
The Star Clustering Algorithm for Information Organization

J.A. Aslam, E. Pelekhev, and D. Rus

Summary. We present the star clustering algorithm for static and dynamic information organization. The offline star algorithm can be used for clustering static information systems, and the online star algorithm can be used for clustering dynamic information systems. These algorithms organize a data collection into a number of clusters that are naturally induced by the collection via a computationally efficient cover by dense subgraphs. We further show a lower bound on the accuracy of the clusters produced by these algorithms as well as demonstrate that these algorithms are computationally efficient. Finally, we discuss a number of applications of the star clustering algorithm and provide results from a number of experiments with the Text Retrieval Conference data.

1 Introduction

We consider the problem of automatic information organization and present the star clustering algorithm for static and dynamic information organization. Offline information organization algorithms are useful for organizing static collections of data, for example, large-scale legacy collections. Online information organization algorithms are useful for keeping dynamic corpora, such as news feeds, organized. Information retrieval (IR) systems such as Inquery [427], Smart [378], and Google provide automation by computing ranked lists of documents sorted by relevance; however, it is often ineffective for users to scan through lists of hundreds of document titles in search of an information need. Clustering algorithms are often used as a preprocessing step to organize data for browsing or as a postprocessing step to help alleviate the “information overload” that many modern IR systems engender.

There has been extensive research on clustering and its applications to many domains [17, 231]. For a good overview see [242]. For a good overview of using clustering in IR see [455]. The use of clustering in IR was

mostly driven by *the cluster hypothesis* [429], which states that “closely associated documents tend to be related to the same requests.” Jardine and van Rijsbergen [246] show some evidence that search results could be improved by clustering. Hearst and Pedersen [225] re-examine the cluster hypothesis by focusing on the Scatter/Gather system [121] and conclude that it holds for browsing tasks.

Systems like Scatter/Gather [121] provide a mechanism for user-driven organization of data in a fixed number of clusters but the users need to be in the loop and the computed clusters do not have accuracy guarantees. Scatter/Gather uses fractionation to compute nearest-neighbor clusters. Charika et al. [104] consider a dynamic clustering algorithm to partition a collection of text documents into a *fixed* number of clusters. Since in dynamic information systems the number of topics is not known a priori, a fixed number of clusters cannot generate a natural partition of the information.

In this chapter, we provide an overview of our work on clustering algorithms and their applications [26–33]. We propose an offline algorithm for clustering static information and an online version of this algorithm for clustering dynamic information. These two algorithms compute clusters induced by the natural topic structure of the information space. Thus, this work is different from [104, 121] in that we do not impose the constraint to use a fixed number of clusters. As a result, we can guarantee a lower bound on the topic similarity between the documents in each cluster. The model for topic similarity is the standard vector space model used in the IR community [377], which is explained in more detail in Sect. 2 of this chapter.

While the clustering document represented in the vector space model is our primary motivating example, our algorithms can be applied to clustering any set of objects for which a similarity measure is defined, and the performance results stated largely apply whenever the objects themselves are represented in a feature space in which similarity is defined by the cosine metric.

To compute accurate clusters, we formalize clustering as covering graphs by cliques [256] (where the cover is a vertex cover). Covering by cliques is NP complete and thus intractable for large document collections. Unfortunately, it has also been shown that the problem cannot be approximated even in polynomial time [322, 465]. We instead use a cover by *dense subgraphs* that are *star shaped* and that can be computed *offline* for static data and *online* for dynamic data. We show that the offline and the online algorithms produce correct clusters efficiently. Asymptotically, the running time of both algorithms is roughly linear in the size of the similarity graph that defines the information space (explained in detail in Sect. 2). We also show lower bounds on the topic similarity within the computed clusters (a measure of the accuracy of our clustering algorithm) as well as provide experimental data.

We further compare the performance of the star algorithm to two widely used algorithms for clustering in IR and other settings: the single link

method¹ [118] and the average link algorithm² [434]. Neither algorithm provides guarantees for the topic similarity within a cluster. The single link algorithm can be used in offline and online modes, and it is faster than the average link algorithm, but it produces poorer clusters than the average link algorithm. The average link algorithm can only be used offline to process static data. The star clustering algorithm, on the other hand, computes topic clusters that are naturally induced by the collection, provides guarantees on cluster quality, computes more accurate clusters than either the single link or the average link methods, is efficient, admits an efficient and simple online version, and can perform hierarchical data organization. We describe experiments in this chapter with the TREC³ collection demonstrating these abilities.

Finally, we discuss the use of the star clustering algorithm in a number of different application areas including (1) automatic information organization systems, (2) scalable information organization for large corpora, (3) text filtering, and (4) persistent queries.

2 Motivation for the Star Clustering Algorithm

In this section we describe our clustering model and provide motivation for the star clustering algorithm. We begin by describing the vector space model for document representation and consider an idealized clustering algorithm based on clique covers. Given that clique cover algorithms are computationally infeasible, we redundantly propose an algorithm based on star covers. Finally, we argue that star covers retain many of the desired properties of clique covers in expectation, and we demonstrate in subsequent sections that clusterings based on star covers can be computed very efficiently both online and offline.

2.1 Clique Covers in the Vector Space Model

We formulate our problem by representing a document collection by its *similarity graph*. A similarity graph is an undirected, weighted graph $G = (V, E, w)$, where the vertices in the graph correspond to documents and each weighted edge in the graph corresponds to a measure of similarity between two documents. We measure the similarity between two documents by using a standard metric from the IR community – the cosine metric in the vector space model of the Smart IR system [377, 378].

¹In the single link clustering algorithm a document is part of a cluster if it is “related” to at least *one* document in the cluster

²In the average link clustering algorithm a document is part of a cluster if it is “related” to an average number of documents in the cluster

³TREC is the Annual Text Retrieval Conference. Each participant is given of the order of 5 GB of data and a standard set of queries to test the systems. The results and the system descriptions are presented as papers at the TREC

The vector space model for textual information aggregates statistics on the occurrence of words in documents. The premise of the vector space model is that two documents are similar if they use similar words. A vector space can be created for a collection (or corpus) of documents by associating each important word in the corpus with one dimension in the space. The result is a high-dimensional vector space. Documents are mapped to vectors in this space according to their word frequencies. Similar documents map to nearby vectors. In the vector space model, document similarity is measured by the angle between the corresponding document vectors. The standard in the IR community is to map the angles to the interval $[0, 1]$ by taking the cosine of the vector angles.

G is a complete graph with edges of varying weight. An organization of the graph that produces reliable clusters of similarity σ (i.e., clusters where documents have pairwise similarities of at least σ) can be obtained by (1) thresholding the graph at σ and (2) performing a *minimum clique cover* with maximal cliques on the resulting graph G_σ . The *thresholded graph* G_σ is an undirected graph obtained from G by eliminating all the edges whose weights are lower than σ . The minimum clique cover has two features. First, by using cliques to cover the similarity graph, we are guaranteed that all the documents in a cluster have the desired degree of similarity. Second, minimal clique covers with maximal cliques allow vertices to belong to *several* clusters. In many information retrieval applications, this is a desirable feature as documents can have multiple subthemes.

Unfortunately, this approach is computationally intractable. For real corpora, similarity graphs can be very large. The clique cover problem is NP-complete, and it does not admit polynomial-time approximation algorithms [322, 465]. While we cannot perform a clique cover or even approximate such a cover, we can instead cover our graph by *dense subgraphs*. What we lose in intracluster similarity guarantees, we gain in computational efficiency.

2.2 Star Covers

We approximate a clique cover by covering the associated thresholded similarity graph with *star-shaped subgraphs*. A star-shaped subgraph on $m+1$ vertices consists of a single *star center* and m *satellite vertices*, where there exist edges between the star center and each of the satellite vertices (see Fig. 1). While finding cliques in the thresholded similarity graph G_σ *guarantees* a pairwise similarity between documents of at least σ , it would appear at first glance that finding star-shaped subgraphs in G_σ would provide similarity guarantees between the star center and each of the satellite vertices, but no such similarity guarantees *between satellite vertices*. However, by investigating the geometry of our problem in the vector space model, we can derive a *lower bound* on the similarity between satellite vertices as well as provide a formula for the *expected* similarity between satellite vertices. The latter formula predicts that the pairwise similarity between satellite vertices in a star-shaped subgraph is

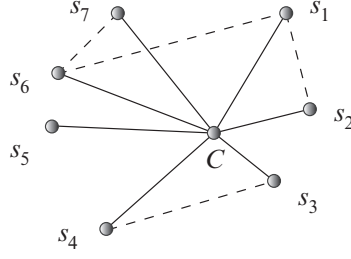


Fig. 1. An example of a star-shaped subgraph with a center vertex C and satellite vertices s_1 – s_7 . The edges are denoted by solid and dashed lines. Note that there is an edge between each satellite and a center, and that edges may also exist between satellite vertices

high, and together with empirical evidence supporting this formula, we conclude that covering G_σ with star-shaped subgraphs is an accurate method for clustering a set of documents.

Consider three documents C , s_1 , and s_2 that are vertices in a star-shaped subgraph of G_σ , where s_1 and s_2 are satellite vertices and C is the star center. By the definition of a star-shaped subgraph of G_σ , we must have that the similarity between C and s_1 is at least σ and that the similarity between C and s_2 is also at least σ . In the vector space model, these similarities are obtained by taking the cosine of the angle between the vectors associated with each document. Let α_1 be the angle between C and s_1 , and let α_2 be the angle between C and s_2 . We then have that $\cos \alpha_1 \geq \sigma$ and $\cos \alpha_2 \geq \sigma$. Note that the angle between s_1 and s_2 can be at most $\alpha_1 + \alpha_2$; we therefore have the following lower bound on the similarity between satellite vertices in a star-shaped subgraph of G_σ .

Theorem 1. *Let G_σ be a similarity graph and let s_1 and s_2 be two satellites in the same star in G_σ . Then the similarity between s_1 and s_2 must be at least*

$$\cos(\alpha_1 + \alpha_2) = \cos \alpha_1 \cos \alpha_2 - \sin \alpha_1 \sin \alpha_2.$$

The use of Theorem 1 to bound the similarity between satellite vertices can yield somewhat disappointing results. For example, if $\sigma = 0.7$, $\cos \alpha_1 = 0.75$, and $\cos \alpha_2 = 0.85$, we can conclude that the similarity between the two satellite vertices must be at least⁴:

$$0.75 \times 0.85 - \sqrt{1 - (0.75)^2} \sqrt{1 - (0.85)^2} \approx 0.29.$$

⁴Note that $\sin \theta = \sqrt{1 - \cos^2 \theta}$

Note that while this may not seem very encouraging, the analysis is based on absolute worst-case assumptions, and in practice, the similarities between satellite vertices are much higher. We can instead reason about the *expected* similarity between two satellite vertices by considering the geometric constraints imposed by the vector space model as follows.

Theorem 2. *Let C be a star center, and let S_1 and S_2 be the satellite vertices of C . Then the similarity between S_1 and S_2 is given by*

$$\cos \alpha_1 \cos \alpha_2 + \cos \theta \sin \alpha_1 \sin \alpha_2,$$

where θ is the dihedral angle⁵ between the planes formed by S_1C and S_2C .

This theorem is a fairly direct consequence of the geometry of C , S_1 , and S_2 in the vector space; details may be found in [31].

How might we eliminate the dependence on $\cos \theta$ in this formula? Consider three vertices from a cluster of similarity σ . Randomly chosen, the pairwise similarities among these vertices should be $\cos \omega$ for some ω satisfying $\cos \omega \geq \sigma$. We then have

$$\cos \omega = \cos \omega \cos \omega + \cos \theta \sin \omega \sin \omega$$

from which it follows that

$$\cos \theta = \frac{\cos \omega - \cos^2 \omega}{\sin^2 \omega} = \frac{\cos \omega(1 - \cos \omega)}{1 - \cos^2 \omega} = \frac{\cos \omega}{1 + \cos \omega}.$$

Substituting for $\cos \theta$ and noting that $\cos \omega \geq \sigma$, we obtain

$$\cos \gamma \geq \cos \alpha_1 \cos \alpha_2 + \frac{\sigma}{1 + \sigma} \sin \alpha_1 \sin \alpha_2. \quad (1)$$

Equation (1) provides an accurate estimate of the similarity between two satellite vertices, as we demonstrate empirically.

Note that for the example given in Sect. 2.2, (1) would predict a similarity between satellite vertices of approximately 0.78. We have tested this formula against real data, and the results of the test with the TREC FBIS data set⁶ are shown in Fig. 2. In this plot, the x -axis and y -axis are similarities between cluster centers and satellite vertices, and the z -axis is the root mean squared prediction error (RMS) of the formula in Theorem 2 for the similarity between satellite vertices. We observe the maximum root mean squared error is quite small (approximately 0.16 in the worst case), and for reasonably high similarities, the error is negligible. From our tests with real data, we have concluded that (1) is quite accurate. We may further conclude that star-shaped subgraphs are reasonably “dense” in the sense that they imply relatively high pairwise similarities between all documents in the star.

⁵The dihedral angle is the angle between two planes on a third plane normal to the intersection of the two planes

⁶Foreign Broadcast Information Service (FBIS) is a large collection of text documents used in TREC

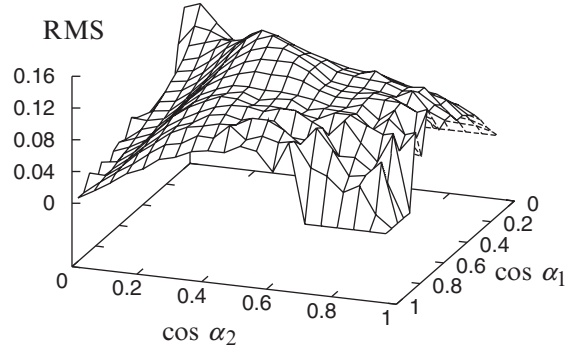


Fig. 2. The RMS prediction error of our expected satellite similarity formula over the TREC FBIS collection containing 21,694 documents

3 The Offline Star Clustering Algorithm

Motivated by the discussion of Sect.2, we now present the *star algorithm*, which can be used to organize documents in an information system. The star algorithm is based on a greedy cover of the thresholded similarity graph by star-shaped subgraphs; the algorithm itself is summarized in Fig.3 below.

Theorem 3. *The running time of the offline star algorithm on a similarity graph G_σ is $\Theta(V + E_\sigma)$.*

For any threshold σ :

1. Let $G_\sigma = (V, E_\sigma)$ where $E_\sigma = \{e \in E : w(e) \geq \sigma\}$.
2. Let each vertex in G_σ initially be *unmarked*.
3. Calculate the degree of each vertex $v \in V$.
4. Let the highest degree unmarked vertex be a star center, and construct a cluster from the star center and its associated satellite vertices. Mark each node in the newly constructed star.
5. Repeat Step 4 until all nodes are marked.
6. Represent each cluster by the document corresponding to its associated star center.

Fig. 3. The star algorithm

Proof. The following implementation of this algorithm has a running time *linear* in the size of the graph. Each vertex v has a data structure associated with it that contains $v.degree$, the degree of the vertex, $v.adj$, the list of adjacent vertices, $v.marked$, which is a bit denoting whether the vertex belongs to a star or not, and $v.center$, which is a bit denoting whether the vertex is a star center. (Computing $v.degree$ for each vertex can be easily performed in $\Theta(V + E_\sigma)$ time.) The implementation starts by sorting the vertices in V by degree ($\Theta(V)$ time since degrees are integers in the range $\{0, |V|\}$). The program then scans the sorted vertices from the highest degree to the lowest as a greedy search for star centers. Only vertices that do not belong to a star already (that is, they are unmarked) can become star centers. Upon selecting a new star center v , its $v.center$ and $v.marked$ bits are set and for all $w \in v.adj$, $w.marked$ is set. Only one scan of V is needed to determine all the star centers. Upon termination, the star centers and only the star centers have the *center* field set. We call the set of star centers the *star cover* of the graph. Each star is fully determined by the star center, as the satellites are contained in the adjacency list of the center vertex. \square

This algorithm has two features of interest. The first feature is that the star cover is not unique. A similarity graph may have several different star covers because when there are several vertices of the same highest degree, the algorithm arbitrarily chooses one of them as a star center (whichever shows up first in the sorted list of vertices). The second feature of this algorithm is that it provides a simple encoding of a star cover by assigning the types “center” and “satellite” (which is the same as “not center” in our implementation) to vertices. We define a *correct star cover* as a star cover that assigns the types “center” and “satellite” in such a way that (1) a star center is not adjacent to any other star center and (2) every satellite vertex is adjacent to at least one center vertex of equal or higher degree.

Figure 4 shows two examples of star covers. The left graph consists of a clique subgraph (first subgraph) and a set of nodes connected to only to the nodes in the clique subgraph (second subgraph). The star cover of the left graph includes one vertex from the 4-clique subgraph (which covers the entire clique and the one nonclique vertex it is connected to), and single-node stars for each of the noncovered vertices in the second set. The addition of a node connected to all the nodes in the second set changes the clique cover dramatically. In this case, the new node becomes a star center. It thus covers all the nodes in the second set. Note that since star centers cannot be adjacent, no vertex from the second set is a star center in this case. One node from the first set (the clique) remains the center of a star that covers that subgraph. This example illustrates the connection between a star cover and other important graph sets, such as set covers and induced dominating sets, which have been studied extensively in the literature [19, 183]. The star cover is related but not identical to a dominating set [183]. Every star cover is a dominating set, but there are dominating sets that are not star covers.

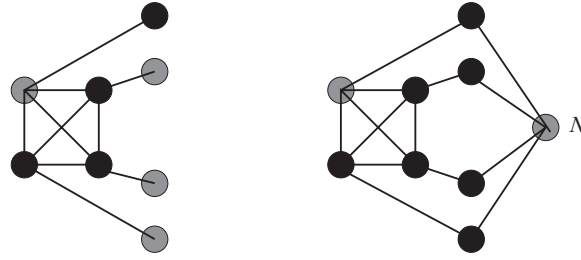


Fig. 4. An example of a star-shaped cover before and after the insertion of the node N in the graph. The dark circles denote satellite vertices. The shaded circles denote star centers

Star covers are useful approximations of clique covers because star graphs are dense subgraphs for which we can infer something about the missing edges as we have shown earlier.

Given this definition for the star cover, it immediately follows that:

Theorem 4. *The offline star algorithm produces a correct star cover.*

We use the two features of the offline algorithm mentioned earlier in the analysis of the online version of the star algorithm in Sect. 4. In Sect. 5, we show that the clusters produced by the star algorithm are quite accurate, exceeding the accuracy produced by widely used clustering algorithms in IR.

4 The Online Star Algorithm

The star clustering algorithm described in Sect. 3 can be used to accurately and efficiently cluster a static collection of documents. However, it is often the case in information systems that documents are added to, or deleted from, a dynamic collection. In this section, we describe an online version of the star clustering algorithm, which can be used to efficiently maintain a star clustering in the presence of document insertions and deletions.

We assume that documents are inserted or deleted from the collection one at a time. We begin by examining INSERT. The intuition behind the incremental computation of the star cover of a graph after a new vertex is inserted is depicted in Fig. 5. The top figure denotes a similarity graph and a correct star cover for this graph. Suppose a new vertex is inserted in the graph, as in the middle figure. The original star cover is no longer correct for the new graph. The bottom figure shows the correct star cover for the new graph. How does the addition of this new vertex affect the correctness of the star cover?

In general, the answer depends on the degree of the new vertex and its adjacency list. If the adjacency list of the new vertex does not contain any star centers, the new vertex can be added in the star cover as a star center. If the adjacency list of the new vertex contains any center vertex c whose degree is equal or higher, the new vertex becomes a satellite vertex of c . The difficult cases that destroy the correctness of the star cover are (1) when the new vertex is adjacent to a collection of star centers, each of whose degree is lower than that of the new vertex and (2) when the new vertex increases the degree of an adjacent satellite vertex beyond the degree of its associated star center. In these situations, the star structure already in place has to be modified; existing stars must be broken. The satellite vertices of these broken stars must be re-evaluated.

Similarly, deleting a vertex from a graph may destroy the correctness of a star cover. An initial change affects a star if (1) its center is removed or (2) the degree of the center has decreased because of a deleted satellite. The satellites in these stars may no longer be adjacent to a center of equal or higher degree, and their status must be reconsidered.

4.1 The Online Algorithm

Motivated by the intuition in the previous section, we now describe a simple online algorithm for incrementally computing star covers of dynamic graphs. The algorithm uses a data structure to efficiently maintain the star covers of an undirected graph $G = (V, E)$. For each vertex $v \in V$, we maintain the following data:

v.type satellite or center
v.degree degree of v
v.adj list of adjacent vertices
v.centers list of adjacent centers
v.inQ flag specifying if v being processed

Note that while *v.type* can be inferred from *v.centers* and *v.degree* can be inferred from *v.adj*, it will be convenient to maintain all five pieces of data in the algorithm.

The basic idea behind the online star algorithm is as follows. When a vertex is inserted into (or deleted from) a thresholded similarity graph G_σ , new stars may need to be created and existing stars may need to be destroyed. An existing star is never destroyed unless a satellite is “promoted” to center status. The online star algorithm functions by maintaining a priority queue (indexed by vertex degree), which contains all satellite vertices that have the possibility of being promoted. So long as these enqueued vertices are indeed properly satellites, the existing star cover is correct. The enqueued satellite vertices are processed in order by degree (highest to lowest), with satellite promotion occurring as necessary. Promoting a satellite vertex may destroy one or more existing stars, creating new satellite vertices that have the possibility of being promoted. These satellites are enqueued, and the process repeats. We

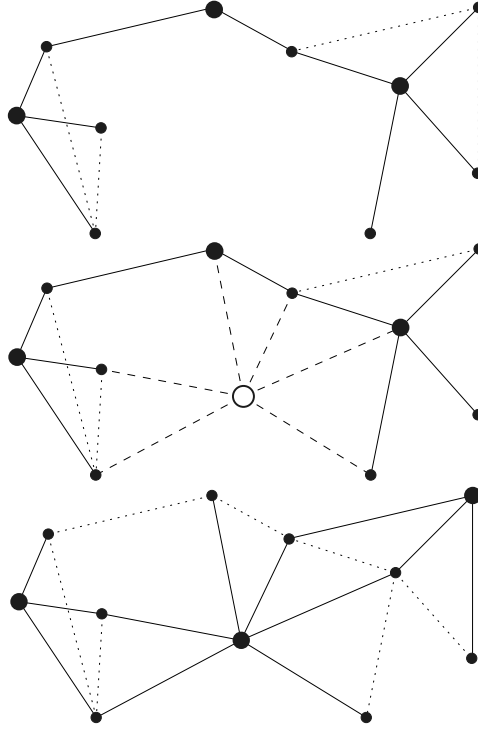


Fig. 5. The star cover change after the insertion of a new vertex. The larger-radius disks denote star centers, the other disks denote satellite vertices. The star edges are denoted by solid lines. The intersatellite edges are denoted by dotted lines. The top figure shows an initial graph and its star cover. The middle figure shows the graph after the insertion of a new document. The bottom figure shows the star cover of the new graph

next describe in some detail the three routines that comprise the online star algorithm.

The INSERT and DELETE procedures are called when a vertex is added to or removed from a thresholded similarity graph, respectively. These procedures appropriately modify the graph structure and initialize the priority queue with all satellite vertices that have the possibility of being promoted. The UPDATE procedure promotes satellites as necessary, destroying existing stars if required, and enqueueing any new satellites that have the possibility of being promoted.

Figure 6 provides the details of the INSERT algorithm. A vertex α with a list of adjacent vertices L is added to a graph G . The priority queue Q is initialized with α (lines 17 and 18) and its adjacent satellite vertices (lines 13 and 14).

```

INSERT( $\alpha, L, G_\sigma$ )
1   $\alpha.type \leftarrow satellite$ 
2   $\alpha.degree \leftarrow 0$ 
3   $\alpha.adj \leftarrow \emptyset$ 
4   $\alpha.centers \leftarrow \emptyset$ 
5  forall  $\beta$  in  $L$ 
6     $\alpha.degree \leftarrow \alpha.degree + 1$ 
7     $\beta.degree \leftarrow \beta.degree + 1$ 
8    INSERT( $\beta, \alpha.adj$ )
9    INSERT( $\alpha, \beta.adj$ )
10   if ( $\beta.type = center$ )
11     INSERT( $\beta, \alpha.centers$ )
12   else
13      $\beta.inQ \leftarrow true$ 
14     ENQUEUE( $\beta, Q$ )
15   endif
16 endfor
17  $\alpha.inQ \leftarrow true$ 
18 ENQUEUE( $\alpha, Q$ )
19 UPDATE( $G_\sigma$ )

```

Fig. 6. Pseudocode for INSERT

```

DELETE( $\alpha, G_\sigma$ )
1  forall  $\beta$  in  $\alpha.adj$ 
2     $\beta.degree \leftarrow \beta.degree - 1$ 
3    DELETE( $\alpha, \beta.adj$ )
4  endfor
5  if ( $\alpha.type = satellite$ )
6    forall  $\beta$  in  $\alpha.centers$ 
7      forall  $\mu$  in  $\beta.adj$ 
8        if ( $\mu.inQ = false$ )
9           $\mu.inQ \leftarrow true$ 
10         ENQUEUE( $\mu, Q$ )
11        endif
12      endfor
13    endfor
14  else
15    forall  $\beta$  in  $\alpha.adj$ 
16      DELETE( $\alpha, \beta.centers$ )
17       $\beta.inQ \leftarrow true$ 
18      ENQUEUE( $\beta, Q$ )
19    endfor
20  endif
21  UPDATE( $G_\sigma$ )

```

Fig. 7. Pseudocode for DELETE

The DELETE algorithm presented in Fig. 7 removes vertex α from the graph data structures, and depending on the type of α enqueues its adjacent satellites (lines 15–19) or the satellites of its adjacent centers (lines 6–13).

Finally, the algorithm for UPDATE is shown in Fig. 8. Vertices are organized in a priority queue, and a vertex ϕ of highest degree is processed in each iteration (line 2). The algorithm creates a new star with center ϕ if ϕ has no adjacent centers (lines 3–7) or if all its adjacent centers have lower degree (lines 9–13). The latter case destroys the stars adjacent to ϕ , and their satellites are enqueued (lines 14–23). The cycle is repeated until the queue is empty.

Correctness and Optimizations

The online star cover algorithm is more complex than its offline counterpart. One can show that the online algorithm is correct by proving that it produces the same star cover as the offline algorithm, when the offline algorithm is run on the final graph considered by the online algorithm. We first note, however, that the offline star algorithm need not produce a unique cover. When there are several unmarked vertices of the same highest degree, the algorithm can arbitrarily choose one of them as the next star center. In this context, one can

```

UPDATE( $G_\sigma$ )
1  while ( $Q \neq \emptyset$ )
2     $\phi \leftarrow \text{EXTRACTMAX}(Q)$ 
3    if ( $\phi.\text{centers} = \emptyset$ )
4       $\phi.\text{type} \leftarrow \text{center}$ 
5      forall  $\beta$  in  $\phi.\text{adj}$ 
6        INSERT( $\phi, \beta.\text{centers}$ )
7      endfor
8    else
9      if ( $\forall \delta \in \phi.\text{centers}, \delta.\text{degree} < \phi.\text{degree}$ )
10        $\phi.\text{type} \leftarrow \text{center}$ 
11       forall  $\beta$  in  $\phi.\text{adj}$ 
12         INSERT( $\phi, \beta.\text{centers}$ )
13       endfor
14       forall  $\delta$  in  $\phi.\text{centers}$ 
15          $\delta.\text{type} \leftarrow \text{satellite}$ 
16         forall  $\mu$  in  $\delta.\text{adj}$ 
17           DELETE( $\delta, \mu.\text{centers}$ )
18           if ( $\mu.\text{degree} \leq \delta.\text{degree} \wedge \mu.\text{in}Q = \text{false}$ )
19              $\mu.\text{in}Q \leftarrow \text{true}$ 
20             ENQUEUE( $\mu, Q$ )
21           endif
22         endfor
23       endfor
24        $\phi.\text{centers} \leftarrow \emptyset$ 
25     endif
26   endif
27    $\phi.\text{in}Q \leftarrow \text{false}$ 
28 endwhile

```

Fig. 8. Pseudocode for UPDATE

show that the cover produced by the online star algorithm is the same as one of the covers that can be produced by the offline algorithm. We can view a star cover of G_σ as a correct assignment of types (that is, “center” or “satellite”) to the vertices of G_σ . The offline star algorithm assigns correct types to the vertices of G_σ . The online star algorithm is proven correct by induction. The induction invariant is that at all times, the types of all vertices in $V - Q$ are correct, *assuming* that the true type of all vertices in Q is “satellite.” This would imply that when Q is empty, all vertices are assigned a correct type, and thus the star cover is correct. Details can be found in [28, 31].

Finally, we note that the online algorithm can be implemented more efficiently than described here. An optimized version of the online algorithm exists, which maintains additional information and uses somewhat different data structures. While the asymptotic running time of the optimized version

of the online algorithm is unchanged, the optimized version is often faster in practice. Details can be found in [31].

4.2 Expected Running Time of the Online Algorithm

In this section, we argue that the running time of the online star algorithm is quite efficient, asymptotically matching the running time of the offline star algorithm within logarithmic factors. We first note, however, that there exist worst-case thresholded similarity graphs and corresponding vertex insertion/deletion sequences that cause the online star algorithm to “thrash” (i.e., which cause the entire star cover to change on each inserted or deleted vertex). These graphs and insertion/deletion sequences rarely arise in practice however. An analysis more closely modeling practice is the random graph model [78] in which G_σ is a random graph and the insertion/deletion sequence is random. In this model, the *expected* running time of the online star algorithm can be determined. In the remainder of this section, we argue that the online star algorithm is quite efficient theoretically. In subsequent sections, we provide empirical results that verify this fact for both random data and a large collection of real documents.

The model we use for expected case analysis is the *random graph model* [78]. A random graph $G_{n,p}$ is an undirected graph with n vertices, where each of its possible edges is inserted randomly and independently with probability p . Our problem fits the random graph model if we make the mathematical assumption that “similar” documents are essentially “random perturbations” of one another in the vector space model. This assumption is equivalent to viewing the similarity between two related documents as a random variable. By thresholding the edges of the similarity graph at a fixed value, for each edge of the graph there is a random chance (depending on whether the value of the corresponding random variable is above or below the threshold value) that the edge remains in the graph. This thresholded similarity graph is thus a random graph. While random graphs do not perfectly model the thresholded similarity graphs obtained from actual document corpora (the actual similarity graphs must satisfy various geometric constraints and will be aggregates of many “sets” of “similar” documents), random graphs are easier to analyze, and our experiments provide evidence that theoretical results obtained for random graphs closely match empirical results obtained for thresholded similarity graphs obtained from actual document corpora. As such, we use the random graph model for analysis and experimental verification of the algorithms presented in this chapter (in addition to experiments on actual corpora).

The time required to insert/delete a vertex and its associated edges and to appropriately update the star cover is largely governed by the number of stars that are broken during the update, since breaking stars requires inserting new elements into the priority queue. In practice, very few stars are broken during any given update. This is partly due to the fact that relatively few stars

exist at any given time (as compared to the number of vertices or edges in the thresholded similarity graph) and partly to the fact that the likelihood of breaking any individual star is also small. We begin by examining the expected size of a star cover in the random graph model.

Theorem 5. *The expected size of the star cover for $G_{n,p}$ is at most $1 + 2 \log n / \log(1/(1-p))$.*

Proof. The star cover algorithm is greedy: it repeatedly selects the unmarked vertex of highest degree as a star center, marking this node and all its adjacent vertices as covered. Each iteration creates a new star. We argue that the number of iterations is at most $1 + 2 \log n / \log(1/(1-p))$ for an even weaker algorithm, which merely selects *any* unmarked vertex (at random) to be the next star. The argument relies on the random graph model described earlier.

Consider the (weak) algorithm described earlier which repeatedly selects stars at random from $G_{n,p}$. After i stars have been created, each of the i star centers is marked, and some of the $n - i$ remaining vertices is marked. For any given noncenter vertex, the probability of being adjacent to any given center vertex is p . The probability that a given noncenter vertex remains unmarked is therefore $(1-p)^i$, and thus its probability of being marked is $1 - (1-p)^i$. The probability that *all* $n - i$ noncenter vertices are marked is then $(1 - (1-p)^i)^{n-i}$. This is the probability that i (random) stars are sufficient to cover $G_{n,p}$. If we let X be a random variable corresponding to the number of star required to cover $G_{n,p}$, we then have

$$\Pr[X \geq i + 1] = 1 - (1 - (1-p)^i)^{n-i}.$$

Using the fact that for any discrete random variable Z whose range is $\{1, 2, \dots, n\}$,

$$\mathbb{E}[Z] = \sum_{i=1}^n i \times \Pr[Z = i] = \sum_{i=1}^n \Pr[Z \geq i],$$

we then have

$$\mathbb{E}[X] = \sum_{i=0}^{n-1} \left[1 - (1 - (1-p)^i)^{n-i} \right].$$

Note that for any $n \geq 1$ and $x \in [0, 1]$, $(1-x)^n \geq 1 - nx$. We may then derive

$$\begin{aligned} \mathbb{E}[X] &= \sum_{i=0}^{n-1} \left[1 - (1 - (1-p)^i)^{n-i} \right] \\ &\leq \sum_{i=0}^{n-1} \left[1 - (1 - (1-p)^i)^n \right] \\ &= \sum_{i=0}^{k-1} \left[1 - (1 - (1-p)^i)^n \right] + \sum_{i=k}^{n-1} \left[1 - (1 - (1-p)^i)^n \right] \end{aligned}$$

$$\begin{aligned}
&\leq \sum_{i=0}^{k-1} 1 + \sum_{i=k}^{n-1} n(1-p)^i \\
&= k + \sum_{i=k}^{n-1} n(1-p)^i
\end{aligned}$$

for any k . Selecting k so that $n(1-p)^k = 1/n$ (i.e., $k = 2 \log n / \log(1/(1-p))$), we have

$$\begin{aligned}
\mathbb{E}[X] &\leq k + \sum_{i=k}^{n-1} n(1-p)^i \\
&\leq 2 \log n / \log(1/(1-p)) + \sum_{i=k}^{n-1} 1/n \\
&\leq 2 \log n / \log(1/(1-p)) + 1. \quad \square
\end{aligned}$$

Combining the above theorem with various facts concerning the behavior of the UPDATE procedure, one can show the following.

Theorem 6. *The expected time required to insert or delete a vertex in a random graph $G_{n,p}$ is $O(np^2 \log^2 n / \log^2(1/(1-p)))$, for any $0 \leq p \leq 1 - \Theta(1)$.*

The proof of this theorem is rather technical; details can be found in [31]. The thresholded similarity graphs obtained in a typical IR setting are almost always dense: there exist many vertices comprising relatively few (but dense) clusters. We obtain dense random graphs when p is a constant. For dense graphs, we have the following corollary.

Corollary 1. *The total expected time to insert n vertices into (an initially empty) dense random graph is $O(n^2 \log^2 n)$.*

Corollary 2. *The total expected time to delete n vertices from (an n vertex) dense random graph is $O(n^2 \log^2 n)$.*

Note that the online insertion result for dense graphs compares favorably to the offline algorithm; both algorithms run in time proportional to the size of the input graph, $\Theta(n^2)$, within logarithmic factors. Empirical results on dense random graphs and actual document collections (detailed in Sect. 4.3) verify this result.

For sparse graphs ($p = \Theta(1/n)$), we note that $1/\ln(1/(1-\epsilon)) \approx 1/\epsilon$ for small ϵ . Thus, the expected time to insert or delete a single vertex is $O(np^2 \log^2 n / \log^2(1/(1-p))) = O(n \log^2 n)$, yielding an asymptotic result identical to that of dense graphs, much larger than what one encounters in practice. This is due to the fact that the number of stars broken (and hence

vertices enqueued) is much smaller than the worst-case assumptions assumed in the analysis of the UPDATE procedure. Empirical results on sparse random graphs (detailed in the following section) verify this fact and imply that the total running time of the online insertion algorithm is also proportional to the size of the input graph, $\Theta(n)$, within lower order factors.

4.3 Experimental Validation

To experimentally validate the theoretical results obtained in the random graph model, we conducted efficiency experiments with the online star clustering algorithm using two types of data. The first type of data matches our random graph model and consists of both sparse and dense random graphs. While this type of data is useful as a benchmark for the running time of the algorithm, it does not satisfy the geometric constraints of the vector space model. We also conducted experiments using 2,000 documents from the TREC FBIS collection.

Aggregate Number of Broken Stars

As discussed earlier, the efficiency of the online star algorithm is largely governed by the number of stars that are broken during a vertex insertion or deletion. In our first set of experiments, we examined the aggregate number of broken stars during the insertion of 2,000 vertices into a sparse random graph ($p = 10/n$), a dense random graph ($p = 0.2$), and a graph corresponding to a subset of the TREC FBIS collection thresholded at the mean similarity. The results are given in Fig. 9.

For the sparse random graph, while inserting 2,000 vertices, 2,572 total stars were broken – approximately 1.3 broken stars per vertex insertion on average. For the dense random graph, while inserting 2,000 vertices, 3,973 total stars were broken – approximately 2 broken stars per vertex insertion on average. The thresholded similarity graph corresponding to the TREC FBIS data was much denser, and there were far fewer stars. While inserting 2,000 vertices, 458 total stars were broken – approximately 23 broken stars per 100 vertex insertions on average. Thus, even for moderately large n , the number of broken stars per vertex insertion is a relatively small constant, though we do note the effect of lower order factors especially in the random graph experiments.

Aggregate Running Time

In our second set of experiments, we examined the aggregate running time during the insertion of 2,000 vertices into a sparse random graph ($p = 10/n$), a dense random graph ($p = 0.2$), and a graph corresponding to a subset of the TREC FBIS collection thresholded at the mean similarity. The results are given in Fig. 10.

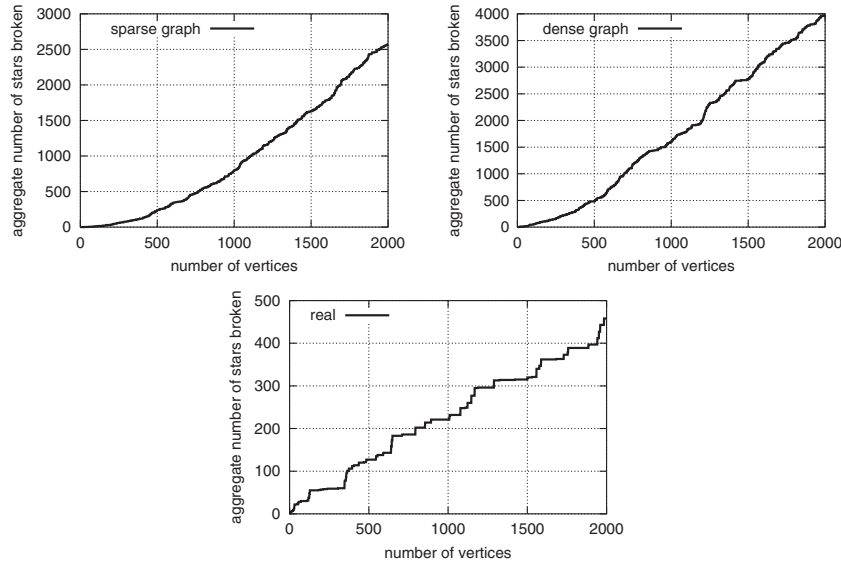


Fig. 9. The dependence of the number of broken stars on the number of inserted vertices in a sparse random graph (top left figure), a dense random graph (top right figure), and the graph corresponding to TREC FBIS data (bottom figure)

Note that for connected input graphs (sparse or dense), the size of the graph is on the order of the number of edges. The experiments depicted in Fig. 10 suggest a running time for the online algorithm, which is linear in the size of the input graph, though lower order factors are presumably present.

5 The Accuracy of Star Clustering

In this section we describe experiments evaluating the performance of the star algorithm with respect to cluster accuracy. We tested the star algorithm against two widely used clustering algorithms in IR: the single link method [429] and the average link method [434]. We used data from the TREC FBIS collection as our testing medium. This TREC collection contains a very large set of documents of which 21,694 have been ascribed relevance judgments with respect to 47 topics. These 21,694 documents were partitioned into 22 separate subcollections of approximately 1,000 documents each for 22 rounds of the following test. For each of the 47 topics, the given collection of documents was clustered with each of the three algorithms, and the cluster that “best” approximated the set of judged relevant documents was returned. To measure the quality of a cluster, we use the standard F measure from IR [429]:

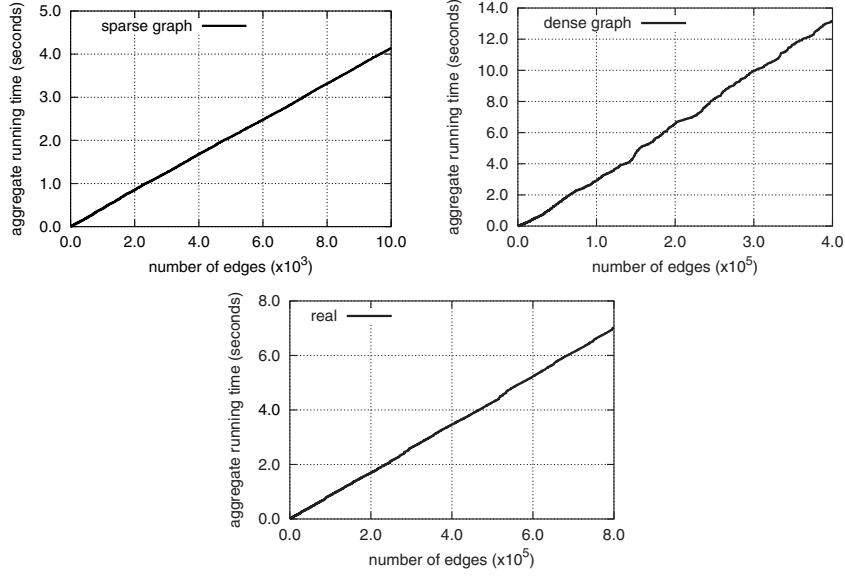


Fig. 10. The dependence of the running time of the online star algorithm on the size of the input graph for a sparse random graph (top left figure), a dense random graph (top right figure), and the graph corresponding to TREC FBIS data (bottom figure)

$$F(p, r) = \frac{2}{(1/p) + (1/r)},$$

where p and r are the *precision* and *recall* of the cluster with respect to the set of documents judged relevant to the topic. Precision is the fraction of returned documents that are correct (i.e., judged relevant), and recall is the fraction of correct documents that are returned. $F(p, r)$ is simply the harmonic mean of the precision and recall; thus, $F(p, r)$ ranges from 0 to 1, where $F(p, r) = 1$ corresponds to perfect precision and recall, and $F(p, r) = 0$ corresponds to either zero precision or zero recall.

For each of the three algorithms, approximately 500 experiments were performed; this is roughly half of the $22 \times 47 = 1,034$ total possible experiments since not all topics were present in all subcollections. In each experiment, the $(p, r, F(p, r))$ values corresponding to the cluster of highest quality were obtained, and these values were averaged over all 500 experiments for each algorithm. The average $(p, r, F(p, r))$ values for the star, average-link, and single-link algorithms were, $(0.77, 0.54, 0.63)$, $(0.83, 0.44, 0.57)$ and $(0.84, 0.41, 0.55)$, respectively. Thus, the star algorithm represents a 10.5% improvement in cluster accuracy with respect to the average-link algorithm and a 14.5% improvement in cluster accuracy with respect to the single-link algorithm.

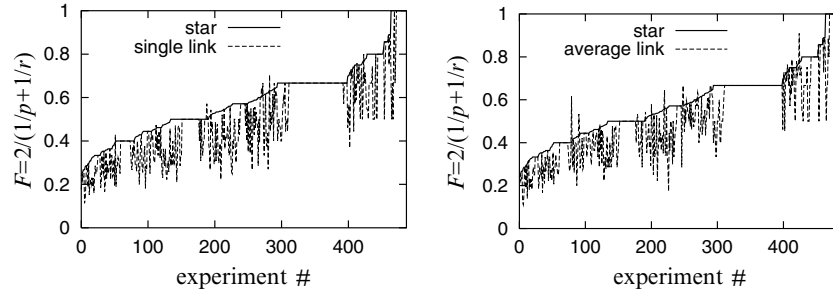


Fig. 11. The F measure for the star clustering algorithm vs. the single link clustering algorithm (left) and the star algorithm vs. the average link algorithm (right). The y axis shows the F measure. The x axis shows the experiment number. Experimental results have been sorted according to the F value for the star algorithm

Figure 11 shows the results of all 500 experiments. The first graph shows the accuracy (F measure) of the star algorithm vs. the single-link algorithm; the second graph shows the accuracy of the star algorithm vs. the average-link algorithm. In each case, the results of the 500 experiments using the star algorithm were sorted according to the F measure (so that the star algorithm results would form a monotonically increasing curve), and the results of both algorithms (star and single-link or star and average-link) were plotted according to this sorted order. While the *average* accuracy of the star algorithm is higher than that of either the single-link or the average-link algorithms, we further note that the star algorithm outperformed each of these algorithms in nearly *every* experiment.

Our experiments show that in general, the star algorithm outperforms single-link by 14.5% and average-link by 10.5%. We repeated this experiment on the same data set, using the entire unpartitioned collection of 21,694 documents, and obtained similar results. The precision, recall, and F values for the star, average-link, and single-link algorithms were (0.53, 0.32, 0.42), (0.63, 0.25, 0.36), and (0.66, 0.20, 0.30), respectively. We note that the F values are worse for all three algorithms on this larger collection and that the star algorithm outperforms the average-link algorithm by 16.7% and the single-link algorithm by 40%. These improvements are significant for IR applications. Given that (1) the star algorithm outperforms the average-link algorithm, (2) it can be used as an online algorithm, (3) it is relatively simple to implement in either of its offline or online forms, and (4) it is efficient, these experiments provide support for using the star algorithm for offline and online information organization.

6 Applications of the Star Clustering Algorithm

We have investigated the use of the star clustering algorithm in a number of different application areas including: (1) automatic information organization systems [26, 27], (2) scalable information organization for large corpora [33], (3) text filtering [29, 30], and (4) persistent queries [32]. In the sections that follow, we briefly highlight this work.

6.1 A System for Information Organization

We have implemented a system for organizing information that uses the star algorithm (see Fig. 12). This organization system consists of an augmented version of the Smart system [18, 378], a user interface we have designed, and an implementation of the star algorithms on top of Smart. To index the documents, we used the Smart search engine with a cosine normalization weighting scheme. We enhanced Smart to compute a document-to-document similarity matrix for a set of retrieved documents or a whole collection. The similarity matrix is used to compute clusters and to visualize the clusters.

The figure shows the interface to the information organization system. The search and organization choices are described at the top. The middle two windows show two views of the organized documents retrieved from the Web or from the database. The left window shows the list of topics, the number of documents in each topic, and a keyword summary for each topic. The right window shows a graphical description of the topics. Each topic corresponds to a disk. The size of the disk is proportional to the number of documents in the topic cluster and the distance between two disks is proportional to the topic similarity between the corresponding topics. The bottom window shows a list of titles for the documents. The three views are connected: a click in one window causes the corresponding information to be highlighted in the other two windows. Double clicking on any cluster (in the right or left middle panes) causes the system to organize and present the documents in that cluster, thus creating a view one level deeper in a hierarchical cluster tree; the “Zoom Out” button allows one to retreat to a higher level in the cluster tree. Details on this system and its variants can be found in [26, 27, 29].

6.2 Scalable Information Organization

The star clustering algorithm implicitly assumes the existence of a thresholded similarity graph. While the running times of the offline and the online star clustering algorithms are linear in the size of the input graph (to within lower order factors), the size of these graphs themselves may be prohibitively large. Consider, for example, an information system containing n documents and a request to organize this system with a relatively low similarity threshold. The resulting graph would in all likelihood be dense, i.e., have $\Omega(n^2)$ edges. If n is large (e.g., millions), just computing the thresholded similarity graph may

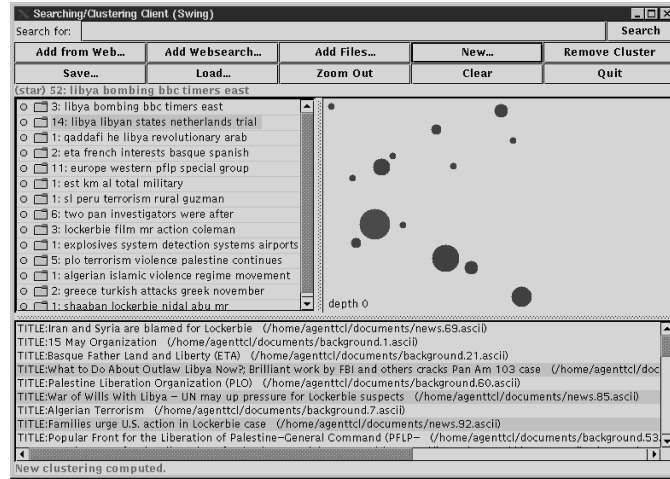


Fig. 12. A system for information organization based on the star clustering algorithm

be prohibitively expensive, let alone running a clustering algorithm on such a graph.

In [33], we propose three methods based on sampling and/or parallelism for generating accurate *approximations* to a star cover in time linear in the number of documents, independent of the size of the thresholded similarity graph.

6.3 Filtering and Persistent Queries

Information filtering and persistent query retrieval are related problems wherein relevant elements of a dynamic stream of documents are sought in order to satisfy a user's information need. The problems differ in how the information need is supplied: in the case of filtering, exemplar documents are supplied by the user, either dynamically or in advance; in the case of persistent query retrieval, a standing query is supplied by the user.

We propose a solution to the problems of information filtering and persistent query retrieval through the use of the star clustering algorithm. The salient features of the systems we propose are (1) the user has access to the *topic structure* of the document collection star clusters; (2) the query (filtering topic) can be formulated as a list of keywords, a set of selected documents, or a set of selected *document clusters*; (3) document filtering is based on *prospective cluster membership*; (4) the user can modify the query by providing relevance feedback on the document clusters and individual documents in the *entire collection*; and (5) the relevant documents adapt as the collection changes. Details can be found in [29, 30, 32].

7 Conclusions

We presented and analyzed an offline clustering algorithm for static information organization and an online clustering algorithm for dynamic information organization. We described a random graph model for analyzing the running times of these algorithms, and we showed that in this model, these algorithms have an expected running time that is linear in the size of the input graph, to within lower order factors. The data we gathered from experiments with TREC data lend support for the validity of our model and analyses. Our empirical tests show that both algorithms exhibit linear time performance in the size of the input graph (to within lower order factors), and that both algorithms produce accurate clusters. In addition, both algorithms are simple and easy to implement. We believe that efficiency, accuracy, and ease of implementation make these algorithms very practical candidates for use in automatic information organization systems.

This work departs from previous clustering algorithms often employed in IR settings, which tend to use a fixed number of clusters for partitioning the document space. Since the number of clusters produced by our algorithms is given by the underlying topic structure in the information system, our clusters are dense and accurate. Our work extends previous results [225] that support using clustering for browsing applications and presents positive evidence for the cluster hypothesis. In [26], we argue that by using a clustering algorithm that guarantees the cluster quality through separation of dissimilar documents and aggregation of similar documents, clustering is beneficial for information retrieval tasks that require both high precision and high recall.

Acknowledgments

This research was supported in part by ONR contract N00014-95-1-1204, DARPA contract F30602-98-2-0107, and NSF grant CCF-0418390.

Grouping Multidimensional Data

Recent Advances in Clustering

Kogan, J.; Nicholas, C.; Teboulle, M. (Eds.)

2006, XII, 268 p., Hardcover

ISBN: 978-3-540-28348-5