A New Algorithm for Clustering Search Results

GIANSALVATORE MECCA, SALVATORE RAUNICH, ALESSANDRO PAPPALARDO
Dipartimento di Matematica e Informatica
Università della Basilicata
Potenza – Italy
`http://www.db.unibas.it/projects/noodles`

Noodles Working Report n. 01-2006
Revision #163 - Apr 3, 2007

ABSTRACT

We develop a new algorithm for clustering search results. Differently from many other clustering systems that have been recently proposed as a post-processing step for Web search engines, our system is not based on phrase analysis inside snippets, but instead uses Latent Semantic Indexing on the whole document content. A main contribution of the paper is a novel strategy – called Dynamic SVD Clustering – to discover the optimal number of singular values to be used for clustering purposes. Moreover, the algorithm is such that the SVD computation step has in practice good performance, which makes it feasible to perform clustering when term vectors are available. We show that the algorithm has very good classification performance, and that it can be effectively used to cluster results of a search engine to make them easier to browse by users. The algorithm has being integrated into the Noodles search engine, a tool for searching and clustering Web and desktop documents.

KEYWORDS:

World Wide Web, search engine, clustering, latent semantic indexing, document

1 Introduction and Motivations

Web search engines like Google [5] can nowadays be considered as a cornerstone service for any Internet user. The keyword-based, boolean search style used by these engines has rapidly permeated user habits, to such an extent that it is now extending to other classes of applications, for example desktop search [12].

A key factor in the success of Web search engines is their ability to rapidly find good quality results to queries that are based on rather specific terms, like "*Java Server Faces*" or "*Juventus Football Club*". On the other side, however, Google-like search services usually fall short when asked to answer much broader queries – those that are often posed by less-experienced users – like, for example, to find documents about the term "*power*" or the term "*amazon*". The poor quality of results in these cases is mainly due to two different factors: (*a*) polysemy and/or synonimity in search terms (*b*) excessively high number of results returned to the user. As a consequence, less skilled users are often frustrated in their research efforts.

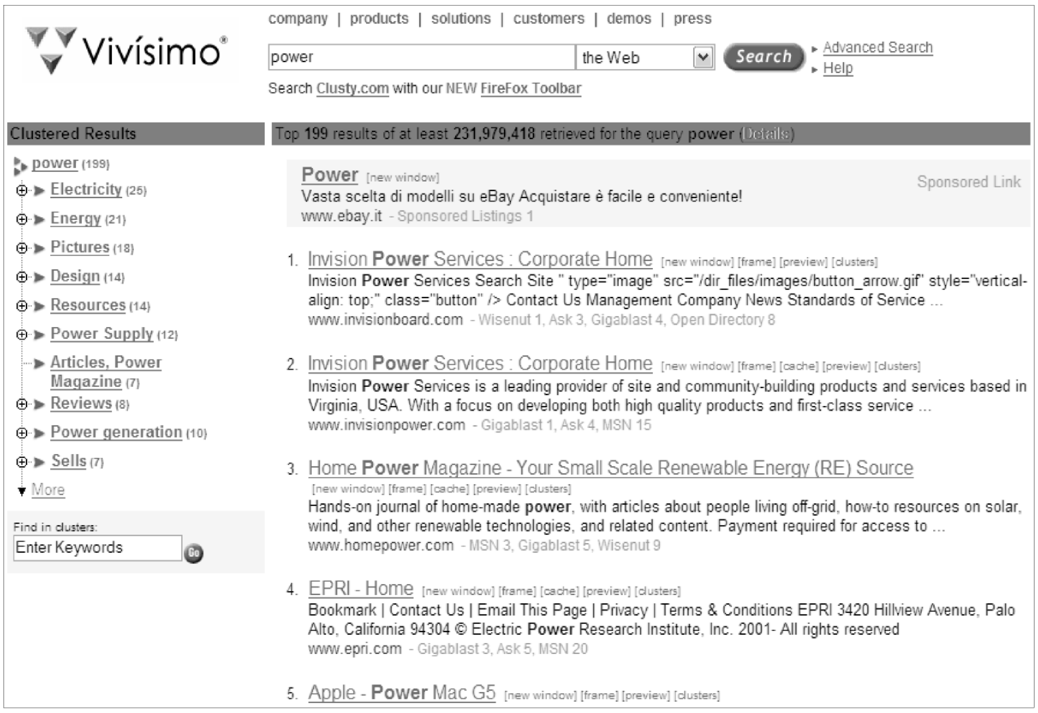
The so-called *Semantic Web* [3] promises to solve most of these problems by adding semantics to Web resources. Although there have been some proposals in the literature towards a semantic Web search engine [14], the transition from the current "syntactic Web" to the next generation Web appears to be a slow process and these proposals can hardly be considered as a ready-made solution to today's search problems.

At the moment, the most promising improvement over traditional search techniques are the so-called *clustering engines.* The idea of clustering search results is not new, and has been investigated quite deeply in Information Retrieval, based on the so called *cluster hypothesis* [34] according to which clustering may be beneficial to users of an information retrieval system since it is likely that results that are relevant to the user are close to each other in the document space, and therefore tend to fall into relatively few clusters.

Several commercial clustering engines have recently emerged on the market. Well known examples are Vivisimo [35] and Grokker [13]. Although the underlying clustering techniques are not fully disclosed to the public, based on the available documentation it is possible to say that these systems share a number of common features with research systems introduced in literature, mainly the Grouper system [38] [39] and Lingo/Carrot Search [23] [25]. We summarize these features in the following.

First, these tools are usually not search engines by themselves. On the contrary, when a user poses a query, the clustering engine uses one or more traditional search engines to gather a number of results; then, it does a form of post-processing on these results in order to cluster them into meaningful groups. The cluster tree is then presented to the user so that s/he can browse it in order to explore the result set. Figure 3.1 shows the clusters produced by Vivisimo for the query term "*power*". It can be seen that such a technique may be helpful to users, since they can quickly grasp the different meanings and articulations of the search terms, and more easily select a subset of relevant clusters.

Being based on a post-processing step, all of these clustering engines work by analyzing *snippets*, i.e., short document abstracts returned by the search engine, usually containing words around query term occurrences. The reason for this is performance: each snippet contains



Top 199 results of at least 231,979,418 retrieved for the query **power** (Details)

Power [new window] Sponsored Link
Vasta scelta di modelli su eBay Acquistare è facile e conveniente!
www.ebay.it - Sponsored Listings 1

1. Invision **Power** Services : Corporate Home [new window] [frame] [preview] [clusters]
Invision **Power** Services Search Site " type="image" src="/dir_files/images/button_arrow.gif" style="vertical-align: top;" class="button" /> Contact Us Management Company News Standards of Service ...
www.invisionboard.com - Wisenut 1, Ask 3, Gigablast 4, Open Directory 8

2. Invision **Power** Services : Corporate Home [new window] [frame] [cache] [preview] [clusters]
Invision **Power** Services is a leading provider of site and community-building products and services based in Virginia, USA. With a focus on developing both high quality products and first-class service ...
www.invisionpower.com - Gigablast 1, Ask 4, MSN 15

3. Home **Power** Magazine - Your Small Scale Renewable Energy (RE) Source
[new window] [frame] [cache] [preview] [clusters]
Hands-on journal of home-made **power**, with articles about people living off-grid, how-to resources on solar, wind, and other renewable technologies, and related content. Payment required for access to ...
www.homepower.com - MSN 3, Gigablast 5, Wisenut 9

4. EPRI - Home [new window] [frame] [cache] [preview] [clusters]
Bookmark | Contact Us | Email This Page | Privacy | Terms & Conditions EPRI 3420 Hillview Avenue, Palo Alto, California 94304 © Electric **Power** Research Institute, Inc. 2001- All rights reserved
www.epri.com - Gigablast 3, Ask 5, MSN 20

5. Apple - **Power** Mac G5 [new window] [frame] [preview] [clusters]

Figure 1.1: Search Results Clustered by Vivisimo

from 0 to 40 words, and therefore can be analyzed very quickly, so that users do not experience excessive delays due to the clustering step. However, snippets are often hardly representative of the whole document content, and this may in some cases seriously worsen the quality of the clusters.

1.1 Contributions of the Paper

This research aims at reconsidering some of the techniques for clustering search results. More specifically, we want to investigate the trade-off between performance and quality of the clustering when choosing snippets versus whole documents. This is particularly relevant if we consider that in some emerging contexts, like for example, desktop search engines, (*a*) it is reasonable to assume that the document-term vectors are available to the clustering engine; (*b*) snippets may not be available at all, based on the different nature of documents.

The main goal of the paper is to evaluate the quality of the clustering in terms of its ability to correctly *classify* documents, i.e., to dynamically build a bunch of clusters that correctly reflect the different categories in the document collection returned by the search engine. Similarly to [7], we believe that this ability may assist less-skilled users in browsing the document set and finding relevant results.

The main contributions of the paper can be summarized as follows:

- we develop a new document clustering algorithm called *Dynamic SVD Clustering* (*DSC*), based on Latent Semantic Indexing [9]; the novelty of the algorithm is twofold: (*a*) first, it is based on an incremental computation of singular values, and does not require to compute the whole SVD of the original matrix; (*b*) second, it uses an original strategy to select *k*, i.e., the number of singular values used to represent the "concepts" in the document space; differently from other proposals in the literature, like for example [30] or [23], our strategy does not assume a fixed value of *k*, neither a fixed approximation threshold;

- based on experimental results, we show that the algorithm has very good classification power; in many cases it is able to cluster pre-classified documents collections with 100% accuracy; it is worth noting that the quality of the classification severely degrades when snippets are used in place of the whole document content, thus providing further evidence that snippets are often too poor and not sufficiently informative; by comparing our results to those of other proposals in the literature, we show that our algorithm has comparatively better classification performance;
- finally, we show that the complexity of the chosen SVD computation strategy is such that the algorithm has in practice good performance, and lends to a very natural clustering strategy based on the minimum spanning tree of the projected document space.

To the best of our knowledge, this is the first paper to propose a dynamic strategy to discover the optimal number of singular values to be used in a classification task. This strategy represents the main contribution of this paper.

The paper is organized as follows. Section 2 introduces a number of preliminary definitions and discusses some of the techniques that will be used in the rest of the paper. Section 3 is devoted to the description of the clustering algorithm and Section 4 discusses implementation and experimental results. Related works are in Section 5.

2 Preliminaries

This section introduces a number of techniques that will be used in the rest of the paper.

Document Indexing

As it is common in information retrieval, we represent documents as vectors in a multidimensional term space [34]. Documents are first preprocessed to remove stop words. Terms may be stemmed. There are several *weighting schemes* [34] [8] that can be used to construct document vectors. Generally speaking, the *weight* w_{ij} of term t_i in document D_j, i.e., the *i*-th coordinate of vector *j,* is given by the product of three different factors:

$$w_{ij}=L_{ij}G_i N_j$$

where L_{ij} is the *local weight* of term *i* in document *j*, G_i is the global weight of term *i* in the document collection, and N_j is the *normalization factor* for document *j*.

We have experimented several weighting schemes, as follows. Let us call f_{ij} the frequency of term *i* in document *j*, F_i the global frequency of term i in the whole document collection, n_i the number of documents in which term i appears, N the total number of documents, and *m* the size of a document vector v_j.

The following table summarizes the three forms of local weight that have been considered in the paper.

Id	Formula	Description
FREQ	$L_{ij}=f_{ij}$	Frequency of term i in document j
LOGA	$L_{ij}=\begin{cases}1+\log(f_{ij}) & if\ f_{ij}>0\\ 0 & if\ f_{ij}=0\end{cases}$	Augmented Logarithmic Local Weight
SQRT	$L_{ij}=\begin{cases}1+\sqrt{(f_{ij}-0.5)} & if\ f_{ij}>0\\ 0 & if\ f_{ij}=0\end{cases}$	Augmented Square Root Local Weight

Table 2.1.Local Weights

With respect to global weights, we have considered the following:

Id	Formula	Description
NONE	$G_i=1$	All terms have the same global weight
IGFF	$G_i=\frac{F_i}{n_i}$	Inverted Global Frequency (a term has higher global weight if its total number of occurrences exceeds the number of documents in which it appears)
IDFB	$G_i=\log(\frac{N}{n_i})$	Inverse Document Frequency (a term has global weight 0 if it appears in every document)

Table 2.2.Global Weights

Finally, with respect to normalization factors, we have considered

Id	Formula	Description
NONE	$N_j=1$	All documents have the same normalization weight
COSN	$N_j=\frac{1}{\sqrt{\sum_{i=0}^{m}(G_i L_{ij})^2}}$	Cosine Normalization Factor

Table 2.3.Normalization Weights

Note that, when using the cosine normalization factor, all term vectors have length 1.

In order to compare distances and similarities between vectors, we use *cosine similarity*, that is, we compute the similarity $s(v_i, v_j)$ between vectors v_i and v_j as the cosine of their angle θ, as follows:

$$s(v_i, v_j)=\cos(\theta)=\frac{v_i \cdot v_j}{\|v_i\|\|v_j\|}$$

here $v_i \cdot v_j$ is the dot product of the two vectors, and $\|v_i\|\|v_j\|$ is the product of their norms. Since all vectors have length 1, we have that $s(v_i, v_j)=\cos(\theta)=v_i \cdot v_j$.

Latent Semantic Indexing

Latent Semantic Indexing (LSI) [9] is a document projection technique based on *Singular Value Decomposition (SVD)*. Suppose we are given *d* documents and *t* terms; let us represent each document as a term vector of *t* coordinates. We call A the $t \times d$ matrix whose columns are the term vectors; let r be the rank of A . SVD decomposes matrix A into the product of three new matrices, as follows:

$$A=U\Sigma V^T=\sum_{i=0}^{r}\sigma_i u_i v_i^T$$

where:

- Σ is a diagonal matrix of size $r\times r$ made of the *singular values* of A in decreasing order: $\Sigma=diag(\sigma_{1,}\sigma_{2,}\ldots\sigma_r),\sigma_1\geqslant\sigma_2\geqslant\ldots\geqslant\sigma_r>0$
- U and V are of size $t\times r$ and $d\times r$ respectively; vectors u_i are called the *left singular vectors* and v_i^T are the *right singular vectors* of A

A singular value and its left and right singular vectors are also called a *singular triplet*. In order to obtain an approximation of A let us fix $k\leqslant r$ and call $\Sigma_k=diag(\sigma_{1,}\sigma_{2,}\cdots,\sigma_k)$, i.e., the $k\times k$ head minor of Σ. Similarly, let us call U_k the restriction of U to the first *k* left singular vectors; similarly for V_k^T; we define:

$$A_k=U_k\Sigma_k V_k^T$$

It is possible to prove [9] that A_k is the best *k*-rank approximation of the original matrix A. Informally speaking, by appropriately choosing a value for *k* we are selecting the largest singular values of the original matrix, and ignoring the smallest ones, therefore somehow preserving the main features of the original vector space while filtering out some “noise”. It can be seen that the smaller is *k* with respect to *r*, the less the new space resembles the original one. In traditional information retrieval applications, in which LSI is used as a means to improve retrieval performance, it is crucial that the essential topological properties of the vector space are preserved; as a consequence, the value of *k* is usually quite high (empirical studies [30] show that for a typical information retrieval application a value between 150 and 300 is usually the best choice).

It is known [4] that the computation of singular triplets of a matrix A may be reduced to the computation of eigenvalues and eigenvectors of matrix AA^T or A^TA. Therefore, in order to discuss the complexity of SVD, we shall refer to the complexity of the eigenvalue problem. In particular, there are several algorithms for computing eigenvalues and eigenvectors of a matrix. One example of this are the *implicitly restarted Arnoldi/Lanczos methods* [32][1]. This family of methods is particularly relevant with respect to this work, since it can be implemented in an *incremental* way, that is, to obtain the first *k* eigenvalues one by one. It has also very interesting computational complexity; more specifically, it has been shown that it can be used to compute the first *k* singular triplets in time $2k^2n+O(k^3)$ using storage $2kn+O(k^3)$ [6].

3 Clustering Algorithm

In this section we introduce the clustering algorithm used by Noodles. We shall first give some insight on the main ideas behind the algorithm, and then elaborate on the technical details.

1 One of these methods is also used in the well known Matlab toolkit to compute eigenvalues/eigenvectors.

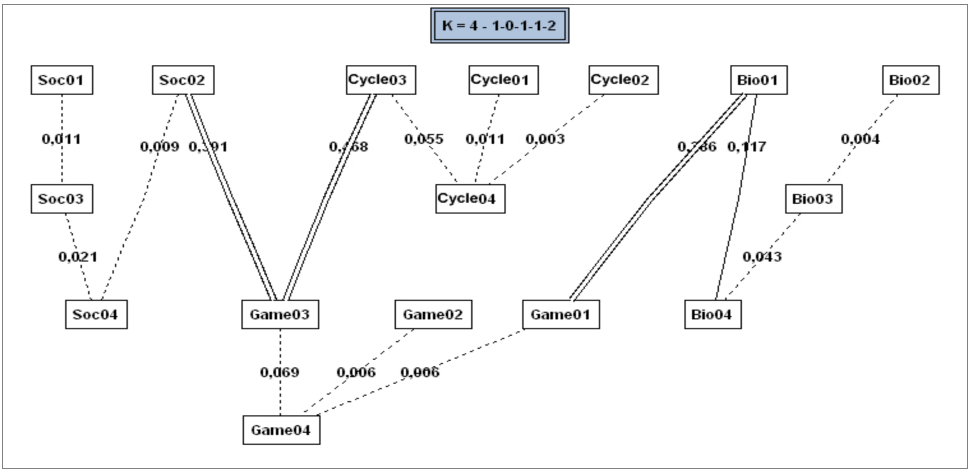


Figure 3.1: Example of a Minimum Spanning Tree for the Life Example

3.1 Intuition

Our algorithm is heavily based on Latent Semantic Indexing. LSI has a natural application in clustering and classification applications. It is often said that, by means of SVD, LSI does perform a transformation of the original vector space – in which each coordinate is a term – into a new vector space, in which each coordinate is some relevant "concept" in the document collection, i.e., some topic that is common to several documents. Note that the coordinates of the original documents in this space are given by matrix $V_k \Sigma_k$. We shall call this $d \times k$ space the *projected document space*.

In this respect, a possible approach to clustering would be the following: (*a*) compute SVD over the original matrix, for some value *k,* to obtain a representation of the original documents in the new "concept" space, $V_k \Sigma_k$, in which each coordinate represents some "topic" in the original document collection, and therefore some cluster of documents; (*b*) run a clustering algorithm in this space to cluster documents with respect to their topics. This method has been used for clustering purposes for example in [23].

A critical step, in this approach, is the selection of *k*. A natural intuition suggests that, assuming the document collection contains *x* hidden clusters, the natural value of *k* to be used for SVD is exactly *x*. This is a consequence of the fact that one of the property of SVD is that of producing in $V_k \Sigma_k$ an optimal alignment of the original documents along the *k* axes [27]. However, such a value is unknown and must be discovered by the clustering engine. There are a few interesting observations with respect to this point:

- first, such a value of *k* can be significantly lower that the number of documents, *d*, and the number of terms, *t,* since it is unlikely that there are more than a dozen relevant clusters among the results of a search; this means that the projected document space does not preserve much of the features of the original space; this is, however, not a concern, since in our case we are using this space only as a means to discover clusters, and not for retrieval purposes;

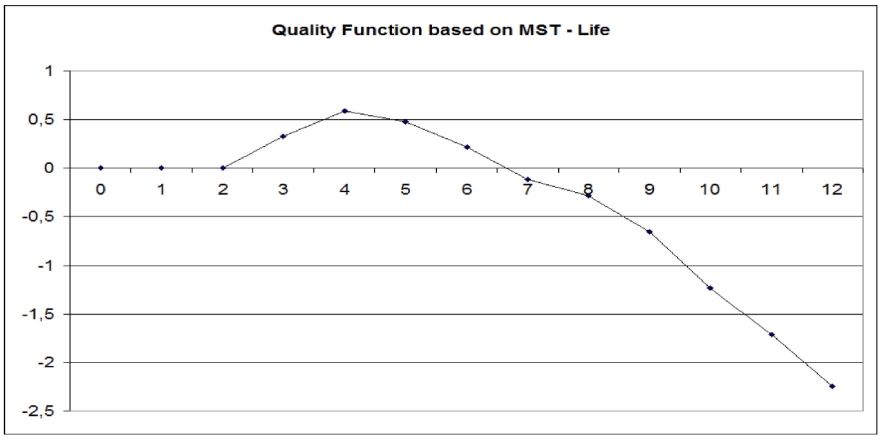


Figure 3.2: Quality Function for the Life Example

- it is therefore apparent that fixing the value of *k*, for example assuming *k* in the order of 150-300*,* as it is often done in the literature, would not give good classification performance; the fact that lower values of *k* are to be preferred in clustering applications is confirmed for example in [30]; nevertheless, in the latter work fixed values of *k=20* and *k=50* are compared, whereas the optimal value should ideally be discovered dynamically;
- in light of these considerations, we also discard the other typical approach used to establish the value of *k*, i.e., that of fixing a threshold for the approximation that A_k gives of the original space A , as it is done, for example, in [23].

In fact, the strategy used to dynamically select the optimal value of *k* is one of the main originality of our algorithm.

The main intuition behind the selection of *k* is that – assuming the original collection contains *x* clusters, and that each cluster informally corresponds to some clearly defined "topic" – then the points in the projected document space should naturally fall into *x* clusters that are reasonably compact and well separated.

To see this, consider for example Figure 4.5, which refers to one of our test document collection, corresponding to search results for the query term "*life*". We have identified four natural clusters in the collection, namely: (*a*) documents about biological life; (*b*) documents about life cycles of a system or a living entity; (*c*) documents about social life and related events; (*d*) documents about the game of life. The figure shows the minimum spanning tree of document distances in space $V_4\Sigma_4$. Edges are drawn differently based on their length; more specifically, the longest ones are drawn as a double line, the shortest ones as a dashed line. It can be seen that, on the one side SVD has clearly brought documents belonging to the same cluster close to each other, and on the other side that an accurate clustering can be obtained simply by removing the $k-1$ longest edges, that is, by applying a variant of a spanning-tree based clustering algorithm [37] [17].

Based on this observation, our algorithm can be informally sketched as follows:

- given the original term-document space A we incrementally compute SVD, starting with a low number of singular values, and, for each value of k, generate the projected space $V_k \Sigma_k$;
- we find the minimum spanning tree of points in the projected space; assuming we have found the optimal value for *k,* we stop the algorithm and obtain our clusters simply by removing the $k-1$ longest edges from the minimum spanning tree, and considering each connected component as a cluster;
- to discover such optimal value of *k,* we define a quality function for the clusters obtained from the minimum spanning tree; intuitively, this function rates higher those trees in which the $k-1$ longest edges are significantly longer than the average (details are given below); based on this, we stop as soon as we find a local maximum of the quality function; this is a hint that we have found a "natural" distribution of documents in the projected space, that would be worsened if we choose higher values of *k,* i.e., more clusters.

A plot of the quality function for the life example discussed above is reported in Figure 4.4. It can be seen that the quality function has a maximum for *k*=4, that is exactly the number of clusters we are seeking in this example. It is also worth noting that such a strategy nicely solves a typical problem of clustering algorithms, i.e., that of dynamically choosing the right number of clusters for a given dataset [17].

The following section describes the clustering algorithm in detail.

3.2 Description of the Dynamic SVD Clustering Algorithm

Let us first formalize the quality function for minimum spanning trees. We fix a value of *k,* and assume we have computed the projected space $V_k \Sigma_k$ by means of SVD. Let us call MST_k the minimum spanning tree of document distances in the projected space. We call the $k-1$ longest edges in the tree $e_1, e_2, \dots e_{k-1}$ respectively.

We find the average avg_k and the standard deviation σ_k of the distance distribution in the projected space. Based on this, we assign to each edge *e* of length $l(e)$ in the spanning tree a *cost* $c(e)$ as follows:

$$c(e)=l(e)-(avg_k+\sigma_k)$$

Then, we define the *quality* $Q(MST_k)$ of MST_k as follows:

$$Q(MST_k)=k\sum_{i=1}^{k-1} c(e_i)$$

Intuitively, the quality function is higher when edges that are candidate to be removed have lengths that are significantly above the average. Term *k* is necessary as a multiplicative factor since we have observed that, by increasing the value of *k*, the distribution of distances in the space changes significantly. More specifically, for higher values of *k* we have both higher average distance and standard deviation – i.e., the overall space is more disperse – and therefore edges tend to bring a smaller contribution to the quality function.

This said, we can formalize the DSC algorithm as follows. The input to the algorithm is a document collection $\Delta=\{D_{0,} D_{1,} \dots D_d\}$. These may either be whole documents, or document summaries, such as snippets. We assume that documents have been previously indexed, i.e., for each document D_i in Δ we have build a term vector after performing stop-word removal. Note that we are not fixing the weighting scheme used to construct the vectors. In Section 4 we compare the impact of the different weighing schemes on the clustering.

The algorithm proceeds as follows.

Step 1: Initialization of the Vector Space Given the term vectors associated to documents in Δ we build matrix A

Step 2: Incremental SVD We incrementally compute SVD for matrix A using the Arnoldi/Lanczos implicitly restarted method, starting with *k = 2*; for each value of *k*, we derive the projected space $V_k \Sigma_k$

Step 3: Quality Evaluation based on Minimum Spanning Trees We build the minimum spanning tree MST_k of the graph of distances in $V_k \Sigma_k$ and calculate $Q(MST_k)$. We stop as soon as we find a value of *k* such that:

$$Q(MST_{k-1}) \leq Q(MST_k) > Q(MST_{k+1})$$

Step 4: Clustering Once we have fixed *k,* we obtain the clusters by removing the $k-1$ longest edges from MST_k and building a cluster for each connected component obtained after the removal. In order to avoid the production of singleton clusters, all clusters containing less than two documents are merged into a single outliers clusters.

Step 5: Cluster Labeling To give a meaningful label to each cluster, we analyze the collection of term vectors associated with documents in the cluster, and select the *n* most frequent terms (currently 4). Those terms will be used as a label for the cluster. The cluster of outliers is labeled as "Other Documents"

Note that our labeling scheme is similar to that used by [15]. As an alternative labeling scheme, we might easily perform phrase analysis on snippets, using for example the algorithm of [38]. However, we have decided not to implement this solution in our system since our main focus is on the clustering algorithm.

3.3 Computational Complexity

A crucial requirement for any clustering algorithm is efficiency. This is particularly true in our case, since: (*a*) document vectors tend to have very high cardinality, and this may in principle produce high overheads in order to calculate distances during the clustering phase; (*b*) performing a full SVD on a large matrix is a demanding computation.

Our algorithm solves these two problems by virtue of its incremental nature. To show this, let us comment on the complexity of the various steps. Let us first note that a critical assumption here is that document indexing has been performed in advance, i.e., that the clustering engine has access to term vectors for the retrieved documents. We call *t* the number of terms occurring in the document collection after stop-word removal. Then, we have the following upper bounds:

- step 1 (*initialization of the vector space*) is linear in time with respect to *t* and produces a matrix of size $t \times d$;
- step 2 (*incremental SVD*) is by far the most delicate step in terms of complexity; as discussed above, its time cost is $O(k^3)$; therefore, the overall time complexity is strongly related to the value of *k*; however, note that in clustering applications *k* is usually quite low – typically less than 10 – and this should guarantee good performance in practical cases;
- to evaluate the cost of step 3 (*quality evaluation based on minimum spanning trees*), please note that the projected space has size $d \times k$, with $k \ll t$; as a consequence, the lower is the value of *k*, the faster we can compute distances between points in the space; in particular, each distance requires exactly *k* products; in order to build the minimum spanning tree, we first need to construct the complete graph of document distances in space $V_k \Sigma_k$; this requires to compute $d(d-1)/2$ distances; then, to build the minimum spanning tree on this graph, we use the classical Prim's algorithm [28]; Prim's algorithm implemented using a Fibonacci heap on a graph of *E* edges and *V* nodes has $O(E+V\log(V))$ time complexity; contextually, we can also find the *k – 1* longest edges in the space, and evaluate function *Q()*;
- step 4 (*clustering*) is linear in *d*, since it requires to visit the minimum spanning tree and find connected components after removing the longest edges.

We will elaborate further on this in Section 4, where we report a study of computing times in practical experiments. For the moment, let us note how this analysis suggests that the SVD computation and the derivation of the minimum spanning tree are likely to have the highest impact in terms of running time. In practice, however, SVD computation is by far the bottleneck in terms of efficiency.

4 Implementation and Experiments

The clustering algorithm has been implemented in the Noodles desktop search engine [21], a snapshot of which is shown in Figure 3.2. The system is written in Java, using Apache Lucene[2] as an indexing engine, and the Spring Rich Client Platform[3] as a desktop application framework. It has been conceived as a general-purpose search tool, that can run both Web and desktop searches. In order to perform desktop searches, it incorporates a crawler module that runs as a daemon and incrementally indexes portions of the local disks that have been specified by the user.

It also incorporates a testbed for the clustering engine, which we have used to conduct a number of experiments. Overall, we have run several hundreds of experiments, combining different datasets, different summarization schemes, different weighting schemes, different criteria for the selection of *k* in SVD, and different clustering algorithms. The most interesting experimental evidences are described in this section.

2 http://lucene.apache.org

3 http://www.springframework.org

4.1 Description of the Datasets

We concentrated on two different categories of datasets. To construct datasets of the first category, we ran queries on Google and selected a number of the top-ranked search results. Those results were manually classified into a number of clusters. Then, the algorithm was run on the document collection to compare the suggested clusters with those identified manually. Although this kind of experiments closely mimic the situation in which a user runs a query on a search engine and then inspects the results browsing the clusters produced by the clustering engine, the manual classification step – necessary to assess the quality of the clustering – tends to be labor-intensive and quite error-prone. As a consequence, document collections of this first category tend to be quite small.

In order to test the classification power of the algorithm on larger document collections, we implemented a data fetching utility for the Open Directory Project (DMOZ) [10]. This utility was used in order to sample categories inside DMOZ, and then run the clustering algorithm on the pre-categorized samples. In this case, the clusters produced by the algorithm were compared with the original DMOZ categories.

Overall, we selected twelve different datasets, ranging from 16 to about 100 documents; three datasets are of the first category, and nine of the second category. The datasets are described in Table 4.1. For each dataset, we have worked both on the full document and on a snippet.

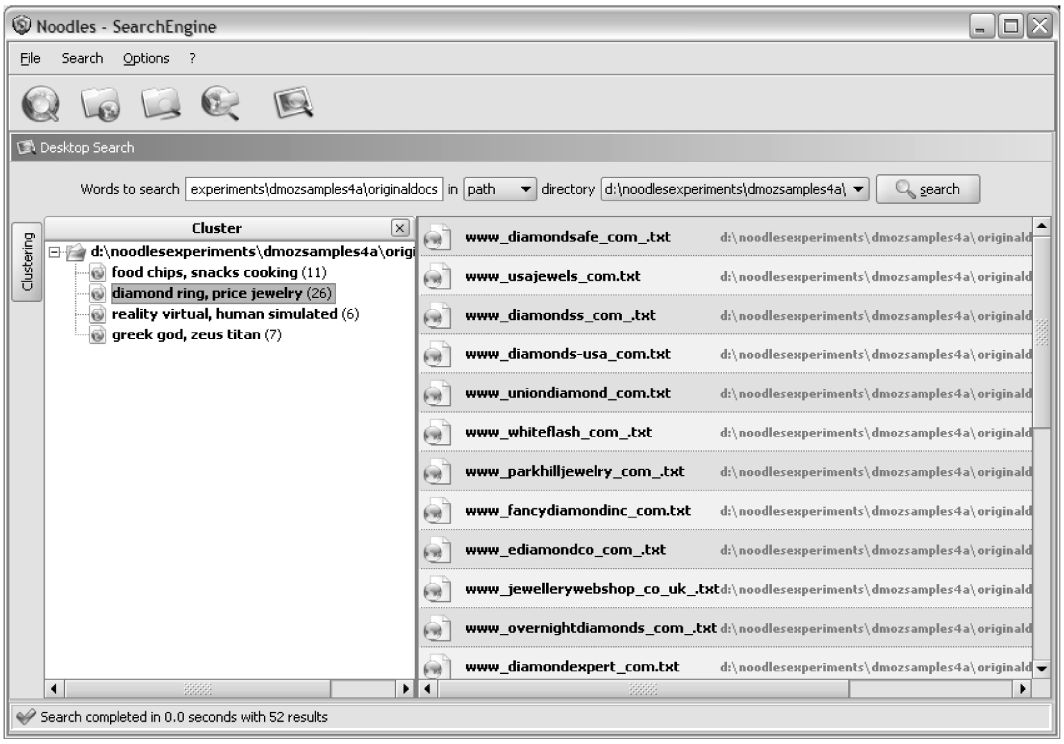
Dataset	Description of the Dataset	# of docs	# of ideal clusters	Description of the clusters
Amazon	Search results for keyword “amazon”	19	3	amazon.com, Amazon forest, Amazons in mythology
Life	Search results for keyword “life”	16	4	Biological life, cycle of life, game of life, social life
Power	Search results for keyword “power”	20	4	Brain power, energy power, military power, numerical power
DMOZsamples3	Sampling of DMOZ categories	69	3	top/shopping/jewelry/diamonds top/shopping/antiques_and_collectibles/coins top/shopping/vehicles/motorcycles
DMOZsamples4A	Sampling of DMOZ categories	52	4	top/arts/literature/myths_and_folktales/myths/greek top/computers/virtual_reality top/recreation/food, top/shopping/jewelry/diamonds
DMOZsamples4B	Sampling of DMOZ categories	97	4	top/news/weather top/regional/caribbean/bahamas/travel_and_tourism top/shopping/antiques_and_collectibles/coins top/shopping/jewelry/diamonds
DMOZsamples4C	Sampling of DMOZ categories	33	4	top/sports/basketball/professional/nba/san_antonio_spurs top/sports/events/olympics/baseball top/sports/volleyball top/sports/water_sports/surfing
DMOZsamples5A	Sampling of DMOZ categories	40	5	top/computers/algorithms top/games/video_games/adventure top/recreation/pets/dogs, top/science/math/number_theory top/sports/volleyball
DMOZsamples5B	Sampling of DMOZ categories	53	5	top/arts/animation/studios/disney top/health/medicine/medical_specialties/neurology top/home/gardening/gardens/public top/shopping/antiques_and_collectibles/coins top/sports/water_sports/surfing

Dataset	Description of the Dataset	# of docs	# of ideal clusters	Description of the clusters
DMOZsamples6	Sampling of DMOZ categories	63	6	top/business/transportation_and_logistics top/regional/caribbean/bahamas/travel_and_tourism top/science/astronomy top/shopping/vehicles/motorcycles top/society/religion_and_spirituality/yoga/practices top/sports/events/olympics/baseball
DMOZsamples7	Sampling of DMOZ categories	90	7	top/arts/movies/history, top/games/video_games/adventure top/recreation/pets/dogs top/shopping/antiques_and_collectibles/coins top/shopping/jewelry/diamonds top/sports/basketball/professional/nba/san_antonio_spurs top/sports/volleyball
DMOZsamples8	Sampling of DMOZ categories	73	8	top/arts/animation/studios/disney top/arts/literature/myths_and_folktales/myths/greek, top/computers/virtual_reality top/health/medicine/medical_specialties/neurology top/home/gardening/gardens/public, top/recreation/food top/shopping/antiques_and_collectibles/coins top/sports/water_sports/surfing

Table 4.1. List of Datasets

4.2 Quality Measures

In order to assess the clusters produced by the algorithm, we have computed several quality measures.



Words to search: experiments\dmozsamples4a\originaldocs in path directory d:\noodlesexperiments\dmozsamples4a\

d:\noodlesexperiments\dmozsamples4a\origi
- food chips, snacks cooking (11)
- diamond ring, price jewelry (26)
- reality virtual, human simulated (6)
- greek god, zeus titan (7)

www_diamondsafe_com_.txt	d:\noodlesexperiments\dmozsamples4a\originald
www_usajewels_com.txt	d:\noodlesexperiments\dmozsamples4a\originald
www_diamondss_com_.txt	d:\noodlesexperiments\dmozsamples4a\originald
www_diamonds-usa_com.txt	d:\noodlesexperiments\dmozsamples4a\originald
www_uniondiamond_com.txt	d:\noodlesexperiments\dmozsamples4a\originald
www_whiteflash_com_.txt	d:\noodlesexperiments\dmozsamples4a\originald
www_parkhilljewelry_com_.txt	d:\noodlesexperiments\dmozsamples4a\originald
www_fancydiamondinc_com.txt	d:\noodlesexperiments\dmozsamples4a\originald
www_ediamondco_com_.txt	d:\noodlesexperiments\dmozsamples4a\originald
www_jewellerywebshop_co_uk_.txt	d:\noodlesexperiments\dmozsamples4a\originald
www_overnightdiamonds_com_.txt	d:\noodlesexperiments\dmozsamples4a\originald
www_diamondexpert_com.txt	d:\noodlesexperiments\dmozsamples4a\originald

Search completed in 0.0 seconds with 52 results

Figure 4.1: A Snapshot of the Noodles Desktop Search Engine

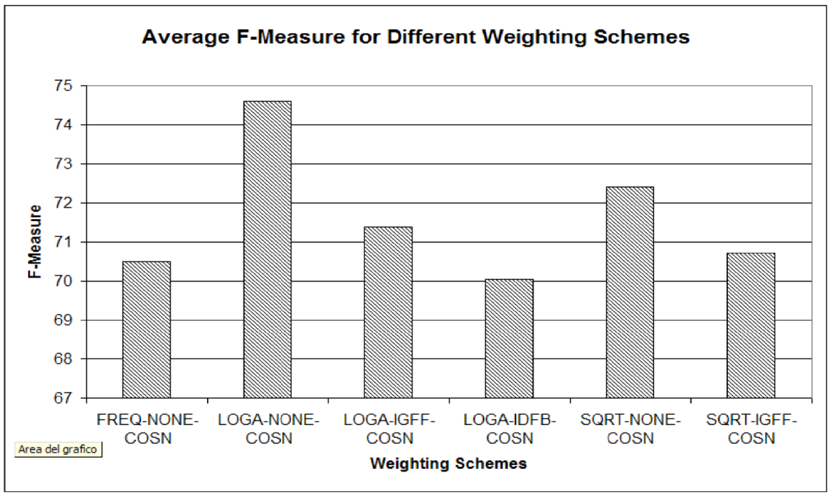


Figure 4.2: Quality Function for the Life Example

As a primary measure of quality we use the *F-measure* [34], i.e., the harmonic means of precision and recall. More specifically, given a cluster of documents C, to evaluate his quality with respect to an ideal cluster C^i we first compute precision and recall as usual:

$$Pr(C,C^i)=\frac{|C\cap C^i|}{|C|} \quad Rec(C,C^i)=\frac{|C\cap C^i|}{|C^i|}$$

Then, we define:

$$F(C,C^i)=\frac{2\,Pr(C,C^i)\,Rec(C,C^i)}{Pr(C,C^i)+Rec(C,C^i)}$$

Given a collection of clusters, $\{C_{1,}C_{2,}\dots C_k\}$, to evaluate its F-measure with respect to a collection of ideal clusters $\{C^i_{1,}C^i_{2,}\dots C^i_h\}$ we do as follows: (*a*) we find for each ideal cluster C^i_n a distinct cluster C_m that best approximates it in the collection to evaluate, and evaluate

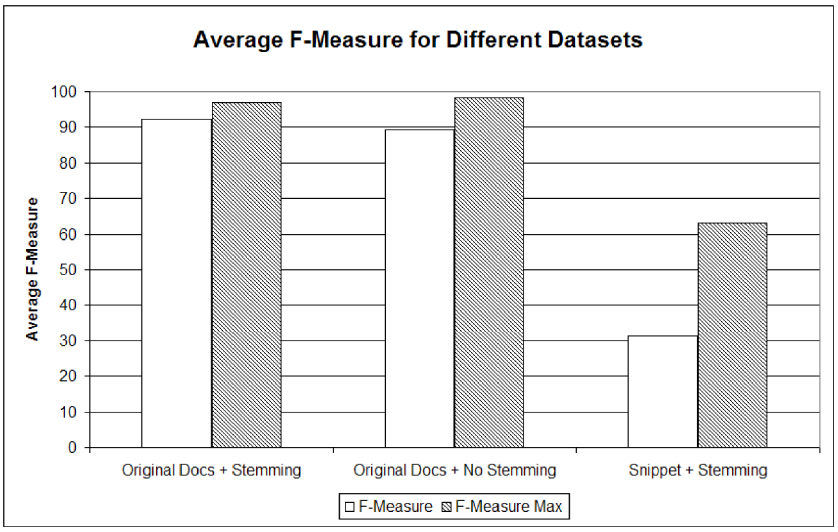


Figure 4.3.: Stemming vs. No Stemming and Full Document vs. Snippet

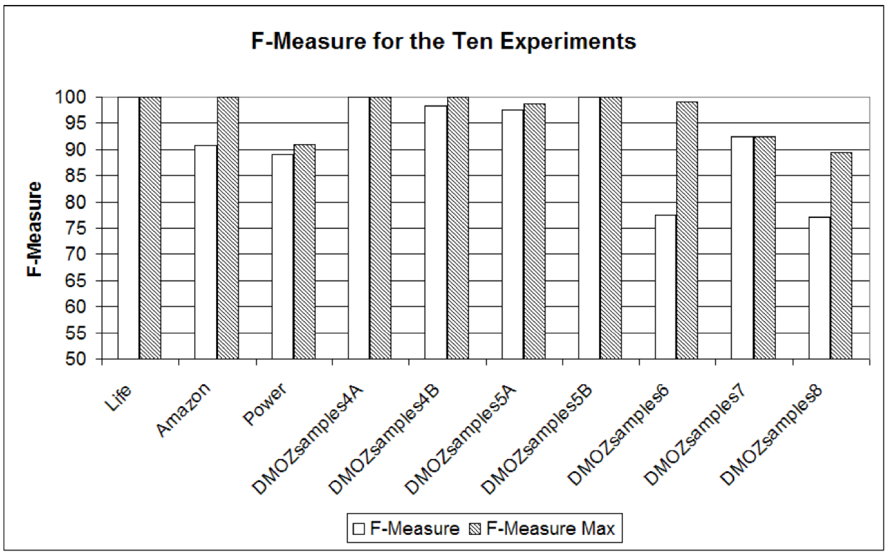


Figure 4.4: Other Quality Measures

$F(C_m, C_n^i)$; (*b*) then, we take as an F-measure of the collection the average value of the F-measures over all ideal clusters.

To gain an in-depth understanding of the classification power of our technique, for each dataset we have computer two different F-measures. First, we ran the clustering algorithm and stopped at the value of *k* chosen by the algorithm; this experiment was used to compute a first F-measure for the clustering algorithm. However, since we were also interested in evaluating the classification power of our SVD-based technique independently of the stop criterion, we also ran a second experiment, in which we computed clusterings for all values of *k,* from 2 to

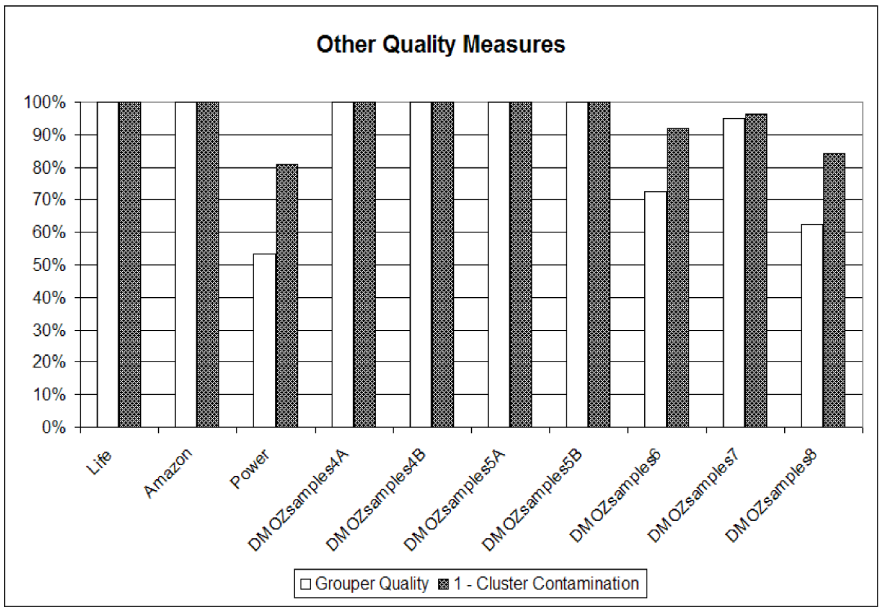


Figure 4.5.: F-Measures for the ten experiments

n, and took the maximum F-measure obtained by the algorithm. In the following, we shall refer to this second measure as “F-measure Max”.

Other works in the literature have used different quality metrics to assess their experimental results. To compare our results to these, we have also computed several other quality measures. More specifically, Grouper [38] introduced a customized quality measure, called *Grouper Quality Function,* which in essence looks at all pair of documents in a single cluster and counts the number of *true positive pairs* (the two documents were also in the same ideal cluster), and *false positive pairs* (they were not in the ideal cluster); then, it combines these two figures to calculate an overall quality function. Lingo [23] uses an alternative measure, *cluster contamination* [26], which intuitively measures how much a cluster produced by the system mixes documents that should ideally belong to different classes; of course, *1 – cluster contamination* can be considered as a measure of the “purity” of the clustering.

Beside F-measure and F-measure Max, we also computed the Grouper Quality Function and cluster contamination for our experiments.

4.3 Indexing Scheme

We are now ready to discuss a number of experimental evidences. As a first step, we dealt with the problem of selecting the correct weighting scheme. We ran several preliminary experiments, both on full documents and on snippets, with and without SVD, comparing six weighting schemes obtained by the combination of different local and global weights. In all cases we used cosine normalization, since we soon noticed that results strongly degrade in absence of normalization. Figure 4.3 shows average F-measures for the six weighting schemes. It can be seen that the best results were obtained using the augmented logarithmic local weight and no global weight. Therefore, we adopted this weighting scheme for the rest of our experiments.

reports for theThis phenomenon seems to be related to the adoption of LSI. In essence, global weights are often used in Information Retrieval as a means to reduce noise in the document space; to give an example, one typical consequence of the adoption of a inverse document frequency global weight is that of assigning weight 0 to all terms that appear in every document. However, similar effects are also obtained by applying SVD. It seems that SVD has indeed superior noise-reduction power in clustering applications, so that global weights are no longer necessary. To confirm this hypothesis, we performed several other experiments on document summaries. Being Web pages, most of our documents typically contain some header – e.g., banner and navigation links – and footer material. We ran the clustering algorithm after removing such sections, which in most cases are rather uninformative; in essence, we extracted what, in information extraction terms, is called a *rich region* from the pages. Also in this case, no significant improvement in F-measures was observed (around 2%), thus confirming the intuition that SVD does a very good job in filtering out this noisy portions of the documents.

A second preliminary evaluation was related to the adoption of stemming after stop-word removal in the document indexing phase. We compared F-measures obtained in the twelve experiments after stemming to those without stemming. As it can be seen in Figure 4.2, there is no significant improvement in quality related to the adoption of stemming. However, we noticed that stemming on average reduced the number of terms by 23%, therefore making matrices smaller and computation more efficient. As a consequence, we decided to consistently perform stemming.

Figure 4.2 also shows a comparison of F-measure and maximum F-measure obtained by the algorithm on full documents and on document snippets. It can be easily seen that performance degrades of more than 40% when analyzing snippets. This seems to confirm one of our initial assumptions, i.e. that snippets are often too short to represent a meaningful summary of the document, and that the analysis of the whole document content is very important in order to improve the quality of the classification. It is worth noting that this is somehow in contrast with other works, for example [38], according to which there is no significant degradation when going from documents to snippets. Such a big discrepancy might be justified by the completely different nature of our approach with respect to theirs.

4.4 Quality of the Algorithm

We are now ready to discuss the quality of the algorithm. Quality figures are reported in detail in Table 4.2 and in Figures 1.1 and . Let us first notice that the algorithm has shown excellent classification performance. In fact, the maximum F-measure was very close to 100% in all experiments.

If we consider the actual clustering produced by the algorithm, we can draw several conclusions. First, the number of generated clusters was in all cases very close to the ideal one; note that in the table, 4+1 means that the algorithm produced 4 labeled clusters plus a cluster of “Other documents” containing outliers, i.e., documents that were considered somehow distant from other clusters. In essence, as it can be seen in Table 4.2, the algorithm was able to choose the right value of *k* for most datasets; in the three cases in which the number of clusters generated by the system was higher than the ideal one (DMOZSamples3, DMOZSamples4C, DMOZSamples7) one or two additional small clusters were produced as a result of splitting larger ones. Note that results on similar experiments reported in [24] with respect to Lingo and Grouper seem to suggest a tendency by their algorithms to over classify DMOZ categories; in these experiments, input documents drawn from 2 to 10 DMOZ categories produced 24 to 36 clusters. Our system does not show this kind of behavior, and is able to reproduce quite accurately the original DMOZ categories.

A second observation is related to the quality of the clusterings that were produced by the algorithm. First, F-measures are on average very high, well above 90%. Moreover, the other quality measures are in both cases higher with respect to those reported by [38] and [26]. For example, papers about Grouper report that average quality has a maximum for document collections with very few ideal clusters; this maximum is below 80%; then, by increasing the number of clusters, the quality measure tends to decrease. Similarly, Lingo reports cluster contamination on average at 25%, whereas in our case we had values below 5% on average. These results further support the idea that clustering based on snippets has inherently lower quality than on full documents, and therefore that in all cases in which the full document is available, it should be used.

Document collection	# of pre-classified clusters	# of generated clusters	F-Measure of the generated clusters	F-Measure Max	Grouper Quality	1 - Cluster Contamination
Life	4	4	100%	100%	100%	100%
Amazon	3	3+1	90,67%	100%	100%	100%
Power	4	4	89%	91%	53,4%	80,8%
DMOZsamples3	3	4+1	91,7%	100%	100%	100%
DMOZsamples4A	4	4	100%	100%	100%	100%
DMOZsamples4B	4	4+1	98,25%	100%	100%	100%
DMOZsamples4C	4	5	96%	96%	100%	100%
DMOZsamples5A	5	5+1	97,6%	98,6%	100%	100%
DMOZsamples5B	5	5	100%	100%	100%	100%
DMOZsamples6	6	6+1	77,5%	99%	72,5%	92%
DMOZsamples7	7	9	92,4%	92,4%	95,1%	96,3%
DMOZsamples8	8	7+1	77%	89,37%	62,5%	84,2%
		Average	**92,5%**	**97,2%**	**90,3%**	**96,1%**

Table 4.2: Quality Measures

4.5 Computing Times

On average, computation times in our experiments were of a few seconds. Also, we developed our user interface in such a way to minimize the latency associated with the display of clusters. More specifically, when a user runs a query, the system very quickly returns the ranked search results. At the same time, it starts to cluster top-ranked results on a background thread, and shows a “Clustering” button on the screen. It the user wants to see the clusters, s/he has to select this button. Considering typical user reaction times, the net effect of such an organization of the user interface is that, in the user perception, the clustering is almost immediate.

Figure 4.1 reports computing times for our experiments. More specifically, for each experiment, we recorded the number of seconds spent in the two most expensive steps, namely computing SVD and generating minimum spanning trees. The remaining steps have negligible running times. As it can be seen, the computation of minimum spanning trees is not a real issue, since in all cases took less than 2 seconds (including the generation of distance matrices in the projected spaces).

On the contrary, SVD computations are significantly more expensive, thus confirming that this is the most delicate task in the algorithm. On average, given the low values of *k,* our incremental SVD computation strategy performed rather well. However, in three of our experiments with the largest documents, we had computing times above 10 seconds. While these times may still be considered acceptable, we believe that performance was negatively influenced by the implementation of SVD used in the system. After a comparative analysis, we selected the COLT[4] matrix manipulation package, one of the open source matrix packages available in Java, and customized the SVD algorithm to fit our needs. However, we noticed

4 http://dsd.lbl.gov/~hoschek/colt

that COLT performance tends to severely degrade on large-size matrices. To confirm this impression, we compared its performance to that of Matlab and JScience[5], the newest math package for the Java language. We noticed that Matlab implementation of the Arnoldi/Lanczos method is orders of magnitude faster than that of COLT. We suspect that this might be a consequence of a poor implementation of the sparse-matrices routines in COLT. To confirm this impressions, we ran a number of comparative tests using JScience; also in this case, JScience – which is based on a high performance library called Javolution[6] – showed significantly better performance both in terms of computing time and heap usage with respect to COLT and other similar packages. Unfortunately, JScience currently does not offer support for SVD, although this might be added to future version. Re-implementing Arnoldi/Lanczos methods in JScience was beyond the purpose of this paper. Nevertheless, we believe that JScience might significantly improve computing times once support for SVD is implemented.

5 Related Works

As we have discussed in Section 1, there are several commercial search engines that incorporate some form of clustering. Besides Vivisimo [35] and Grokker [13], other examples are Ask.com [2], iBoogie [16], Kartoo [18], and WiseNut [36].

In fact, the idea of clustering search results as a means to improve retrieval performance has been investigated quite deeply in Information Retrieval. A seminal work in this respect is the Scatter/Gather project [15] [29]. Scatter-Gather provides a simple graphical user interface to do clustering on a traditional information retrieval system. After the user has posed her/his query, s/he can decide to "scatter" the results into a fixed number of clusters; then, s/he can "gather" the most promising clusters, possibly to scatter them again in order to further refine the search. One limitation in Scatter-Gather is the fact that the system is not able to infer the optimal number of clusters for a given query, and requires that this is specified by the user in advance. This may in some cases have an adverse effect on the quality of the clustering.

Other traditional works on clustering along the same lines include [7], [1], and [30]. In [7] the authors develop a classification algorithm for Web documents, based on Yahoo! categories. The classification algorithm learns representative terms for a number of pre-classified documents, and then tries to assign new documents to the correct category. The reported recall values are in the range 60%-90%. More recently, the issue of classifying Web document into a hierarchy of topics has been studied in [19].

The Paraphrase system [1] introduces Latent Semantic Indexing as a means to cluster search results. The paper focuses on the relationship among clusters produced by LSI and their labels, obtained by taking the most representative words for each cluster. The latter technique is similar to the one we use for labeling our clusters. Besides this, the authors follow a rather typical approach with respect to the selection of the number *k* of singular values. More specifically, *k* is fixed and equals 200; in fact, values in the range 100-200 were considered as optimal in retrieval applications, competitive or even superior to term-based similarity search.

5 http://www.jscience.org

6 http://javolution.org

In [30] the authors further explore the use of projection techniques for document clustering. Their results show that LSI performs better than document truncation. Also in this case, the number *k* of singular values is fixed. The authors compare the quality of the clustering with *k=20*, *k=50* and *k=150*. An interesting point is that, although *k=150* was considered a more typical value, the paper concludes that in clustering applications *k=20* gives better performance and clustering quality. This conclusion is coherent with our idea that the optimal value of *k* for clustering documents is equal to the number of classes (or ideal clusters), and therefore usually much lower than the number of documents.

The advent of Web search engines brought to a shift of perspective about the problem of clustering. The requirements for effective clustering of Web documents are summarized in [20], with emphasis on performance. In fact, most of the recent clustering systems for the Web share a number of common features, namely:

- they work in conjunction with one or more traditional search engines in order to run the queries submitted by users and gather top results
- clustering is based on a form of post-processing of document snippets returned by the back-end search engine
- the clustering algorithm is based on some form of common phrase extraction.

Grouper [38] [39] is a snippet-based clustering engine based on the HuskySearch meta search engine. The main feature of Grouper is the introduction of a phrase-analysis algorithm called *STC* (*Suffix Tree Clustering*). In essence, the algorithm builds a suffix tree of phrases in snippets; each representative phrase becomes a candidate cluster; candidates with large overlap are merged together. The main contribution of Grouper stands in the complexity of the clustering algorithm, which allows for very fast processing of large result sets. However, due to inherent limitations of snippets, quality results are not always excellent. A key observation of the paper is that there is no significant degradation in the quality of clusters when going from full document to snippets. This experimental evidence for the STC algorithm is largely in contrast with our experimental results on DSC.

Grouper has inspired a number of other proposals along the same lines. Two examples are SHOC [41] and Lingo/Carrot Search [23] [25]. Both these works extend the STC algorithm with the use of SVD in order to filter some “noise” in the snippets and improve the quality of the produced clusters. SHOC's clustering is based on two steps: during the first step phrase analysis is used to generate a snippet-topic matrix – in which a topic is either a term or a phrase; then, as a second step, SVD is performed on the matrix in order to identify the most relevant topics. A key difference with our work is that the stop criterion for SVD is based on a fixed approximation threshold. No experimental results are reported in [41] to asses the quality of the clustering algorithm.

Lingo/Carrot Search is similar in spirit, but it uses a different strategy. A primary concern is to produce meaningful descriptions for the clusters. To do this, first SVD is performed on a snippet-term matrix to identify a number of relevant topics. Also in this case, the selection of *k* is based on a fixed approximation threshold specified by the user. Then, phrase analysis is done to identify, for each of the selected topics, a phrase that represents a good description.

Finally, documents are assigned to clusters based on the contained phrases. Experimental results in terms of cluster contamination have been reported in [26].

Other works along the same lines are [40] and [11]. In all these cases variants of phrase-analysis algorithms are developed in order to improve the quality of the original STC algorithm. As an example, in [40] the problem of improving cluster labels is studied. A machine learning approach is used: given a corpus of training data, the system is able to identify the most salient phrases and use them as cluster titles. The algorithm is therefore supervised. It is worth noting that this work was developed by Microsoft Research Asia, and has been used to provide an experimental clustering service based on MSN contents [33].

SnakeT [11] introduces advanced cluster labeling with variable-length sentences, based on the use of "gapped sentences", i.e., sentences made of terms that may not appear contiguously in the original snippet. The main focus of the system in on personalization: the authors show how to plug the clustering engine into a search engine to obtain a form of personalization.

A different approach has been recently undertaken in the Infocious System [22]. Infocious is a full-fledged search engine based on natural language processing techniques. Linguistic processing is built into the search engine in order to provide a deeper form of processing of documents and search terms, and a much richer search interface to users. The authors compare the classification power of a Naive Bayes Classifier to that of a classifier enhanced with NLP techniques, and show that this may reduce error rate of about 7%.

6 Conclusions

The paper has introduced a new algorithm for clustering the results of a search engine. With respect to snippet-based clustering algorithms, we have shown that, by considering the whole document content and employing appropriate SVD-based compression techniques, it is possible to achieve very good classification results, significantly better than those obtained by analyzing document snippets only.

We believe that these results represent promising directions to improve the quality of clustering in all context in which it is reasonable to assume that document vectors are available to the clustering system. One of these cases is that of desktop search engines. This impression is confirmed by our experiences with a prototypical implementation of the algorithm in the Noodles desktop search engine, which showed how the proposed approach represents a good compromise between quality and efficiency.

Another possible context of application is that in which the clustering algorithm is integrated into the search engine, and not run as a postprocessing step. In fact, Google has experienced for a while with forms of clustering of their search results [31], and has recently introduced a "Refine Your Query" feature[7] that essentially allows to select one of a few topics to narrow a search. Similar experiments are also being conducted by Microsoft [33]. We believe that this might draw more attention in the future around Web document clustering.

7 http://www.google.com/help/features.html#refine

Acknowledgments
The authors would like to thank Martin Funk, Donatella Occorsio, and Maria Grazia Russo for the insightful discussions during the early stages of this research. Thanks also go to Donatello Santoro for his excellent work in the implementation of the desktop search engine.

Bibliography

[1] P. Anick, S. Vaithyanathan. Exploiting Clustering and Phrases for Context-Based Information Retrieval. In *ACM SIGIR*, 1997.

[2] The Ask.com Search Engine. http://search.ask.com.

[3] T. Berners-Lee, J. Hendler, O. Lassila. The Semantic Web. *Scientific American* (2001) 284(5):34-43.

[4] M. W. Berry. Large-Scale Sparse Singular Value Computations. *The International Journal of Supercomputer Applications* (1992) 6(1):13--49.

[5] S. Brin, L. Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Computer Networks and ISDN Systems* (1998) 30(1-7):107-117.

[6] D. Calvetti, L. Reichel, D. C. Sorensen. An Implicitly Restarted Lanczos Method for Large Symmetric Eigenvalue Problems. *Electronic Transactions on Numerical Analysis* (1994) 2:1-21.

[7] C. Chekuri, P. Raghavan. Web Search Using Automatic Classification. In *Proceedings of the World Wide Web Conference*, 1997.

[8] E. Chisholm, T. G. Kolda. New Term Weighting Formulas for the Vector Space Method in Information Retrieval. *Technical Report n. ORNL/TM-13756* - Computer Science and Mathematics Division - Oak Ridge National Laboratory (1999)

[9] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, R. Harshman. Indexing by Latent Semantic Analysis. *Journal of the American Society for Information Sciences* (1990) 41(6):391-407.

[10] The Open Directory Project (DMOZ). http://www.dmoz.org.

[11] P. Ferragina, A. Gulli. A Personalized Search Engine Based on Web Snippet Hierarchical Clustering. In *Proceedings of the World Wide Web Conference*, 2005.

[12] Google Desktop Search. http://desktop.google.com.

[13] The Grokker Search Engine. http://www.grokker.com.

[14] R. Guha, R. Mc Cool, E. Miller. Semantic Search. In *Proceedings of the World Wide Web Conference*, 2003.

[15] M. A. Hearst, J. O. Pedersen. Re-examining the Cluster Hypothesis: Scatter/Gather on Retrieval Results. In *Proceedings of the ACM SIGIR Conference*, 1996.

[16] The iBoogie Search Engine. http://www.iboogie.com.

[17] A. K. Jain, M. N. Murty, P. J. Flynn. Data Clustering: A Review. *ACM Computing Surveys* (1999) 31(3):265-323.

[18] The Kartoo Search Engine. http://www.kartoo.com.

[19] K. Kummamuru, R. Lotlikar, S. Roy, K. Singal, R. Krishnapuram. A Hierarchical Monothetic Document Clustering Algorithm for Summarization and Browsing Search Results. In *Proceedings of the World Wide Web Conference*, 2004.

[20] Y. S. Maarek, R. Fagin, I. Z. Ben-Shaul, D. Pelleg. Ephemeral Document Clustering for Web Applications. *IBM Research Report RJ 10186* - IBM (2000)

[21] The Noodles Project Web Site. http://www.db.unibas.it/projects/noodles.

[22] A. Ntoulas, G. Chao, J. Cho. The Infocious Web Search Engine: Improving Web Searching through Linguistic Analysis. In *Proceedings of the World Wide Web Conference*, 2005.

[23] S. Osinski, J. Stefanowski, D. Weiss. Lingo: Search Results Clustering Algorithm Based on Singular Value Decomposition. In *Proceedings of the International Conference on Intelligent Information Systems (IIPWM)*, 2004.

[24] S. Osinski, D. Weiss. Conceptual Clustering Using Lingo Algorithm: Evaluation on Open Directory Project Data. In *Proceedings of the International Conference on Intelligent Information Systems (IIPWM)*, 2004.

[25] S. Osinski, D. Weiss. A Concept-Driven Algorithm for Clustering Search Results. *IEEE Intelligent Systems* (2005) 20(3):48-54.

[26] S. Osinski. Dimensionality Reduction Techniques for Search Result Clustering. Master Thesis. Department of Computer Science - University of Sheffield. 2004.

[27] C. Papadimitriou, P. Raghavan, H. Tamaki, S. Vempala. Latent Semantic Indexing: a Probabilistic Analysis. *Journal of Computer and System Sciences* (2000) 61:217 - 235 .

[28] R. C. Prim. Shortest Connection Networks and Some Generalisations. *Bell Systems Technical Journal* (1957) 36:1389-1401.

[29] The Scatter-Gather Project Web Site. http://www.sims.berkeley.edu/~hearst/sg-overview.html.

[30] H. Schutze, C. Silverstein. Projections for Efficient Document Clustering. In *Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, 1997.

[31] Web 2.0 - Exclusive Demonstration of Clustering from Google. http://www.searchenginelowdown.com/2004/10/web-20-exclusive-demonstration-of.html.

[32] D. C. Sorensen. Implicitly Restarted Arnoldi/Lanczos Methods for Large Scale Eigenvalue Calculations. *TR-96-40* - Department of Computational and Applied Mathematics - Rice Universiry (1996)

[33] The SRC Search Engine. http://rwsm.directtaps.net/.

[34] C. J. van Rijsbergen. *Information Retrieval, Second Edition*. London, Butterworths, 1979.

[35] The Vivisimo Search Engine. http://www.vivisimo.com.

[36] The WiseNut Search Engine. http://www.wisenut.com.

[37] C. T. Zahn. Graph-Theoretical Methods for Detecting and Describing Gestalt Clusters. *IEEE Transactions on Computers* (1971) C-20:68-86.

[38] O. Zamir, O. Etzioni. Web Document Clustering: A Feasibility Demonstration. In *Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, 1998.

[39] O. Zamir, O. Etzioni. Grouper: A Dynamic Clustering Interface for Web Search Results. *Computer Networks* (1999) 31(11-16):1361-1374.

[40] H. J. Zeng, Q. C. He, Z. Chen, W. Y. Ma, J. Ma. Learning to Cluster Web Search Results. In *Proceedings of the ACM SIGIR Conference*, 2004.

[41] D. Zhang, Y. Dong. Semantic, Hierarchical, Online Clustering of Web Search Results. In *Proceedings of the Asia-Pacific Web Conference*, 2004.