

COFE: A Scalable Method for Feature Extraction from Complex Objects

Gabriela Hristescu and Martin Farach-Colton

{hristesc, farach}@cs.rutgers.edu *

Rutgers University, Dept. of Computer Science, Piscataway NJ 08854, USA

Abstract. Feature Extraction, also known as Multidimensional Scaling, is a basic primitive associated with indexing, clustering, nearest neighbor searching and visualization. We consider the problem of feature extraction when the data-points are complex and the distance evaluation function is very expensive to evaluate. Examples of expensive distance evaluations include those for computing the Hausdorff distance between polygons in a spatial database, or the edit distance between macromolecules in a DNA or protein database.

While feature extraction is a well-studied problem in the databases and statistics communities, almost all methods known require that the distance between every pair of points be evaluated. This is prohibitive, even for small databases, when the distance function is expensive.

We propose COFE, a method for sparse feature extraction which is based on novel random non-linear projections. We evaluate COFE on real data and find that it performs very well in terms of quality of features extracted, number of distances evaluated, number of database scans performed and total run time. We further propose COFE-GR, which matches COFE in terms of distance evaluations and run-time, but outperforms it in terms of quality of features extracted.

1 Introduction

Feature Extraction, also known as Multidimensional Scaling (MDS), is a basic primitive associated with indexing, clustering, nearest neighbor searching and visualization. The simplest instance of feature extraction arises when the points of a data set are defined by a large number, k , of features. We say that such points are *embedded in k -dimensional space*. Picking out $k' \ll k$ features to represent the data-points, while preserving distances between points, is a feature extraction problem called the *dimensionality reduction* problem.

The most straightforward and intuitively appealing way to reduce the number of dimensions is to pick some subset of size k' of the k initial dimensions. However, taking k' linear combinations of the original k dimensions can often produce substantially better features than this naïve approach. Such approaches are at the heart of methods like Single Value Decomposition (SVD) [10] or the Karhunen-Loève transform [8]. Of course, linear combinations of original dimensions are but one way to pick features. Non-linear functions of features have the potential to give even better embeddings since they are more general than linear combinations.

* Supported by NSF Award CCR-9820879

In many other cases the points are complex objects which are not embedded. For example, if the dataset consists of DNA or protein sequences, then there is no natural notion of a set, large or otherwise, of orthogonal features describing the objects. Similarly, in multimedia applications, the data may include polygons as part of the description of an object. Once again, polygonal shapes can be described as a sequence of points along the convex hull, but such a description does not constitute a description of the object in a feature space. While such data types are not represented in a feature space, they are typically endowed with a *distance function*, which together with the dataset of objects define a *distance space*. For example, the distance between biological macromolecules is taken to be some variant of the edit distance between them. For geometric objects in 2- or 3-dimensions, the distance is often measured as the Hausdorff distance. These distance functions are very expensive to compute. The edit distance between two sequences of length m takes $O(m^2)$ time to compute, while the Hausdorff distance between two geometric objects, each with m points, takes time $O(m^5)$ to compute. Even though complex objects have a finite representation in the computer, the natural feature space this representation describes does not preserve distances between objects. For example, the k points of the polygon can be trivially represented by $O(k)$ dimensions by a straightforward $O(k)$ bit computer representation, but the vector distance between such embeddings is unrelated to any natural geometric distance between the polygons.

The *Complex Object Multidimensional Scaling (COMDS)* problem is then the problem of extracting features from objects given an expensive distance function between them. A good solution to the *COMDS* problem has to have good quality: **Quality:** The k features extracted should reflect, as closely as possible, the underlying distances between the objects. Furthermore, the extracted features should be good even with small k . If we are interested in visualization $k = 2, 3$. Clustering and nearest neighbor searching becomes prohibitively expensive, or the quality of the clustering degrades, if k is more than about 10. Thus the quality of a *COMDS* algorithm depends on the quality of a small number of extracted features. There is a tradeoff between the quality and the scalability of a solution to the *COMDS* problem.

A scalable solution should have the following characteristics:

Sparsity: Since the distance function is very expensive to evaluate, it is not feasible to compute all $\binom{n}{2}$ pairwise distances, where n is the number of elements in the database. Thus, the method must compute only a sparse subset of all possible distances.

Locality: As many databases continue to grow faster than memory capacity, the performance of any *COMDS* solution will ultimately depend on the number of accesses to secondary storage. Therefore, a *COMDS* solution should have good locality of object references. This can be measured by the number of database scans necessary to compute the embedding. Thus, this number should be low.

We address these issues in designing COFE, an algorithm for the *COMDS* problem. We evaluate COFE and compare it with FastMap [7], a previously proposed solution for *COMDS*.

Features define a metric space. The standard way to define the distance between two k -dimensional feature vectors is through their l_2 (Euclidean) distance, that is, if point p has features p_1, \dots, p_k and point q has features q_1, \dots, q_k , we can interpret the “feature distance” $d'(p, q) = \sqrt{\sum_{i=1}^k (p_i - q_i)^2}$. Taken in this view, we seek the embedding of the real distance function $d(\cdot, \cdot)$ into \mathbb{R}^k , such that the induced l_2 distance function $d'(\cdot, \cdot)$ is as similar to $d(\cdot, \cdot)$ as possible. Why should we choose the Euclidean distance between feature vectors? The reason is clear: a considerable amount of database work has focused on indexing and related tasks for low dimensional Euclidean data. No other metric combines the robustness of a low dimensional Euclidean space with its tractability. For a survey of techniques which can be applied to low-dimensional Euclidean data, see [7].

Measuring the Quality of an Embedding. We compared embeddings based on the *Stress* criterion proposed in [13], which is a standard way to measure the quality of an embedding. The *Stress* of a distance function $d(\cdot, \cdot)$ with respect to

another distance function $d'(\cdot, \cdot)$ is defined as: $Stress(d, d') = \sqrt{\frac{\sum_{i,j} (d(i,j) - d'(i,j))^2}{\sum_{i,j} d'(i,j)^2}}$.

2 Related Work

Multidimensional Scaling (MDS) has been used in the literature with different meanings, though in this paper we restrict ourselves to the standard meaning within the database community [14], as described above. MDS can be applied as a preprocessing step for data to be used in indexing, clustering and related tasks. The method we propose, called COFE, is a scalable MDS method.

Traditional MDS methods [15] do not offer a scalable solution to the *COMDS* problem, because of high complexity and unrealistic resource assumptions. In [7], Faloutsos and Lin proposed the FastMap method for MDS. Their innovation was to consider sparsity as an important factor for MDS and FastMap is the first method we know of which maintains quality and sparsity simultaneously. However, Faloutsos and Lin did not consider the problem of scalability – the largest instance which they used to evaluate their method had 154 points, consisted of 13-dimensional categorical data – and FastMap is not optimized for complex objects. Faloutsos and Lin [7] discuss traditional MDS methods [15], their drawbacks and compare them with FastMap. We describe FastMap in detail below, and make extensive comparisons between FastMap and COFE.

Once an MDS method has embedded the database in low-dimensional Euclidean space, efficient methods for indexing or clustering, to support nearest neighbor search or visualization, can be applied. A considerable amount of database work has focused on indexing, clustering and related tasks for low dimensional Euclidean data. Here, we mention briefly some of the key results in this area, in particular for the problem of *similarity querying*, which has been used to mean nearest neighbor searching in a database.

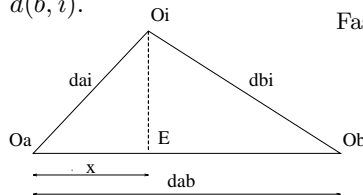
Multi-dimensional index structures called Spatial Access Methods (*SAMs*) were designed to rely on certain clustering properties of objects in low-dimensional space. If these assumptions hold, we would expect objects to occupy regions of the multi-dimensional space that can be captured by the multi-dimensional in-

dex. The result of previous research [3, 4, 20] indicated that the indexing techniques, which are designed for low-dimensional data, do not perform equally well on high-dimensional data. Some of the most popular database indexing structures used for similarity querying are the R^* -tree [5], X -tree [4], SS -tree [20], SR -tree [12], TV -tree [16], the Pyramid Technique [2], PK -tree [21]. BUBBLE and BUBBLE-FM [9] are two recently proposed algorithms for clustering datasets in arbitrary distance spaces, based on BIRCH [22], a scalable clustering algorithm. BUBBLE-FM uses FastMap to improve the scalability of the method.

3 Embedding Methods

FastMap

The approach taken in FastMap [7] for embedding points into k -dimensional Euclidean space is based on 3-point linear projections. Consider three objects O_a , O_b and O_i , and the distances, $d(a, b)$, $d(a, i)$ and $d(b, i)$, between them. Any such distances which satisfy the triangle inequality can be represented exactly in two-dimensional Euclidean space (Figure 3). O_a can be assumed to be at position $(0, 0)$, and O_b at $(0, d(a, b))$. To find O_i 's two coordinates (x_i, y_i) , we can solve two equations with two unknowns, plugging in the values $d(a, i)$ and $d(b, i)$.



FastMap uses the following procedure to find a feature:

1. Pick a set of reference points, O_a and O_b .
2. For each other point O_i , its feature is the projection x onto the line (O_a, O_b) .

Fig. 1. Projecting point O_i on the line $O_a O_b$

So each feature is a linear projection defined by a reference set O_a and O_b . The question is then how to pick such a reference pair. Faloutsos and Lin's aim was to select a pair along the principal component, *a la* S.V.D. Since it would be computationally expensive to find the principal component of the points, they suggest finding the pair which is furthest apart, which they suggest should lie more or less along the principal component. But finding such pair is also computationally expensive! So the following heuristic is used: Start by picking a point O_a at random. Find the furthest point to O_a by computing its distance to all others and consider it O_b . Compute O_b 's distance to all others and replace O_a with the most distant point. Repeat the last two steps t times, where t is a parameter of the algorithm. The last pair of points O_a and O_b found this way are taken to be the reference points. Note that this sketch only describes how the first feature is selected. All distance calculations for the second feature factor out the part of the distance function which has already been captured by the first feature. Otherwise, if O_a and O_b are the furthest points originally, they will remain so and all features will be identical! The selection of a reference set always requires t one-against-all sets of distance evaluations. Thus, selecting the reference points requires $O(tn)$ distance evaluations. Computing the feature once the reference set is known requires another $2n$ distance evaluations. Therefore, k dimensions require $O(tkn)$ distance evaluations.

FastMap-GR

Once a method has extracted k features, the dimensionality of the embedding space can be further reduced by selecting a smaller set k' of high quality features from the initial k . The quality of a feature can be quantified in terms of how well it reduces the *Stress*. We propose the *Greedy Resampling* algorithm for picking out the best subset of features, which has the advantage of sparsity over traditional methods like SVD [10], the Karhunen-Loève transform [8] and even the recent and more efficient SVD-based method [11].

FastMap-GR starts by picking out k features using FastMap. Then, each single feature is compared in terms of what *Stress* it produces. Notice that computing the *Stress* directly seems to require $\binom{n}{2}$ distance evaluations, where n is the size of the dataset, but we can simply pick out a random sample of distances and compare the *Stress* on just these distances. Thus, picking out the best feature is very fast compared to finding the k features to begin with. Once some feature f_1 is selected as the best, we can pick the second feature as the one which produces the best *Stress* when combined with f_1 . We can proceed in this greedy manner until we have reordered all k features by decreasing order of quality.

Having done this, if we need $k' < k$ features, we can simply take the first k' features of the reordered embedding. Thus, *Greedy Resampling* is a dimensionality reduction method. Note that it does not guarantee to pick out the best k' features, but picking out the best k' features takes exponential time, and so we suggest greedy resampling as a heuristic.

Bourgain

Bourgain's method [6] is not a sparse method, that is, it evaluates all $\binom{n}{2}$ possible distances between the n objects of the database. We will therefore not compare its performance with those of the other methods presented in this section. However, COFE is based on the Bourgain embedding, so we present this method for the sake of exposition.

Suppose we have a set X of n elements. Let the distance function between elements of X be $d : X \times X \rightarrow \mathfrak{R}^+$. We define the distance function D between an element of X and a subset $X' \subseteq X$ as $D(x, X') = \min_{y \in X'} \{d(x, y)\}$, that is, $D(x, X')$ is the distance from x to its closest neighbor in X' . Let $R = \{X_1, X_2, \dots, X_k\}$ be a set of subsets of X . Then we can define an embedding with respect to R as follows: $E_R(x) = [D(x, X_1), D(x, X_2), \dots, D(x, X_k)]$. It is not obvious at first why such an embedding might have any reasonable properties. For one thing, it is highly non-linear, and so visual intuition tends not to be too helpful in understanding the behavior of such an embedding. However, we can gain some understanding of this type of embedding by considering the case where the points of the metric space are packed tightly into well separated clusters. Then, if some set X_1 has a point which is in one cluster C_1 and not in another C_2 , the distance $D(x, X_1)$ will be small for $x \in C_1$ and large for $x \in C_2$. The dimension corresponding to X_1 will then induce a large distance between points in C_1 and C_2 . Obviously, we cannot count on the input points being packed into well-separated clusters, or that the sets X_i have the right ar-

rangements with respect to the clusters. However, Bourgain showed that there is a set of reference sets which works for general inputs.

The Bourgain embedding, then, is a choice of reference sets, R , and the corresponding embedding E_R . The choice of the sets in R will be randomized via a procedure described below: R consists of $O(\log^2 n)$ sets $X_{i,j}$, which we can think of as naturally organized into *columns* (κ) and *rows* (β). Let $R = \{X_{1,1}, X_{1,2}, \dots, X_{1,\kappa}, X_{2,1}, \dots, X_{2,\kappa}, X_{\beta,1}, \dots, X_{\beta,\kappa}\}$, where $\kappa = O(\log n)$ and $\beta = O(\log n)$. The Bourgain embedding will thus define $O(\log^2 n)$ dimensions. We select the elements of $X_{i,j}$ to be any random subset of X of size 2^i . Thus, we get κ sets of size 2, κ of size 4, etc., up to β of size approximately n . Here is an example of the Bourgain embedding. Let $d(\cdot, \cdot)$ be given by the following matrix:

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
x_1	0	12	10	9	10	6	3	10
x_2		0	8	13	14	10	13	8
x_3			0	11	12	8	11	2
x_4				0	3	7	10	11
x_5					0	8	11	12
x_6						0	7	8
x_7							0	11
x_8								0

The algorithm picks 6 reference sets at random, 3 with 2 elements, and 3 with 4 elements:

x_3, x_4	x_1, x_8	x_2, x_6
x_4, x_5, x_2, x_3	x_1, x_4, x_6, x_3	x_7, x_5, x_3, x_6

The embedding of x_3 and x_5 is computed as:

$$E_R(x_3) = \begin{bmatrix} 0 & 2 & 8 \\ 0 & 0 & 0 \end{bmatrix} \quad \text{and} \quad E_R(x_5) = \begin{bmatrix} 3 & 10 & 8 \\ 0 & 3 & 0 \end{bmatrix}$$

$$l_2(E_R(x_3), E_R(x_5)) = \sqrt{3^2 + 8^2 + 0 + 0 + 3^2 + 0} \approx 9.055 \text{ and the original distance is } 12.$$

An excellent presentation of the proof of the correctness for the Bourgain embedding can be found in [17]. Bourgain showed [6] that if R is selected as above, then the embedding E_R has *distortion* $O(\log n)$, where the distortion of an embedding is the maximum stretch of any distance, that is $d(x_i, x_j) \leq \log n l_2(E_R(x_i), E_R(x_j))$, and $d(x_i, x_j) \geq l_2(E_R(x_i), E_R(x_j)) / \log n$.

It seems at first glance that modifying a distance by as much as a $\log n$ *multiplicative* factor would almost completely destroy the original information contained in the distance function $d(\cdot, \cdot)$. For example, if we are embedding 1024 points, and $d(\cdot, \cdot)$ ranges over a factor of 10, a $\log n$ distortion embedding could arbitrarily reorder the distance between objects in our dataset. However, as with many worst case bounds, the Bourgain guarantee of no more than $\log n$ distortion is often very far from the actual behavior on real data. Furthermore, the $\log n$ bound is tight in the sense that there are metrics which require this much distortion for any Euclidean embedding. Experimental analysis on meaningful data rather than on concocted counterexamples is the *sine qua non* of feature extraction evaluation.

The Bourgain embedding has two very serious drawbacks. First, it produces $O(\log^2 n)$ dimensions. Again, if we are embedding 1024 points, we could end up with 100 dimensions, which would be far too many. Second, with high probability, every point in the dataset will be selected in some reference set $X_{i,j}$. Thus, we will need to compute all $\binom{n}{2}$ distances in order to perform the Bourgain embedding. If we are using the embedding for similarity searching, we would need to compute $E_R(q)$ for a query point q . This would require a distance evaluation between q

and every point in X . There would be no point in performing an embedding if we were willing to evaluate all n distances between q and the points in X , since we are using the embedding in order to facilitate a nearest neighbor search.

Similarly, the Karhunen-Loève is completely impractical. FastMap was, in fact, designed to give a practical alternative to the Karhunen-Loève transform.

Cofe

COFE is a heuristic modification of the Bourgain method which is designed to perform very few distance evaluations. The key idea in COFE is a loop interchange. Bourgain’s method is typically thought of as having an outer loop which runs through all the points, and then an inner loop which computes all the features. We can equivalently imagine computing the first feature for every point, then the second feature, etc., until we have computed however many features we want. So far, we have not changed anything substantial: whether we loop on all points and then on all reference sets, or on all reference sets and then on all points, we end up doing the same distance evaluations. Notice that the features are defined by a two-dimensional array. Thus, the order in which we loop through the features is not determined. For reasons which are suggested by the proof of the distortion bound of Bourgain’s algorithm, we will proceed row-by-row, that is, we will compute features for all reference sets of size 2, then for size 4, etc.

So finally, we need to know how to actually save on distance evaluations. The main idea is as follows: Suppose we already computed k' features for every point. Then we have a rough estimate on the distances between points, that is $d'_{k'}(p, q) = \sqrt{\sum_{l=1}^{k'} (p_l - q_l)^2}$, where p_l is the l^{th} feature of p , and similarly for q . If the first k' features are good enough, then we can very quickly compute the approximate distance from some point to all points in a reference set, using these first k' features. We can then consider just the best few candidates, and perform full distance evaluations to these points. Concretely, for every point p in the dataset X and every reference set X_k (taken in row-major order) the approximate distance $\tilde{D}(p, X_k)$ is computed as follows:

- For every point $q \in X_k$, compute the approximate distance $d'_{k-1}(p, q)$.
- Select the σ points in X_k with smallest d' distance to p .
- For each point q of these σ points, evaluate the true distance $d(p, q)$.
- Then $\tilde{D}(p, X_k) = d(p, q')$, where q' is the q with smallest $d(p, q)$.

$\tilde{D}(p, X_k) = \min_{y \in S} \{d(p, y) \mid S \subseteq X_k, |S| = \sigma, \forall a \in S, \forall b \in X_k \setminus S, d'(p, a) \leq d'(p, b)\}$. Thus, for every point, and for every feature, we compute σ distances. We get $O(nk\sigma)$ distance evaluations, if there are n points and k features.

There are two questions one might ask about the COFE embedding: does it do well in theory and does it do well in practice. The second question, which is the subject of this paper, is addressed in Section 4. As far as the theory goes, COFE does not provide the distortion guarantees of Bourgain. Indeed, we have shown that *no* algorithm which evaluates any proper subset of the values of $d(\cdot, \cdot)$ can give such a guarantee. However, as noted above, a guarantee of $\log n$ distortion is not really all that useful, so we turn our attention to the matter of evaluating the performance of COFE on data.

Cofe-GR

Recall FastMap-GR, which uses FastMap to find k features and then reorders them using the Greedy Resampling heuristic. COFE-GR is analogous: it uses COFE to find k features and then uses Greedy Resampling to reorder the features. We will show that COFE-GR does a very good job of picking out the best features from a COFE embedding.

4 Experimental Comparison

We compared COFE, COFE-GR, FastMap and FastMap-GR in terms of the quality of the computed embedding, the execution time for computing the embedding and the number of distance evaluations performed during the computation.

Case study: Protein datasets

We chose proteins as test data, because the distance function between them is very expensive to compute. If proteins are compared at the structure level, just one distance computation can take minutes and even hours. Comparing proteins at sequence level is faster, but still computationally high, if no specialized hardware is used. We ran experiments on datasets selected randomly from the SwissProt protein database [1]. In [7], only very low dimensional data was used, such as 3 to 13 dimensional vector data, with the exception of text data. However, computing the distance between text documents, once such preprocessing as finding word-counts has been completed, is not expensive and thus not as relevant to our method.

The size of the protein datasets we used ranged from 48 to the whole SwissProt database consisting of 66,745 proteins. The platform we used for these experiments was a dual-processor 300 MHz PentiumII with 512 MB of memory. COFE and COFE-GR were implemented in C and tested under Linux. We used the original FastMap code supplied by Faloutsos and Lin [7]. Due to excessive memory requirements we were not able to run FastMap on datasets larger than 511. Thus, most of our comparisons will be the average of 10 random subsets of size 255 from SwissProt. We find the performance of FastMap and COFE not to degrade significantly with size, and so these data are quite representative.

We used the Smith-Waterman [19] measure for protein similarity, which is based on a dynamic programming method that takes $O(m_1m_2)$ time, where m_1 and m_2 are the lengths of the two protein sequences. Since this is a very expensive operation, it will significantly affect the running time of the methods that use it. If $s(a, b)$ is the similarity value between proteins a and b , the distance measure between the two proteins we considered is $d(a, b) = s(a, a) + s(b, b) - 2s(a, b)$ as proposed by Linial & al. [18].

The Quality of Embeddings

For the quality of the embedding we used the *Stress* measure suggested by Faloutsos et al. [7] which gives the relative error the embedded distances suffer from on average. The figures report results for embeddings of sizes ranging from one to the maximum number of features considered. We used $t=2$ for FastMap.

Notice that in COFE we can select a number of rows (α) and columns (κ) to evaluate. The number of features extracted will then be $k = \alpha\kappa$. We used the full Bourgain dimensionality with $\kappa = \alpha = \log n$ and $\sigma = 1$ for COFE.

We measured the *Stress* on 4,000 distances between randomly selected pairs of elements from the dataset. In our comparisons we present the average over results on 10 sets of 255 proteins.

Experimental Results

Comparing the Quality of the Embedding. We begin by comparing COFE with FastMap and COFE-GR with FastMap-GR. Results show that for FastMap (Fig-

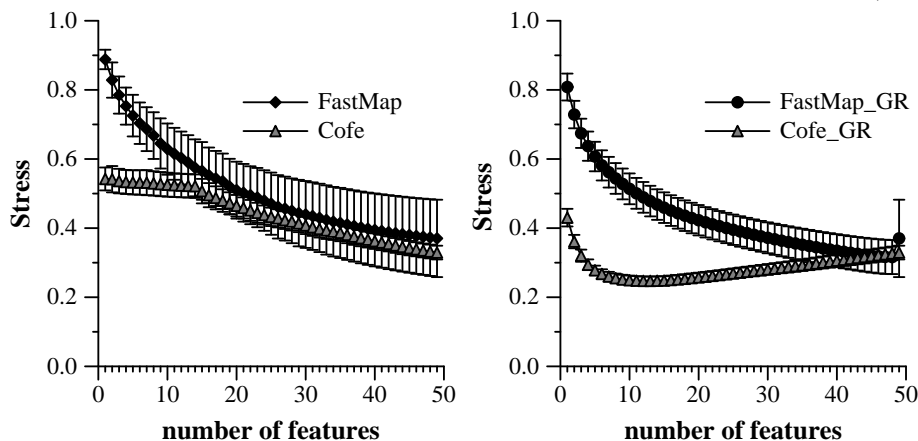


Fig. 2. Average quality of COFE vs. FastMap and COFE-GR vs. FastMap-GR on 10 sets of 255 proteins, with error bars

ure 2) the *Stress* starts at 0.88 for the first feature extracted and constantly decreases with increasing number of features, reaching 0.37 for 49 features, which constitutes a 58% reduction. For the same range of features, COFE starts with a much lower *Stress* value (0.54) and ends at 0.33, although the *Stress* between the two methods after 14 features is basically indistinguishable. Comparing COFE-GR with FastMap-GR (Figure 2) we notice the quality difference between them. COFE-GR starts by being 46% better than FastMap for the first extracted feature and stays around 50% for the first 10 features, after which the gap in quality decreases to almost 0 for 46 features.

The thing to note is that COFE-GR has its best *Stress* at 10 dimensions, and this *Stress* is not matched by FastMap-GR, even out to 49 dimensions (though if sufficient dimensions were used, FastMap would probably produce a lower-*Stress* embedding). As noted above, it is quality of an embedding with a small number of dimensions which is most important, and so we conclude that COFE-GR produces very high quality embeddings with a small number of features.

Experiments on larger and smaller sets showed basically no difference in the relative performance of the methods and were thus omitted from this paper.

The Effects of Greedy Resampling. Intuitively, the greedy resampled versions of the algorithms should have lower *Stress* for any given number of features, unless, of course, we use all features, in which case the greedy resampled algorithms will match their non-resampled versions. The comparison of the two plots in Figure 2 confirms this, although we note that greedy resampling is a heuristic and that it is possible to come up with pathological embeddings where greedy

resampling does not yield good reduction of dimensionality. A striking aspect of the results is that COFE-GR is a significant improvement over COFE, while FastMap and FastMap-GR have roughly similar performance. COFE-GR starts by improving the quality of the first selected feature by 20% over COFE and reaches the minimum *Stress* of 0.25 for 10 features, whereas FastMap-GR starts with a 9% improvement over FastMap and reaches the minimum value of 0.31 for 46 features. This comparison shows that COFE can significantly benefit from resampling, while FastMap selects the sequence of choices of reference sets very close to the one picked through greedy resampling on the same reference set.

Comparing the Execution Time. In addition to comparing the quality of the two methods, we were also interested in comparing their execution time. We

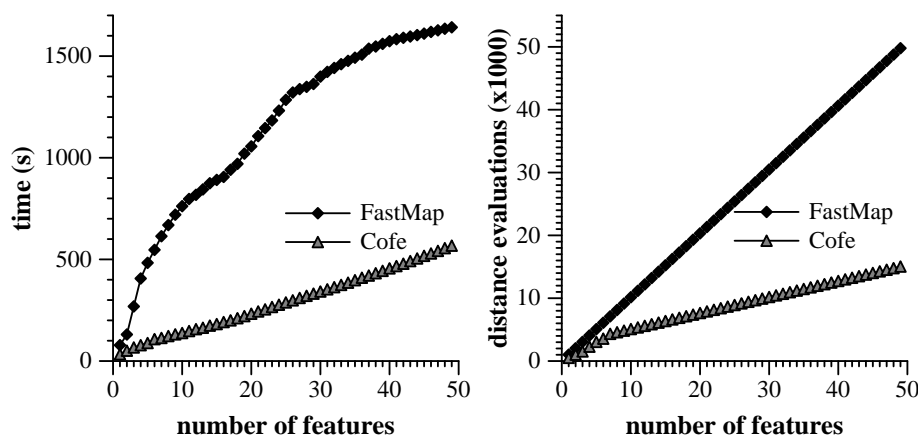


Fig. 3. The average run time and the number of distance evaluations for COFE vs. FastMap for 255 proteins

instrumented the code to get timing information and also profiled the execution in order to understand where the time was spent. Recall that greedy resampling does not require a significant number of distance evaluations and very little run time, so we compare only COFE with FastMap.

Figure 3 presents the average execution time over runs on 10 datasets of size 255. It shows results for increasing number of features for COFE and FastMap. COFE outperforms FastMap by being 2.3 times faster than it for the first feature and 2.8 times faster for all 49 features.

We profiled both methods to determine where the bottlenecks were in the computation. As expected, in both cases, more than 90% of the execution time was spent in the computation of the distance between data points (proteins). Therefore, we were able to explain the difference in execution time by the difference in the frequency of distance computations between the two methods. However, the run times measured require some explanation. We would have expected FastMap to run about 33% slower than COFE since it performs about 4/3 as many distance evaluations in theory. However, two effects come into play here. First, FastMap performs more scans of the data, and we would expect it to pay a penalty in terms of slower memory access. Also, the reference set is

known ahead of time in COFE. This allows many optimizations, both in terms of possible parallelization, as well as in savings of repeated distance evaluations. Such optimizations are not possible for FastMap because the reference sets are only determined dynamically. Thus, Figure 3 presents actual measured data of optimized code.

In Figure 3 we compare the number of distance calls for a dataset of 255 protein sequences. For a 49 feature embedding 46% of the total number of pairwise distances were computed for COFE and 153% for FastMap. Even for one feature FastMap computes 3.1% of distances compared to 1.5% for COFE. Even if we use the lowest value for $t = 1$, which means that one reference point is completely random, FastMap makes 115% of the total number of pairwise distance evaluations for 49 features extracted and 2.3% for the first feature.

Scalability. We studied how the COFE method scales with increasing dataset sizes both in terms of quality and running time. Figure 4 shows the quality of the first

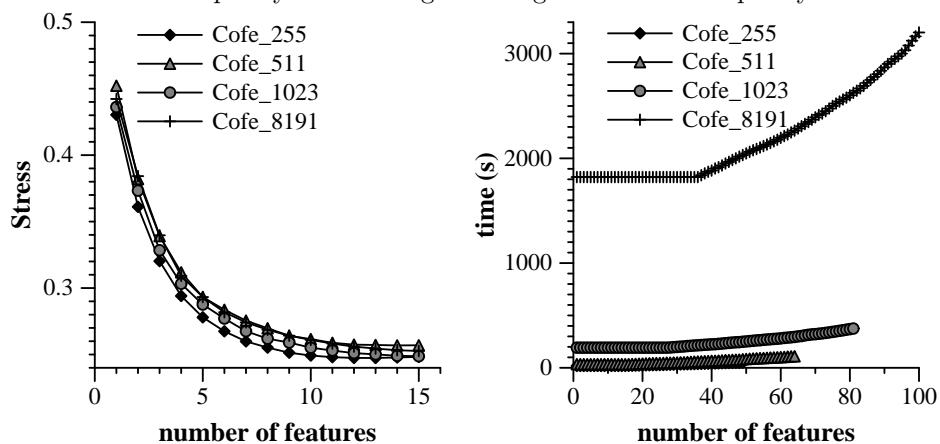


Fig. 4. Quality of COFE-GR features and run time for COFE for protein sets of sizes ranging from 255 to 8191

15 dimensions of COFE-GR. We note very consistent behavior, with basically no degradation in the solution over a wide range of dataset sizes. Figure 4 shows the measured run time of COFE over a variety of data set sizes and number of features extracted. The flat initial part of the curve is due to startup costs.

5 Conclusions

We have proposed fast algorithms, COFE and COFE-GR which map points into k -dimensional space so that distances are well preserved. We assume only that we have a distance function between pairs of data-points. Such a setting is very natural for data types such as biological sequences and multimedia data. Since distance evaluations are very expensive, reducing the number of distance evaluations, while preserving the quality of our embedding was our main goal.

We compared our COFE method with FastMap [7] on sets of protein sequences, but due to excessive memory requirements, we were not able to run FastMap on datasets larger than 511. Therefore, most of our comparisons are the average over 10 protein datasets, each of size 255. We found that our method

performs substantially fewer distance evaluations than FastMap. Furthermore, the *Stress* of the embedding is about half that of FastMap's for a reasonable number of dimensions. FastMap and COFE have very different approaches: COFE performs random non-linear projections while FastMap performs geometric projections. We conclude that COFE is an extremely effective solution to the *Complex Object Multidimensional Scaling* problem, exhibiting excellent scalability and yielding high quality features.

References

1. Bairoch, A., Apweiler, R.: The SWISS-PROT protein sequence data bank and its supplement TrEMBL in 1998. *Nucleic Acids Res.* **26** (1998) 38–42
2. Berchtold, S., Böhm, C.: The Pyramid-Technique: Towards breaking the curse of dimensionality. *Proc. ACM SIGMOD Conf.* (1998) 142–176
3. Berchtold, S., Böhm, C., Keim, D.A., Kriegel, H.-P.: A cost model for nearest neighbor search in high-dimensional data space. *Proc. ACM PODS Symposium* (1997)
4. Berchtold, S., Keim, D.A., Kriegel, H.-P.: The X-tree: An index structure for high-dimensional data. *Proc. 22nd VLDB Conf.* (1996) 28–39
5. Beckmann, N., Kriegel, H.-P., Schneider, R., Seeger, B.: The R^* tree: An Efficient and Robust Access Method for Points and Rectangles. *Proc. ACM SIGMOD Conf.* (1990) 322–331
6. Bourgain, J.: On Lipschitz embedding of finite metric spaces in Hilbert space. *Israel J. of Math.* (1985) 52:46–52
7. Faloutsos, C., Lin, K.-I.: FastMap: a fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. *Proc. ACM SIGMOD* **24(2)** (1995) 163–174
8. Fukunaga, K.: *Introduction to Statistical Pattern Recognition*. Academic Press, 2nd Edition (1990)
9. Ganti, V., Ramakrishnan, R., Gehrke, J., Powell, A., French, J.: Clustering large datasets in arbitrary metric spaces. *Proc. 15th ICDE Conf.* (1999) 502–511
10. Golub, G. H., Van Loan, C. F.: *Matrix computations*. Johns Hopkins University Press, 2nd Edition (1989)
11. Kanth, K. V. R., Agrawal, D., Singh, A.: Dimensionality reduction for similarity searching in dynamic databases. *Proc. ACM SIGMOD Conf.* (1998) 142–176
12. Katayama, N., Satoh, S.: The SR-tree: An index structure for high-dimensional nearest neighbor queries. *Proc. ACM SIGMOD Conf.* (1997) 369–380
13. Kruskal, J.B.: Multidimensional Scaling by Optimizing Goodness of Fit to a Non-metric Hypothesis. *Psychometrika* **29** (1964) 1–27
14. Kruskal, J.B.: *Multidimensional Scaling and other Methods for Discovering Structure*. *Stat. Meth. for Digital Computers*, Wiley, New York (1977) 296–339
15. Kruskal, J.B., Wish, M.: *Multidimensional Scaling*. Sage University Paper series on Quantitative Applications in the Social Sciences, Beverly Hills, CA (1978) 7–11
16. Lin, K.-I., Jagadish, H. V., Faloutsos, C.: The TV-tree: An index structure for high-dimensional data. *Proc. 20th VLDB Conf.* **3(4)** (1994) 517–542
17. Linial, N., London, E., Rabinovich, Y.: The geometry of graphs and some of its algorithmic applications. *Proc. 35th IEEE FOCS Symp.* (1994) 577–591
18. Linial, M., Linial, N., Tishby, N., Yona, G.: Global self organization of all known protein sequences reveals inherent biological signatures. *J. Mol. Biol.* **268** (1997) 539–556
19. Smith, T., Waterman, M.: The identification of common molecular subsequences. *J. Mol. Biol.* **147** (1981) 195–197

20. White, D. A., Jain, R.: Similarity Indexing with the SS-tree. Proc. 12th ICDE Conf. (1996) 516–523
21. Wang, W., Yang, J., Muntz, R. R.: PK-tree: A Spatial Index Structure for High Dimensional Point Data. 5th Intl. FODO Conf. (1998)
22. Zhang, T., Ramkarishnan, R., Livny, M.: Birch: An efficient data clustering method for large databases. Proc. ACM SIGMOD Conf. (1996) 103–114

A Analytical Comparison

In this Appendix we compare COFE, COFE-GR, FastMap and FastMap-GR in terms of the following three criteria: quality, sparsity and locality.

Quality. As stated in Section 3, COFE is based on the Bourgain algorithm, which guarantees a distortion of $O(\log n)$ for any metric, and there are some metrics which require $\Omega(\log n)$ distortion. It is easy to show that no sparse method can have such a bound. Consider, for example, the *Needle-in-a-haystack* distance function. All distances are some large Δ , except, perhaps, for some unknown pair a, b which are at distance $\epsilon \ll \Delta$ apart. Unless we find the pair a, b , we do not know on any unevaluated distance if it is Δ or ϵ . Thus we cannot provide an embedding with distortion of less than Δ/ϵ , a quantity which is not even bounded. But this is not a shortcoming of COFE, it is a shortcoming of *every sparse method*, including FastMap.

Similarly, FastMap is based on the Karhunen-Loève transform, which gives some optimality guarantees about picking out the best *linear projections*. Notice that it gives no guarantees on finding the best features overall, only the best linear features, under some criterion. FastMap is a heuristic and gives no guarantees. We conclude by noting that there is no sparse method which gives a guarantee under any criterion.

Sparsity. Above, we showed that FastMap performs $O(tkn)$ distance evaluations, where t is a parameter of the algorithm and COFE performs $O(\sigma kn)$ evaluations. The greedy resampling versions of these algorithms do not perform any significant number of extra distance evaluations.

A comparison of the exact number of distance evaluations performed requires knowing t and σ , and analyzing the hidden constants in the O notation. If we consider a dataset with n elements, the average computational cost of a distance evaluation in the distance space to be D and the cost of computing the Euclidean distance in an i -dimensional space to be $d_i = id$ where d is the cost of computing a squared difference. We consider the number of features extracted to be k .

FastMap first chooses a pair of points that define a reference line and then projects all points in the dataset on that line. At the next step, this procedure is repeated, except that the distances between points are considered projected in a hyperplane orthogonal to the chosen reference line. This introduces, in addition to a distance evaluation in the distance space, an extra d computation for each feature already extracted. The time to execute FastMap consists of the time $T_{extract}$ to extract the k features and the time $T_{project}$ to project the n points onto them. If for each feature extracted, a number of t passes over the dataset are executed while on each pass the furthest point to a selected reference point is computed, then $T_{extract} = t \sum_{i=1}^n \sum_{j=1}^k (D + (j-1)d)$ and if c distance computations are needed to project a point onto a reference line, then $T_{embed} = c \sum_{i=1}^n \sum_{j=1}^k (D + (j-1)d)$. The number of distance computations c for projecting a point on a line is 2, considering that the distance between the reference points is known. Thus: $T_{FastMap} = T_{extract} + T_{embed} = nk(t+c) \left(D + \frac{k-1}{2}d \right)$.

Computing the Bourgain transform would require randomly selecting the reference sets and then computing the embedding of a point in the image space as the smallest

distance between the point and the points in the reference sets. Selecting the reference sets does not require any computation, and therefore for an embedding with κ columns and α rows: $T_{Bourgain} = 2n\kappa(2^\alpha - 1)D$.

Instead of the α rows Bourgain would compute, COFE computes only β rows using the distance in the distance space. For the rest of $\alpha - \beta$ rows, the coordinate for the corresponding dimension i is approximated by first selecting the σ closest points in the already computed $(i - 1)$ -dimensional image space, then computing the smallest distance in the distance space among the σ points.

Extending the β row embedding to an α row embedding takes therefore:

$T_{extension} = n\kappa\sigma(\alpha - \beta)D + n \sum_{i=\beta+1}^{\alpha} \left(2^i \sum_{j=\kappa(i-1)}^{\kappa i - 1} (jd) \right)$. Approximating $j \leq \kappa i$ we get: $T_{Cofe} \leq n\kappa (2^{\beta+1} - 2 + \sigma(\alpha - \beta)) D + n\kappa ((\alpha - 1)2^{\alpha+1} - \beta 2^{\beta+2} + 4) d$.

Because computing D is by far more expensive than d , the dominant terms for the two methods are: $T'_{FastMap} = n\kappa\alpha(t + c)D$ and $T'_{Cofe} = n\kappa (2^{\beta+1} + \sigma(\alpha - \beta))D$, where the size of the embedding is $k = \kappa\alpha$.

For $\alpha = \log n$ and $\beta = \log \log n$:

$T'_{FastMap} = n\kappa(t + c)\log n D$ and $T'_{Cofe} = n\kappa((\sigma + 2)\log n - \sigma \log \log n)D$. For this choice of parameters, both methods compute $O(\log n)$ real distances. Considering $c = 2$, and $k = \kappa \log n$, FastMap performs more or less $(t + 2)kn$ distance evaluations and COFE performs approximately $(\sigma + 2)kn$ distance evaluations. The values for t that would give comparable numbers of distance computations in the distance space are $t = \sigma$ or $t = \sigma + 1$.

Faloutsos and Lin suggested that $t = 7$, though experiments showed that on the data used, very little advantage is gained by setting $t > 2$. We found that setting $\sigma = 1$ gave very reasonable result, so we set $t = 2$ and conclude that, as n increases, FastMap will perform $\frac{t+2}{\sigma+2} = \frac{4}{3}$ times as many distance evaluations.

Locality. In order to select one feature, FastMap has to scan the data t times and then compute the embedding (projection) using one more scan. For extracting k features, the number of scans is: $S_{FastMap} = (t + 1)k$.

COFE randomly selects the reference sets that correspond to features. We distinguish two cases: either the reference sets fit in memory or they don't. Consider the second case in which not all reference sets fit in memory. The size of the reference sets grows exponentially with the row index of the feature. The smallest sets can be considered to fit in memory. Since the only information needed to compute the embedding corresponding to the bootstrapping features are the points in the reference sets, two scans, one to load them and one to compute the embedding for each point in the dataset, are needed. To compute the embedding of a point for each extension feature, the embedding for the previous features for that point and for the reference set points are needed. This can be achieved by loading a few reference sets during a scan and then computing their embedding, followed by another scan to compute the embedding of the rest of the points in the dataset. The reference sets for the next feature can also be prefetched during the computation of the embedding for the current feature. The worst case number of scans is therefore: $S_{Cofe} = k + 1$.

When all points in the reference sets fit in memory, that is, when we compute sufficiently few features so that we need not generate very large reference sets, a scan for loading the reference set and the computation of their embedding followed by a scan during which the whole embedding for each point in the dataset is computed suffices. Only 2 scans are required in this case.

In the worst case and setting $t = 2$, FastMap performs 3 times as many scans.