A Practical Clustering Algorithm for Static and Dynamic Information Organization*

Javed Aslam Katya Pelekhov Daniela Rus

Dartmouth College†

Abstract

We present and analyze the off-line star algorithm for clustering static information systems and the on-line star algorithm for clustering dynamic information systems. These algorithms organize a document collection into a number of clusters that is 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 efficient (running times roughly linear in the size of the problem). Finally, we provide data from a number of experiments.

1 Introduction

We wish to create more versatile information capture and access systems for digital libraries by using information organization: thousands of electronic documents will be organized automatically as a hierarchy of topics and subtopics, using algorithms grounded in geometry, probabilities, and statistics. Off-line information organization algorithms will be useful for organizing static collections (for example, large-scale legacy data). Incremental, on-line information organization algorithms will be useful to keep dynamic corpora, such as news feeds, organized. Current information systems such as Inquery [Tur90], Smart [Sal91], or Alta Vista provide some simple automation by computing ranked (sorted) lists of documents, but it is ineffective for users to scan a list of hundreds of document titles. To cull the relevant information out of a large set of potentially useful dynamic sources we need methods for organizing and reorganizing dynamic information as accurate clusters, and ways of presenting users with the topic summaries at various levels of detail.

There has been extensive research on clustering and applications to many domains [HS86, AB84]. For a good overview see [JD88]. For a good overview of using clustering in Information Retrieval (IR) see [Wil88]. The use of clustering in IR was mostly driven by *the cluster hypothesis* [Rij79] which states that relevant documents tend to be more closely related to each other than to non-relevant documents. Jardine and van Rijsbergen [JR71] show some evidence that search results could be improved by clustering. Hearst and Pedersen [HP96] re-examine the cluster hypothesis by focusing on the Scatter/Gather system [CKP93] and conclude that it holds for browsing tasks.

Systems like Scatter/Gather [CKP93] 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. [CCFM97] 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.

Our work on clustering presented in this paper and in [APR97] provides positive evidence for the cluster hypothesis. We propose an off-line algorithm for clustering static information and an on-line version of this algorithm for clustering dynamic information. These two algorithms compute clusters induced by the natural topic structure of the space. Thus, this work is different than [CKP93, CCFM97] 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 information retrieval community [Sal89] which is explained in more detail in this paper in Section 2.

To compute accurate clusters, we formalize clustering as covering graphs by cliques. Covering by cliques is NP-complete, and thus intractable for large document collections. Unfortunately, it has also been shown that the problem cannot even be approximated in polynomial

*Research partially supported by ONR contract N00014-95-1-1204, Rome Labs contract F30602-98-C-0006, and Air Force MURI contract F49620-97-1-0382.

†Email: {jaa,katya,rus}@cs.dartmouth.edu

time [LY94, Zuc93]. We instead use a cover by *dense subgraphs* that are *star-shaped* and that can be computed *off-line* for static data and *on-line* for dynamic data. We show that the off-line and on-line 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 Section 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.

Finally, we compare the performance of the star algorithm to two widely used algorithms for clustering in IR and other settings: the single link method[1] [Cro77] and the average link algorithm[2] [Voo85]. Neither algorithm provides guarantees for the topic similarity within a cluster. The single link algorithm can be used in off-line and on-line mode, 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 off-line 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 average link methods, is efficient, admits an efficient and simple on-line version, and can perform hierarchical data organization. We describe experiments in this paper with the TREC[3] database demonstrating these abilities.

Our algorithms for organizing information systems can be used in several ways. The off-line algorithm can be used as a pre-processing step in a static information system or as a post-processing step on the specific documents retrieved by a query. As a pre-processor, this system assists users with deciding how to browse a database of free text documents by highlighting relevant topics and irrelevant subtopics. Such clustered data is useful for narrowing down the database over which detailed queries can be formulated. As a post-processor, this system classifies the retrieved data into clusters that capture topic categories and subcategories. The on-line algorithm can be used as a basis for constructing self-organizing information systems. As the content of dynamic information systems changes, the on-line algorithm can efficiently automate the process of organization and re-organization to compute accurate topic summaries at various level of similarity.

[1]In the single link clustering algorithm a document is part of a cluster if it is "related" to at least *one* document in the cluster.

[2]In 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.

[3]Text Retrieval Conference

2 Clustering static data with star-shaped subgraphs

In this section we motivate and present an off-line algorithm for organizing information systems. The algorithm is very simple and efficient, and it computes high-density clusters.

We formulate our problem by representing an information system by its *similarity graph*. A similarity graph is an undirected, weighted graph $G = (V, E, w)$ where 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 the cosine metric in the vector space model of the Smart information retrieval system [Sal91, Sal89].

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 the same 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 as points in this space according to their word frequencies. Similar documents map to nearby points. In the vector space model, document similarity is measured by the angle between the corresponding document vectors. The standard in the information retrieval 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 that 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 our information retrieval application 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 [LY94, Zuc93]. While we cannot perform a clique cover nor even approximate such a cover, we can instead cover our graph by *dense subgraphs*. What we lose in intra-cluster similarity guarantees, we gain in computational efficiency. In the sections that follow, we describe off-line and on-line covering algorithms and analyze their performance and efficiency.

2.1 Dense Star-Shaped 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. 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 high, and together with empirical evidence supporting this formula, we shall 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 which 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 $\alpha_1 + \alpha_2$, and therefore we have the following lower bound on the similarity between satellite vertices in a star-shaped subgraph of G_σ.

Proposition 2.1. *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*

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

If $\sigma = 0.7$, $\cos\alpha_1 = 0.75$ and $\cos\alpha_2 = 0.85$, for instance, we can conclude that the similarity between the two satellite vertices must be at least[4]

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

Note that while this may not seem very encouraging, the above analysis is based on absolute worst-case assumptions, and in practice, the similarities between satellite vertices are much higher. We further undertook a study to determine the *expected* similarity between two satellite vertices.

2.2 The random graph model

The model we use for analysis is the *random graph model* [Bol95]. 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 results obtained for random graphs closely match those obtained for thresholded similarity graphs obtained from actual document corpora. As such, we will use the random graph model for analysis and for experimental verification of the algorithms presented in this paper (in addition to experiments on actual corpora).

The following upper bound on the expected similarity between two satellite vertices holds:

Proposition 2.2. *The* expected *similarity between two satellite vertices S_1 and S_2 in the same star in a similarity graph G_σ is:*

$$\cos\alpha_1\cos\alpha_2 + \frac{\sigma}{1+\sigma}\sin\alpha_1\sin\alpha_2.$$

Proof. (Omitted for space considerations.)

Note that for the previous example, the above formula would predict a similarity between satellite

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

vertices of approximately 0.78. We have tested this formula against real data, and the results of the test with the MEDLINE data set[5] are shown in Figure 1. In this plot, the x- and y-axes are similarities between cluster centers and satellite vertices, and the z-axis is the actual mean squared prediction error (MSE) of the above formula for the similarity between satellite vertices. Note that the maximum root mean squared error (RMS) is quite small (approximately 0.13 in the worst case), and for reasonably high similarities, the error is negligible. From our tests with real data, we have concluded that the random graph model holds and that this formula is quite accurate. We can conclude that star-shaped subgraphs are reasonably "dense" in the sense that they imply relatively high pairwise similarities between documents.

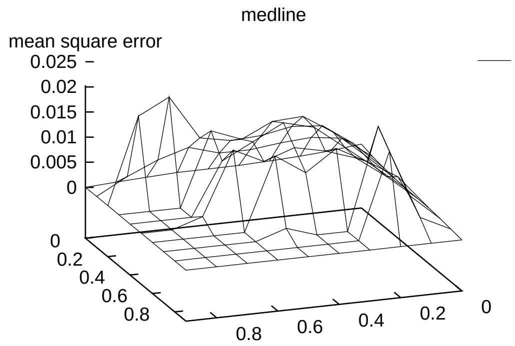

Figure 1: This figure shows the error for a 6000 abstract subset of MEDLINE.

3 The off-line star algorithm

Motivated by the discussion of the previous section, 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 Figure 2 below.

THEOREM 3.1. *The running time of the off-line star algorithm on a similarity graph G_σ is $\Theta(V + E_\sigma)$.*

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 associate 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

For any threshold σ:

1. Let $G_\sigma = (V, E_\sigma)$ where $E_\sigma = \{e : w(e) \geq \sigma\}$.
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 cluster.
5. Repeat step 4 until all nodes are marked.
6. Represent each cluster by the document corresponding to its associated star center.

Figure 2: The star algorithm

$v.center$, which is a bit denoting whether the vertex is a star center. (Computing $v.degree$ for each vertex can easily be 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.

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 higher degree. It immediately follows that:

[5] MEDLINE is a large collection of medical abstracts that is often used as benchmark in information retrieval experiments.

THEOREM 3.2. *The off-line star algorithm produces a correct star cover.*

We will use the two features of the off-line algorithm mentioned above in the analysis of the on-line version of the star algorithm, in the next section.

4 Clustering dynamic data with the star algorithm

In this section we consider algorithms for computing the organization of a dynamic information system. We derive an on-line version of the star algorithm for information organization that can incrementally compute clusters of similar documents. We continue assuming the vector space model and the cosine metric to capture the pairwise similarity between the documents of the corpus, and the random graph model for analyzing the expected behavior of the new algorithm.

We assume that documents are inserted or deleted from the collection one at a time. For simplicity, we will focus our discussion on adding documents to the collection. The delete algorithm is similar. The intuition behind the incremental computation of the star cover of a graph after a new vertex is inserted is depicted in Figure 3. 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 on 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 higher, the new vertex becomes a satellite vertex of c. The difficult case that destroys the correctness of the star cover is when the new vertex is adjacent to a collection of star centers, each of whose degree is lower than that of the new vertex. In this situation, the star structure already in place has to be modified to assign the new vertex as a star center. The satellite vertices in the stars that are broken as a result have to be re-evaluated.

4.1 The on-line star algorithm

Motivated by the intuition in the previous section, we now describe an on-line algorithm for incrementally computing star covers of dynamic graphs. The algorithm is shown in Figure 4. This 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.

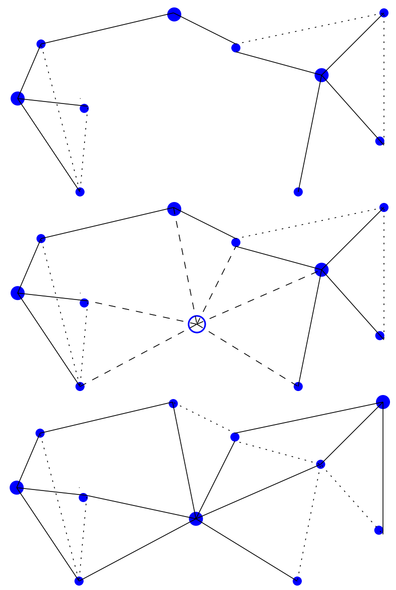

Figure 3: This figure shows the star cover change after the insertion of a new vertex. The larger-radius disks denote star centers, the other disks denote satellite vertices. The star edges are denoted by solid lines. The inter-satellite edges are denoted by dotted lines. The top figure shows an initial 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.

$v.type$	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 have all five pieces of data in the algorithm. Let α be a vertex to be added to G, and let L be the list of vertices in G which are adjacent to α. The algorithm in Figure 4 will appropriately update the star cover of G.

The algorithm maintains a priority queue Q of vertices not yet correctly placed in the star cover. When a star is broken, its center and satellites are placed in Q.

The on-line star cover algorithm is more complex than its off-line counterpart. We devote the rest of this section to proving that the algorithm is correct and to analyzing its expected running time.

```
    UPDATE(α, L)
 1  α.type ← satellite
 2  α.degree ← 0
 3  α.adj ← ∅
 4  α.centers ← ∅
 5  forall β in L
 6     α.degree ← α.degree + 1
 7     β.degree ← β.degree + 1
 8     INSERT(β, α.adj)
 9     INSERT(α, β.adj)
10     if (β.type = center)
11        INSERT(β, α.centers)
12     else
13        β.inQ ← true
14        ENQUEUE(β, Q)
15     endif
16  endfor
17  α.inQ ← true
18  ENQUEUE(α, Q)
19  while (Q ≠ ∅)
20     φ ← EXTRACTMAX(Q)
21     if (φ.centers = ∅)
22        φ.type ← center
23        forall β in φ.adj
24           INSERT(φ, β.centers)
25        endfor
26     else
27        if (∀δ ∈ φ.centers, δ.degree < φ.degree)
28           φ.type ← center
29           forall β in φ.adj
30              INSERT(φ, β.centers)
31           endfor
32           forall δ in φ.centers
33              δ.type ← satellite
34              forall μ in δ.adj
35                 DELETE(δ, μ.centers)
36                 if (μ.inQ = false)
37                    μ.inQ ← true
38                    ENQUEUE(μ, Q)
39                 endif
40              endfor
41           endfor
42        endif
43     endif
44     φ.inQ ← false
45  endwhile
```

Figure 4: The on-line star algorithm for clustering.

4.2 Correctness

In this section we show that the on-line algorithm is correct by proving that it produces the same star cover as the off-line algorithm, when the off-line algorithm is run on the final graph considered by the on-line algorithm. Before we state the result, we note that the off-line star algorithm does not produce a unique cover. When there are several vertices of the same highest degree, the algorithm arbitrarily chooses one of them as the next star center. We will show that the cover produced by the on-line star algorithm is the same as one of the covers that can be produced by the off-line algorithm

THEOREM 4.1. *The cover generated by the on-line star algorithm when $G_\sigma = (V, E_\sigma)$ is constructed incrementally (by inserting its vertices one at a time) is identical to some legal cover generated by the off-line star algorithm on G_σ.*

Proof. 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 off-line star algorithm assigns correct types to the vertices of G_σ. We will prove the correctness of the on-line star cover by induction. The induction invariant is that at all times, the types of vertices not in Q are correct, *assuming* that the true type of 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.

The invariant is true initially: as the type of the new node α is unknown and α is in Q; the type of all the satellite neighbors of α are unknown and these neighbors are in Q; and all the other vertices have correct types from the original cover, assuming that the nodes in the queue are correctly satellite. We now show that the induction invariant is maintained throughout the algorithm. Consider Figure 4. The first thing to note is that the type of all the vertices in Q is "satellite"; statements 14, 18 and 33 enqueue satellite vertices. We now argue that every time a vertex ϕ of highest degree is pulled out of Q, it is assigned a correct type. When ϕ has no centers on its adjacency list, its type should be "center" (which is assigned correctly by statement 22). When ϕ is adjacent to star centers δ_i, each of which has a lower degree that ϕ, the correct type for ϕ is "center" (statement 28). This action has a side effect: all δ_i cease to be star centers and thus get enqueued for further evaluation (statements 32-39). Otherwise, the correct type for ϕ is the default "satellite". Since ϕ was extracted from Q and all vertices in Q are satellites, the type of ϕ is correct in this case as well.

To complete the argument, what remains to be shown is that eventually the queue Q becomes empty. The termination of the while loop at statement 19 in Figure 4 is guaranteed by the following result.

LEMMA 4.1. *The degree of the stars broken by the on-line star algorithm is strictly decreasing.*

The lemma is equivalent to the following statement: node ϕ in Q has the potential of becoming a star center and has the capability of adding new nodes γ to Q that can become stars of degree strictly less than the degree of node ϕ.

Suppose ϕ becomes a new star center. We show than its satellite neighbors γ cannot become star centers. Two cases arise. (Case 1) γ_i is not a star center because its degree is smaller than the degree of the new star center that covers ϕ in the new cover. (Case 2) γ_i is not a star center because it is a satellite of a much larger star, so its degree is larger than the degree of the new star that covers ϕ. But this condition still holds after making the new star. This completes the proof sketch for the termination lemma and it follows that the types assigned by the on-line algorithm are correct; in other words, that there exists an off-line algorithm that produces the same cover.

4.3 Running Time Analysis and Experimental Results

In this section, we argue that the running time of the on-line star algorithm is efficient in practice, asymptotically matching the running time of the off-line star algorithm ($\Theta(V + E)$) to within lower order factors. We first note, however, that there exist worst-case thresholded similarity graphs G_σ and corresponding vertex insertion sequences which cause the on-line star algorithm to run in $\Theta(V^2)$ time.[6] These graphs and insertion sequences rarely arise in practice though. An analysis more closely modeling practice is the random graph model in which G_σ is a random graph and the insertion sequence is random. In this model, the *expected* running time of the on-line star algorithm can be determined.

In the sections that follow, we first give intuition for the expected running time of the on-line star algorithm. In subsequent sections, we give experimental results showing that the on-line star algorithm is quite efficient with respect to both random data and a large collection of real documents.

4.3.1 Intuition

We have implemented the on-line star algorithm using a heap for the priority queue and simple linked lists for the various lists required. The time required to insert a new vertex and associated edges into a thresholded similarity graph 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 due partly 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 with the former, noting that the number of stars expected to cover a random graph $G_{n,p}$ is only $\Theta(\log n)$.

THEOREM 4.2. *The expected size of the star cover for $G_{n,p}$ is $\frac{\log n}{\log(\frac{1}{1-p})}$.*

Proof. The star cover algorithm is greedy: it iterates by selecting the highest degree vertex not yet covered as a star center and marking this node and all its adjacent vertices as covered. Each iteration creates a new star. We will argue that the number of iterations is $\frac{\log n}{\log(\frac{1}{1-p})}$. The argument relies on the random graph model described in Section 2.1.

Let $G_{n,p}$ correspond to a random graph on n vertices where each edge exists with probability p. The degree of each vertex of G is distributed binomially: $Pr[\deg = k] = bin(k; n-1, p) = \binom{n-1}{k}\ p^k (1-p)^{n-1-k}$. The mean of this distribution is $\mu = (n-1)p$ and its variance is $\sigma = \sqrt{(n-1)p(1-p)}$. Note that while the the degrees of the vertices do exhibit some dependence, for practical purposes they can be considered independent [Bol95]. This means that on average, each star covers $(n-1)p + 1 \geq np$ vertices.[7] Since the np vertices covered by each star are randomly chosen, there will be some overlap between the star covers. Each new star leaves uncovered a $(1-p)$ fraction of the previously uncovered vertices. In other words, after the first iteration, $(1-p)n$ vertices remain uncovered. After i iterations, $(1-p)^i n$ vertices remain uncovered. The algorithm terminates when all the vertices are covered, or $(1-p)^i n < 1$. By taking logs of both sides of this inequality, it follows that $i > \frac{\log n}{\log(\frac{1}{1-p})}$ is sufficient.

Thus, the number of stars is expected to be relatively small. Furthermore, the probability any individual star will be broken is quite small as well. A star can only be broken if the star center has the same degree as one of its associated satellite vertices and if the vertex being added to the graph is connected to that satellite but not to the star center.[8] In practice, the expected number of stars broken during an update is a small constant even for graphs containing thousands of vertices (though asymptotically it is certainly a slowly

[6] An example is a graph consisting of two connected vertices A and B of very high but identical degree (not both of which can be star centers) and an insertion sequence which causes the "local" center to repeatedly switch between A and B.

[7] The star covers its center and $(n-1)p$ satellites.

[8] Once a star is broken during an update, however, other stars can be broken in different ways via a cascading effect.

growing function of n). In Figure 5, we give experimental results showing that the total number of stars broken during runs on two different types of data is roughly a linear function of the number of vertices; thus, the expected number of stars broken during any given update is roughly a constant (or more likely a slowly growing function of n).

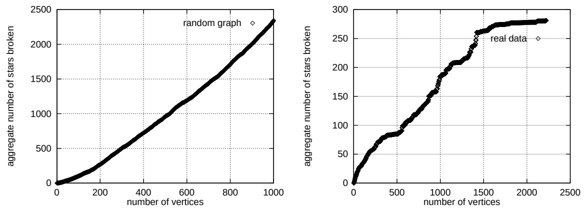

Figure 5: (The dependence of the number of broken stars on the number of vertices in a random graph (left) and for text data (right).

The time to break a star is roughly proportional to its size (the degree of its associated star center), and since the degrees of all vertices are expected to be similar in distribution ($bin(k; n-1, p)$), this is on the order of the number of edges being inserted into the graph. Since only a constant number of stars are expected to be broken, the expected time to perform an update will be roughly proportional to the number of edges inserted in the graph during the update. Thus, the total time to perform n updates should be roughly proportional to the total number of edges in the final graph. In the sections that follow, we give experimental results which confirm this fact.

4.3.2 Experimental results

We have experimented with the on-line clustering algorithm in two scenarios. The first type of data matches our random graph model and consists of random similarity 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 real data from the TREC collection[9] as a second type of benchmark for the algorithm.

We now detail our data generation procedure and the experimental running time of the on-line star algorithm on each data type.

[9] TREC is the annual text retrieval conference. TREC is organized as a competition. Each participant is given on the order of 5 gigabytes of data and a standard set of queries on which to test their systems. The results and the system descriptions are presented as papers at the TREC conference.

Generating Random Data. We ran the on-line star cover algorithm on a random graph with 1000 nodes. The edges in this graph were inserted randomly with probability $p = 0.2$. The on-line algorithm was run 30 times. Each time, the vertices of the random graph were inserted in random order. The results were averaged over the 30 experiments. Figure 6 shows the data from these experiments. Note that the the running time is roughly linear in the number of edges in the graph, and we can see the effects of lower order terms.

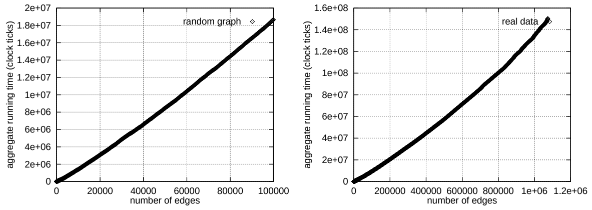

Figure 6: This figure shows the dependence of the running time of the on-line star algorithm on the number of edges in a random graph (left) and for text data (right).

Experiments with real data. We ran the on-line star cover algorithm on a document collection that consists of a slice of TREC documents augmented with our department's technical reports. The resulting collection consists of 2224 documents. We ran four experiments. Each time we set a different threshold and added the similarity graph nodes in random order. The results of these experiments were averaged and the running time measurements appear to be linear in the number of edges of the similarity graph. Figure 6 shows the data from these experiments. Note that the the running time is roughly linear in the number of edges in the graph, and we can see the effects of lower order terms.

Comparion between Star, Single Link, and Average Link on TREC Data. We have implemented a system for organizing information that uses the star algorithm. Figure 7 shows the user interface to this system [APR97]. In order to evaluate the performance of our system, we tested the star algorithm against two widely used clustering algorithms in IR: the single link method [Rij79] and the average link method [Voo85]. We used data from the TREC-6 conference as our testing medium. The TREC collection contains a very large set of documents of which 21,694 have been ascribed relevance data 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 experiment. For each of the 47 topics the given collection of documents was clustered with each of the three algorithms and the best cluster was returned. To measure the quality of a cluster, we use the common E measure [Rij79] defined as: $E(p,r) = 1 - 2/(1/p + 1/r)$, where p and r are the standard *precision* and *recall* of the cluster with respect to the set of documents relevant to the topic.[10] It is worthwhile to note that in viewing data comparing two clustering methods, *lower* $E(p,r)$ values correspond to *better* performance. Averaging over all 22 experiments, we find that the mean $(p, r, E(p,r))$ values for star, average-link and single-link are $(0.77, 0.54, 0.36)$, $(0.83, 0.44, 0.42)$ and $(0.84, 0.41, 0.45)$, respectively. Thus, the star algorithm represents a 16.6% improvement in performance with respect to average-link and an 25% improvement with respect to single-link.

Figure 8 shows the detailed E measures for the star algorithm vs. the single link algorithms and for the star algorithm vs. the average link algorithm. We collected all the topic clusters computed in this experiment. We sorted the clusters produced by the star algorithm according to their E-values. We plotted the E value for the coresponding cluster computed by the single link algorithm (see the oscillating line in Figure 8–left) and for the average link algorithm (see the oscillating line in Figure 8–right). We note that the star algorithm outperforms both the single link algorithm and the average link algorithm, because the E values for the star clusters are almost everywhere lower than the corresponding values for the other two algorithms. Note that not all topics are present in all 22 experiments, which is why we have only approximately 500 clusters in these graphs.

Our experiments show that in general, the star algorithm outperforms single link by 25% and that it outperforms average link by 16.6%. We repeated this experiment on the same data, using one collection only (of 21,694 documents), and obtained similar results.[11] These improvements are significant for text applications. Considering that the star algorithm outperforms the average link algorithm, it is easier to implement than the average link algorithm, it can be used as an on-line algorithm, and it runs much faster, these experiments provide support for using the star algorithm in clustering and off-line information organization.

[10] Precision is the fraction of returned documents that are correct. Recall is the fraction correct documents that are returned.

[11] The precision, recall, and E values for star, average link, and single link were (.53, .32, .61), (.63, .25, .64), and (.66, .20, .70), respectively. We note that the E measures are worse for all three algorithms on this larger collection and that the star algorithm outperforms average link by 10.3% and single link by 20.7%.

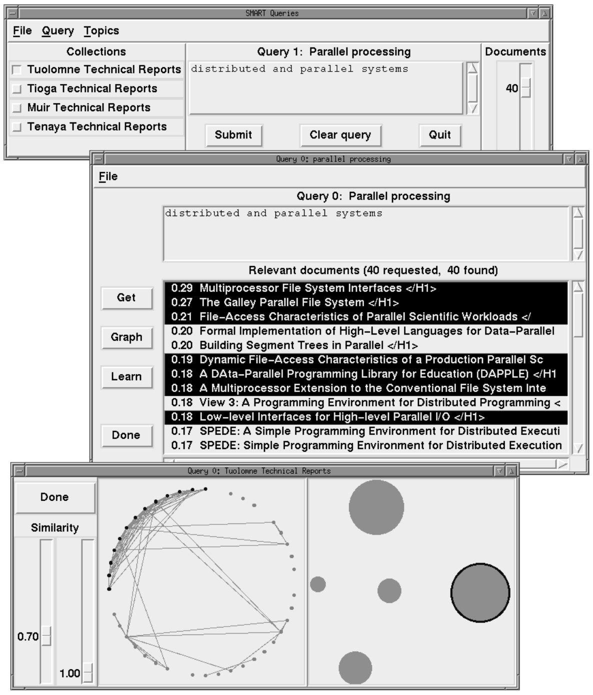

Figure 7: This is a screen snapshot from a clustering experiment. The top window is the query window. The middle window consists of a ranked list of documents that were retrieved in response to the user query. The user my select "get" to fetch a document or "graph" to request a graphical visualization of the clusters as in the bottom window. The left graph displays all the documents as dots around a circle. Clusters are separated by gaps. The edges denote pairs of documents whose similarity falls between the slider parameters. The right graph displays all the clusters as disks. The radius of a disk is proportional to the size of the cluster. The distance between the disks is proportional to the similarity distance between the clusters.

5 Discussion

We presented and analyzed an off-line clustering algorithm for static information organization and an on-line clustering algorithm for dynamic information organization. We discussed the random graph model for analyzing these algorithms and showed that in this model, the algorithms have expected running time that is roughly linear in the number of edges. The data we gathered from experimenting with these algorithms provides support for the validity of our model and analysis. The empirical tests show that both algorithms have an asymptotic linear time performance in the number of edges in the graph. In addition, both algorithms are simple and easy to implement. We believe that the fast running

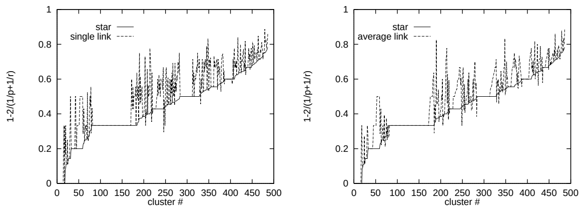

Figure 8: This figure shows the E measure for the star clustering algorithm vs. the single link clustering algorithm (left) and the star algorithm vs. the average link algorithm (right). The y axis shows the E measure. The x axis shows the cluster number. Clusters have been sorted according to the E value for the star algorithm.

time and the ease of implementation make these algorithms very practical candidates for use in automatically organizing digital libraries.

This work departs from previous clustering algorithms used in information retrieval that use a fixed number of clusters for partitioning the 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 [HP96] that support using clustering for browsing applications and presents positive evidence for the cluster hypothesis. In [APR97], 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 high precision and high recall. Precision-recall are the standard measurements for the performance of an information retrieval algorithm [Sal89].

References

[AB84] M. Aldenderfer and R. Blashfield, *Cluster Analysis*, Sage, Beverly Hills, 1984.

[APR97] J. Aslam, K. Pelekhov, and D. Rus, Generating, visualizing, and evaluating high-accuracy clusters for information organization, in *Principles of Digital Document Processing*, eds. C. Nicholas, LNCS Springer Verlag 1998.

[Bol95] B. Bollobás, *Random Graphs*, Academic Press, London, 1995.

[Can93] F. Can, Incremental clustering for dynamic information processing, in *ACM Transactions on Information Systems*, no. 11, pp143-164, 1993.

[CCFM97] M. Charikar, C. Chekuri, T. Feder, and R. Motwani, Incremental clustering and dynamic information retrieval, in *Proceedings of the* 29^{th} *Symposium on Theory of Computing*, 1997.

[Cro80] W. B. Croft. A model of cluster searching based on classification. *Information Systems*, 5:189-195, 1980.

[Cro77] W. B. Croft. Clustering large files of documents using the single-link method. *Journal of the American Society for Information Science*, pp189-195, November 1977.

[CKP93] D. Cutting, D. Karger, and J. Pedersen. Constant interaction-time scatter/gather browsing of very large document collections. In *Proceedings of the* 16^{th} *SIGIR*, 1993.

[FG88] T. Feder and D. Greene, Optimal algorithms for approximate clustering, in *Proceedings of the* 20^{th} *Symposium on Theory of Computing*, pp 434-444, 1988.

[HP96] M. Hearst and J. Pedersen. Reexamining the cluster hypothesis: Scatter/Gather on Retrieval Results. In *Proceedings of the* 19^{th} *SIGIR*, 1996.

[HS86] D. Hochbaum and D. Shmoys, A unified approach to approximation algorithms for bottleneck problems, *Journal of the ACM*, no. 33, pp533-550, 1986.

[JD88] A. Jain and R. Dubes. *Algorithms for Clustering Data*, Prentice Hall 1988.

[JR71] N. Jardine and C.J. van Rijsbergen. The use of hierarchical clustering in information retrieval, 7:217-240, 1971.

[KP93] G. Kortsarz and D. Peleg. On choosing a dense subgraph. In *Proceedings of the 34th Annual Symposium on Foundations of Computer Science (FOCS)*, 1993.

[LY94] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. *Journal of the ACM* 41, 960–981, 1994.

[Rij79] C.J. van Rijsbergen. *Information Retrieval.* Butterworths, London, 1979.

[Sal89] G. Salton. *Automatic Text Processing: the transformation, analysis, and retrieval of information by computer*, Addison-Wesley, 1989.

[Sal91] G. Salton. The Smart document retrieval project. In *Proceedings of the Fourteenth Annual International ACM/SIGIR Conference on Research and Development in Information Retrieval*, pages 356-358.

[Tur90] H. Turtle. Inference networks for document retrieval. PhD thesis. University of Massachusetts, Amherst, 1990.

[Voo85] E. Voorhees. The cluster hypothesis revisited. In *Proceedings of the* 8^{th} *SIGIR*, pp 95-104, 1985.

[Wil88] P. Willett. Recent trends in hierarchical document clustering: A critical review. *Information Processing and Management*, 24:(5):577-597, 1988.

[Wor71] S. Worona. Query clustering in a large document space. In Ed. G. Salton, *The SMART Retrieval System*, pp 298-310. Prentice-Hall, 1971.

[Zuc93] D. Zuckerman. NP-complete problems have a version that's hard to approximate. In *Proceedings of the Eight Annual Structure in Complexity Theory Conference*, IEEE Computer Society, 305–312, 1993.