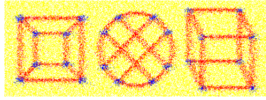

Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 8, no. 1, pp. 95–129 (2004)

The Star Clustering Algorithm for Static and Dynamic Information Organization

Javed A. Aslam

College of Computer and Information Science
Northeastern University
http://www.ccs.neu.edu/home/jaa
jaa@ccs.neu.edu

Ekaterina Pelekhov

Department of Computer Science
Dartmouth College
http://www.cs.dartmouth.edu/~katya
ekaterina.pelekhov@alum.dartmouth.org

Daniela Rus

Department of Computer Science and CSAIL
Dartmouth College and MIT
http://www.cs.dartmouth.edu/~rus
rus@csail.mit.edu

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 quality 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.

Article Type	Communicated by	Submitted	Revised
regular paper	S. Khuller	December 2003	August 2004

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

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, probability, 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 [32], Smart [31], 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 its applications to many domains [18, 2]. For a good overview see [19]. For a good overview of using clustering in Information Retrieval (IR) see [34]. The use of clustering in IR was mostly driven by *the cluster hypothesis* [28] which states that "closely associated documents tend to be related to the same requests". Jardine and van Rijsbergen [20] show some evidence that search results could be improved by clustering. Hearst and Pedersen [17] re-examine the cluster hypothesis by focusing on the Scatter/Gather system [14] and conclude that it holds for browsing tasks.

Systems like Scatter/Gather [14] 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. [10] 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 [4] 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 [14, 10] 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 [30] which is explained in more detail in Section 2 of this paper.

To compute accurate clusters, we formalize clustering as covering graphs by cliques [21] (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 even be approximated in poly-

nomial time [25, 36]. 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] [13] and the average link algorithm[2] [33]. 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] collection 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 and irrelevant topics. 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. The on-line algorithm can be used as a basis for constructing self-organizing information systems. As the content of a dynamic information system changes, the on-line algorithm can efficiently automate the process of organization and reorganization 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] TREC is the annual text retrieval conference. 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.

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 a standard metric from the IR community—the cosine metric in the vector space model of the Smart information retrieval system [31, 30].

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 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 [25, 36]. 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.

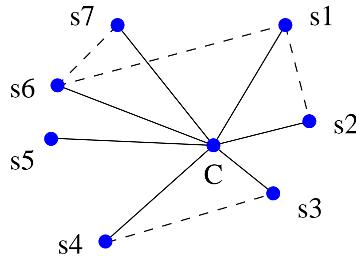


Figure 1: An example of a star-shaped subgraph with a center vertex C and satellite vertices s_1 through s_7. The edges are denoted by solid and dashed lines. Note that there is an edge between each satellite and a center, and that edges may also exist between satellite vertices.

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 (see Figure 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 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 $\alpha_1 + \alpha_2$; we therefore have the following lower bound on the similarity between satellite vertices in a star-shaped subgraph of G_σ.

Fact 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 determined the *expected* similarity between two satellite vertices.

2.2 Expected satellite similarity in the vector space model

In this section, we derive a formula for the expected similarity between two satellite vertices given the geometric constraints of the vector space model, and we give empirical evidence that this formula is accurate in practice.

Theorem 2.2 *Let C be a star center, and let S_1 and S_2 be 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 angle[5] *between the planes formed by S_1C and S_2C.*

Proof: Let C be a unit vector corresponding to a star center, and let S_1 and S_2 be unit vectors corresponding to satellites in the same star. Let $\alpha_1 = \angle S_1C$, $\alpha_2 = \angle S_2C$ and $\gamma = \angle S_1S_2$ be the pairwise angles between vectors. Let θ, $0 \le \theta \le \pi$, be the dihedral angle between the planes formed by S_1C and S_2C. We seek a formula for $\cos\gamma$.

First, we observe that θ is related to the angle between the vectors normal to the planes formed by S_1C and S_2C.

$$\pi - \theta = \angle(S_1 \times C)(C \times S_2)$$

Consider the dot product of these normal vectors.

$$(S_1 \times C)\cdot(C \times S_2) = \|S_1 \times C\|\ \|C \times S_2\|\ \cos(\pi - \theta) = -\cos\theta \sin\alpha_1 \sin\alpha_2$$

On the other hand, standard results from geometry dictate the following.

$$(S_1 \times C)\cdot(C \times S_2) = (S_1 \cdot C)(C \cdot S_2) - (S_1 \cdot S_2)(C \cdot C) = \cos\alpha_1 \cos\alpha_2 - \cos\gamma$$

Combining these two equalities, we obtain the result in question. □

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

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

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

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

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

Note that for the example given in the previous section, Equation 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 set[6] are shown in Figure 2. In this plot, the x- and y-axes 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 2.2 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 Equation 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.

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 3 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

[6]FBIS is a large collection of text documents used in TREC.

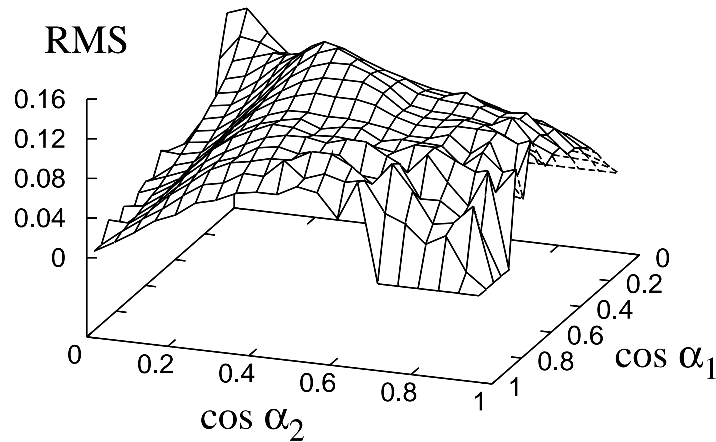
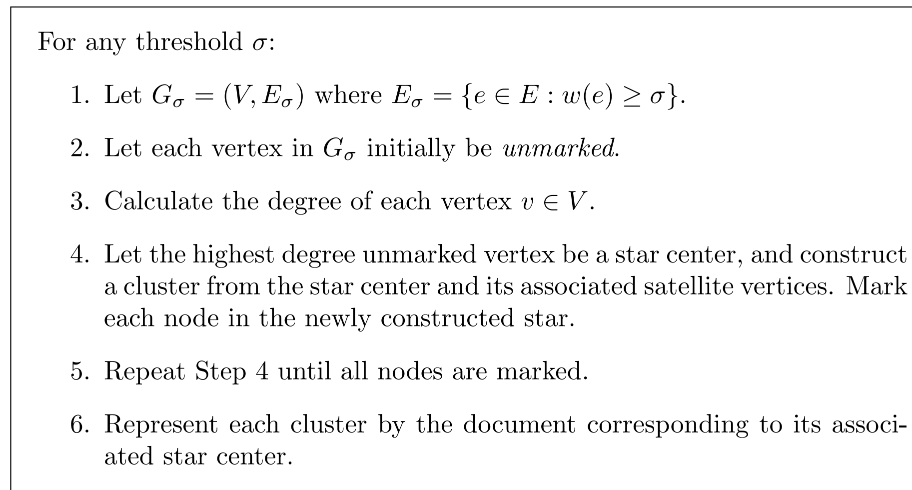


Figure 2: The RMS prediction error of our expected satellite similarity formula over the TREC FBIS collection containing 21,694 documents.

or not, and *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 equal or higher degree.



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.

Figure 3: The star algorithm

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 non-clique vertex it is connected to), and single-node stars for each of the non-covered 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 can not 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 studies extensively in the literature [16, 1]. The star cover is related but not identical to a dominating set [16]. Every star cover is a dominating set, but there are dominating sets that are not star covers. 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 showed above.

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

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. In a subsequent section, we will show that the clusters produced by the star algorithm are quite accurate, exceeding the accuracy produced by widely used clustering algorithms in information retrieval.

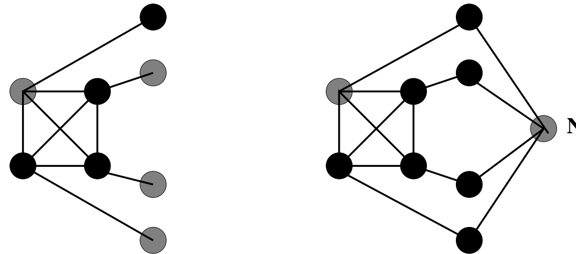


Figure 4: An example of a star-shaped covers before and after the insertion of the node N in the graph. The dark circles denote satellite vertices. The shaded circles denote star centers.

4 The On-line Star Algorithm

In this section we consider algorithms for computing the organization of a dynamic information system. We consider a document collection where new documents arrive incrementally over time, and they need to be inserted in the collection. Existing documents can become obsolete and thus need to be removed. We derive an on-line version of the star algorithm for information organization that can incrementally compute clusters of similar documents, supporting both insertion and deletion. We continue assuming the vector space model and its associated cosine metric for capturing the pairwise similarity between the documents of the corpus as well as 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. 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 Figure 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 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 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.

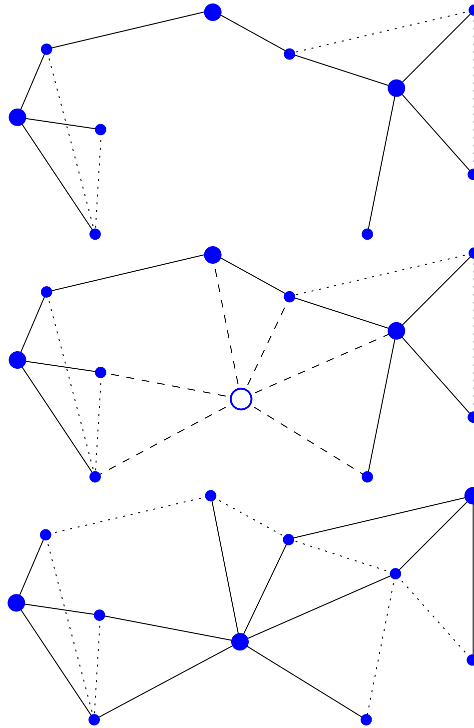


Figure 5: The star cover change after the insertion of a new vertex. The larger-radius disks denote star centers, the other disks denote satellite vertices. The star edges are denoted by solid lines. The inter-satellite edges are denoted by dotted lines. The top figure shows an initial graph and its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

4.1 The on-line algorithm

Motivated by the intuition in the previous section, we now describe a simple on-line algorithm for incrementally computing star covers of dynamic graphs; a more optimized version of this algorithm is given in Appendix A. 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.

```
INSERT(α, L, G_σ)
 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  UPDATE(G_σ)
```

Figure 6: Pseudocode for Insert.

```
DELETE(α, G_σ)
 1  forall β in α.adj
 2     β.degree ← β.degree − 1
 3     DELETE(α, β.adj)
 4  endfor
 5  if (α.type = satellite)
 6     forall β in α.centers
 7        forall μ in β.adj
 8           if (μ.inQ = false)
 9              μ.inQ ← true
10              ENQUEUE(μ, Q)
11           endif
12        endfor
13     endfor
14  else
15     forall β in α.adj
16        DELETE(α, β.centers)
17        β.inQ ← true
18        ENQUEUE(β, Q)
19     endfor
20  endif
21  UPDATE(G_σ)
```

Figure 7: Pseudocode for Delete.

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 on-line 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 on-line 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 next describe in some detail the three routines which comprise the on-line star algorithm.

```
UPDATE(G_σ)
 1  while (Q ≠ ∅)
 2    φ ← EXTRACTMAX(Q)
 3    if (φ.centers = ∅)
 4      φ.type ← center
 5      forall β in φ.adj
 6        INSERT(φ, β.centers)
 7      endfor
 8    else
 9      if (∀δ ∈ φ.centers, δ.degree < φ.degree)
10        φ.type ← center
11        forall β in φ.adj
12          INSERT(φ, β.centers)
13        endfor
14        forall δ in φ.centers
15          δ.type ← satellite
16          forall μ in δ.adj
17            DELETE(δ, μ.centers)
18            if (μ.degree ≤ δ.degree ∧ μ.inQ = false)
19              μ.inQ ← true
20              ENQUEUE(μ, Q)
21            endif
22          endfor
23        endfor
24        φ.centers ← ∅
25      endif
26    endif
27    φ.inQ ← false
28  endwhile
```

Figure 8: Pseudocode for Update.

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 enqueuing 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 17–18) and its adjacent satellite vertices (lines 13–14).

The DELETE algorithm presented in Figure 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 Figure 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.

The on-line star cover algorithm is more complex than its off-line counterpart. We devote the next two sections to proving that the algorithm is correct and to analyzing its expected running time. A more optimized version of the on-line algorithm is given and analyzed in the appendix.

4.2 Correctness of the on-line algorithm

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 unmarked 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 or deleting 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 algorithm 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.

The invariant is true for the INSERT procedure: the correct type of the new node α is unknown, and α is in Q; the correct types of all adjacent satellites of α are unknown, and these satellites are in Q; all other vertices have correct types from the original star cover, assuming that the nodes in Q are correctly satellite. DELETE places the satellites of all affected centers into the queue. The correct types of these satellites are unknown, but all other vertices have correct types from the original star cover, assuming that the vertices in Q are properly satellite. Thus, the invariant is true for DELETE as well.

We now show that the induction invariant is maintained throughout the UPDATE procedure; consider the pseudocode given in Figure 8. First note that the assigned type of all the vertices in Q is "satellite;" lines 14 and 18 in INSERT, lines 10 and 18 in DELETE, and line 20 in UPDATE enqueue satellite vertices. We now argue that every time a vertex ϕ of highest degree is extracted from Q, it is assigned a correct type. When ϕ has no centers in its adjacency list, its type should be "center" (line 4). When ϕ is adjacent to star centers δ_i, each of which has a strictly smaller degree that ϕ, the correct type for ϕ is "center" (line 10). This action has a side effect: all δ_i cease to be star centers, and thus their satellites must be enqueued for further evaluation (lines 14–23). (Note that if the center δ is adjacent to a satellite μ of greater degree, then μ must be adjacent to another center whose degree is equal to or greater than its own. Thus, breaking the star associated with δ cannot lead to the promotion of μ, so it need not be enqueued.) Otherwise, ϕ is adjacent to some center of equal or higher degree, and a correct type for ϕ is the default "satellite."

To complete the argument, we need only show that the UPDATE procedure eventually terminates. In the analysis of the expected running time of the UPDATE procedure (given in the next section), it is proven that no vertex can be enqueued more than once. Thus, the UPDATE procedure is guaranteed to terminate after at most V iterations. □

4.3 Expected running time of the on-line algorithm

In this section, we argue that the running time of the on-line star algorithm is quite efficient, asymptotically matching the running time of the off-line star algorithm within logarithmic factors. We first note, however, that there exist worst-case thresholded similarity graphs and corresponding vertex insertion/deletion sequences which cause the on-line 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 [7] in which G_σ is a random graph and the insertion/deletion sequence is random. In this model, the *expected* running time of the on-line star algorithm can be determined. In the remainder of this section, we argue that the on-line star algorithm is quite efficient theoretically. In subsequent sections, we provide empirical results which 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* [7]. 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 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 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 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.

Theorem 4.2 *The expected size of the star cover for $G_{n,p}$ is at most $1 + 2\log(n)/\log(\frac{1}{1-p})$.*

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 will argue that the number of iterations is at most $1 + 2\log(n)/\log(\frac{1}{1-p})$ 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 above.

Consider the (weak) algorithm described above which repeatedly selects stars at random from $G_{n,p}$. After i stars have been created, each of the i star centers will be marked, and some number of the $n-i$ remaining vertices will be marked. For any given non-center vertex, the probability of being adjacent to any given center vertex is p. The probability that a given non-center vertex remains unmarked is therefore $(1-p)^i$, and thus its probability of being marked is $1-(1-p)^i$. The probability that *all* $n-i$ non-center vertices are marked is then $\left(1-(1-p)^i\right)^{n-i}$. 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

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

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

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

we then have the following.

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

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

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

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

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

□

We next note the following facts about the Update procedure given in Figure 8, which repeatedly extracts vertices ϕ from a priority queue Q (line 2). First, the vertices enqueued within the Update procedure (line 20) must be of degree strictly less than the current extracted vertex ϕ (line 2). This is so because each enqueued vertex μ must satisfy the following set of inequalities

$$\mu.degree \leq \delta.degree < \phi.degree$$

where the first and second inequalities are dictated by lines 18 and 9, respectively. This implies that the degrees of the vertices ϕ extracted from the priority queue Q must monotonically decrease; ϕ is the current vertex of highest degree in Q, and any vertex μ added to Q must have strictly smaller degree. This further implies that no vertex can be enqueued more than once. Once a vertex v is

enqueued, it cannot be enqueued again while v is still present in the queue due to the test of the inQ flag (line 18). Once v is extracted, it cannot be enqueued again since all vertices enqueued after v is extracted must have degrees strictly less than v. Thus, no more than $|V|$ vertices can ever be enqueued in Q.

Second, any star created within the UPDATE procedure cannot be destroyed within the UPDATE procedure. This is so because any star δ broken within the UPDATE procedure must have degree strictly less than the current extracted vertex ϕ (line 9). Thus, any star created within the UPDATE procedure (lines 4 and 10) cannot be subsequently broken since the degrees of extracted vertices monotonically decrease.

Combining the above facts with Theorem 4.2, we have the following.

Theorem 4.3 *The expected time required to insert or delete a vertex in a random graph* $G_{n,p}$ *is* $O(np^2 \log^2(n)/\log^2(\frac{1}{1-p}))$*, for any* $0 \leq p \leq 1 - \Theta(1)$.

Proof: For simplicity of analysis, we assume that n is large enough so that all quantities which are random variables are on the order of their respective expectations.

The running time of insertion or deletion is dominated by the running time of the UPDATE procedure. We account for the work performed in each line of the UPDATE procedure as follows. Each vertex ever present in the queue must be enqueued once (line 20 or within INSERT/DELETE) and extracted once (lines 2 and 27). Since at most $n+1$ (INSERT) or $n-1$ (DELETE) vertices are ever enqueued, we can perform this work in $O(n \log n)$ time total by implementing the queue with any standard heap. All centers adjacent to any extracted vertex must also be examined (lines 3 and 9). Since the expected size of a *centers* list is p times the number of stars, $O(p \log(n)/\log(\frac{1}{1-p}))$, we can perform this work in $O(np \log(n)/\log(\frac{1}{1-p}))$ expected time.

For each star created (lines 4–7, 10–13, and 24), we must process the newly created star center and satellite vertices. The expected size of a newly created star is $\Theta(np)$. Implementing *centers* as a standard linked list, we can perform this processing in $\Theta(np)$ expected time. Since no star created within the UPDATE procedure is ever destroyed within the UPDATE procedure and since the expected number of stars is $O(\log(n)/\log(\frac{1}{1-p}))$, the total expected time to process all created stars is $O(np \log(n)/\log(\frac{1}{1-p}))$.

For each star destroyed (lines 14–19), we must process the star center and its satellite vertices. The expected size of a star is $\Theta(np)$, and the expected size of a *centers* list is p times the number of stars; hence, $O(p \log(n)/\log(\frac{1}{1-p}))$. Thus, the expected time required to process a star to be destroyed is $O(np^2 \log(n)/\log(\frac{1}{1-p}))$. Since no star created within the UPDATE procedure is ever destroyed within the UPDATE procedure and since the expected number of stars is $O(\log(n)/\log(\frac{1}{1-p}))$, the total expected time to process all destroyed stars is $O(np^2 \log^2(n)/\log^2(\frac{1}{1-p}))$.

Note that for any p bounded away from 1 by a constant, the largest of these terms is $O(np^2 \log^2(n)/\log^2(\frac{1}{1-p}))$. $\square$

The thresholded similarity graphs obtained in a typical IR setting are almost always dense: there exist many vertices comprised of relatively few (but dense) clusters. We obtain dense random graphs when p is a constant. For dense graphs, we have the following corollary.

Corollary 4.4 *The total expected time to insert n vertices into (an initially empty) dense random graph is* $O(n^2 \log^2 n)$.

Corollary 4.5 *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 on-line insertion result for dense graphs compares favorably to the off-line 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 the next section) verify this result.

For sparse graphs ($p = \Theta(1/n)$), the analogous results are asymptotically 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 above analysis of the UPDATE procedure. Empirical results on sparse random graphs (detailed in the next section) verify this fact and imply that the total running time of the on-line insertion algorithm is also proportional to the size of the input graph, $\Theta(n)$, within lower order factors.

4.4 Efficiency experiments

We have conducted efficiency experiments with the on-line 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.

4.4.1 Aggregate number of broken stars

The efficiency of the on-line 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 Figure 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.

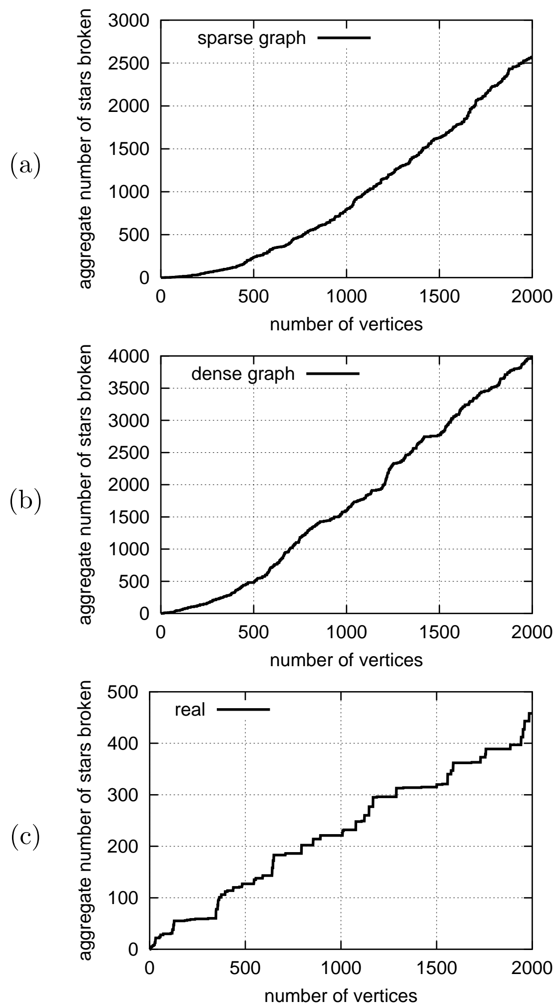


Figure 9: The dependence of the number of broken stars on the number of inserted vertices in (a) a sparse random graph, (b) a dense random graph, and (c) the graph corresponding to TREC FBIS data.

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.

4.4.2 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 Figure 10.

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 Figure 10 suggest a running time for the on-line algorithm which is linear in the size of the input graph, though lower order factors are presumably present.

4.5 Cluster accuracy experiments

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 [28] and the average link method [33]. 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 which "best" approximated the set of judged relevant documents was returned. To measure the quality of a cluster, we use the standard F measure from Information Retrieval [28],

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

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

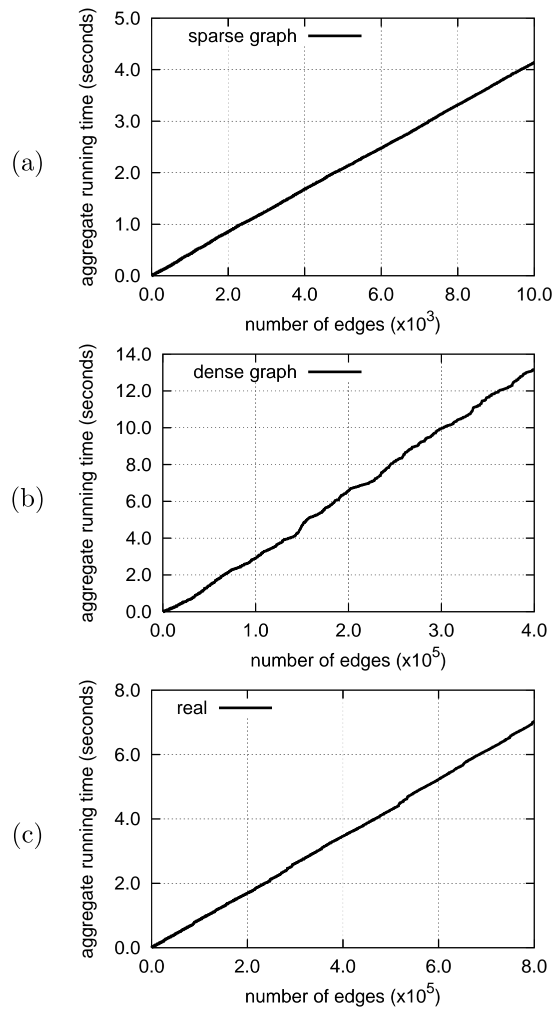


Figure 10: The dependence of the running time of the on-line star algorithm on the size of the input graph for (a) a sparse random graph, (b) a dense random graph, and (c) the graph corresponding to TREC FBIS data.

$(p, r, F(p, r))$ 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, respectively, $(.77, .54, .63)$, $(.83, .44, .57)$ and $(.84, .41, .55)$. 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.

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 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 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 $(.53, .32, .42)$, $(.63, .25, .36)$, and $(.66, .20, .30)$, 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 Information Retrieval applications. Given that (1) the star algorithm outperforms the average-link algorithm, (2) it can be used as an on-line algorithm, (3) it is relatively simple to implement in either of its off-line or on-line forms, and (4) it is efficient, these experiments provide support for using the star algorithm for off-line and on-line information organization.

5 A System for Information Organization

We have implemented a system for organizing information that uses the star algorithm. Figure 12 shows the user interface to this system.

This organization system (that is the basis for the experiments described in this paper) consists of an augmented version of the Smart system [31, 3], 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 user interface is implemented in Tcl/Tk.

The organization system can be run on a whole collection, on a specified

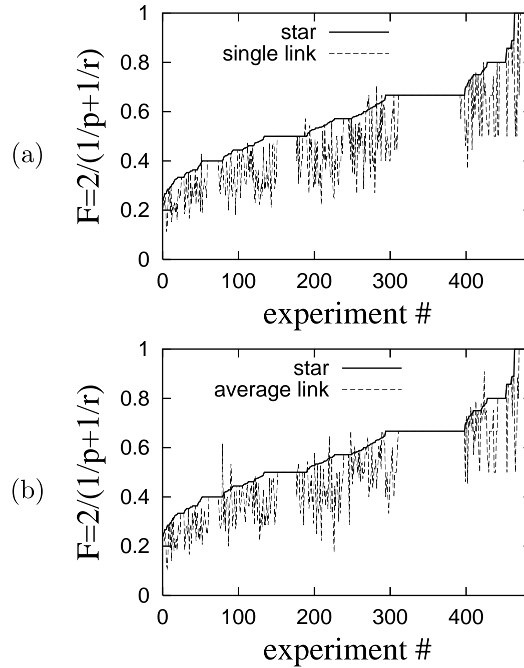


Figure 11: The F measure for (a) the star clustering algorithm vs. the single link clustering algorithm and (b) the star algorithm vs. the average link algorithm (right). The y axis shows the F measure. The x axis shows the experiment number. Experimental results have been sorted according to the F value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

subcollection, or on the collection of documents retrieved in response to a user query. Users can input queries by entering free text. They have the choice of specifying several corpora. This system supports distributed information retrieval, but in this paper we do not focus on this feature, and we assume only one centrally located corpus. In response to a user query, Smart is invoked to produce a ranked list of the most relevant documents, their titles, locations and document-to-document similarity information. The similarity information for the entire collection, or for the collection computed by the query engine is provided as input to the star algorithm. This algorithm returns a list of clusters and marks their centers.

5.1 Visualization

We developed a visualization method for organized data that presents users with three views of the data (see Figure 12): a list of text titles, a graph that shows the similarity relationship between the documents, and a graph that shows the

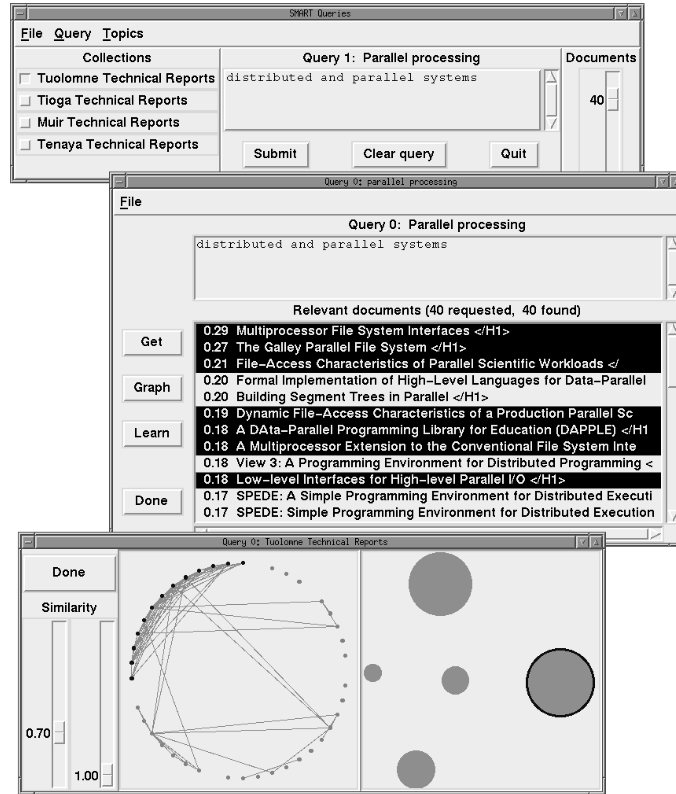


Figure 12: This is a screen snapshot from a clustering experiment. The top window is the query window. The middle window consists of a ranked list of documents that were retrieved in response to the user query. The user may select "get" to fetch a document or "graph" to request a graphical visualization 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.

similarity relationship between the clusters. These views provide users with summaries of the data at different levels of detail (text, document and topic) and facilitate browsing by topic structure.

The connected graph view (inspired by [3]) has nodes corresponding to the retrieved documents. The nodes are placed in a circle, with nodes corresponding to the same cluster placed together. Gaps between the nodes allow us to identify clusters easily. Edges between nodes are color coded according to the similarity between the documents. Two slider bars allow the user to establish minimal and maximal weight of edges to be shown.

Another view presents clusters as solid disks whose diameters are proportional to the sizes of the corresponding clusters. The Euclidean distance between the centers of two disks is meant to capture the topic separation between the corresponding clusters. Ideally, the distance between two disks would be proportional to the dissimilarity between the corresponding clusters C_1 and C_2; in other words, $1-sim(c(C_1), c(C_2))$ where $c(C_i)$ is the star center associated with cluster C_i. However, such an arrangement of disks may not be possible in only two dimensions, so an arrangement which *approximately* preserves distance relationships is required. The problem of finding such "distance preserving" arrangements arises in many fields, including data analysis (*e.g.*, multidimensional scaling [23, 9]) and computational biology (*e.g.*, distance geometry [11]). We employ techniques from distance geometry [11] which principally rely on eigenvalue decompositions of distance matrices, for which efficient algorithms are easily found [26].

All three views and a title window allow the user to select an individual document or a cluster. Selections made in one window are simultaneously reflected in the others. For example, the user may select the largest cluster (as is shown in the figure) which causes the corresponding documents to be highlighted in the other views. This user interface facilitates browsing by topic structure.

6 Conclusion

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 an expected running time that is linear in the size of the input graph (within logarithmic factors). The data we gathered from experimenting with these algorithms provides 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 (within lower order factors), and that they 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 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 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 [17] that support using clustering for browsing applications and presents positive evidence for the cluster hypothesis. In [4], 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.

The on-line star algorithm described in Section 4 can be optimized somewhat for efficiency, and such an optimization is given in the appendix. Both the off-line and on-line star algorithms can be further optimized in their use of similarity matrices. Similarity matrices can be very large for real document corpora, and the cost of computing the similarity matrix can be much more expensive than the basic cost of either the off-line or on-line star algorithms. At least two possible methods can be employed to eliminate this bottleneck to overall efficiency. One method would be to employ *random sampling of the vertices*. A random sample consisting of $\Theta(\sqrt{n})$ vertices could be chosen, the similarity matrix for these vertices computed, and the star cover created. Note that the size of the computed similarity matrix is $\Theta(n)$, linear in the size of the input problem. Further note that the expected size of the star cover on n vertices is $O(\log(n))$, and thus the size of the star cover on the $\Theta(\sqrt{n})$ sampled vertices is expected to be only a constant factor smaller. Thus, the star cover on the $\Theta(\sqrt{n})$ sampled vertices may very well contain many of the clusters from the full cover (though the clusters may be somewhat different, of course). Once the sampled vertices are clustered, the remaining vertices may be efficiently added to this clustering by comparing to the current cluster centers (simply add a vertex to a current cluster if its similarity to the cluster center is above the chosen threshold). Another approach applicable to the on-line algorithm would be to *infer* the vertices adjacent to a newly inserted vertex in the thresholded similarity graph while only minimally referring to the similarity matrix. For each vertex to be inserted, compute its similarity to each of the current star centers. The expected similarity of this vertex to any of the satellites of a given star center can then be inferred from Equation 1. Satellites of centers "distant" from the vertex to be inserted will likely not be adjacent in the thresholded similarity graph and can be assumed non-adjacent; satellites of centers "near" the vertex to be inserted will likely be adjacent in the thresholded similarity graph and can be assumed so. We are currently analyzing, implementing and testing algorithms based on these optimizations.

We are currently pursuing several extensions for this work. We are developing a faster algorithm for computing the star cover using sampling. We are also considering algorithms for computing star covers over distributed collections.

Acknowledgements

The authors would like to thank Ken Yasuhara who implemented the optimized version of the on-line star algorithm found in the appendix.

References

[1] N. Alon, B. Awerbuch, Y. Azar, N. Buchbinder and J. Naor. The Online Set Cover Problem. In *Proceedings of the thirty-fifth annual ACM Symposium on Theory of Computing*, pp 100–105, San Diego, CA, 2003.

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

[3] J. Allan. *Automatic hypertext construction*. PhD thesis. Department of Computer Science, Cornell University, January 1995.

[4] J. Aslam, K. Pelekhov, and D. Rus, Generating, visualizing, and evaluating high-accuracy clusters for information organization, in *Principles of Digital Document Processing*, eds. E. Munson, C. Nicholas, D. Wood, Lecture Notes in Computer Science 1481, Springer Verlag 1998.

[5] J. Aslam, K. Pelekhov, and D. Rus, Static and Dynamic Information Organization with Star Clusters. In *Proceedings of the 1998 Conference on Information Knowledge Management*, Bethesda, MD, 1998.

[6] J. Aslam, K. Pelekhov, and D. Rus, A Practical Clustering Algorithm for Static and Dynamic Information Organization. In *Proceedings of the 1999 Symposium on Discrete Algorithms*, Baltimore, MD, 1999.

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

[8] F. Can, Incremental clustering for dynamic information processing, in *ACM Transactions on Information Systems*, no. 11, pp 143–164, 1993.

[9] J. Carroll and P. Arabie. Multidimensional Scaling. *Ann. Rev. Psych.*, vol. 31, pp 607–649, 1980.

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

[11] G. Crippen and T. Havel. *Distance Geometry and Molecular Conformation.* John Wiley & Sons Inc., 1988.

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

[13] W. B. Croft. Clustering large files of documents using the single-link method. *Journal of the American Society for Information Science*, pp 189-195, November 1977.

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

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

[16] M. Garey and D. Johnson. *Computers and Intractability: a Guide to the Theory of NP-Completness.* W. H. Freeman and Company, New York. 1979.

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

[18] D. Hochbaum and D. Shmoys, A unified approach to approximation algorithms for bottleneck problems, *Journal of the ACM*, no. 33, pp 533-550, 1986.

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

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

[21] R. Karp. Reducibility among combinatorial problems. *Computer Computations*, pp 85–104, Plenum Press, NY, 1972.

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

[23] J. Kruskal and M. Wish. *Multidimensional scaling.* Sage Publications, Beverly Hills, CA, 1978.

[24] N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica* 15(2):215-245, 1995.

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

[26] W. Press, B. Flannery, S. Teukolsky, and W. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, 1988.

[27] W. Pugh. SkipLists: a probabilistic alternative to balanced trees. in *Communications of the ACM*, vol. 33, no. 6, pp 668-676, 1990.

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

[29] D. Rus, R. Gray, and D. Kotz. Transportable Information Agents. *Journal of Intelligent Information Systems*, vol 9. pp 215-238, 1997.

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

[31] 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.

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

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

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

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

[36] 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.

A The Optimized On-line Star Algorithm

A careful consideration of the on-line star algorithm presented earlier in this paper reveals a few possible optimizations. One of the parameters that regulates the running time of our algorithm is the size of the priority queue. Considering the UPDATE procedure that appears in Figure 8, we may observe that most satellites of a destroyed star are added to the queue, but few of those satellites are promoted to centers. The ability to predict the future status of a satellite can save queuing operations.

Let us maintain for each satellite vertex a *dominant center*; *i.e.*, an adjacent center of highest degree. Let us maintain for each center vertex a list of *dominated satellites*; *i.e.*, a list of those satellite vertices for which the center in question is the dominant center. Line 9 of the original UPDATE procedure, which determines whether a satellite should be promoted to a center, is equivalent to checking if the degree of the satellite is greater than the degree of its dominant center. In lines 16–22 of the original UPDATE procedure, we note that only satellites dominated by the destroyed center need to be enqueued. A list of dominated satellites maintained for each center vertex will help to eliminate unnecessary operations.

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
$v.domsats$	satellites dominated by this vertex
$v.domcenter$	dominant center
$v.inQ$	flag specifying if v being processed

In our implementation, lists of adjacent vertices and adjacent centers are SkipList data structures [27], which support INSERT and DELETE operations in expected $\Theta(\log n)$ time. We will also define a MAX operation which simply finds the vertex of highest degree in a list. This operation runs in time linear in the size of a list; however, we will use it only on lists of adjacent centers, which are expected to be relatively small.

The dominant center of a satellite is the adjacent center of highest degree. The list of dominated satellites is implemented as a heap keyed by vertex degree. The heap supports the standard operations INSERT, DELETE, MAX and EXTRACTMAX; the ADJUST function maintains the heap's internal structure in response to changes in vertex degree.

The procedures for INSERT, DELETE and UPDATE presented here are structurally similar to the versions described earlier in this paper. A few changes need to be made to incorporate new pieces of data.

The INSERT procedure in Figure 13 adds a vertex α with a list of adjacent vertices L to a thresholded similarity graph G_σ and updates the star cover of G_σ.

Lines 1–7 of the algorithm initialize various data fields of α. Lines 10–14

update the degrees and adjacency lists of α and its adjacent vertices β. Lists of dominated satellites that contain β are maintained in lines 13–15 to reflect the change in degree of β. Lines 16–18 build the list of centers adjacent to α.

Any satellite vertex adjacent to α may potentially be placed into Q. However, as discussed earlier in this section, only satellites that have a degree higher than the degree of their dominant centers need to be enqueued (lines 19–22). Similarly, α is enqueued only if it has no adjacent centers lines 25–27, or if it has a greater degree than its dominant center (lines 31–34). The Update procedure called in line 36 computes a correct star cover.

The Delete procedure in Figure 14 removes vertex α from the appropriate data structures and modifies the star cover. Lines 1–7 remove α from adjacency lists (line 3), modify degrees of affected vertices (line 2), and maintain correct lists of dominated satellites (lines 4–6). A different course of action should be taken upon removing a satellite vertex (lines 8–24) or a center vertex (lines 25–36).

A satellite vertex should be removed from the dominated satellite list of its dominant center (lines 10–12). The degrees of the centers adjacent to α have decreased, leading to a possible conflict between a center and its dominated satellites. The dominated satellites that have degrees greater than the degrees of their dominant centers are located in lines 13–23 of the code, removed from dominated satellites lists (line 16) and enqueued (lines 18–21).

If a center vertex is being removed, it is deleted from the lists of centers of its adjacent vertices (lines 26–28), and its dominated satellites are enqueued (lines 29–35). The Update procedure is then called to recompute the star cover.

The Update procedure is given in Figure 15. Vertices are organized in a priority queue, and a vertex ϕ of highest degree is considered in each iteration (line 2). The algorithm assigns an appropriate type to the vertex and updates the graph data structures appropriately.

If ϕ has no adjacent centers (lines 3–8), it becomes a center and is included in the centers lists of its adjacent vertices (lines 6–8). Otherwise, a dominant center of ϕ is found and assigned to λ (line 10). If λ is a true dominant center (*i.e.*, the degree of λ is greater than the degree of ϕ), we set the dominant center of ϕ to λ (lines 12–16). Otherwise (lines 17–40), the degree of ϕ is greater than the degrees of all its adjacent centers, and thus ϕ is promoted to a center. The centers adjacent to *phi* are destroyed (lines 23–38), and the satellites dominated by these stars are enqueued (lines 30–36). The cycle is repeated until the queue is empty.

```
Insert(α, L, G_σ)
 1  α.type ← satellite
 2  α.degree ← 0
 3  α.adj ← ∅
 4  α.centers ← ∅
 5  α.domcenter ← NIL
 6  α.domsats ← ∅
 7  α.inQ ← false
 8  forall β in L
 9     α.degree ← α.degree + 1
10     β.degree ← β.degree + 1
11     Insert(β, α.adj)
12     Insert(α, β.adj)
13     if (β.domcenter ≠ NIL)
14        Adjust(β, β.domcenter.domsats)
15     endif
16     if (β.type = center)
17        Insert(β, α.centers)
18     else
19        if (β.degree > β.domcenter.degree)
20              β.inQ ← true
21              Enqueue(β, Q)
22        endif
23     endif
24  endfor
25  if (α.centers = ∅)
26     α.inQ ← true
27     Enqueue(α, Q)
28  else
29     α.domcenter ← Max(α.centers)
30     Insert(α, α.domcenter.domsats)
31     if (α.degree > α.domcenter.degree)
32        α.inQ ← true
33        Enqueue(α, Q)
34     endif
35  endif
36  Update(G_σ)
```

Figure 13: The details of the Insert operation for the optimized on-line star algorithm.

```
Delete(α, G_σ)
 1  forall β in α.adj
 2     β.degree ← β.degree − 1
 3     Delete(α, β.adj)
 4     if (β.domcenter ≠ NIL)
 5        Adjust(β, β.domcenter.domsats)
 6     endif
 7  endfor
 8  if (α.type = satellite)
 9     forall β in α.centers
10        if (β = α.domcenter)
11           Delete(α, β.domsats)
12        endif
13        γ ← Max(β.domsats)
14        while (β.domsats ≠ ∅ and
15                 γ.degree > β.degree)
16           ExtractMax(β.domsats)
17           γ.domcenter ← NIL
18           if (γ.inQ = false)
19              γ.inQ ← true
20              Enqueue(γ, Q)
21           endif
22           γ ← Max(β.domsats)
23        endwhile
24     endfor
25  else
26     forall β in α.adj
27        Delete(α, β.centers)
28     endfor
29     forall ν in α.domsats
30        ν.domcenter ← NIL
31        if (ν.inQ = false)
32           ν.inQ ← true
33           Enqueue(ν, Q)
34        endif
35     endfor
36  endif
37  Update(G_σ)
```

Figure 14: The details of the Delete operation for the optimized on-line star algorithm.

```
Update(G_σ)
 1  while (Q ≠ ∅)
 2     φ ← ExtractMax(Q)
 3     if (φ.centers = ∅)
 4        φ.type ← center
 5        φ.domcenter ← NIL
 6        forall β in φ.adj
 7           Insert(φ, β.centers)
 8        endfor
 9     else
10        λ ← Max(φ.centers)
11        if (λ.degree ≥ φ.degree)
12           if (φ.domcenter ≠ NIL)
13              Delete(φ, φ.domcenter.domsats)
14           endif
15           φ.domcenter ← λ
16           Insert(φ, λ.domsats)
17        else
18           φ.type ← center
19           φ.domcenter ← NIL
20           forall β in φ.adj
21              Insert(φ, β.centers)
22           endfor
23           forall δ in φ.centers
24              δ.type ← satellite
25              δ.domcenter ← φ
26              Insert(δ, φ.domsats)
27              forall μ in δ.adj
28                 Delete(δ, μ.centers)
29              endfor
30              forall ν in δ.domsats
31                 ν.domcenter ← NIL
32                 if (ν.inQ = false)
33                    ν.inQ ← true
34                    Enqueue(ν, Q)
35                 endif
36              endfor
37              δ.domsats ← ∅
38           endfor
39           φ.centers ← ∅
40        endif
41     endif
42     φ.inQ ← false
43  endwhile
```

Figure 15: The details of the Update operation for the optimized on-line star algorithm.