Dynamic User-Defined Similarity Searching in Semi-Structured Text Retrieval

Filippo Geraci
Istituto di Informatica e Telematica, CNR
Via G. Moruzzi 1
Pisa (Italy)
filippo.geraci@iit.cnr.it

Marco Pellegrini
Istituto di Informatica e Telematica, CNR
Via G. Moruzzi 1
Pisa (Italy)
marco.pellegrini@iit.cnr.it

ABSTRACT

Modern text retrieval systems often provide a *similarity search utility*, that allows the user to find efficiently a fixed number h of documents in the data set that are the *most similar* to a given query (here a query is either a simple sequence of keywords or a full document). We consider the case of a textual database made of semi-structured documents. For example, in a corpus of bibliographic records any record may be structured into three fields: title, authors and abstract, where each field is an unstructured free text. Each field, in turns, may be modelled with a specific vector space. The problem is more complex when we also allow users to associate at query time to each vector space a weight influencing its contribution to the overall *dynamic aggregated and weighted similarity*. We investigate the use of metric k-center clustering to prune the search space at query time. The embedding of the weights in the data structure is investigated with the purpose of allowing users query customization without any data replication. The validity of our approach is demonstrated experimentally by showing significant quality/time performance improvements over two state of the art methods. We also speed up the pre-processing time by a factor at least thirty with respect to a method based on k-means clustering.

Categories and Subject Descriptors

H.3.3 [**Information Storage And Retrieval**]: Information Search and Retrieval—*Clustering, Retrieval models*

General Terms

Information retrieval; Clustering

Keywords

Semi-Structured Text; personalized search

1. INTRODUCTION

Motivations. Collections of semi-structured texts and XML annotated text files are growing in size and scope. It is thus important to exploit structural information when searching these collections in order to better match the user's information need. Different structural elements can be weighted differently according to the specific search. For example, suppose one wants to retrieve from a database of books: "a book authored by *Aho*, with *Algorithm* in the title, with author score twice as important as the title score." Traditional un-weighted text similarity search mechanisms are unable to handle such query when the relative weight is defined on-the-fly by the user.

In a larger perspective we can envision three levels of usage for our dynamic searching tool. In the first level the user is an expert and he/she supplies directly weights to each field in a composite query. In the second level the query interface provides a few *adjustable templates* so that the user provides directly the parameters of these templates and the system maps them into proper weights. In the third level, the query interface has a feedback loop in which user's ok/not ok tagging is used to learn weights so to converge to the implicitly defined user metric. In all three scenarios it is essential that the underlying search mechanism to be able to adjust its weighted metric on-the-fly and independently for each user.

Background. Similarity Searching in an ubiquitous operation much studied in a variety of research communities including: data bases, spatial data bases, data structures, computational geometry, information retrieval (see recent surveys in [3, 14] and books [20]). In this paper we explore the relationship between the approximate similarity searching for textual data and the k-center problem [12]. We show how the cluster pruning strategy for approximate similarity searching can be improved using a clustering algorithm which approximate an optimal solution to the k-center problem. Moreover, since we observe that most of the nearest neighbors are found visiting the first few closest clusters we modify the cluster pruning approach to work with a multi-clustering obtained by merging few independent clusterings. Using properties of the cosine distance to measure the distance among pairs of documents, we derived a new scheme for the dynamic vector score aggregation problem. Our method is able to deal with weights given at query time with scalable preprocessing time and storage. Our method scales well also with the number of attributes (i.e. vector spaces) to be aggregated.

Infoscale2008, June 4-6, 2008, Vico Equense, Napoli, Italy

2. PREVIOUS WORK

Similarity searching with a fixed metric is a much studied problem and a survey of results is beyond the scope of this paper we mention few results most relevant to our research. Instead for dynamically user-defined metrics fewer results are known.

Cluster pruning is an approach to similarity searching that is rather simple but, because of its simplicity, it is suitable to handling very large data sets in very high dimensional spaces (as arise for example in handling large corpora of free textual information). Consider the data items as points in a high dimensional space endowed with a distance function. Subdivide the data points into many small compact clusters and elect a representative point in each cluster. When a query point q is given, q is compared with the representatives and based on this comparison one decides either to explore further the cluster or to disregard completely the associated cluster. The heuristic step (no guarantee) is that the selected clusters contain the exact (or approximate) answer we are looking for. There are many algorithmic design choices one has to take (see a recent paper [4] exploring many of these choice). The cluster pruning approach has been used in [13, 19].

Locality Sensitive Hashing proposed by Indyk and Motwani [16] reduces the approximate similarity searching problem to the problem of *point location in equal balls* (PLEB). Given a set $C = \{c_1, \ldots, c_n\}$ of points that are centers of balls $B(c_i, r)$ of a fixed radius r in a metric space and a query point q, the basic PLEB routine returns c_i if there exist an index i such that $q \in B(c_i, r)$. The authors introduce a family of hashing functions $\mathcal{H}$ and the main property of LSH scheme is that similar objects hashed with $\mathcal{H}$ are much more likely to collide than dissimilar objects. At query time the query q is hashed and only the points in the ball $B(c_i, r)$ such that $h(q) = h(c_i)$, for $h \in \mathcal{H}$ are scanned to find the points most similar to q. In [1] the LSH technique is extended and made more data adaptive. LSH is very sensitive to the metric used since a specific hashing function family has to be built for every metric. It is not clear how to extend this scheme to user-defined metrics.

Random Projections. Fagin et al. in [8] show that one can solve approximate Euclidean nearest neighbor by first projecting data points and query points onto a set of 1-dimensional flats (lines), compute the rank of the query in each 1-dimensional space, and then combine (aggregate) these ranks using deep techniques from *voting theory.* Experiments reported in [8] are on dense data sets in dimension 100 (stock market time series) and dimension 784 (vectorialization of digital images). It is not clear how to extend this technique to different metrics and to dynamically changing metrics.

2.1 User Defined and Aggregated Metrics

In the standard version of the similarity searching problem the user is allowed to choose the queries, but not the underlying distance function that is fixed at pre-processing time. Suppose now that we want to give the user the possibility of choosing a metric of his/her own choice at query time. One special case of this scenario is for example when the objects to be searched have an internal structure and the overall distance function is an *aggregation* of the outcome of several distance functions defined on the components of the structure. Even if the data set to be searched is the same, at different times the user might prefer a different relative weight of the individual factors within the overall aggregated distance function depending on the purpose of the query. This choice is taken at query time and the data structure must be flexible in this respect. This scenario has been considered in a recent paper by Singitham et al. [18] (See section 5). In [5] Ciaccia and Patella discuss which general relations should hold between two metrics that allow to build a data structure using the first metric, but perform searches using the second (e.g. a user defined) one. They propose a method that can be applied to any generic distance based search tree. Its performance analysis is based on probabilistic distance distributions and shows a degradation of performance depending on the "distance" among the metrics used.

Another scheme in [17] handles metrics d_A parameterized by a matrix A: $d_A(p, q) = (p-q)A(p-q)^T$, where the Euclidean metric is a special case corresponding to the identity matrix. This approach is very general, however the evaluation of the basic point-box-distance primitive involves an iterative steepest descent approach, thus it is unsuitable for handling high dimensional data in real time.

Bustos and Skopal [2] extend the M-tree data structure to handle multiple metrics. The proposed data structure is tested on a data set of images mapped as points in 89-dimensional real space and to a collection of 3d-models mapped as points 84-dimensional real space. In both cases four L_1 metrics are applied to a 4-partition of the 89 (resp 84) dimensions. Such an approach might not be suitable to the much higher dimensional spaces found when textual data is considered.

3. OUR CONTRIBUTION

In this paper we first deal with the problem of fast approximate similarity searching for semi-structured text documents. Then we focus on the more general problem in which users can decide to dynamically assign different weights to each field of the searched text (Dynamic Vector Score Aggregation). We observed an analogy between the similarity searching problem and the k-center objective function for clustering. Thus, according to the cluster pruning approach, we used a clustering algorithm for the k-center problem for approximate similarity searching. Moreover, we derived a new and simple way for managing dynamic weights: we avoid to consider them altogether in the preprocessing phase and thus we avoid *duplication of vectors* that wastes storage. Finally, by using a multi-clustering strategy (in which we introduced a certain data redundancy but we *duplicate only indices*) we obtain further benefits in terms of precision. In particular we will describe alternatives for the following key aspects:

The ground clustering algorithm. When searching for nearest neighbors of a query point q using the cluster pruning approach it is natural to consider a clustering good for such a search when the clusters are well separated and the

diameter of the clusters is small. This observation leads to considering the optimal k-center problem (i.e. finding a decomposition minimizing the maximum diameter of any cluster produced) as a natural objective function to optimize. Thus we are led to consider the Furthest-Point-First (FPF) heuristic of Gonzalez [12], that is 2-competitive for the metric k-center problem. We will show (Theorem 1) that when the cosine distance, which is not a metric, is used it is possible to attain a solution with maximum diameter within a factor 4 of the optimum possible. Experiments in Section 5 comparing FPF with a randomized variant M-FPF show that we attain two practical benefits: (1) quality of the output is increased and (2) preprocessing time is reduced by orders of magnitude since we can use fast variants of the FPF algorithm.

How weights are embedded in the scheme. In the general Vector Score Aggregation problem the user supplies a query (this can be either a document in the database or a collection of keywords that capture the concept being searched for) and a weight-vector that express the user's perception of the relative importance of the document features in capturing the informal notion of "similarity". We show in this paper that, surprisingly, one need not be concerned with dynamic weights at all during pre-processing, the solution for the un-weighted case is good also for the weighted one since tight upper and lower bounds can be derived for the weighted distance in terms of the un-weighted one (Theorem 2). Experiments in Section 5 show that there is no loss of performance when handling weights with the proposed scheme. An additional benefit is that the number of vector spaces to be aggregated can be arbitrarily large without any performance penalty, or extra storage needed.

Multiple clusterings. In cluster pruning search one decides beforehand to visit a certain number of clusters whose "leaders" are closest to the query point. However, there is a hidden law of diminishing returns, that our experiments highlight: clusters further away from the query are less likely to contain good neighbors. We use a different strategy: we form not one but several (three in our experiments) different independent clusterings and we search all three of them but looking into fewer clusters in each clustering. This scheme leads to higher output quality without significant time and storage penalties since documents/vectors do not need to be replicated (see Section 5).

By introducing these three new concepts, our experiments show that we can significantly improve over the performance of state of the art algorithms for this problem.

3.1 Clustering Algorithms

Basic FPF Algorithm [12]. Given the set S of n points and a metric distance D, FPF incrementally computes the set of centers $C_1 \subset \cdots \subset C_k \subseteq S$, where C_k is the solution to the problem and $C_1 = \{c_1\}$ is the starting set, built by randomly choosing c_1 in S. At a generic iteration $1 < i \leq k$, the algorithm knows the set of centers C_{i-1} (computed at the previous iteration) and a mapping μ that associates, to each point $p \in S$, its closest center $\mu(p) \in C_{i-1}$. Iteration i consists of the following two steps:

1. Find the point $p \in S$ for which the distance to its closest center, $D(p, \mu(p))$, is maximum; make p a new center c_i and let $C_i = C_{i-1} \cup \{c_i\}$.
2. Compute the distance of c_i to all points in $S \setminus C_{i-1}$ to update the mapping μ of points to their closest center.

After k iterations, the set of center $C_k = \{c_1, \ldots, c_k\}$ and mapping μ define the clustering: cluster $\mathcal{C}_i$ is defined as the set $\{p \in S \setminus C_k \mid \mu(p) = c_i\}$, for $i = 1, \ldots, k$. Each iteration can be done in time $O(n)$, hence the overall cost of the algorithm is $O(kn)$. Experiments have shown that the random choice of c_1 to initialize C_1 does not affect neither the effectiveness nor the efficiency of the algorithm. We introduce a randomized variant of FPF, called M-FPF, as follows: apply FPF not to the whole set S but only to a random sample $R \subset S$ of size $\sqrt{nk}$ of the input points (this sample size suggested in [15]), afterwards add the other points in $S \setminus R$ to the cluster of their closest center, one by one.

Despite no formal properties has yet been demonstrated, experimentally M-FPF has consistently shown to be more robust to outliers than FPF. This is due to the fact that outliers are typically far from all the other points, therefore in the procedure of selection of a new center, FPF is likely to select one of them as next center. The sampling applied to M-FPF tends to mitigate this effect. Moreover M-FPF can be easily executed in parallel in the MapReduce model [7]. In section 5.3 we compare FPF and M-FPF for the similarity searching task. Our experiments confirm that M-FPF consistently attains better quality with respect to the FPF.

FPF and the cosine distance. In this paper we will use mostly distance measures, therefore we will convert all results and algorithms for similarity measures into distances. As noted in [6] the inner product of two positive vectors x and y of length 1 (in norm L_2) that is the standard cosine similarity of two normalized vector used in Information Retrieval is turned into a *cosine distance* $d(x, y) = 1 - x \cdot y$. This cosine distance function is not a metric in a strict sense since the triangular inequality is not satisfied, however the following derivation $\|x - y\|_2^2 = x \cdot x + y \cdot y - 2x \cdot y = 2(1 - x \cdot y) = 2d(x, y)$ shows that the square root $b(x, y) = \sqrt{d(x, y)}$ of the cosine distance is indeed a metric. Equivalently one can say that the cosine distance satisfies the extended triangular inequality $(d(x, y)^\alpha + d(y, z)^\alpha \geq d(x, z)^\alpha)$ with parameter $\alpha = 1/2$. Let P be a set of unit vectors, and $Q_C(P, k)$ be the maximum diameter of any set in a partition C of P into k sets. The k-center optimality criterion is given by the clustering that minimizes $Q_C(P, k)$ over all possible choices of C, for a given input set P and cluster number k.

THEOREM 1. *The FPF algorithm in [12] with the cosine distance produces a clustering whose maximum diameter is 4-competitive for k-center optimality criterion.*

Proof sketch. The algorithm FPF uses the inter-point distances only for comparisons, therefore the same output clustering is obtained applying the cosine distance $d(x, y)$ or its square root $b(x, y)$. The proof of 2-competitiveness in [12] that holds for any metric can be used to determine that

the computed maximum diameter w.r.t metric b (denoted as D_b^{comp}) is within a factor two of the value given by the optimality criterion (denoted by D_b^{opt}). Since the square root is a monotone transformation, it holds that $(D_b^{opt})^2 = D_d^{opt}$, therefore $D_d^{comp} \leq 4D_d^{opt}$. □

Theorem 1 indicates that FPF maintains its good properties when the cosine distance function is used.

Clustering Algorithms and Similarity searching. The cluster pruning approach to similarity searching leaves open the issue of which is the best clustering strategy to adopt. In [4] good probabilistic bounds are proved within a generative model for the input data that assumes the data to be generated according to a hierarchical Gaussian distribution. Devising a method provably good for any input is an hard open problem. However, some simple heuristic considerations can be done. Recalling that the optimal k-center problem is to find an assignment of cluster centers that minimizes the maximum diameter of the clusters, a bound is also imposed to the radius. The found clusters are intuitively rather compact. If the query point q is exactly the center of cluster C_i, its elements are the first $|C_i|$ nearest neighbors of q. The more a query point q is close to a cluster center, the more it is likely that its nearest neighbors are points of the cluster. More specifically, if the cluster radius is r and the distance $d(q, c_i) < r$, all the nearest neighbors at distance at most $r - d(q, c_i)$ are points of C_i. Since k-center imposes that the wider cluster diameter is the smallest possible, in the case queries are points of the clustering, this means that also the maximal distance between the query and the closest center is minimized.

3.2 Embedding weights in the scheme

In the aggregated vector score model the queries are points of the form $q = (q_1, .., q_s)$ where each q_i is a vector of unit length, moreover the user supplies a weight vector $w = (w_1, .., w_s)$ where each w_i is a positive scalar weight, and the weights sum to 1. The point p_j in the input set P is of the form $((p_j)_1, \ldots, (p_j)_s)$ where each $(p_j)_i$ is a vector of unit length. The aggregate similarity is: $s_A(q, p_j) = 1 - d_A(q, p_j) = \sum_i w_i(q_i \cdot (p_j)_i)$ where $q_i \cdot (p_j)_i$ is the cosine similarity between q_i and $(p_j)_i$. The aggregate distance function is $d_A(q, p_j) = 1 - \sum_i w_i(q_i \cdot (p_j)_i)$. One should notice that because of the linearity of the summation and the inner product operators the weights can be associated to the data vectors or to the query vectors : $\sum_i w_i(q_i \cdot (p_j)_i) = \sum_i q_i \cdot w_i(p_j)_i = q \cdot wp_j$. This association has been chosen in [18] thus the challenge arises from the fact that one has to do pre-processing of P without knowing the real weights that are supplied on-line at query time. Now we establish a theorem that gives upper and lower bounds to the weighted distance between a query point to a data point in terms of the properties of an *unweighted* clustering. Given vectors q and w, let $Q_w = [w_1 q_1, .., w_s q_s]$ the concatenation of s weighted vectors, and define the *normalized equivalent point* $Q'_w = Q_w/|Q_w|$.

THEOREM 2. *Let q be a query point and w a query weight-vector and Q'_w the corresponding normalized equivalent point. For any two points p and c it holds: $d(Q'_w, p)^\alpha \leq d(Q'_w, c)^\alpha + d(c, p)^\alpha$ (upper bound) and $d(Q'_w, p)^\alpha \geq d(Q'_w, c)^\alpha - d(c, p)^\alpha$ (lower bound).*

Proof sketch. Consider now the *weighted similarity* WS: $WS(w, q, p) = \sum_i w_i(p_i \cdot q_i) = \sum_i (w_i q_i) \cdot p_i = Q_w \cdot p$. Since the linear combination of weights and queries might not result in a unit length vector we perform a normalization (depending only in the weights and query point) and obtain a *normalized weighted distance* NWD: $NWD(w, q, p) = 1 - WS(w, q, p)/|Q_w| = 1 - Q_w/|Q_w| \cdot p = d(Q'_w, p)$. Now we are in the condition of using the generalized triangular inequality for the cosine distance and establish the upper bound:

$$NWD(w, q, p) = d(Q'_w, p) \leq (d(Q'_w, c)^\alpha + d(c, p)^\alpha))^{1/\alpha}.$$

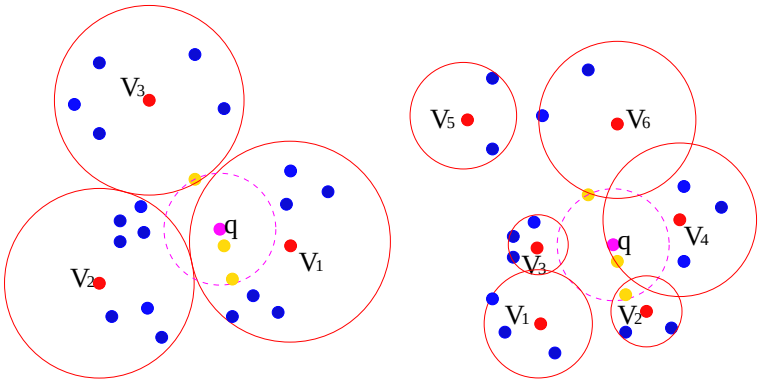
The lower bound is obtained symmetrically by exchanging the role of c and p. □

Theorem 2 shows upper and lower bounds for the weighted distance between a query point q and a data point p that have two components: the first one is the weighted distance to another point c (typically the center of a cluster), while the second component depends only on the un-weighted distance of p and c. This result suggests the following algorithm for searching the nearest neighbors of q in P according to the weighted cosine distance. Off-line we compute a k-clustering of P with the FPF algorithm using the unweighted cosine distance. For each cluster index $i = [1, .., k]$, let c_i be the center of the cluster, and d_i be its radius (i.e. the distance from c_i to the furthest point in the i-th cluster). On line, when (q, w) are given, compute the lower bound $d(Q'_w, c_i) - d_i$ for all clusters, in time $O(k)$. Visit the clusters according to the increasing order of lower bounds performing the cluster pruning search. The performance of this scheme is explored experimentally in section 5 and the empirical results indicate that the recall and normalized aggregated goodness do not suffer any performance degradation with respect to the un-weighted similarity searching problem.

3.3 Multiple clusterings

Unfortunately, it does not suffice to run a k-center algorithm on the input data (instead of k-means like in [18]) to get better quality. The situation is slightly more complex.

Generally speaking k-means is much slower than k-center, but typically produces higher quality clusters (the iterative process has a smoothing effect). We have found that, while a single application of M-FPF is not able to beat k-means in quality, a few independent applications of M-FPF to different random samples of the input points (*multi-clustering*) produce sets of overlapping clusters that yield better output quality. As a preliminary step we studied how helpful it is the visit of one more cluster after examining a certain number t of clusters. The *marginal utility* is computed for both Normalized Aggregated Goodness and Recall (see definitions in Section 4) and it is expressed as the difference between the value of the measure visiting t and $t+1$ clusters. In Figure 2 we show the *marginal utility* at the t-th cluster, for increasing values of t. Figure 2 demonstrates that the



(a) CL1 (b) CL2

Figure 1: The same set of points is divided in 3 clusters at the top (a) and 6 at the bottom (b). The query point q is shown in magenta and the 3 nearest neighbors yellow.

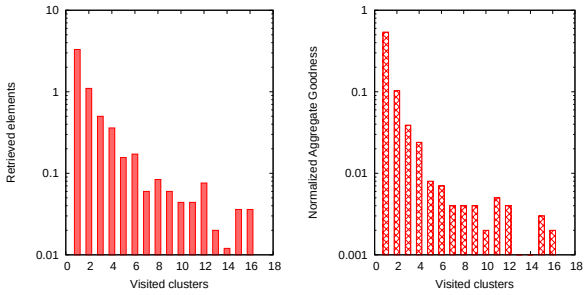


Figure 2: The *marginal utility* of examining the cluster $t+1$ after the inspection of t clusters. On the x axe the number of visited clusters, on the y axe the *marginal utility* in terms of recall (left) and normalized aggregate Goodness (right).

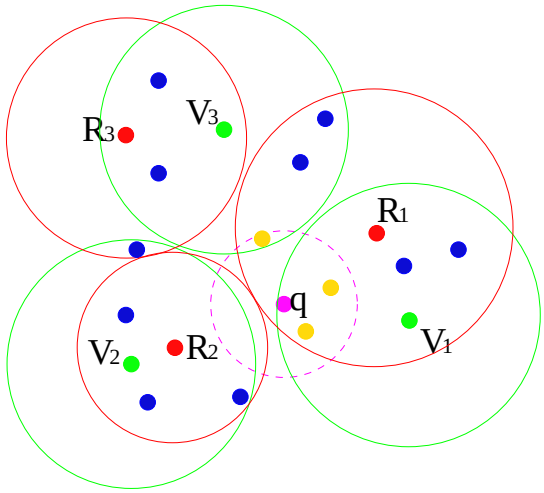


Figure 3: Two independent clusterings (one in red and one in green) of the same set of points. In magenta a query point and in yellow its nearest neighbors.

visit of the first cluster is the most important and the *marginal utility* of examining a new cluster decays drastically after just the visit of three or four clusters. This suggests that it might be better to examine fewer clusters in some independent clustering than many clusters of the same clustering. Experiments in section 5 show that we have a gain in quality.

A second observations that suggested us to use multi-clustering was that the redundancy of points makes the system more stable in terms of query performance. This effect can be better explained with the example of Figures 3 and 1.

In Figure 3 the red circles and the green circles represent two independent clusterings (the center of each cluster is the point of the same color). Let the point q be the query, and the yellow points its nearest neighbors. Suppose we want to visit two clusters searching for nearest neighbors. If one considers only red clusters (corresponding to centers R1, R2, and R3), q is closer to R_1 and R_2, thus all nearest neighbors are found. Instead, considering the green clustering (corresponding to centers V1, V2 and V3), point q is closer to V_1 and V_2, but one of the nearest neighbors is in V_3. If one considers all the red and green clusters as a single clustering, then R_1 and V_1 are the closest centers and all the nearest neighbors are found. It is easy to note that in the last case the clusters with highest *marginal utility* are selected, thus increasing the final expected quality. The example of Figure 3 also shows that the mean distance among the query point and the nearest centers in the multi-clustering scheme can be equal or smaller with respect to the case of a single clustering scheme.

At this point one could, erroneously, think that the advantage we gain is just due to the higher number of centers and not to the combination of independent clusterings. More-

over, multi-clustering has the disadvantage that there could be points whose distance to the query is evaluated more then once. An alternative could be to make a single clustering with twice as many smaller clusters and increase the number of visited clusters balancing the final computational cost of the two approaches.

The example of Figure 1 shows on the left (a) a set of points divided in three clusters (CL1) and on the right (b) the same set split in six clusters (CL2). Given the query q, we examine 2 clusters on CL1 and 4 on CL2. Note that only the cost of finding the nearest centers increase searching the structure on the right. In fact, since visited clusters in CL2 are twice those visited in CL1, this balance the lower expected number of elements in each visited cluster. In both cases there is a nearest neighbor that is not found and the higher number of clusters in (b) does not help.

Comparing k-means and Multi-clustering, the latter has an extra cost at query time in terms of number of distance computations to determine the t clusters to explore. However, since each distance computations in M-FPF involves only sparse vectors (i.e. there are no dense centroids), the two effects balance out in the query time. In our experiments three applications of the M-FPF algorithm for k-center to different random samples of the input suffice. There are two possibilities in querying a multi-clustering. Suppose one wants to examine t different clusters at query time, the system can consider the three k-clusterings as a single clustering structure with $3k$ clusters (and centers) and examine the t clusters closest to the query point, or it can query independently the three clusterings visiting $t/3$ clusters for each structure. We observed that this choice does not affect the final result neither in terms of quality nor in terms of speed, thus we decided to use the latter strategy.

4. MEASURING OUTPUT QUALITY

Two popular quality indexes for approximate similarity searching are often used: the *mean competitive recall* and the *mean normalized aggregate goodness* [18][4].

Mean Competitive Recall. Let h be the number of similar documents we want to find (in our experiments h=10) and $A(P, q, h)$ the set of the h retrieved documents for a query q by algorithm A on data set P, and the *Ground Truth* $GT(P, q, h)$, the set of the h closest points in P to the query q which is found through an exhaustive search; the competitive recall is $CR_A(P, q, h) = |A(P, q, h) \cap GT(P, q, h)|$. Note that competitive recall is an integer number in the range $[0, \ldots, h]$ and a higher value indicated higher quality. The Mean Competitive Recall $\overline{CR}(P, Q, h)$ is the average of the competitive recall over a set of queries Q. This measure tell us how many of the true h nearest neighbors an algorithm was able to find.

Mean Normalized Aggregate Goodness. We define as the Farthest Set $FS(P, q, h)$ the set of h points in P farthest from q. Let the sum of distances of the h furthest points from q be $W(P, q, h) = \sum_{p \in FS(P,q,h)} d(q, p)$. The normalized aggregate goodness:

$$NAG_A(P, q, h) = \frac{W(P, q, h) - \sum_{p \in A(P,q,h)} d(q, p)}{W(P, q, h) - \sum_{p \in GT(P,q,h)} d(q, p)}.$$

Note that the Normalized Aggregate goodness is a real number in the range $[0, .., 1]$ and a higher value indicated higher quality. The Mean Normalized Aggregate Goodness is the average of the normalized aggregate goodness over a set of queries Q (denoted with $\overline{NAG}(P, Q, h)$). This normalization allows us a finer appreciation of the different algorithms by factoring out border effects.

5. EXPERIMENTS

In this section we will first compare M-FPF against the standard FPF showing that the former consistently beat the latter in terms of quality. Then we show benefits obtained using multiple clustering strategy with respect to standard clustering. Finally, we compare our solution for the weighted problem against two baselines: the algorithm in [18] that uses as a subroutine k-means clustering and the algorithm [18] modified so to use a simple cluster pruning randomized strategy proposed in [4] in place of k-means.

5.1 Baseline algorithms

In [18] several schemes and variants are compared but experiments show that the best performance is consistently attained by Query Algorithm 3 (*CellDec*) described in [18, Section 5.4]. The preprocessing is as follows. For simplicity we consider the 3-dimensional case, that is a data set where each record has 3 distinct fields (e.g. in our tests, title, authors and abstract of a paper). We consider the set T of positive weight-vectors summing to one (this is the intersection of the hyperplane $w_1 + w_2 + w_3 = 1$ with the positive coordinate octant). We split T into 4 equal regular triangles T_1, T_2 and T_3 each incident to a vertex of T and the central region T_4. For each region we build a different vector space. Let $V_{i,j}$ be the vector corresponding to point p_j and field i. Since the value of weights in region T_4 (the central one) are comparable, we form a composite vector as follows $V(T_4)_j = V_{1,j} + V_{2,j} + V_{3,j}$. For the other three regions we simply apply a squeeze factor θ in correspondence of the two lowest weights. Thus we have $V(T_1)_j = V_{1_j} + \theta V_{2,j} + \theta V_{3,j}$, $V(T_2)_j = \theta V_{1_j} + V_{2,j} + \theta V_{3,j}$ and $V(T_3)_j = \theta V_{1_j} + \theta V_{2,j} + V_{3,j}$. Experiments in [18] show that a value of $\theta = 0.5$ attains the best results. At query time, given the query (q, w) one first detects the region of T containing w, then uses q in the associated indexing data structure for cluster-pruning.

In [4] Chierichetti et al. propose a very simple but effective scheme for doing approximate nearest neighbor search for documents. In a nutshell, after mapping n documents in a vector space they choose randomly $K = \sqrt{n}$ such documents as representatives, and associated each other document to its closest representative. Afterwards, for each group the *centroid* is computed as "leader" of the group to be used during the search. In [4] the authors are able to prove probabilistic bounds on the size of each group which is an important parameter that directly influences the time complexity of the cluster prune search. Dynamically weighted queries are not treated in [4], therefore we choose as a second base-line to employ [4], in place of k-means, within the weighting framework of [18]. We will refer to it as *PODS07* for lack of a better name.

# Visited	NAG		Recall		Time	
	Multi	M-FPF	Multi	M-FPF	Multi	M-FPF
3	0.842	0.755	6.884	5.612	0.619	0.298
6	**0.887**	0.786	**7.688**	6.352	**0.692**	0.384
9	0.907	0.796	8.096	6.28	0.828	0.440
12	0.915	0.808	8.292	6.448	0.886	0.539
15	0.921	0.810	8.408	6.488	0.942	0.551
18	0.925	0.812	8.508	6.548	0.988	0.593
21	0.927	0.816	8.528	6.616	1.068	0.636
24	0.928	**0.818**	8.548	**6.628**	1.087	**0.691**

Table 1: Comparison of multi-clustering (multi) and M-FPF algorithms on TS2. Highlighted entries are results obtained in the same query time

5.2 Experimental setup

We implemented all the algorithms in Python. Data were stored in textual *bsd* databases. Tests have been run on a Intel(R) Pentium(R) D CPU 3.2GHz with 3GB of RAM and with operating System Linux. Following [18] we have downloaded the first one hundred thousands Citeseer bibliographic records[1]. Each record contains three fields: paper title, authors and abstract. We built two data sets described in Table 2.

Dataset	TS1	TS2
Input size (MB)	41.80	76.13
# Records	53722	100000
# Clusters	500	1000

Table 2: Dataset description

This 100K collection, that has been used for tests in [18], is not very large in absolute terms: the complete Citeseer has has over 700,000 documents. However, it is large enough to form a basis for comparing different algorithmic designs. Note that the Citeseer collection is formed of rather high quality documents, and this is consistent with a scenario where similarity search utilities are indeed more useful to the users. Extensions to other collections of lower quality documents (e.g. web pages) require the adoption of specific data cleaning phases and will be left as future research.

In Table 2 is also reported the number k of clusters made by all the considered algorithms. Note that the average cluster size is roughly preserved. After applying standard stemming and stop words removal, three vector spaces were created: one for each field of the documents. Terms in the vector are weighted according to the standard *tf-idf* scheme. Without loss of generality, as queries we used documents extracted from the data set. Test queries have been selected by picking a random set of 250 documents. During searches the exact match of the query document is not counted. In our experiments we used the seven weight-schemes reported in Table 3 for the three fields (title, authors and abstract), adopted also in [18]. For each set of weights, we always used the same query set.

[1] `http://citeseer.ist.psu.edu/`

Weight clue	Author	Title	Abstract
1	0.33	0.33	0.33
2	0.4	0.4	0.2
3	0.4	0.2	0.4
4	0.2	0.4	0.4
5	0.6	0.2	0.2
6	0.2	0.6	0.2
7	0.2	0.2	0.6

Table 3: Weights used for queries

5.3 Comparison among different k-center algorithms

Figure 4 shows a comparison of the two clustering algorithms for the k-center problem: FPF and M-FPF, using the largest TS2 dataset. We obtained analogous results using the TS1 dataset. Observe that M-FPF obtain better results in all cases in terms of both Normalized Aggregate Goodness and Recall. We obtained similar results also comparing M-FPF with respect to several other variants of FPF described in [10, 11, 9] (data not shown here).

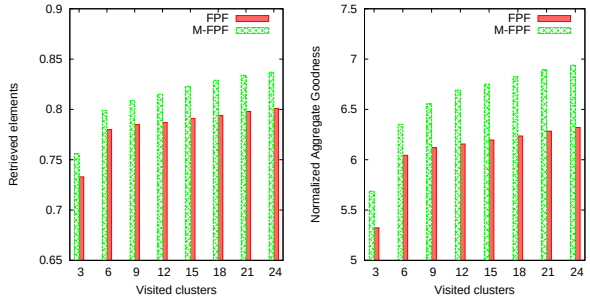
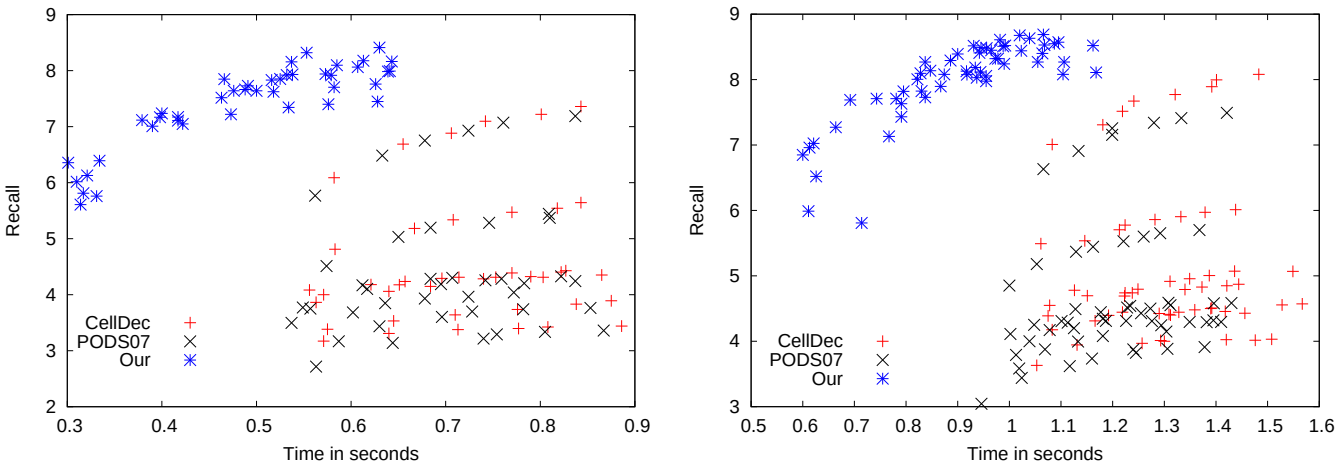


Figure 4: Comparison of FPF, M-FPF, algorithms on TS2. Recall (left) is a number in [0,10], Normalized Aggregated Goodness (right) is a number in [0,1].

The query time of FPF and M-FPF is very similar: the time varies in the range of 0.3 seconds for visiting 3 clusters up to 0.9 seconds for visiting 24 clusters.

5.4 Improved precision using multi-clustering

Results in previous section shows that M-FPF achieves a better performance with respect to FPF. In this section we



(a) Data Set TS1

(b) Data Set TS2

Figure 5: Recall of 10 nearest neighbors as a function of query time. Each point in the graph is the average of measurements of all queries for a class of weights and a number of visited clusters. The points in the upper left corner of the graphs corresponding to our algorithm show clear dominance.

discuss experimental results that show how multi-clustering works better than simple clustering. As remarked in Section 3, there are two main ways to query multi-clustering: query independently each clustering or merge them as a whole single clustering. Experiments (not reported here) show that in most of the cases both these querying strategies return exactly the same set of objects in comparable time. We report in Table 1 a comparison between multi-clustering and single clustering (both based on M-FPF) in terms of Normalized Aggregate Goodness, Recall and query time. Results show that multi-clustering achieve better quality than M-FPF but spends more time. Clearly, visiting the same number of clusters, multi-clustering querying is slower than querying a single clustering. This is due to the higher number of centers against which query must be compared. In our case we made three independent clusterings of 1000 elements in the case of TS2 (500 in the case of TS1). Thus an additional cost of 2000 distance invocations is paid (equivalent on average to about 0.4 seconds). It is interesting to note that querying multi-clustering remain qualitatively better than M-FPF even in the case the querying processes on the two clusterings are constrained to spend the same amount of time. In Table 1 we highlight those results in which M-FPF and multi-clustering spend about the same time for querying. In this case multi-clustering had time to visit only 6 clusters, while M-FPF visited 24 clusters. Also in this case multi-clustering performs better then M-FPF in terms of both Normalized Aggregated Goodness and Competitive Recall.

5.5 Comparison of the whole systems

In this section we compare our Multi-clustering algorithm against the two baseline algorithms in the most general setting in which dynamically user-defined score are allowed. As shown in Table 4, the simpler clustering strategy in [4] has preprocessing time close to our while [18] is orders of magnitude slower. In a test with 100,000 documents, we gain a speedup factor of 30. In practice we could complete the preprocessing in one day compared to one month required by [18] on a single processor machine. However note that in Table 5 and Figure 5, the quality/cost performance of PODS07 is inferior to our scheme and to that in [18] both for the weighted and the un-weighted cases.

Dataset	TS1		
Algorithm	Our	CellDec	PODS07
Preprocessing time	5:28	215:48	7:18
Space (MB)	332.078	1407.656	1402.140
Dataset	TS2		
Algorithm	Our	CellDec	PODS07
Preprocessing time	20:13	636.80	22:56
Space (MB)	645.765	2738.671	2725.078

Table 4: Preprocessing time (in hours and minutes) and storage (in Megabytes) of the data structures generated by *CellDec*, PODS07 and our algorithm.

Query quality and speed. Figure 5 shows in synthesis the query time/recall tradeoff of the three methods. In the graph each dot represent the mean of 250 queries for a given choice of number of clusters to visit and user weights (see Table 5). The 250 queries were selected only once and submitted to each algorithm and weights assignment. Our method is clearly dominant giving consistently better quality results in less time. Quality data are also given in tabular form in Table 5. Although we test on average more points, we have a speed up due to the fact of avoiding the use dense vectors (centroids). The top portion of Table 5 corresponds

to the case of equal weights. The entries of Table 5 for unequal weights indicate that our scheme is quite superior in recall, even doubling the number of true k-nearest neighbors found using less time over both baselines. The overall quality of the retrieved nearest neighbors, as measure via the *normalized aggregated goodness*, is also generally improved: this indicates that our method is robust and stable relative to the baselines.

6. CONCLUSIONS

In this paper we tackled the similarity searching problem in semi-structured text documents with user-defined metrics. We provided an approximated solution based on cluster pruning. Our method is inspired by the observation that there is an intuitive relationship between the k-center problem and the similarity searching problem, thus the former can be helpful to approximate the latter. We also introduced multi-clustering that, by introducing redundancy in the input data, makes the whole system more stable in terms of output quality. Moreover, we have shown that a difficult searching problem with dynamically chosen weights can be reduced, thanks to the linearity properties of the cosine similarity metric, to a simpler static search problem. For this problem we provide efficient and effective method that is competitive with state of the art techniques for large semi-structured textual databases.

7. REFERENCES

[1] M. Bawa, T. Condie, and P. Ganesan. Lsh forest: self-tuning indexes for similarity search. In *Proceedings WWW 2005*, pages 651–660, 2005.

[2] B. Bustos and T. Skopal. Dynamic similarity search in multi-metric spaces. In *Proceedings of MIR '06*, pages 137–146, 2006.

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

[4] F. Chierichetti, A. Panconesi, P. Raghavan, M. Sozio, A. Tiberi, and E. Upfal. Finding near neighbors through cluster pruning. In *Proceedings of ACM PODS*, pages 103–112, 2007.

[5] P. Ciaccia and M. Patella. Searching in metric spaces with user-defined and approximate distances. *ACM Trans. Database Syst.*, 27(4):398–437, 2002.

[6] K. Clarkson. Nearest neighbour searching and metric space dimensions. In G. Shakhnarovich, T. Darrell, and P. Indyk, editors, *Nearest-Neighbor Methods for Learning and Vision: Theory and Practice*, pages 15–59. MIT Press, 2006.

[7] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

[8] R. Fagin, R. Kumar, and D. Sivakumar. Efficient similarity search and classification via rank aggregation. In *Proceedings of ACM SIGMOD '03*, pages 301–312. ACM Press, 2003.

[9] M. Furini, F. Geraci, M. Montangero, and M. Pellegrini. Visto: Visual storyboard for web video browsing. In *In Proceedings of the ACM International Conference on Image and Video Retrieval (CIVR 2007)*, pages 635–642, 2007.

[10] F. Geraci, M. Pellegrini, P. Pisati, and F. Sebastiani. A scalable algorithm for high-quality clustering of Web snippets. In *Proceedings of ACM SAC'06*, pages 1058–1062, 2006.

[11] F. Geraci, M. Pellegrini, F. Sebastiani, and M. Maggini. Cluster generation and cluster labelling for web snippets. In *Proceedings of SPIRE '06*, pages 25–36, 2006.

[12] T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38(2/3):293–306, 1985.

[13] J. Hafner, N. Megiddo, and E. Upfal. Fast query search in large dimansion database. US patent 5848404, 1998.

[14] G. R. Hjaltason and H. Samet. Index-driven similarity search in metric spaces. *ACM Trans. Database Syst.*, 28(4):517–580, 2003.

[15] P. Indyk. Sublinear time algorithms for metric space problems. In *Proceedings of STOC-99, ACM Symposium on Theory of Computing*, pages 428–434, 1999.

[16] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proc. of 30th STOC*, pages 604–613, 1998.

[17] T. Seidl and H.-P. Kriegel. Efficient user-adaptable similarity search in large multimedia databases. In *The VLDB Journal*, pages 506–515, 1997.

[18] P. K. C. Singitham, M. S. Mahabhashyam, and P. Raghavan. Efficiency-quality tradeoffs for vector score aggregation. In *VLDB*, pages 624–635, 2004.

[19] S. Sitarama, U. Mahadevan, and M. Abrol. Efficient cluster representation in similar document search. In *WWW (Alternate Paper Tracks)*, 2003.

[20] P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity Search: The Metric Space Approach*, volume 32 of *Advances in Database Systems*. Springer Verlag, 2006.

		Data Set TS1 = 50K docs.						Data Set TS2 = 100K docs.						
Visited clusters		3	6	9	12	15	18	3	6	9	12	15	18	21
		Query weights 0.33-0.33-0.34 - CellDec weights 1-1-1												
Recall	CellDec	6.088	6.688	6.884	7.096	7.22	7.36	7.008	7.308	7.516	7.672	7.772	7.892	7.996
	PODS07	5.768	6.484	6.752	6.928	7.072	7.188	6.044	6.632	6.908	7.156	7.256	7.34	7.412
	Our	6.016	7.172	7.64	7.852	7.94	7.992	6.884	7.688	8.096	8.292	8.408	8.508	8.528
NAG	CellDec	0.779	0.822	0.841	0.854	0.865	0.876	0.858	0.876	0.891	0.905	0.914	0.919	0.924
	PODS07	0.753	0.816	0.831	0.842	0.852	0.863	0.779	0.827	0.851	0.867	0.874	0.881	0.887
	Our	0.776	0.838	0.863	0.876	0.879	0.882	0.842	0.887	0.907	0.915	0.921	0.925	0.927
		Query weights 0.4-0.4-0.2 - CellDec weights 1-1-1												
Recall	CellDec	4.812	5.184	5.336	5.472	5.544	5.644	5.492	5.536	5.704	5.776	5.86	5.904	5.972
	PODS07	4.512	5.032	5.196	5.284	5.372	5.444	4.852	5.18	5.368	5.444	5.528	5.6	5.652
	Our	6.128	7.168	7.64	7.832	7.916	7.984	6.848	7.708	8.08	8.268	8.392	8.448	8.48
NAG	CellDec	0.769	0.811	0.830	0.844	0.855	0.866	0.852	0.869	0.884	0.899	0.908	0.914	0.918
	PODS07	0.743	0.807	0.821	0.832	0.842	0.853	0.771	0.819	0.843	0.860	0.867	0.875	0.881
	Our	0.778	0.833	0.856	0.869	0.872	0.875	0.836	0.883	0.903	0.909	0.916	0.919	0.921
		Query weights 0.2-0.4-0.4 - CellDec weights 1-1-1												
Recall	CellDec	3.864	4.06	4.148	4.284	4.312	4.404	4.78	4.692	4.796	4.916	4.956	5.004	5.072
	PODS07	3.772	4.168	4.284	4.3	4.284	4.328	4.0	4.2	4.344	4.428	4.5	4.552	4.58
	Our	6.356	7.116	7.516	7.624	7.704	7.76	6.96	7.708	8.004	8.076	8.184	8.24	8.268
NAG	CellDec	0.698	0.737	0.756	0.774	0.786	0.797	0.798	0.811	0.828	0.847	0.857	0.863	0.868
	PODS07	0.679	0.738	0.753	0.763	0.772	0.783	0.725	0.763	0.785	0.800	0.808	0.817	0.825
	Our	0.762	0.807	0.827	0.836	0.840	0.842	0.819	0.870	0.883	0.887	0.896	0.898	0.900
		Query weights 0.4-0.2-0.4 -CellDec weights 1-1-1												
Recall	CellDec	4.0	4.176	4.292	4.312	4.324	4.352	4.388	4.396	4.444	4.4	4.412	4.444	4.456
	PODS07	3.752	4.104	4.188	4.256	4.204	4.244	3.792	4.172	4.284	4.312	4.312	4.308	4.296
	Our	5.608	7.048	7.664	7.932	8.096	8.176	5.988	7.272	7.82	8.136	8.44	8.516	8.608
NAG	CellDec	0.791	0.830	0.845	0.851	0.858	0.865	0.834	0.855	0.863	0.868	0.874	0.877	0.880
	PODS07	0.757	0.815	0.828	0.839	0.849	0.856	0.762	0.813	0.834	0.848	0.852	0.855	0.858
	Our	0.786	0.869	0.901	0.916	0.922	0.926	0.817	0.895	0.924	0.934	0.943	0.946	0.949
		Query weights 0.2-0.6-0.2 - CellDec weights 0.5-1-0.5												
Recall	CellDec	4.084	4.18	4.236	4.312	4.388	4.428	4.548	4.696	4.74	4.74	4.792	4.828	4.848
	PODS07	3.496	3.684	3.848	3.932	3.964	4.04	4.112	4.252	4.308	4.444	4.492	4.516	4.54
	Our	6.392	7.008	7.22	7.344	7.4	7.448	7.024	7.632	7.824	7.976	8.028	8.056	8.08
NAG	CellDec	0.770	0.802	0.818	0.828	0.842	0.848	0.870	0.900	0.914	0.923	0.927	0.928	0.930
	PODS07	0.668	0.702	0.734	0.751	0.762	0.769	0.770	0.795	0.816	0.835	0.844	0.852	0.859
	Our	0.740	0.775	0.788	0.799	0.801	0.805	0.814	0.849	0.861	0.867	0.873	0.876	0.878
		Query weights 0.6-0.2-0.2 - CellDec weights 1-0.5-0.5												
Recall	CellDec	3.172	3.308	3.376	3.396	3.424	3.44	3.632	3.944	3.968	4.012	4.0	4.024	4.016
	PODS07	2.716	3.14	3.216	3.292	3.336	3.36	3.044	3.44	3.62	3.736	3.824	3.876	3.884
	Our	5.76	7.236	7.848	8.156	8.32	8.412	5.808	7.132	7.728	8.128	8.32	8.488	8.632
NAG	CellDec	0.809	0.845	0.861	0.867	0.870	0.874	0.803	0.852	0.861	0.865	0.869	0.874	0.874
	PODS07	0.725	0.793	0.823	0.839	0.849	0.856	0.702	0.784	0.823	0.836	0.849	0.860	0.862
	Our	0.795	0.883	0.913	0.930	0.936	0.939	0.812	0.891	0.921	0.936	0.945	0.953	0.957
		Query weights 0.2-0.2-0.6 - CellDec weights 0.5-0.5-1												
Recall	CellDec	3.384	3.532	3.64	3.736	3.832	3.892	4.176	4.312	4.424	4.48	4.5	4.508	4.556
	PODS07	3.168	3.436	3.604	3.7	3.74	3.764	3.584	3.876	3.996	4.08	4.148	4.244	4.292
	Our	5.812	7.108	7.728	7.92	8.064	8.164	6.52	7.432	7.896	8.116	8.32	8.4	8.52
NAG	CellDec	0.773	0.806	0.828	0.840	0.856	0.866	0.853	0.869	0.884	0.889	0.894	0.896	0.903
	PODS07	0.737	0.785	0.812	0.825	0.835	0.840	0.755	0.807	0.834	0.845	0.852	0.862	0.866
	Our	0.773	0.859	0.887	0.896	0.902	0.908	0.837	0.889	0.914	0.923	0.933	0.936	0.939

Table 5: Quality results of the compared algorithms. Recall is a number in [0,10], Normalized Aggregated Goodness is a number in [0,1]. Data as a function of the number of visited clusters.