Information Retrieval and Clustering (pp. – - –)
W. Wu, H. Xiong and S. Shekhar(Eds.)

Clustering in Metric Spaces with Applications to Information Retrieval

Ricardo Baeza-Yates
Benjamín Bustos
Center for Web Research, Dept. of Computer Science
Universidad de Chile, Blanco Encalada 2120, Santiago, Chile
E-mail: {rbaeza,bebustos}@dcc.uchile.cl

Edgar Chávez
Universidad Michoacana, Morelia, México
E-mail: elchavez@fismat.umich.mx

Norma Herrera
Univ. Nacional de San Luis, San Luis, Argentina
E-mail: nherrera@unsl.edu.ar

Gonzalo Navarro
Center for Web Research, Dept. of Computer Science
Universidad de Chile, Blanco Encalada 2120; Santiago, Chile
E-mail: gnavarro@dcc.uchile.cl

Contents

1 Introduction

The concept of cluster is elusive. The direct implication of this is a large diversity of clustering techniques, each serving a particular definition. Most techniques focus on a global optimization function. The general procedure is to propose a clustering (using a suitable algorithm), then to measure the quality and amount of clusters, and repeat the procedure (proposing a new clustering structure, using for example new parameters) until satisfied.

This setup is satisfactory for many applications. Traditional clustering has been used in Information Retrieval (IR) for many different purposes, such as query expansion, document categorization and classification, relevance feedback, visualization of results, and so on. However, there are a number of applications in IR where new clustering methods would be useful.

In this chapter we propose a non-traditional clustering method, which is aimed at clustering data from a general *metric space*. This means that all the information we can use for discovering the clustering structure is the matrix of distances between all the data pairs in the set. Moreover, we cannot create new data points such as centroids. This is in contrast with traditional clustering methods (such as the k-means technique), which operate on coordinate spaces and may require to create new data points. As a result, our technique is more general and can be used in applications where traditional clustering methods cannot be applied.

Our approach is aimed at clustering multimedia data. For this we mean data with no coordinates or dimensional information whatsoever. Several clustering algorithms exist for this scenario, for example, graph theoretical methods such as Zhan's algorithm [Zha71]. However, in several IR applications we need to go further. First, we may need to obtain the cluster of a given element (which will be regarded as its "equivalence class") without having to compute the clusters for all the data. In applications such as stemming and conflation we just want to obtain the class of a given word, which does not depend on the surrounding data. Even more difficult, we may want to compute the class of a word not in the data set (think, for example, of a misspelled word). A second requirement is the need to compute a *cluster hierarchy* by scaling a parameter. This hierarchy must start with the individual elements and end (possibly) with an equivalence class of the entire sample.

A third and very important requirement is that of having a *robust* clustering method. Data samples are often contaminated with spurious data objects, that is, objects not following the same distribution as the rest. These are called *outliers*. Deciding that an object is an outlier is rather

risky[1]. We require a robust method that assigns an equivalence class for each element. Querying the system with any object will produce its equivalence class, and the result must be the same for any object in the class used as a query. If the equivalence class of the object is the object itself, then we will call it an outlier.

In the next section we present our clustering technique for metric spaces. Then, we show three different applications of clustering in IR, all them under the metric space model.

In Section 3 we apply the clustering method to the problem of conflating words that share a common root, which is a relevant task in IR. The standard solution for stemming is a rule based approach inherited from Porter's algorithm. This type of solution closely follows the possible variations of a single stem into all its morphological instances. This technique suffers from several drawbacks, in particular its inability to cope with noisy data. For example, when a word is misspelled, finding its correct stem by following rules is at least risky. A single edit error in a word may trigger the wrong rule and cause a mistake in the conflation. We propose a more robust solution based in clustering on metric spaces.

In Section 4 we use a simplification of the clustering technique to build a data structure (index) on a space of text documents. This data structure permits us to search for documents similar to a given one or to a text query, under the classical cosine similarity measure used in IR. The search technique exploits the clustering properties of the index structure by scanning the most promising clusters first. The approach is an approximation: with a very low probability of missing relevant documents, we can answer queries by making only a fraction of comparisons (distance computations) needed for an *deterministic* algorithm.

Finally, Section 5 resorts to a more general form of clustering: the data is partitioned according to an arbitrary property, not only to spatial closeness. This is used to partition a data set into subsets that behave different from each other in terms of their distances to the rest. The goal is again to build a faster index using information of the cluster structure of the data. Now the goal is to improve a deterministic algorithm by fine tuning the algorithm for each discovered cluster. In the example presented, different index parameters are used on each partition, each best suited to the local properties of each group. In a general perspective these indexes may have different

[1]There are a number of horror stories about automatic data cleaning (outlier removal) that show that, under an unknown distribution, it is better to be cautious. An example is the ozone layer measurements, where the computer filtered out large holes as outliers, but the holes were actually there.

local parameterizations of a single scheme, or even be completely different approaches, such as a an approximated approach in one partitions and an exact approach on a different partition. We demonstrate the effectiveness of the approach on an application to find the correct versions of misspelled words.

2 Our Clustering Method

Cluster and outlier detection are aimed at producing a model of the data. The primary objective of an application is to partition the data into a meaningful collection of sets. A partition, in the mathematical sense, is a collection of non-intersecting subsets whose union is the whole set. If the clustering defines a partition, then we will have a *crisp* clustering. If the subsets have non-empty intersection then we will have a *fuzzy* clustering. In this chapter we are interested in crisp clusters.

The data model obtained from cluster detection may be very sensitive to outliers. The outliers could be considered as "rare" data, not following the overall tendencies of the group. In order to design a robust clustering algorithm we need to properly define what a cluster is. This definition should follow some intuition and be computable in a reasonable amount of time. In addition, the clustering should be able to give the cluster of a given element without clustering all the data, and it should be able to detect outliers.

Two additional restrictions are of importance. The clustering should be carried out using only the distances between objects, and no new objects should be created. These restrictions are a fair way to mask the application level. All the domain knowledge will be encapsulated in the distance computation. The resulting clustering technique will be suitable for document retrieval, for browsing images, or for handling multimedia data in general.

Cluster and outlier detection is a classic problem of non-parametric statistics. Cluster analysis is the detection of groups that reduce the inter-group similarity and increase the intra-group similarity. In this section we explore several approaches based on graphs for cluster analysis. The strategy is to define the clustering property and then to identify a random graph capturing it. This generic algorithm will satisfy our restrictions and hence will be suitable for metric spaces.

The ultimate goal of any clustering algorithm is to postulate a model of the data. Different applications define different models and hence different objective functions, each of which will be optimized by a different clustering. This optimization goal function may be implicit or explicit. For

exact indexing purposes (such as the application of Section 5), the implicit goal function is the number of distance computations performed to solve a proximity query. A good clustering will produce a partition that minimizes the number of distance computations performed at search time. For approximate searching (as described in Section 4), the goal is to optimize the tradeoff between accuracy of the result and number of distance evaluations. For classification purposes (described in Section 3) the goal will be just the accuracy of the classification.

Most clustering methods considered in the statistical literature and in statistical software, namely partitioning methods such as k-means and k-medians [JD88], and agglomerative methods such as dendograms [Har75], share a "two-stage" nature. First, by assigning different values to some parameter (for instance, k in k-means clustering), the algorithm produces a family of clustering structures, that is, a family of partitions of the original data into relatively homogeneous disjoint blocks. Second, the user must resort to some validation criterion, such as those studied in [MC85], or use some exploratory analysis of the results, in order to decide which is the "right" clustering structure. It is desirable to have clustering procedures that are "automatic", in the sense of providing the user at once with the amount and identity of the clusters present in the data. This automatic procedure will be appealing only if we may find a way to disaggregate those objects not belonging to any group. This feature will be crucial for data cleaning applications.

On the other hand, most classical clustering algorithms have quadratic or even higher complexity, because they must measure global properties involving every distance pair. Clustering is an inverse problem, just like interpolation. This means that it is not well defined in the mathematical sense unless we add additional constrains, called "regularity assumptions", which distinguish between good and bad solutions. A rather classic example of regularity conditions for a problem is the problem of fitting a curve to a set of discrete points, or to interpolate the data. An infinite number of solutions exist for this problem, but we select a single *smooth* curve as *the solution*. In detecting the cluster structure of multivariate data we must make some kind of regularity assumptions. Most of the proposed algorithms in the literature are based on heuristic regularity assumptions about the nature of the clusters and their shape. The user iterates several times, accepting some hypotheses and rejecting others, and each assumption leads to a different clustering of the data. A difficult problem remains: naming clusters, particularly in IR.

2.1 Clustering in Metric Spaces

We will focus on applications where the objects and their similarity define a *metric space*. This is formalized as a universe of valid objects $\mathbb{X}$ and a distance function d that satisfies the triangle inequality and is typically expensive to compute. We manage a finite data set $\mathbb{U} \subseteq \mathbb{X}$, of size n. The problem of clustering can then be defined as partitioning $\mathbb{U}$ into sets such that the intra-cluster distances are "small" and inter-cluster distances are "large". In several applications, clustering is used as a tool for answering proximity queries of the form (q, r), where $q \in \mathbb{X}$ and $r \in \mathbb{R}^+$. This query defines a *query ball* $(q, r) = \{x \in \mathbb{X}, d(q, x) \leq r\}$, and the goal is to retrieve the elements of the database that are inside the query ball, that is, $(q, r) \cap \mathbb{U}$. Other types of queries are possible, but we will focus on the simplest case. The aim of *metric space searching* is to preprocess the database $\mathbb{U}$ so as to build an index that answers queries (q, r) performing as few distance computations as possible [CNBYM01]. We will show that clustering $\mathbb{U}$ yields large performance improvements on the searching performance of the indexes.

In cluster analysis one often assumes that the data set comes from a d-dimensional real vector space $\mathbb{R}^d$. Some algorithms, like k-means, make heavy use of the coordinate information; assuming, for example, that new centroids can be created. If neither the coordinates of the data, nor explicit object creation is used, then the algorithm is likely to be extensible to metric data sets. Further care must be taken if one wishes to extend a clustering algorithm to massive data sets, as the algorithm must scale in performance as the input data size grows. It is important to note that the distance function could be very expensive to compute, and therefore the complexity must take into account this leading complexity term.

Recent work has been done on clustering metric spaces by directly using a mapping from the original metric space into a low dimensional vector space, and then clustering the data using a traditional technique or an ad-hoc variation. This approach was tested in [GRG+99] using a generalization of the well known BIRCH clustering procedure [ZRL96], originally designed for vector spaces. The strategy is to map the metric space into a vector space using FastMap [FL95], and then to use BIRCH in the mapped space. To obtain a good clustering the mapping must be accurate. Unfortunately, FastMap is neither contractive nor expansive if the original space is not $\mathbb{R}^d$ with the Euclidean distance. Hence the distortion in the distances is unbounded. The algorithm proposed in [GRG+99] is linear in time. Our clustering approaches will make a single pass over the data set to discover each grouping, hence their total complexity is at most quadratic.

2.2 Mutual k-Nearest Neighbor Graph Clustering Algorithm

The problem of identifying the clustering structure is hard. We may focus instead on the converse problem, identifying the *absence of clusters* in a sample data set. If we have a small amount of sample data, we cannot decide if they cluster together, since we have little implicit information about their relative proximity. So we begin by asking for a significant number of sample points. We believe that most readers will agree that if the data set is drawn from a uniform distribution, then the data set is cluster-free. This can be generalized to distributions bounded away from zero and from infinity, that are almost-uniform distributions. If the data set is cluster-free, we say there is a single cluster integrated by all the elements of the sample. If the data have a cluster structure, it is reasonable to expect the data to be almost-uniform inside each clusters. We use this observation to base our approach in the null hypothesis testing "the data is cluster free".

In [BCQY97], a technique is proposed to detect clusters and outliers based on a graph theoretical-approach. This technique has provable cluster-detection power under some regularity conditions: (a) the clusters to be detected correspond to different connected components in the support[2] of the sample distribution, and these connected components are at strict positive distance from each other, (b) the sample distribution has a density bounded away from zero and infinity on each component of the support and (c) the connected components of the support of the distribution are grid compatible[3]. We call a distribution with this properties a *regular* distribution.

We will call clusters the connected components of the support. Since it is assumed that connected components are at a strictly positive distance from each other, the algorithm can detect "crisp" clusters, as opposed to "fuzzy" clusters with overlapping support.

The technique just described postulates a graph, over random points in $\mathbb{R}^d$. The random graph has a controlling parameter τ for its density (e.g. the number of arcs for each node). Above certain threshold the graph is connected with high probability. The threshold for τ depends in turn on the underlying point distribution, the dimension of the space and the number of points. In short, we can always have a value of τ such that

[2]The support of a random variable x, with density function $f(x)$, is the set S such that $f(x) > 0$. We strength this condition by assuming $f(x) > \epsilon$ (bounded away from 0).

[3]See [BCQY97] for details, the essential idea is that the support accepts a discretization (it cannot be infinitely thin) and it is completely contained in a discrete grid, such that each grid cell has neighboring non-empty cells.

the graph will be connected with high probability if the sample points are drawn from a distribution that is bounded away from zero and from infinity and will not be connected if the points come from a distribution vanishing asymptotically in the support. If we have a sample data set coming from a regular distribution our goal will be to detect the clusters (or equivalently to detect the components of the support). Many graphs could be defined for clustering purposes under the above setup. In [BCQY97] they use the Mutual Nearest Neighbor graph, defined below. In this chapter we also describe the range-r graph.

The controlling parameter τ of the random graph depends on the characteristics of the probability space. The general technique relies on the proper estimation of the connectivity threshold of τ in the random graph. In general, the estimation could be two fold, a theoretical bound and a practical recipe based on a Montecarlo simulation. To find the threshold, the procedure is to generate particular instances of cluster-free datasets. In these datasets we observe the minimum value of τ such that the random graph is connected.

For example, we may choose uniformly distributed points in $\mathbb{R}^2$ and estimate the minimum value of the parameter τ to obtain a single connected component. After estimating τ we will be able to detect clusters (i.e. connected components) in arbitrary data, if the number of points and the dimension of the sample data, are both similar to the estimated values.

The technique described above can be applied to data sets with the same topological characteristics. For example the parameter τ for $\mathbb{R}^2$ and $n = 1000$ points in the sample may not be applied to data sampled from $\mathbb{R}^{23}$ and $n = 50,000$ points. Hence the controlling parameter τ will be a function of the space topology.

After this digression, the central question is: how to apply the procedure to general metric spaces? In this case we will need to find out an invariant in the sampled data and, for clustering, the parameter τ must be applied to data sets sampled from a probability space with the same invariant. We will use as invariant the intrinsic dimension of the data, defined as the minimum d such that the data can be embedded in $\mathbb{R}^d$ without distortion. This approach can be used as long as the random graph can be computed using only the distances among the sample data. Note, however, that in most cases computing the distances between sample elements is very expensive; hence the algorithm performance is measured in the number of distance computations. Moreover, if the data is *massive* (gigabytes to terabytes of data) it is advisable to use a technique that can work in secondary memory, because disk accesses could be far more expensive than distance computations.

Since the above embedding may be hard to find (as hard as finding the clustering structure), one may resort to hierarchical clustering, by moving the parameter τ to find the proper threshold. In this section we describe a clustering algorithm based on the *Mutual Nearest Neighbor Graph* and the *Range r Graph* respectively. This clustering procedure is able to discover clusters at different resolutions, and has subquadratic complexity measured in terms of the number of distance computations. To achieve this complexity the algorithm relies on the use of an indexing algorithm. This indexing can be implemented using any indexing data structure. A better indexing algorithm will lead to faster clustering, without impact on the quality of the clusters obtained.

2.2.1 The Clustering Algorithm

From an algorithmic point of view the procedure is to find the k nearest neighbors of a seed point s, the set $kNN(s) = \{s_1, s_2, \cdots, s_k\}$, and to include in the cluster of s those points $s' \in kNN(s)$ having the reciprocal property $s \in kNN(s')$; and then to proceed iteratively until no more points could be added to the current cluster, restarting with a new seed point until no more seed points are left to visit. This algorithm is shown in figure 2.2.1 with S being the set of points, and k the number of neighbors considered. The output is a partition of the sample data, and each partition element is either a cluster or an outlier.

```
MkNNCluster (k, s, S)
1.
2.      let S'_s = {s' ∈ kNN(s) || s ∈ kNN(s')}
3.      return s∪ MkNNCluster (k, s'_1, S) ∪ ··· ∪ MkNNCluster (k, s'_k', S)
4.
Partition (k, S)
5.      let S = {s_1, ··· , s_n}
6.      while(|S| > 0)
7.          output MkNNCluster(k, s' ← pick(S), S)
8.          S = S−MkNNCluster(k, s' ← pick(S), S)
9.      endwhile
```

Figure 1: The $MkNN$ clustering algorithm, k is the number of neighbors to consider.

Calling the procedure **Partition** of figure 2.2.1 will identify all the connected components of a graph defined implicitly, called the mutual k nearest-neighbor graph. A technical drawback in this approach, is to find out the correct value for k. It is clear that, as k increases, the number of connected components decreases; this gives a natural way to define a hierarchical structure. Using brute force for finding the k nearest neighbor of each site in the data set, takes $O(n^2)$ distance computations. In the next subsection we discuss in some detail how the above algorithm, and a family of algorithms based on similar assumptions, can be converted on a clustering detection procedure.

2.2.2 Connectivity Properties

The algorithm just described depends heavily on the parameter k, the number of neighbors to use. For $k = 0$ the partition consist in one point for each partition element. If $k = n - 1$ then we have surely one single set in the partition. For any number of neighbors k in between we will have a finer or a coarser partition. We are interested in finding the "right" number of clusters and we have as a parameter the number of neighbors, k, to use. If we cannot postulate any a priory knowledge neither about the type nor the shape of support of the distribution, we can always build a hierarchy of partitions.

With a large enough k we can compute the k-nearest neighbor graph of the data set and for each $k' \leq k$ we don't need any additional distance computation to find the k'-nearest neighbor. Each partition in the hierarchy is a coarsification of the preceding partition. In other words, if $k_1 \leq k_2$ then each equivalence class in the partition induced by k_1 is completely contained in one equivalence class of the partition induced by k_2. Once the hierarchy is postulated, the "right" number of clusters could be found using well known intracluster/intercluster stress measures.

If our goal is to produce an "automatic" clustering procedure, we need to find a priory the value of the parameter k; and bound the search for a particular class of distributions *inside* the connected components of the support. In particular we can restrict ourselves to vector spaces, or to metric spaces that can be embedded into a low dimensional vector space. It is not necessary to explicitly find the mapping, instead we can postulate a null hypothesis.

In [BCQY97] are proved both analytical and experimental bounds for k, depending on both the size of the sample and the dimension of the embedded vector space. distribution is regular enough. The bounds proved

are not tight, but they postulate a compact support and a uniform distribution inside the clusters. Under this setup, the connectivity constant can be estimated using a Montecarlo approach. The idea is to estimate the minimum k such that there is only one cluster for a given sample size, obtained from a uniform distribution. The data obtained in the Montecarlo simulation is fitted to a model of the same family of the theoretical bound, of the form $k_d = a_d + b_d \log(n)$, having different results for each different intrinsic dimension d.

2.3 The Range r Graph

We propose an alternative procedure to detect clusters and outliers, based on the same foundations of the $MkNN$ graph. The central idea is to use *range queries* instead of k-nearest neighbor queries. Each site s will share an edge with s' in the G_r graph if $s' \in (s,r)_d$, with $(s,r)_d = \{s' \| d(s,s') \leq r\}$.

With the above definition of G_r we can use algorithm 2.2.1 with essentially no modifications to partition the data into clusters and outliers. The graph G_r is simpler than $MkNN$, and not checking for symmetry speeds up the construction. In both approaches the use of an index to satisfy either k-nearest neighbor queries or range queries is advised to achieve subquadratic complexity.

RangeCluster (r, s, S)
1.
2. let $S'_s = \{s' \in (s,r)_d\}$
3. return $s \cup$ RangeCluster $(r, s'_1, S) \cup \cdots \cup$ RangeCluster $(r, s'_{j'}, S)$
4.

Partition (r, S)
5. let $S = \{s_1, \cdots, s_n\}$
6. while$(|S| > 0)$
7. output RangeCluster$(r, s' \leftarrow pick(S), S)$
8. $S = S -$RangeCluster$(r, s' \leftarrow pick(S), S)$
9. endwhile

Figure 2: The range clustering algorithm, r is the ball radius.

In algorithm **RangeCluster** presented in figure 2.3, we can see that the procedure is almost the same for our clustering strategy and the one obtained using the $MkNN$ graph. We proceed to obtain an adequate r according

to our hypothesis that inside clusters the data distributes almost uniformly. The procedure is to make a Montecarlo simulation of the algorithm to obtain statistics on the minimum r such that there is only one cluster in the sample.

This algorithm performs on average $O\left(n^{1+\alpha}\right)$ distance computations (the operation of leading complexity), with $0 \leq \alpha \leq 1$ a constant depending on the intrinsic dimension of the sample data. The algorithm can detect outliers in the sample data and, if desired, it can produce a hierarchical structure (a dendogram) pointing to clusters at different resolutions.

2.3.1 Outliers, Equivalence Classes and Stability

The random graphs discussed in our scope are uniquely defined once the parameter τ (k and r, respectively) and the point set are fixed. The two examples we present in subsection 2.2.1 have additional advantages. One nice feature is that the random graph can be computed incrementally. In other words, we start with a single point and begin adding edges and vertices until no more vertices appear. We may think clusters as equivalence classes, and the incremental procedure ensures obtaining the same class for any representative. Hence, *the method is stable and deterministic*.

Certain points will be isolated in the MkNNG or the Range r graphs, forming a connected component of low cardinality. If the equivalence class of a point is the point itself (or a very small number of elements, compared with the complete data set), this equivalence class will be considered as an *outlier of the sample*. The discussion about the validity of this assumption is beyond the scope of this book. A common sense argument can be used however. A small equivalence class corresponds (according to our hypothesis) to a very small component of the support of the distribution. This implies either a very small probability of having points in that particular location, or having points where the support vanishes. In either case, the presence of these points does not follow the same general rule of the rest of the sample.

A final remark on the outlier issue. If *most* of the equivalence classes detected have a small cardinality, this naturally implies a small value (underestimated) of the parameter τ, and *not* the presence of "many" outliers.

2.4 Radius vs. Neighbors

The controlling parameter of both clustering approaches (Range and MkNN graphs), the number of neighbors k and the radius r, must be estimated beforehand. We face here a different problem, because we do not have a theoretical model on the searching radius, moreover we do not expect to

obtain a fixed radius for every data set. The searching radius must scale with the density of the points.

The goal of our Montecarlo analysis is to find an invariant giving us a rule to estimate the radius of the RangeCluster procedure. We will have at hand only the sampled data, with no good information about the intrinsic dimension of the data. We can use only inter-point statistics to estimate the parameter r. If the particular case of spatial data (sampled from an Euclidean d-dimensional space) we can use intrinsic dimensionality estimators, but this does not apply to metric data from an arbitrary space.

To guide the search we ran the clustering algorithm in uniformly distributed data in the d-dimensional unitary cube. The goal is to estimate the minimum radius to obtain a single cluster. The variables of the experiment are the number of points and the dimension. We had to take into account that as the number of points increased the minimum inter point distance decreases, since the cube becomes more dense. Note that the number of points in a ball grows exponentially with the ball radius.

The relation found was the *average number of points* inside the $(s,r)_d$ ball. For a fixed dimension, independently of the number of points (the data density), we found that when the average cardinality of the $(s,r)_d$ ball is above certain threshold there is always a single cluster in our setup. The radius of the $(s,r)_d$ ball was variable, revealing small changes in the local density even in the uniformly distributed data.

The rule is then simply stated as searching with a radius such that the ball captures at least n_d points. If the data is very concentrated (with high local density) then the searching radius is small and conversely. This implies that we need inter-point statistics, to estimate the radius of the $(s,r)_d$ ball for the algorithm. Fortunately we can use statistics from the index construction to propose the appropriate searching radius, since the construction of the index uses inter point distances in the construction process. In other words it does not imply an overhead in the number of distance computations.

2.5 The Connectivity Parameters

We include the results of the Montecarlo simulation for the connectivity of the MkNN graph from [BCQY97] below, for low dimensional data. The R^2 column is the fitness value, high values are better. The sampled standard deviation $\hat{\sigma}_d$ is also shown.

Once the intrinsic dimension of the data is estimated, we can use table 2.5 to compute the connected components of the MkNN graph, as described. The user will judge if the obtained equivalence classes correspond to clusters

dim	adjusted model	R^2	$\hat{\sigma}_d$
2	$4.341 + 0.430 \ln n$	0.979	1.313
3	$3.422 + 0.646 \ln n$	0.984	1.744
4	$2.651 + 0.924 \ln n$	0.983	2.266
5	$1.859 + 1.219 \ln n$	0.976	2.674
6	$0.268 + 1.687 \ln n$	0.994	3.092
7	$-0.770 + 2.019 \ln n$	0.989	3.364
8	$-2.453 + 2.474 \ln n$	0.994	3.666

Table 1: Adjusted models for k (as described) in the MkNN random graph (uniform data in $[0,1]^d$).

Dimension	2	3	4	5	6	7	8	9	10
Sites	4.37	5.07	5.51	6.05	6.22	8.03	7.78	8.59	10.32

Table 2: The minimum number of points to have a single cluster, in different dimensions.

or outliers, based on the cardinality and the relative position of the data.

For the Range r graph a similar table can be obtained after the corresponding Montecarlo simulation. A threshold parameter τ must be found depending on the number of points, and the (intrinsic) dimension of the data. In this case the threshold parameter cannot be easyly stated in absolute numbers, since the inter-point distance is variable. If we fix the number of points in the sample, and regard the "minimum radius such that m points are captured" we will obtain a table similar to table 2. We can also fix the radius as a percentage of the minimum inter-point distance, the maximum inter-point distance, etc. The proper invariant should be selected according to the user needs.

2.6 Intrinsic Dimension

The intrinsic dimension of the data is an important invariant in metric space samples. Since one cannot assign directly coordinates or a dimension notion to the data, we have to resort to vector spaces where the notion of dimension is properly defined. If the intrinsic dimension of the data is known the clustering structure could be detected. Conversely, if the cluster structure is known, the intrinsic dimension could be found.

Both arguments are valid. When discovering the clustering structure of a sample we are postulating a particular distribution of the data. This implies that we have an inverse problem, since the intrinsic dimension, and distribution are unknown, we have to fix one of them. The clustering procedure described could be used also as an intrinsic dimensionality estimator. The Range r graph, or unit distance graph, and the MkNN graph could be used as an intrinsic dimensionality estimator. The procedure will be the converse of the clustering procedure. We discover the clustering structure using a hierarchical approach. After this we obtain (from the appropriate table) the intrinsic dimension corresponding to the value of the parameter, and the number of points used.

3 Morphological Stemming and the Holomorphic Distance

3.1 Motivation

The problem of conflating words sharing a common root is an important task in information retrieval. The standard solution for stemming is a rule based approach inherited from Porter's algorithm. This type of solution closely follows the possible variations of a single stem into all its morphological instances. This technique suffer from several drawbacks, being the main problem the presence of *noise* in the pattern, i.e. spelling errors. When a word is misspelled then finding its correct stem by following rules is at least risky. A single edit error in a word may trigger the wrong rule and that will cause a sure mistake in the conflation.

In this section we describe a different alternative for stemming and conflation. This technique has been proposed in [Cha02]. The idea is to use the so called holomorphic distance to detect words with similar stems, and then to use a clustering algorithm to isolate clusters. The input is a dictionary of words and the output is a partition of the dictionary into words sharing a stem.

Traditional sequence comparison is done by carefully aligning the formant symbols of a sequence to obtain the minimum number of atomic operations (insert, delete, substitute) to convert one sequence into another. The de-facto choice for comparing sequences has been the Levenshtein or *edit distance* and variations obtained by adding or inhibiting atomic operations.

This naturally implies that the distance takes into account only local information. Moreover, our sequence comparison does not use the context of

strings, as the expert knowledge in some very focused fields cannot be meaningfully mapped into insertion/deletions/substitutions or other operations, mainly because the similarity perceived by the expert is not local.

The holomorphic distance for sequences is not based on the edit distance for words. The idea is to extract a number of features of the sequence (in this case a word in a document), and to use the features to build a vector, in the same fashion that is done for documents in IR. Below we give a more detailed description of the distance.

For information retrieval it is often desirable to conflate all the variations of a word into a single stem. This operation conflates, for example, `house, housing, houses` into the single stem `hous`. In this way, when the user wants to query the system with any of the variations of the word, all the documents containing the same stem will be retrieved. In a regular language (like English) there will be no problem in building a stemmer using the Porter's algorithm. In a Semitic or romance language the stem may not be a prefix of the word, and the construction of a stemmer using the Porter's approach may not lead to an efficient solution. Other problem often faced in either language is querying or indexing misspelled words. In this case even a perfect stemmer would fail to conflate to the proper stem. A *morphological* stemmer, on the other hand, will make use of the word structure to guide the conflation process. To this end it would be necessary to assign a small distance to all the words with the same structure and a large distance to words with different structure.

3.2 The Holomorphic Transformation

The holomorphic transformation generalizes the notion of similarity developed for documents. The general idea is to decompose the original sequence in a number of sub-sequences using a number of fixed rules. This rules will be *feature* extractors of the sequences. The rule will be to obtain sub-sequences when a feature extractor is applied to a sequence. If Σ is the alphabet, and $s \in \Sigma^*$ is a sequence, then a function $\Gamma : 2^{\Sigma^*} \to 2^{\Sigma^*}$ is a *feature extractor*. We put no constraints to the type of constructions we allow for extracting features. Any function will fit our needs.

The limits imposed will be (if any) the computational complexity to obtain the transformation, and the a priori knowledge of the designer.

The *holomorphic transformation* will be a function $\mathcal{H} : \Sigma^\dagger \to \mathbb{R}^m$ with m the size of the context $[\Gamma^\dagger, \mathbb{S}]$. This transformation depends heavily on the set of feature extractors. The context will be the vehicle to introduce domain knowledge to the problem of sequence comparison. Note that once

a sequence is transformed using the holomorphic transformation, we can use the cosine distance to compare the corresponding vectors.

The context is the generalization of the vocabulary of a document collection. Under this point of view, the distance between a given pair of sequences will depend on the context. The distance will not be absolute any longer. Note also that the positional information may be lost after the holomorphic transformation. This is the case of transforming documents into vectors. If we want to recover the positional information we need to be very careful with the design of the feature extraction functions. In the following subsection we will discuss some examples of this.

3.3 A Morphological Stemmer Using Clustering

We can use the holomorphic distance to build a morphological stemmer. For a romance language the stem of a word is almost surely certain prefix, and the syntactical category is certain suffix. If we use q-grams, prefixes and swaps (Γ_q^q, Γ_j^p and $\Gamma_j^{2!}$), and carefully select the relative weights of the individual features, we are in position of measure the morphological closeness for word pairs. Tables 3 and 4 are examples of the closeness using this approach. The idea of the distance between the words is to find the maximum prefix, allowing misspellings, between two words. This maximum prefix is weighted by the presence of similar prefixes in the corpus.

The complete algorithm for stemming using the holomorphic transformation has two stages. First, we find the clusters of the corpus using the holomorphic distance. These clusters coincide with the words conflated for sharing the same stem. This defines equivalence classes by morphological closeness. In the second stage we query the corpus and find the closest equivalence class, tp report the same representative for each equivalence class member. This representative may be some normal form of the stem (e.g. the verb in infinitive) or the largest common stem in the equivalence class.

A related problem in information retrieval and computational linguistics, is syntactical tagging. Here the problem consist in assigning a label to the word, indicating if it is a verb, adverb, noun, etc. A similar scheme as the one described for stemming may be followed to tag words.

4 Clustering for Approximate Proximity Search

In this section we describe an approximate proximity search algorithm based on a clustering data structure. The clustering method we use is in the same

	H. non weight	H. weight	Edit distance
1	apetitosa	apetitosa	apetitosa
2	apetito	apetitoso	apetitoso
3	apetitoso	apetito	aceitosa
4	apetite	apetite	apestosa
5	apetitiva	apetitiva	apetitiva
6	apetitivo	apetitivo	apetito
7	apetible	apetible	aceitoso
8	apetecer	apetecer	acetoso
9	apetecedor	apea	alentosa
10	apetencia	apetencia	aparatosa

Table 3: The 10 nearest neighbors of the Spanish word *apetitosa* with two different weighted holomorphic distances, and the edit distance. The edit distance reports words with different meaning (3-4, 7-10), while the holomorphic distance provides very strong closeness (except 9 in the weighted case).

	H. non weight	H. weight	Edit distance
1	dettagli	dettagli	dettagli
2	dettaglio	dettaglio	dettaglio
3	dettagliato	dettagliata	dettai
4	dettagliati	dettagliate	dentali
5	dettagliate	dettagliati	deragli
6	dettagliare	dettagliato	dettami
7	dettagliata	dettagliare	dettati
8	dettagliante	dettagliante	dettavi
9	dettaglianti	dettaglianti	rettali
10	dettagliatamente	detta	battagli

Table 4: The 10 nearest neighbors of the Italian word *dettagli* with two different weighted holomorphic distances, and the edit distance, with results similar to the Spanish case.

spirit of that described in Section 2.2, although simplified.

The *List of Clusters* data structure [CN00] is a list of "zones". Each zone has a center c and a *covering radius* $cr(c)$, defined as the maximum distance between c and any element of the zone. A parameter of the structure is the number m of elements inside each zone (counting the center).

To build the structure, a first center $c \in \mathbb{U}$ is chosen at random, and its $m-1$ nearest neighbors are found. A zone is made up of c and these $m-1$ neighbors. Its covering radius $cr(c)$ is defined as the distance to the $(m-1)$-th neighbor. Once this first zone, I, is determined, we remove the elements of the set and build the rest of the list over $E = \mathbb{U} - I$. The next center is chosen as the element that maximizes the sum of distances to the previous centers.

The construction process returns a list of triples (c_i, cr_i, I_i) (center, radius, zone elements). Figure 3 (left) shows the pseudocode for the construction algorithm. On the right we show an example with three zones. Note that this data structure is asymmetric, because the first center chosen has preference over the next centers in case of overlapping balls. The brute force algorithm for constructing the list takes $O(n^2/m)$ distance computations, but it can be improved using auxiliary data structures to find the neighbors.

```
BuildList (Set of objects 𝕌)

1.  if 𝕌 = ∅ then return empty list
2.  Select c ∈ 𝕌
3.  I ← the m elements in 𝕌
    closest to c
4.  cr ← max_{u∈I} d(c,u)
5.  E ← 𝕌 − I
6.  return (c, cr(c), I) : BuildList(E)
```

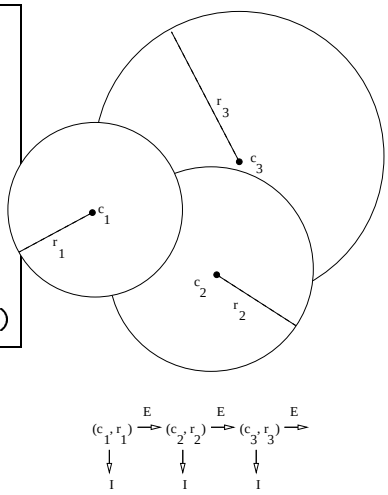


Figure 3: On the left, construction algorithm of List of Clusters. Note that I includes c. The operator ":" is the list concatenation. On the right, a graphical example.

To solve a query (q, r), $d(q, c)$ is computed, reporting c if it is within the

query ball. Then, we search exhaustively inside I only if $d(q,c) - cr(c) \leq r$, because otherwise the zone with center c cannot have intersection with the query ball. Also, given the asymmetry of the data structure, we can prune the search in the other direction: if the query ball is totally contained in $(c, cr(c))$, that is, if $cr(c) - d(q,c) < r$, then we do not need to consider E, because all the elements that are inside the query ball have been inserted in I. Figure4 depicts the search algorithm.

```
SearchList (List L, Query q, Radius r)

1.  if L = ∅ then return /* empty list */
2.  Let L ← (c, cr(c), I) : E
3.  Evaluate d(c, q)
4.  if d(c, q) ≤ r then report c
5.  if d(c, q) ≤ cr(c) + r then search I exhaustively
6.  if d(c, q) > cr(c) − r then SearchList(E, q, r)
```

Figure 4: Search algorithm on List of Clusters.

The search cost for this data structure has a form close to $O(n^\alpha)$ for some $0.5 < \alpha < 1.0$ [CN00].

4.1 The Vector Model for Information Retrieval

In IR [BYRN99] a *document* is defined as the *recovery unit*. This can be a paragraph, an article, a Web page, etc. The classical models on IR consider that every document is described by a representative set of keywords called *terms*. A term is a word whose semantics help describe the principal topics of a document.

The most popular of these models, the *vectorial model*, considers a document as a t-dimensional vector, where t is the total number of terms in the system. Each vector coordinate i is associated to a term of the document, whose value corresponds to a positive *weight* w_{ij} if that term belongs to the document or zero if not. If $\mathbb{D}$ is the set of documents and d_j is the j-th document in $\mathbb{D}$, then $d_j = (w_{1j}, w_{2j}, \ldots, w_{tj})$.

In the vectorial model, a query object q can be a list of a few terms or even a whole document. The *similarity* between a document d and a query q is taken as the similarity between vectors $\vec{d_j}$ and $\vec{q}$, and quantified as the

cosine of the angle between the two vectors:

$$sim(d_j, q) = \frac{\vec{d_j} \cdot \vec{q}}{|\vec{d_j}| \times |\vec{q}|} = \frac{\sum_{i=1}^{t} w_{ij} \times w_{iq}}{\sqrt{\sum_{i=1}^{t} w_{ij}^2 \times \sum_{i=1}^{t} w_{iq}^2}} \quad (1)$$

where w_{iq} is the weight of the i-th term of the query q. The weights of the terms can be calculated using *tf-idf* schemes:

$$w_{ij} = f_{i,j} \times \log\left(\frac{N}{n_i}\right) \qquad f_{i,j} = \frac{freq_{i,j}}{\max_{\ell=1\ldots t}(freq_{\ell,j})}$$

where N is the total number of documents, n_i is the number of documents where the i-th term appears, and f_{ij} is the *normalized frequency* of the i-th term: $freq_{i,j}$ is the frequency of the i-th term in d_j.

If we consider that the documents are points in a metric space, then the problem of searching for documents similar to a given query is reduced to a proximity search in the metric space. Since $sim(d_j, q)$ is only a similarity function, we will use the angle between vectors $\vec{d_j}$ and $\vec{q}$, $d(d_j, q) = \arccos(sim(d_j, q))$, as our distance metric, so $(\mathbb{D}, d)$ will be our metric space.

Despite of this clear link, metric space techniques have seldom been used for this purpose. One reason is that the metric space of documents has a very high dimension, which makes any exact search approach unaffordable. In fact, there is no exact algorithm that can avoid an almost exhaustive search on the database to answer proximity queries in this particular metric space.

The standard practice is to use an inverted index to find the documents that share terms with the query and then search this set of candidate documents by brute force. This technique behaves well on queries of a few terms, but it becomes impractical when queries are documents. This is required, for example, in relevance feedback scenarios.

In the latter case, it is customary to use heuristics, because the definition of relevance is already fuzzy. In most cases, finding some good answers is as good as finding them all, particularly when the search cost is drastically reduced. This is a case where metric space search with approximation algorithms would be of great value, as it is much better to use an approximation where the quality and degree of exactness is well understood, than a heuristic which resists analysis.

4.2 Techniques for Approximate Proximity Searching

We describe a search technique where the user can tune between efficiency and quality of the answer. The maximum number of distance computations

we can perform is fixed and denoted by *quota*. Once *quota* has been reached, no more elements can be considered. The search algorithm is approximated in the sense that some relevant elements can be missed because we could not find them before we exhausted the *quota*. Hence, it is crucial to use the allotted *quota* efficiently, that is, to find as soon as possible as many elements of the result as possible.

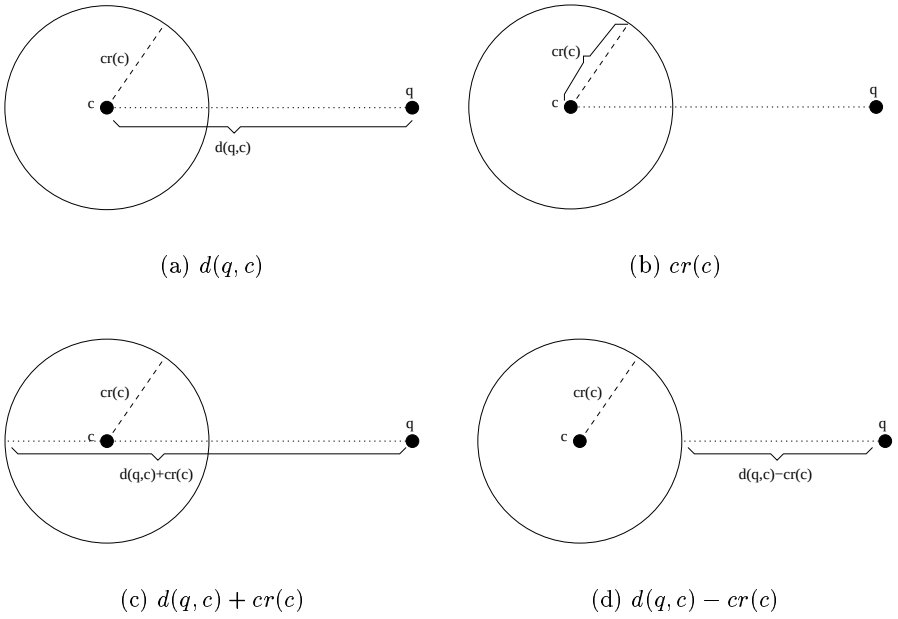
The technique described in this section is called *ranking of zones* [BN02]. The idea is to sort the zones of the List of Clusters in order to favor the most promising, and then to traverse the list in that order. The sorting criterion must aim at quickly finding elements that are close to the query object. As the space is partitioned into zones, we must sort these zones using the information given by the index data structure.

We compute $d(q, c)$ for every center, and estimate how promising a zone is using only $d(q, c)$ and $cr(c)$ (which is precomputed). One not only would like to search first the zones closer to the query, but also to search first the zones that are more compact, that is, the zones which have smaller covering radii (since all the zones have the same number of elements). Some zone ranking criteria are (all in increasing order, see Figure5):

- $d(q, c)$: the distance from q to each zone center.
- $cr(c)$: the covering radius of each zone.
- $d(q, c) + cr(c)$: an upper bound of the distance from q to the farthest element in the zone.
- $d(q, c) - cr(c)$: a lower bound of the distance from q to the closest element in the zone.
- $\beta(d(q, c) - cr(c))$: what we call *dynamic beta*.

The first two techniques are the simplest ranking criteria. The third technique aims to search first those zones that are closer to q and also are compact. The fourth technique uses the lower bound of the distance between the query object and any element of the zone, given by the index structure.

If factor β is fixed, then the last technique is equivalent to the criterion $d(q, c) - cr(c)$, because the ordering is the same. However, instead of using a constant factor $\beta \in [0..1]$, we use a *dynamic factor* of the form $\beta = 1/(1.0 - \frac{cr(c)}{mcr})$, where mcr is the maximum size of the covering radius of all zones. This implies that we reduce more the search radii in zones of larger covering radii.



(a) $d(q,c)$ (b) $cr(c)$

(c) $d(q,c)+cr(c)$ (d) $d(q,c)-cr(c)$

Figure 5: Some zone sorting criteria.

4.3 Experimental Results

Figure 6 shows the results of experiments on a subset of 25,000 documents from *The Wall Street Journal* 1987–1989, from TREC-3 [Har95]. We compare the approximate algorithm using different ranking criteria. We used clusters of $m = 10$ elements and show queries with search radii that return, on average, 9 and 16 documents from the set. For example, on the left we see that, using the criterion $d(q,c)+cr(c)$, we can retrieve 90% of the results using 10,000 distance computations, that is, examining 40% of the space. We recall that all the exact algorithms require examining almost 100% of this space.

The results show that the approximate algorithms can handle well this space, and that the best criteria were $cr(c)$ and *dynamic beta*. We could retrieve more than 99% of the relevant objects while traversing merely a 17% of the database. This is the first feasible metric space approach to this long standing problem.

To show up to which point the concept of clustering has been essential for this good result, we consider a tempting declustering idea. Instead of

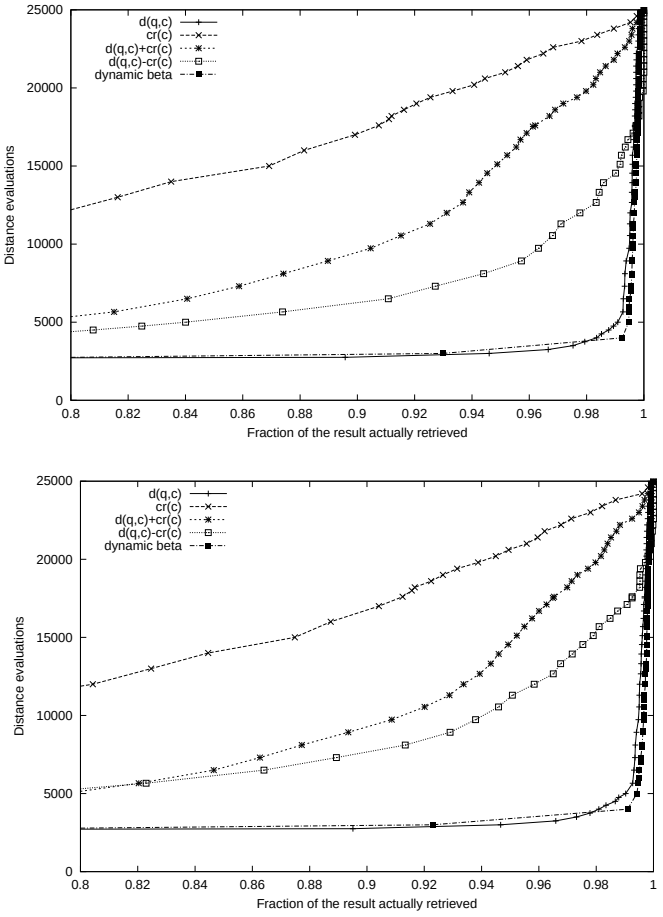


Figure 6: Comparison among different criteria in a document space, retrieving on average 9 elements (top) and 16 elements (bottom).

sorting the clusters according to how promising the center looks, let us sort all the elements separately according to the extra information given by their distance to the center. For example, the lower bound criterion groups all the elements of a zone of center c under the value $d(q,c) - cr(c)$. If we knew that an element u in this zone is at distance $d(c,u)$ of its center, we could refine its estimation to $d(q,c) - d(c,u)$. This, however, performs worse than the original method. We conjecture that the reason is that we lose valuable clustering information when we rank each element separately.

Finally, we note that this data structure reminds the clustering techniques based on nearest neighbors covered in previous subsections, but it is more rough and simple. It is likely that better results can be obtained with a more sophisticated clustering algorithm.

5 Clustering for Metric Index Boosting

In this section we explore a different alternative for clustering. Instead of directly detect the data grouping by distance, we will detect the clusters in a different domain. For indexing purposes we are interested in characterizing the intrinsic difficulty of a given dataset. A step towards this is by detecting segments of the dataset which are more difficult to index than others. Here we propose a particular procedure obtaining clusters of data which are *not necessarily* close to each other in the distance domain. This can be put in the following terms, for clustering we will be interested in minimizing the intra cluster distance $d(x,y)$, while maximizing the inter cluster distance $d(x,y)$. For our version of clustering we are interested in a different property, namely to minimize the intra cluster *measure* $f(x,y)$ and hence to maximize the inter cluster measure. The goal is to split the data in difficult-to-index and easy-to-index sets. We begin proposing a two way split, but this can be generalized to m-way splits as well.

Most indexing algorithms for proximity searching have tuning parameters. These parameters allow one to balance construction time, memory usage and search time, adapting the performance of the index depending on the characteristics of the data. The most relevant feature in a metric data set is how the data is distributed. Finding the underlying structure of a data set is very useful to design an indexing algorithm. In particular, knowing how the elements are clustered in the metric space help us identify the hardest region to be searched. Once the regions are categorized as easy, medium or hard for searching, we can locally tune the parameters for each region. Moreover, we can build independent indexes for each region, and

search each index separately at search time. This has proven to be more efficient than using global parameters and a single index. Another application of the local parameterization technique could be, for example, to index one part of the database with an exact searching index and another part with an approximated index that will give good answers almost all the time.

In this section we use the data distribution to segment the database into just two parts: the hardest to be searched and the rest. This can be generalized to a finer partition, but we content ourselves with illustrating the technique. Notice that this does not correspond any more to a traditional clustering where we aim at grouping data that is spatially close. Rather, we group data that share some common properties. This can be seen as grouping data that are close after applying a change of domain.

One way of visualizing the data distribution is by using distance histograms. Given a metric space $(\mathbb{X}, d)$ and an element $p \in \mathbb{X}$, the *local histogram* with respect to the reference point p is the distribution of the distances from p to all the elements $x \in \mathbb{X}$. The local histogram can be very different from the *global histogram* of all the distance pairs in $\mathbb{X}$. However, if several local histograms are similar, then we can predict the behavior of the global histogram of a data set $\mathbb{U} \subseteq \mathbb{X}$.

One of the main difficulties in metric space searching is the so-called *curse of dimensionality*. Some metric spaces (called "high dimensional") have a very concentrated histogram, with small variance and typically large mean. This means that random pairs of distances are very similar from each other, or alternatively, that from the point of view of a given element p, all the others are more or less at the same distance. All indexing methods are based on precomputing some distances and inferring lower bounds on other distances [CNBYM01]. For example, if the index has precomputed $d(p, u)$ and, when searching for (q, r), we compute $d(p, q)$, then we know by the triangle inequality that $d(q, u) \geq |d(p, u) - d(p, q)|$, so we can discard u without ever computing $d(q, u)$ if it turns out that $|d(p, u) - d(p, q)| > r$. However, this (and any other attempt to avoid computing $d(q, u)$ for every u) becomes useless if the space is high dimensional, since in this case $d(p, u)$ will be very close to $d(p, q)$. However, for those elements u far away from the central region in the local histogram of p, the reference point p can be a good tool to discard them.

If a group of elements is at the same time in the central region of the histograms of several reference points, then those elements represent a subset where searching is inherently difficult. We call this group of elements the *hard kernel* of the space and denote it $hk(\mathbb{X}, d)$. The remaining elements belong then to a *soft kernel* denoted by $sk(\mathbb{X}, d)$. The idea is then to index

and search separately the hard and soft kernels. That is:

- Partition the data set $\mathbb{U}$ into $hk(\mathbb{U}, d)$ and $sk(\mathbb{U}, d)$.
- Index separately $hk(\mathbb{U}, d)$ and $sk(\mathbb{U}, d)$.
- Solve (q, r) in $\mathbb{U}$ by searching $hk(\mathbb{U}, d)$ and $sk(\mathbb{U}, d)$ separately.

Detecting $hk(\mathbb{U}, d)$ is simple. We just intersect the central regions of local histograms for several different reference points p. After finding $hk(\mathbb{U}, d)$, we have that $sk(\mathbb{U}, d)$ is the complement.

Figure 7 describes the detection process of the hard kernel of a data set $\mathbb{U}$. The parameter s is the fraction of elements that should belong to the hard kernel. The parameter cr is the cutting radius used to delimit the central region in the local histogram of the reference point p. The idea is to take the elements surrounding the median of the histogram.

```
Compute_hk(Set of objects U, Fraction s, Radius cr)

1. hk(U,d) ← U
2. Choose a point p ∈ U
3. while |hk(U,d)| > s·|U| do
4.     m ← median{d(p,u), u ∈ U}
5.     hk(U,d) ← hk(U,d) ∩ {x ∈ U, d(x,p) ∈ [m − cr, m + cr]}
6.     if U ∩ hk(U,d) ≠ φ
7.       Choose a point p ∈ U − hk(U,d)
8.     else
9.       Choose a point p ∈ U
10. return hk(U,d)
```

Figure 7: Algorithm that finds $hk(\mathbb{U}, d)$

To test this algorithm we used a data structure for metric spaces called GNAT, explained next.

5.1 GNATs

GNATs (Geometric Near-neighbor Access Trees [Bri95]) are m-ary trees built as follows. We select, for the root node, m centers $c_1 \ldots c_m$, and define $\mathbb{U}_i = \{u \in \mathbb{U}, d(c_i, u) < d(c_j, u), \forall j \neq i\}$. That is, $\mathbb{U}_i$ are the elements closer to c_i than to any other c_j. From the root, m children numbered $i = 1 \ldots m$ are built recursively as GNATs for $\mathbb{U}_i$. Figure 8 shows a simple example of the first level of a GNAT.

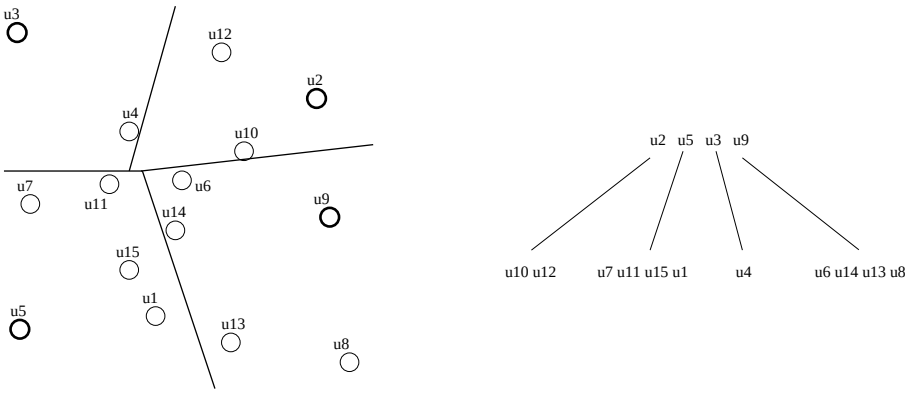


Figure 8: Example of the first level of a GNAT with $m = 4$.

The GNAT stores at each node an $O(m^2)$ size table $range_{ij} = [\min_{u \in \mathbb{U}_j}(c_i, u), \max_{u \in \mathbb{U}_j}(c_i, u)]$, with minimum and maximum distances from each center to each class. The tree requires $O(nm^2)$ space and is built in close to $O(nm \log_m n)$ time.

At search time, the query q is compared against some center c_i and we discard any other center c_j such that $d(q, c_i) \pm r$ does not intersect $range_{i,j}$, since all the subtree $\mathbb{U}_j$ can be discarded by the triangle inequality. The process is repeated with random centers until no one can be discarded. The search then enters recursively into each non discarded subtree. In the process, any center close enough to q is reported.

The performance of the GNAT heavily depends on the arity of the tree. The best arity is different for each metric space, and even for different subsets of the data set. In particular, the hard kernel happens to require a large arity, while the soft kernel is searched better with a smaller arity tree. Hence, we will illustrate our technique by choosing different arities for hk and sk.

5.2 Experimental Analysis

We experimented with a metric space of strings under the *edit distance* (also called "Levenshtein distance"). This function is discrete and computes the minimum number of characters that we have to append, change and/or delete from one word to obtain the other. This distance has applications in information retrieval, signal processing and computational biology.

We have used a Spanish dictionary of 86,061 words, and experimented with different values for s and cr. The experimental setup consist in finding the best arity for the whole dictionary, and then splitting it in several ways

to find a good proportion of soft/hard kernel tuning the individual arities of the proportions. Each combination is compared against the best arity for the whole dictionary. With this we ensure a fair comparison, since we compete with the best possible tuning of the GNAT for the whole dictionary, against individual tuning for the hard/soft kernels.

We chose 500 random words from the dictionary and searched for them using distance radii $r \in 1 \ldots 4$. For each search we computed the ratio of the search cost using separate indexes versus one global index. A value of 1 indicates the same performance in either approach, while a smaller value implies that the cluster-based approach is better than the standard one. Each point in the graph corresponds to the average measure of 500 queries, hence we expect a low variance for this measure.

Figure 9 shows the tuning of the GNATs, and we observe that an arity of 256 is the best we can do for low selectivity queries, while an arity of 128 is the best for high selectivity queries. We *must* choose a given arity if we will use only one index for all type of queries. We chose an arity of 256 to compare with the clustered approach.

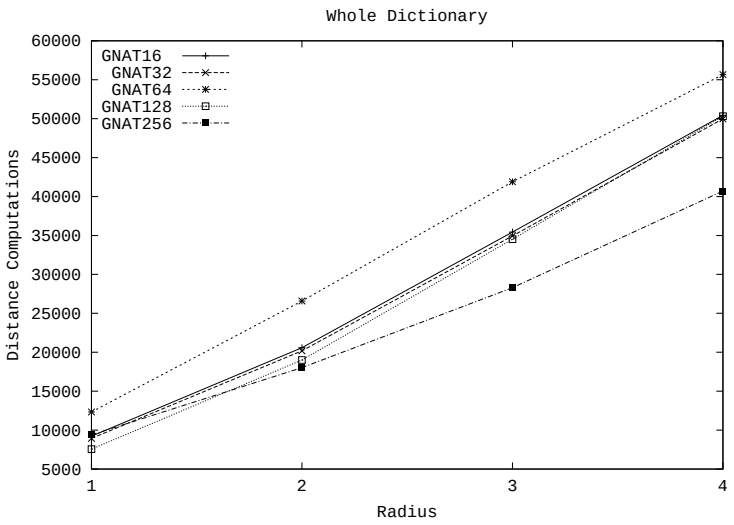


Figure 9: Tuning the GNAT. We try to find the best arity for the GNAT and found that this number is not the same for low/high selectivity queries. The best is GNAT-256 for most of the radii.

For each cutting radius, and each proportion we tested all the combinations of arities for the cutting proportion. In figure 10 we show an example of such an experiment, for a cutting radius of 2, and a proportion of 0.4 in

the hard kernel and 0.6 in the soft kernel and a fixed arity of 128 in the hard kernel, to find the best arity for the soft kernel. This experiment was exhaustive in all the possible combinations.

In figure 11 we show how the different proportions are compared to the original dictionary. We observe a systematic improvement for high selectivity queries, while the improvement proportion is smaller in low selectivity queries. This is a natural consequence of the non-monotonicity of the tuning of the GNAT, but it is not explained solely by this, since for metric range queries of radius 2, both arities 128 and 256 in figure 9 have the same performance. The clustered index is better even in this case. We also observe that a balanced partition is the best choice, using an arity of 128 in each segment.

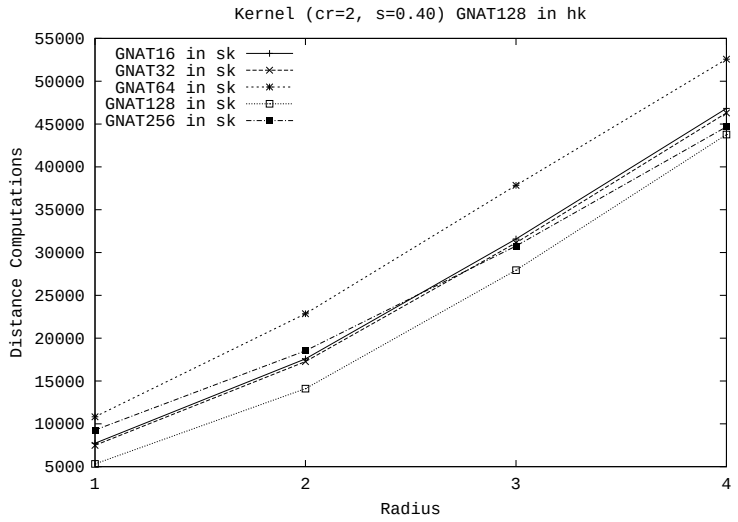


Figure 10: For each cutting radius and kernel proportion we selected the best combination of arities.

The example we have presented can be improved in a number of ways, for example by partitioning the data into more than two clusters, or by building a cluster hierarchy. More improvements can be expected by using different clustering strategies. The use of local histograms is a fast technique, but more costly techniques may produce a better segmentation.

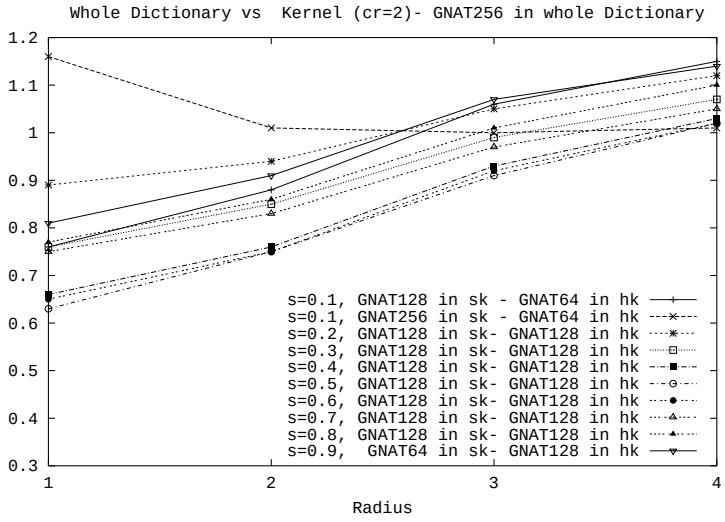


Figure 11: Once selected the best combination for each kernel proportion we compare it with the best-tuned GNAT. A balanced partition with arity 128 seems to be the best choice.

Acknowledgements

We acknowledge the support of CYTED Project VII.19 RIBIDI. The first and last author also acknowledges the support of the Center for Web Research, Millenium Research Initiative, Mideplan, Chile.

References

[BCQY97] M. R. Brito, E. L. Chávez, A. J. Quiroz, and J. E. Yukich. Connectivity of the mutual k-nearest-neighbor graph in clustering and outlier detection. *Statistics & Probability Letters*, 35:33–42, 1997.

[BN02] B. Bustos and G. Navarro. Probabilistic proximity searching based on compact partitions. In *Proc. 9th International Symposium on String Processing and Information Retrieval (SPIRE'02)*. LNCS, Springer-Verlag, 2002.

[Bri95] S. Brin. Near neighbor search in large metric spaces. In *Proc. 21st Conference on Very Large Databases (VLDB'95)*, pages 574–584, 1995.

[BYRN99] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, 1999.

[Cha02] E. Chávez. Knowledge based distances for sequence comparison. In *Sistemi Evoluti per Basi di Dati (SEBD)*, pages 34–47, 2002.

[CN00] E. Chávez and G. Navarro. An effective clustering algorithm to index high dimensional metric spaces. In *Proc. 7th International Symposium on String Processing and Information Retrieval (SPIRE'00)*, pages 75–86. IEEE CS Press, 2000.

[CNBYM01] E. Chávez, G. Navarro, R. Baeza-Yates, and J. Marroquín. Proximity searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.

[FL95] C. Faloutsos and K. Lin. Fastmap: a fast algorithm for indexing, data mining and visualization of traditional and multimedia datasets. *ACM SIGMOD Record*, 24(2):163–174, 1995.

[GRG+99] V. Ganti, R. Ramakrishnan, J. Gehrke, A. L. Powell, and J. C. French. Clustering large datasets in arbitrary metric spaces. In *International Conference of Data Engineering*, pages 502–511, 1999.

[Har75] J Hartigan. *Clustering Algorithms*. John Wiley & Sons, New York, 1975.

[Har95] D. Harman. Overview of the Third Text REtrieval Conference. In *Proc. Third Text REtrieval Conference (TREC-3)*, pages 1–19, 1995. NIST Special Publication 500-207.

[JD88] A.K. Jain and R.C. Dubes. *Algorithms for Clustering Data*. Prentice Hall Advanced Reference Series, 1988.

[MC85] G. W. Milligan and M. C. Cooper. An examination of procedures for determining the number of clusters in a data set. *Psychometrica*, 50:159–179, 1985.

[Zha71] C. T. Zhan. Graph theoretical methods for detecting and describing gestalt clusters. *IEEE Transactions on Computing*, 20:68–86, 1971.

[ZRL96] T. Zhang, R. Ramakrishman, and M. Livny. Birch: An efficient data clustering method for very large databases. In *ACM SIGMOD International Conference on Management of Data*, pages 103–114, 1996.