Distributed Dimension Reduction Algorithms for Widely Dispersed Data*

Faisal N. Abu-Khzam†, Nagiza Samatova‡, George Ostrouchov‡,
Michael A. Langston†§, and Al Geist‡

Abstract

It is well known that information retrieval, clustering and visualization can often be improved by reducing the dimensionality of high dimensional data. Classical techniques offer optimality but are much too slow for extremely large databases. The problem becomes harder yet when data are distributed across geographically dispersed machines. To address this need, an effective distributed dimension reduction algorithm is developed. Motivated by the success of the serial (non-distributed) FastMap heuristic of Faloutsos and Lin, the distributed method presented here is intended to be fast, accurate and reliable. It runs in linear time and requires very little data transmission. A series of experiments is conducted to gauge how the algorithm's emphasis on minimal data transmission affects solution quality. Stress function measurements indicate that the distributed algorithm is highly competitive with the original FastMap heuristic.

Keywords: Data Mining, Distributed Databases, Information Systems, Parallel and Distributed Algorithms

1 Introduction

A set S of points in a d-dimensional space often belong to an embedded manifold of dimension $d' \ll d$. Classic dimension reduction techniques [3, 8, 5] compute an optimal k-dimensional representation of S for a specified $k \leq d$ and a given optimality criterion. Techniques related to principal components [3] begin with coordinates of the points, whereas those related to multidimensional scaling [8, 5] begin with a complete set of pairwise distances. All of these require at least quadratic running time, making them reasonable reduction candidates only as long as S is not too large. The focus of this paper, however, is on the case in which S is of some immense size N, with its elements distributed across a modest number s of locations. This models a variety of timely environments, for example, when massive data sets reside on a number of different, geographically dispersed machines. It is usually impractical or impossible to bring such data sets to a central location. Thus, our main objective is to reduce dimensionality in a way that does not require moving all the data, rather only some much smaller representation of the data. A similar approach is taken in [7]. A reduction in dimensionality has been shown to help in data mining and related

*Research sponsored by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory (ORNL), managed by UT-Battelle, LLC for the U. S. Department of Energy under Contract No.DE-AC05-00OR22725.

†Department of Computer Science, University of Tennessee, Knoxville, TN 37996–3450.

‡Computer Science and Mathematics Division, Oak Ridge National Laboratory, P.O.Box 2008, Oak Ridge, TN 37831–6367.

§This author's research is supported in part by the National Science Foundation under grants EIA–9972889 and CCR–0075792, by the Office of Naval Research under grant N00014–01–1–0608, and by the Tennessee Center for Information Technology Research under award E01–0178–081.

applications. For example, it can assist in effective data visualization and reveal the way the data are clustered [4, 6].

One of the major challenges researchers face in dealing with massive sets of data is algorithm scalability as the sets grow in size. Algorithms that scale as $\Omega(N^2)$ or higher quickly become computationally infeasible. Moreover, in parallel and distributed algorithms, the cost of data transmission often dominates the execution time. For these reasons, we seek a distributed dimension reduction algorithm that not only runs in linear or almost-linear time, but also requires as little data communication as possible.

Among the various alternatives available, we have chosen for exploitation the attractive FastMap heuristic [2]. It can be interpreted as an approximation to principal components that operates on pairwise distances rather than coordinates. FastMap is a linear-time serial algorithm. Even when data objects (points) are specified only by their d-dimensional coordinates, as they are in our case, FastMap runs in linear time and can serve as a dimension reduction algorithm [6]. We therefore wish to study the potential feasibility of distributed versions of this handy heuristic. Of course the naive method of bringing all data to a central location and then running FastMap requires a prohibitive amount of data transfers. We call this method Centralized FastMap, as opposed to our versions of Distributed FastMap. FastMap gained popularity in part because of empirical demonstrations of the quality of its solutions. For example, its quality was tested in [2] against that of Multi-Dimensional Scaling [8] by measuring the price/performance of each algorithm. We will similarly measure the quality of our versions of Distributed FastMap by comparing their results to those of Centralized FastMap.

In the next section, we describe FastMap in detail and discuss how it can be used as a linear-time dimension reduction technique. In Section 3, we devise two versions of Distributed FastMap. Experimental results are presented in Section 4. A final section provides a summary of our work and some insights on the performance of our algorithms.

2 An Overview of FastMap

Assuming the distance between any two elements of S is given and obeys the triangle inequality, FastMap produces a k-dimensional representation of S by projecting its points onto k carefully selected orthogonal lines. In selecting each suitable line, three points are chosen: the first is arbitrary; the second, O_a, is farthest from the first; the third, O_b, is farthest from the second. The axis, analogous to a principal component axis, is then defined solely by the pair (O_a, O_b), whose elements are henceforth termed "pivots."

FastMap proceeds iteratively. At the ith step, $1 \leq i < k$, it finds pivots to form the axis (O_{ai}, O_{bi}), and operates on the projection of S on a hyperplane, H_i, orthogonal to all previously selected axes. Let $d_0(P, Q)$ denote the original distance between points P and Q, and let $d_i(P, Q)$ denote the distance between the projections of points P and Q on H_i. Using elementary Euclidean geometry as in [2], point P's ith coordinate, P_i, is determined by using d_{i-1} in the formula

$$P_i = \frac{d_{i-1}(O_{ai}, P)^2 + d_{i-1}(O_{ai}, O_{bi})^2 - d_{i-1}(O_{bi}, P)^2}{2d_{i-1}(O_{ai}, O_{bi})}$$

and $d_i(P, Q)$ is determined by using d_0 and the coordinates just computed in the formula

$$d_i(P, Q) = \sqrt{d_0(P, Q)^2 - \Sigma_{j=1}^{i-1}(P_j - Q_j)^2}.$$

Recall that we are interested in the features problem. We must avoid computing all pairwise distances between the elements of S, a task that would consume quadratic time. To ensure a linear

```
FastMap(S, k)
begin
let ProjectionMatrix be a k × |S| matrix
let PivotsMatrix be a 2k × d matrix
for i = 1 to k do
   begin
   (O_ai, O_bi) ← ChooseObjects(S, i)
   store (O_ai, O_bi) in PivotsMatrix
   compute P_i for each point P in S
   store all P_i values in the ith row of ProjectionMatrix
   end
return ProjectionMatrix and PivotsMatrix
end
```

Figure 1: The FastMap Heuristic

```
ChooseObjects(S, i)
begin
choose arbitrary point O_c
compute distance d_{i-1}(O_c, P) for each point P in S
select point O_ai for which d_{i-1}(O_c, O_ai) is maximum
compute distance d_{i-1}(O_ai, P) for each point P in S
select point O_bi for which d_{i-1}(O_bi, O_ai) is maximum
return O_ai and O_bi as the ith pivot pair
end
```

Figure 2: The ChooseObjects Subroutine

running time, distances are therefore computed only as they are needed. Pseudo code for (the features version of) FastMap and its ancillary routine ChooseObjects is in Figures 1 and 2.

3 Distributed FastMap

We assume that S is stored as a collection of disjoint data sets, one for each of s distinct machines. Thus subset S_i is assumed to be resident on machine M_i for $i \in [1, s]$. Each element is stored in some d-dimensional representation. Pairwise distances are not given, but can be computed as previously discussed. The objective is to find k global axes of projection so that, in a new k-dimensional representation, the original distances are preserved as much as possible.

The intuition behind our approach is as follows. FastMap tends to select each pair of pivots so that they are widely separated and among the extreme points of a data set. If we have several data subsets then, by strategically choosing a few points from each one, the user might in general expect to wind up with a reasonable collection of points from which to select pivot pairs for the combined data set. We present two approaches. In each, one of the machines, say M_1, will serve as a "merger." It will obtain pivots generated locally on each machine (including the merger machine itself) and use them to choose global pivots.

Our first algorithm uses all the chosen points at one time. Each machine first runs FastMap, then sends its k local pivot pairs to the merger machine. When all pairs are received, the merger

```
OneTime(S_j, k)
begin
LocalPivots ← FastMap(S_j, k)
if j ≠ 1 then begin
   send LocalPivots to M_1
   receive GlobalPivots from M_1
   end
else begin
   Points ← LocalPivots
   for i = 2 to s do
      Points ← Points ∪ LocalPivots received
                    from M_i
   GlobalPivots ← FastMap(Points,k)
   for i = 2 to s do
      send GlobalPivots to M_i
   end
for i = 1 to k do
   compute ith global coordinate for all points in
         S_j ∪ GlobalPivots
end
```

Figure 3: The OneTime Algorithm

runs FastMap on the complete set of pivots, generates k global pairs, and broadcasts them to all other machines. It is easy to see that this strategy runs in linear time and incurs communication cost O(ksd). Our second algorithm is to iterate at each coordinate. This of course requires more send/receive cycles. It also allows all machines to work from the same projection at each iteration and so may provide better solutions. Pseudo code for each process is in Figures 3 and 4.

4 Experimental Results

We seek to compare the performance of these two fast versions of Distributed FastMap with Centralized FastMap, bearing in mind that our objective is to preserve distances as much as possible. To accomplish this we employ, as did the work reported in [2], the following well-known stress function

$$\text{stress} = \sqrt{\frac{\Sigma_{P,Q}(d'(P,Q) - d_0(P,Q))^2}{\Sigma_{P,Q} d_0(P,Q)^2}}$$

where $d_0(P,Q)$ is the original distance between points P and Q and $d'(P,Q)$ is the distance between their images in the new k-dimensional space. We refer the reader to [1] for a review of stress functions and their applications.

We performed a variety of experiments, using both real and synthetic data. We ran our distributed algorithms on different machines by randomly splitting each data set into s equal parts. Some data sets were ordered by clustering. Thus random splitting had the added benefit of ensuring that our results did not unintentionally take advantage of pre-computed structures.

Our results were roughly the same on all inputs. We illustrate with three sets of real data from the UC-Irvine repository of machine learning databases and domain theories [9]. From the data

```
Iterative(S_j, k)
begin
if j ≠ 1 then begin
  for i = 1 to k do
    begin
    (O_ai, O_bi) ← ChooseObjects(S_j, i)
    send (O_ai, O_bi) together with their new i − 1
                components to M_1
    receive new values for (O_ai, O_bi) from M_1
    compute the new ith component, P_i, for
                all P ∈ S_j
    end
  end
else begin
  Points ← φ
  for i = 1 to k do
    begin
    (O_a1, O_b1) ← ChooseObjects(S_1, i)
    Points ← Points ∪ {O_a1, O_b1}
    for j = 2 to s do
      begin
      receive (O_aj, O_bj) from M_j along with
              their new i − 1 components
      Points ← Points ∪ {O_aj, O_bj}
      end
    (O_ai, O_bi) ← ChooseObjects(Points,i)
    for j = 2 to s do
      send (O_ai, O_bi) to M_j
    end
  end
end
```

Figure 4: The Iterative Algorithm

available at this site, we show representative results on the files Pendigits.data, Glass.data, and Wine.data in Table 1.

The tables reported here bear out a common theme. In all experiments, stress values remained within about 20 percent of one another. Moreover, in a few cases, one or the other version of Distributed FastMap even provided better results than did Centralized FastMap. For example, the Iterative algorithm performed best for large values of k and s on the Pendigits data. On the other hand, for small k and large s, the Iterative version performed worst with about 20 percent increase in stress value.

The highly competitive behavior of all three algorithms may be due to the following tradeoffs. Although the Iterative version works from the same projection on each machine on every iteration, the number of points available at each iteration on the merger machine is always larger for the OneTime version (except on the last iteration). Also, both distributed versions get several pivot

Table 1: Comparison of algorithms on data from UC-Irvine repository.

Pendigits Data, Original Dimension $d = 16$

Reduced Dimension	s = 1	s = 4		s = 8	
	Centralized FastMap	OneTime	Iterative	OneTime	Iterative
k = 2	0.434022	0.434359	0.43434	0.503303	0.528707
k = 3	0.31065	0.378843	0.365004	0.366627	0.34608
k = 4	0.271444	0.261565	0.263866	0.320987	0.224068
k = 5	0.199701	0.248389	0.206782	0.209538	0.153991

Glass Data, Original Dimension $d = 9$

Reduced Dimension	s = 1	s = 4		s = 8	
	Centralized FastMap	OneTime	Iterative	OneTime	Iterative
k = 2	0.482818	0.476746	0.481473	0.476746	0.482818
k = 3	0.397119	0.397119	0.373308	0.397119	0.397119
k = 4	0.154114	0.154114	0.185656	0.154114	0.165018

Wine Data, Original Dimension $d = 13$

Reduced Dimension	s = 1	s = 4		s = 8	
	Centralized FastMap	OneTime	Iterative	OneTime	Iterative
k = 2	0.0014384	0.00143578	0.00142489	0.00142489	0.00142489
k = 3	0.00116799	0.00117015	0.00115875	0.00115625	0.00115625
k = 4	0.00107843	0.00107104	0.00106995	0.0010678	0.0010678

point pairs on the merger machine at each iteration. Choosing the best of these may outweigh the disadvantage of not considering all data points at once.

5 Conclusions

In this paper we present two Distributed FastMap algorithms for mapping high dimensional objects distributed across geographically dispersed machines into points in lower dimensional space, so that distances between the objects are preserved as much as possible. Transferring all local data to a central location and running the Centralized FastMap would require O(nd) data transmission, where n is the number of objects and d is the number of features. Our Distributed FastMap algorithms require only O(ksd) data transmission, where s is the number of data locations and k is the dimensionality of the projected space. Empirical results on both synthetic and real datasets show that our Distributed FastMap algorithms differ by at most 20 percent in accuracy, sometimes giving a loss and sometimes some gain, when compared to the Centralized FastMap.

References

[1] T. F. Cox and M. A. A. Cox. *Multidimensional scaling.* Chapman & Hall, Boca Raton, 2001.

[2] C. Faloutsos and K. Lin. FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In M. J. Carey and D. A. Schneider, editors, *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, 1995.

[3] H. Hotelling. Analysis of a complex of statistical variables into principal components. *J. Educ. Psych.*, 24:417–441,498–520, 1933.

[4] H. Kargupta, W. Huang, K. Sivakumar, and E. Johnson. Principal component analysis for dimension reduction in massive distributed data sets. *Knowledge and Information Systems*, 3:422–448, 2001.

[5] J. B. Kruskal. Nonmetric multidimensional scaling: a numerical method. *Psychometrica*, 29:115–129, 1964.

[6] J. E. Otoo, A. Shoshani, and S. W. Hwang. Clustering high dimensional massive scientific dataset. *JIIS*, 17:147–168, 2001.

[7] Y. Qu, G. Ostrouchov, N.F. Samatova, and A. Geist. Principal component analysis for dimension reduction in massive distributed data sets. In *Workshop on High Performance Data Mining at the Second SIAM International Conference on Data Mining, Washington, DC*, page in press, 2002.

[8] W. S. Torgerson. Multidimensional scaling: I. theory and method. *Psychometrica.*, 17:401–419, 1952.

[9] University of California, Irvine. Repository of machine learning databases and domain theories. See `http://ics.uci.edu/pub/machine-learning-databases`.