancarlo. Improving table compression with combinatorial optimization. In *Proc. of the 13th Annual ACM-SIAM Symp. on Discrete Algorithms*, pages 213–222, 2002.

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

[6] G. Cormode, M. Paterson, S. Sahinalp, and U. Vishkin. Communication complexity of document exchange. In *Proc. of the ACM–SIAM Symp. on Discrete Algorithms*, January 2000.

[7] F. Douglis and A. Iyengar. Application-specific delta-encoding via resemblance detection. In *Proc. of the USENIX Annual Technical Conference*, June 2003.

[8] L. Gravano, P. Ipeirotis, H. Jagadish, N. Koudas, S. Muthukrishnan, L. Pietarinen, and D. Srivastava. Using q-grams in a DBMS for approximate string processing. *IEEE Data Engineering Bulletin*, 24(4):28–34, December 2001.

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

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

[11] 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.

[12] P. Kulkarni, F. Douglis, J. LaVoie, and J. Tracey. Redundancy elimination within large collections of files. IBM Research Report RC23042, IBM T.J. Watson Research Center, December 2003.

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

[14] U. Manber. Finding similar files in a large file system. In *Proc. of the 1994 Winter USENIX Conference*, pages 1–10, January 1994.

[15] 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.

[16] T. Suel and N. Memon. Algorithms for delta compression and remote file synchronization. In Khalid Sayood, editor, *Lossless Compression Handbook*. Academic Press, 2002.

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

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

[19] 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.

[20] B.-D. Vo and K.-P. Vo. Using column dependency to compress tables. In *Proc. of the IEEE Data Compression Conference*, pages 92–101, 2004.