

Clustering with Lower Bound on Similarity [★]

Mohammad Al Hasan¹, Saeed Salem¹,
Benjarath Pupacdi², and Mohammed J. Zaki¹

¹ Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY

² Chulabhorn Research Institute, Laksi, Bangkok, Thailand

Abstract. We propose a new method, called **SimClus**, for clustering with lower bound on similarity. Instead of accepting k the number of clusters to find, the alternative similarity-based approach imposes a lower bound on the similarity between an object and its corresponding cluster representative (with one representative per cluster). **SimClus** achieves a $O(\log n)$ approximation bound on the number of clusters, whereas for the best previous algorithm the bound can be as poor as $O(n)$. Experiments on real and synthetic datasets show that our algorithm produces more than 40% fewer representative objects, yet offers the same or better clustering quality. We also propose a dynamic variant of the algorithm, which can be effectively used in an on-line setting.

1 Introduction

In many application domains that involve clustering, it is difficult to guess the number of clusters. It is even more challenging in dynamic domains like news-groups and blogosphere clustering, where the number of topics is typically unknown and may even change.

An alternative to the parameter k that defines the number of clusters is to provide a lower bound, β that defines the desired (minimum) similarity between a cluster member and a representative object in that cluster. This similarity-based formulation has several benefits. Firstly, the lower bound β automatically imposes a similarity bound among the members of a cluster. Secondly, it supports overlapping clusters in a very natural way. Thirdly, this paradigm can easily be adapted to work in a dynamic setting, where new objects are added in or existing objects are removed.

The optimization problem of clustering with a lower bound on similarity is in fact NP-Hard (as we show below). We thus propose a greedy solution, named **SimClus**, that achieves a $O(\log n)$ approximation bound (where n is the size of the object-set). We also show that the approximation bound for the existing best algorithm, *star clustering* [2], is as worse as $O(n)$. We then propose a variant of **SimClus** that is suitable for clustering in a dynamic setting. We experiment with different synthetic and real-world datasets like random graphs and newsgroup

[★] This work was supported in part by NSF Grants EMT-0829835, and CNS-0103708, and NIH Grant 1R01EB0080161-01A1.

documents. In all experiments, **SimClus** achieves similar or better clustering performance in comparison to Star clustering algorithm with more than 40% fewer representatives. It also outperforms traditional clustering algorithms (k -Medoids, hierarchical clustering) in terms of cluster quality and execution time.

2 Background

Consider a set of objects, \mathcal{O} , and a similarity function, $sim : \mathcal{O} \times \mathcal{O} \rightarrow [0, 1]$ such that for any $x \in \mathcal{O} : sim(x, x) = 1$. Assume that the objective is to cluster the objects in \mathcal{O} such that the objects in a cluster are at least β -similar for a user defined $\beta \in [0, 1]$ (with minimum number of clusters). This formulation leads to an interesting graph problem, given as follows: Let $G(V, E)$ be a graph whose vertices are the objects, and an edge $e(u, v) \in E$ implies that the similarity between vertices u and v is at least β . Below we shall refer to it as the (β) *similarity graph*. Now, any clique in this graph can be taken as one cluster in some clustering, since the distances between the elements in a clique would satisfy the required pair-wise similarity constraints. The clustering objective then becomes to cover the entire graph G by a minimum number of cliques. However, this formulation is difficult to solve; in fact, it leads to an NP-Complete problem, named covering a graph by cliques, which can not be approximated in polynomial time [11].

A relaxation of the above problem can be obtained which requires the similarity bound (β) to hold only between the cluster elements and a fixed center object belonging to that cluster. The center object is the representative for the corresponding cluster. Thus if a cluster has m elements, out of all $\binom{m}{2}$ similarities, $m - 1$ are guaranteed to be greater than or equal to β . Nevertheless, all the m objects in the cluster would be sufficiently similar due to triangular inequality or other forms of transitive bounds. We call this relaxed formulation the **lower bound similarity clustering (LBSC)** problem, and it is the main focus of this paper. LBSC seeks exactly one cluster center (representative object) for every cluster. Thus a center object c together with all the objects s such that $sim(c, s) \geq \beta$, form a cluster. All objects s that satisfy the above inequality are called β -similar objects with respect to the object c . If s is β -similar to multiple centers, it belongs to multiple clusters. Thus this model naturally supports overlapping clusters. Since every object s belongs to at least one cluster, s is β -similar to at least one representative object, say c . In that case, we say that c *covers* s . Thus, the clustering objective is to cover all the objects with the smallest number of center objects. A center always covers itself. Unfortunately, the optimization task of LBSC is also NP-Hard, since the corresponding decision problem is NP-Complete.

Lemma 1. *For a given collection of objects \mathcal{O} , and a user-defined similarity threshold β , to determine whether there exists a set of representative objects, $\mathcal{C} \subseteq \mathcal{O}$, of size k is NP-Complete.*

PROOF: Given graph $G = (V, E)$, the vertex dominating set problem, a known

NP-Complete problem [7], asks whether there exists a subset $C \subseteq V$ (called dominating set), of size at most k , such that every vertex in $V \setminus C$ is adjacent to at least one element in C . If G is the β -similarity graph then each center set C corresponds to a dominating set, since every object in \mathcal{O} is either in the center set or is covered by at least one cluster center. ■

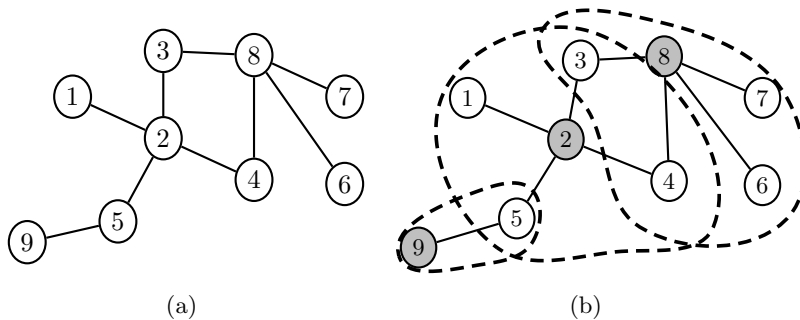


Fig. 1. Lower bound similarity clustering. (a) β -similar graph. (b) overlapping clusters

The above relation to the dominating set problem suggests a graph-based formulation. From $\binom{n}{2}$ similarity values for a set of n objects, we can first construct a β -similarity graph. Then, in this graph we need to find a vertex dominating set, which would constitute the representative center set. Figure 1(a) shows an example β -similarity graph. Figure 1(b) shows a LBSC of these objects with 3 clusters (dotted closed curves, with gray circles as center objects).

2.1 Related Work

LBSC did not get much attention in the past. Star clustering [2] is the leading algorithm in this paradigm (we discuss it in detail below). [6] highlighted some of the problems of the Star algorithm and proposed an alternative solution to improve it, however they did not provide any approximation bounds. Furthermore, they considered only the static scenario. In terms of dynamic clustering, [3] propose an online version of the popular EM-based approach. [1] solves it by using an online component and an off-line component, whereas [5] proposed an online variant of different mixture model clustering for text data. One may consider the streaming model as dynamic clustering, however, this model imposes very strict requirements [4] on available memory and the number of passes over the data. Hence, the approximation quality is generally poor.

Star Clustering: Aslam et. al. [2] proposed a greedy solution to LBSC called *Star clustering*. It sorts the vertices in the descending order of degree; it then selects the first vertex in the sorted order as one of the cluster centers. Any other vertex covered by this one is deleted from the sorted list and the procedure is repeated until all the vertices are covered. Provided the similarity matrix is given, the computational complexity of this method is $O(|V| \lg |V|)$. Figure 1(b)

shows an example of Star clustering. The greedy solution of the Star algorithm is obviously not optimal. Moreover, the approximation bound can be very bad.

Lemma 2. *The greedy Star clustering algorithm generates an independent set.*

PROOF: Once it selects an object as a center-node, it removes all adjacent (covered) nodes. In later iterations they never appear in the dominating set, thus it is an independent set. ■

The independent set restriction is the main reason why Star clustering yields unnecessarily many star-centers. In Figure 2, we show an example that illustrates the poor bound of the Star clustering algorithm. Instead of choosing $\{2, 8\}$ as center-nodes (in gray), Star algorithm chooses $\{1, 3, 4, 5, 8\}$, to respect the independence set condition. This example can be generalized to obtain a

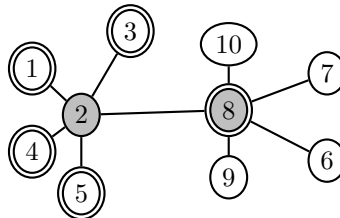


Fig. 2. Star clustering: poor bound

bound in an asymptotic sense. Assume, we have n vertices arranged in two star-shapes (as in Figure 2), each with $\frac{n}{2} - 1$ satellites and one star-center. Assume further that the two star-centers are adjacent to each other. In this case, Star clustering would return $\frac{n}{2}$ centers, whereas the optimal LBSC would have only 2 centers; thus the approximation ratio can be as bad as $O(n)$.

3 SimClus: Lower Bound Similarity Clustering

We now describe our set cover formulation for LBSC, named **SimClus**. It is also greedy in nature, but with a better $O(\log n)$ approximation bound. It thus produces fewer cluster centers, and better representative objects for each cluster.

Lemma 3. *LBSC can be reduced to a set cover problem.*

PROOF: Let $G = (V, E)$ be the β -similarity graph. For each vertex $u \in V$, we construct a new set $s_u = \{u\} \cup \{v : (u, v) \in E\}$. Let $\mathcal{S} = \{s_u : u \in V\}$. Obviously, $|\mathcal{S}| = |V|$ and there exists a one-to-one correspondence between the elements in \mathcal{S} and the vertices in V . The set cover problem is to find the smallest set of elements $\mathcal{T} \subseteq \mathcal{S}$, that covers all the vertices, i.e., $\bigcup_{u \in \mathcal{T}} s_u = V$. If for each $s_x \in \mathcal{T}$, we insert the corresponding vertex $x \in V$ in the center-set \mathcal{C} , then \mathcal{C} is also an optimal solution for the LBSC problem, since the minimality of $|\mathcal{T}|$ ensures the minimality of $|\mathcal{C}|$. ■

The set-cover problem is NP-Complete, but it can be solved using an efficient greedy algorithm that achieves $O(\log n)$ approximation bound [10]. The greedy heuristic iteratively chooses the set that covers the largest number of uncovered elements until all the elements are covered. Using the same heuristic for LBSC then yields the $O(\log n)$ approximation bound with much fewer representatives. For the dominating set problem, the hardness of approximation result is also available, that proves that unless $P = NP$, no polynomial time algorithm can

have better than $O(\log n)$ approximation [9]. So, it is highly unlikely that a better bound than $\log n$ can be achieved for LBSC.

The main difference between Star clustering and SimClus is that instead of choosing the uncovered object with the highest degree (as done in Star), SimClus chooses the object that can cover the most uncovered elements. This drastically reduces the number of clusters; the clusters are more dense and hence, more informative.

SimClus follows the reduction shown in Lemma 3. At the beginning, all the objects are not covered and the center-set is empty. We then map each vertex, u of LBSC similarity graph to a set s_u that contains itself and all objects that it can cover. To facilitate the greedy heuristic, the set $s_u, \forall u \in V$ contains only those objects that are uncovered. So, we call the set s_u as the *uncovered cover-set* of u , which intuitively means that it holds those objects that are uncovered and can be covered by choosing u as a center. Hence, once an object is chosen as a center, the *uncovered cover-set* of all the non-center objects are updated by removing any object that is covered by the newly chosen center. In every iteration a new center is selected using the above greedy criterion and the process is repeated until all the objects are covered. If there is a tie in the above criterion, we break the tie by selecting the object that has the largest degree in the similarity graph. If there is still a tie, we break it arbitrarily.

Example: Consider the similarity graph in Figure 2. The uncovered cover-set of vertex 1 is $\{1, 2\}$, and for vertex 2 it is $\{1, 2, 3, 4, 5, 8\}$. At the beginning, the uncovered cover-set of both the vertices 2 and 8 have the same size. They also have equal degrees. So, we arbitrarily choose, say, vertex 2 as a center. After updating the uncovered cover-set for all the objects, we have empty uncovered cover-set for 1, 3, 4 and 5 and only one object for 6, 7, 9 and 10. But, the uncovered cover-set for 8 contains $\{6, 7, 9, 10\}$. So, 8 is selected as the next center. At this point all the objects are covered, and the algorithm terminates by returning $\{2, 8\}$ as representatives.

Complexity Analysis: SimClus computes the entire similarity matrix to find the β -similarity graph. So, the overall complexity of the algorithm is at least $O(|V|^2)$. If we consider that the similarity matrix is given, then the overall worst-case complexity can be computed in terms of the number of the centers obtained. Assume, that the final clustering has k centers. In each iteration, the best center can be obtained in $O(\lg |V|)$ time by using a `max_heap` data structure. The cover-set update affects only those vertices that are adjacent to the newly covered objects. For the sake of worst case analysis, if we assume that to be $O(|V|)$, the worst case complexity of the algorithm is $O(k|V|)$.

4 Dynamic SimClus: Dynamic Similarity Clustering

The static algorithm that is provided in the previous section requires that the entire β -similarity graph is available before the algorithm is applied. However, in many practical application scenarios, this requirement does not hold. For information retrieval, new documents can be added or old documents may be

deleted from the repositories and the clustering may need to be updated. One option is to re-cluster the objects by running the static clustering algorithm for every change in the collection. But, for most of the cases, the changes are only local; so re-computing the entire clustering is wasteful. Also note that re-computation dramatically changes the cluster-centers; so if the objective of LBSC is to find representative objects in a dynamic setting, one may prefer the dynamic algorithm over the static algorithm, since the former retains a large portion of the representative set. It is also useful in adopting a lazy update scheme for a static version. Since, the dynamic version is generally worse in terms of number of clusters compared to the static version (the $O(\log n)$ approximation guarantee may not hold), a static update may follow after a batch of dynamic updates to re-optimize the clustering.

Dynamic SimClus: In a dynamic environment, we assume that we have an initial solution for the lower bound similarity clustering. New requests for insertion or deletion of an object come in an arbitrary manner. We need to satisfy the requests efficiently while maintaining the minimality of the representative set as much as possible. We are allowed to change the status of an object in either direction (a center can be made non-center and vice-versa). It is worth pointing that this model is not the same as the *online model* that is studied in theoretical computer science. In the online model, the decision that is made based on present information cannot be altered later when further information is available at future. So in a pure online setting, if a node is made representative, its status cannot be changed later. Nevertheless, the online version of LBSC is very difficult; in fact its *competitive ratio* is as worse as $O(n - 1)$ [8].

Since the dynamic model assumes that a valid solution to the lower bound similarity clustering exists, addition of new objects cannot be made based on the greedy criteria of the static algorithm, as only one object (the new object) is uncovered and all vertices adjacent to it have exactly the same size (one) for the *uncovered cover-set*. So, we propose to obtain cluster-centers in such way that the following three conditions are satisfied.

1. Every object is either a cluster-center or is adjacent to at least one cluster center.
2. A cluster center with *degree* > 0 must be adjacent to at least one non-center object of smaller or equal degree.
3. No cluster-center is redundant, i.e. every cluster-center covers at least one object exclusively.

Note that the first of the above conditions is from the definition of LBSC. However, the second and the third conditions are chosen for LBSC to have a reasonably good solution in a dynamic setting. In the experimental section, we shall show that the above dynamic algorithm generates less centers in comparison to both static and dynamic Star clustering algorithm.

Inserting a new object: A center is called *illegal* if it does not satisfy condition (2) above, i.e. all its adjacent vertices have degree strictly higher than it. A center is called *redundant*, if its removal does not change (increase) the number of nodes

that need to be covered. The dynamic SimClus accepts as input the similarity graph and the *covered* vector, which stores the current coverage of each existing vertex. Once a new object v is added, the adjacency list of similarity graph G is updated. If v is adjacent to any existing center, it is properly reflected in the *covered* vector. Now, we have two different cases to consider.

In the first case, when v is covered, condition 1 is already satisfied. Then, we check condition 2 (illegal center) for all the centers that are adjacent to v . Since addition of new vertices can change the adjacency list of some of the existing vertices, this may change a legal center into an illegal one. If this check succeeds (some illegal neighboring center is found), we make the illegal center as non-center and v becomes a center. This step is called recursively on success. In case the above step fails (v is not a center yet), we test whether v should be a center as it has strictly higher degree than any of its neighboring centers. Note that this step is not essential for correctness according to our conditions, but this heuristic has potential to generate better centers.

For the second case, when v is not covered, then none of its adjacent vertices is a center and to fulfill the coverage requirements, at least one of these vertices should be a center. We first test whether it should be some $x \in adj(v)$ by checking whether it has any illegal neighbors. If not, then we choose the vertex with the highest degree as a center.

When a node is made a center, some auxiliary updates are required. First, redundancy check is required for all other centers that are adjacent to it to satisfy the condition 3. Moreover, some adjacent non-centers can also become a center as one of its neighboring center becomes illegal. Figure 3 shows an example of insertion, where we explain the steps of the insertion algorithm.

Deleting an existing object: Deletion is comparably easier. Like insertion, we first update the adjacency and covered vector as necessary. Then, we consider the following two cases. The first is when the deleted node v was a center. In that case, we need to check if any of its adjacent vertex becomes isolated. All of those become centers. For the remaining, if they are still covered, we return immediately. Assume x is not covered, then we cover x by making the highest degree vertex (among itself and its neighbors) a center. For the second case, when v was not a center, its removal does not violate condition (1), so we check for condition (2) and (3) by calling the same methods as in insert routine.

Complexity Analysis: Complexity of dynamic SimClus is difficult to analysis, since it depends on the degree of the vertices that are adjacent to the inserted (or deleted) vertex. For the sake of average-case analysis, we assume Erdos-Renyi random graph model, where the (expected) degree of a node is equal to $d = p \cdot |V|$; p is the probability of an edge to exist. The redundancy checking of a node takes $O(d)$ time as it just reads the *covered* array of its adjacent nodes. Similarly, the illegal center condition checking of one node also takes $O(d)$ time, as it reads the degree of the adjacent nodes. So, considering the above check for all the nodes adjacent to the inserted (deleted) node, the total cost is $O(d^2)$. But, this check is done recursively on success. If the recursion is successful for I times, the total

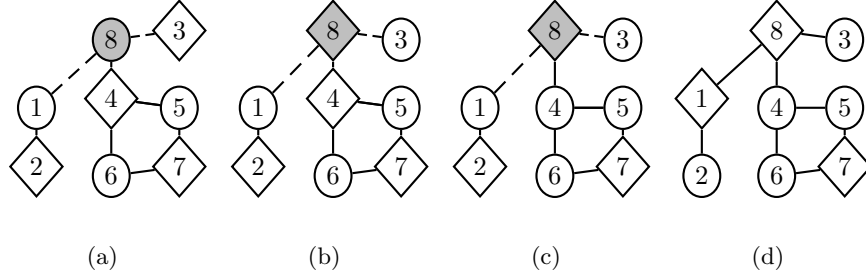


Fig. 3. Insertion example; a diamond shape indicates a center. (a) after inserting node 8, node 3 is illegal center. (b) node 3 becomes a center; now, node 4, a center-neighbor of 8 is redundant. (c) redundant center removed; a non-center neighbor of 8, namely 1, has an illegal center 2 as neighbor. (d) Final result; an off-line algorithm will produce identical result.

cost is $O(Id^2)$. But, generally the value of I is very close to $O(1)$ as we show next.

From the definition, an illegal center has the smallest degree among all its neighbors. Assume u is a node, $deg(u)$ is its degree and $adj(u)$ is its adjacency list. The probability that u is an illegal node is given as:

$$\begin{aligned}
 &= \sum_{i=1}^{n-2} P[\forall v \in adj(u) : deg(v) \geq i + 1 \mid deg(u) = i] \\
 &= \sum_{i=1}^{n-2} (P[deg(v) \geq i + 1 \mid v \in adj(u)])^i \cdot P[deg(u) = i] \\
 &= \sum_{i=1}^{n-2} \left[\left(\sum_{k=i+1}^{n-2} \binom{n-2}{k} p^k (1-p)^{n-k-2} \right)^i \cdot \binom{n-1}{i} p^i (1-p)^{n-i-1} \right]
 \end{aligned}$$

The probability that the node u has degree i is equal to $\binom{n-1}{i} p^i (1-p)^{n-i-1}$ and the probability that a node $v \in adj(u)$ has degree greater than i is equal to $\sum_{k=i+1}^{n-2} \binom{n-2}{k} p^k (1-p)^{n-k-2}$. For a node to be illegal, all its neighbors should have greater degree. Since an illegal node can have degree from 1 to $n-2$, the outer sum aggregates these probabilities. If the above probability is p_{repeat} , using geometric distribution, the average value of I is $\frac{1}{1-p_{repeat}}$. For all possible values of p , the value for I is less than 2 (found with direct calculation in Matlab) when $n \geq 10$. For example, the highest value of I from n is 10 is around 1.75 and for $n = 1000$, it is around 1.015 and with larger n , the value diminishes further. Intuitively, this means that for random graphs, every node is generally connected to some vertices with higher and some vertices with lower degree; hence, very few nodes are illegal. For real-life graphs, the constant may be a bit higher. So,

considering I a constant, the complexity of dynamic algorithm for one insertion or deletion is $O(d^2)$.

5 Experiments

The objective of the experiments is to show the following. Firstly, that SimClus returns a smaller number of representatives for LBSC, yet it yields similar quality as Star clustering. Secondly, that the representative objects are of good quality and can generate clustering that is better than the traditional clustering algorithms. Finally, that the LBSC approach is particularly suitable for clustering multi-label dataset.

Besides SimClus and Star, the experiments use two other traditional clustering algorithms: k -Medoids, and hierarchical (UPGMA). For all the algorithms, the document similarity is modeled by using cosine similarity and a similarity matrix is provided as input. For k -Medoids, we use our own implementation which works very similar to k -Means, but in each iteration, when k -Means chooses a new center by averaging the vectors in that cluster, it chooses it by finding the object with the best average similarity to all the objects in that cluster (thus it has a quadratic time complexity). For hierarchical clustering, we use the CLUTO implementation (glaros.dtc.umn.edu/gkhome/views/cluto/). The Star algorithm was implemented as described in [2].

5.1 Synthetic data: Random Graphs

The first experiment considers synthetic data, in the form of random similarity graph of various size and types for both static and dynamic scenario. For the dynamic experiments, we shuffle the vertices of the similarity graphs randomly and insert them in the existing graph (starting from an empty graph) in that order. With a small probability (0.05), we also delete a random vertex after every insertion. For random graph type, we consider the following two models: (1) Erdos-Renyi and (2) Power-law graphs. Table 1 shows the center-set sizes. For both static and dynamic versions, SimClus achieves a smaller number

vertex	edge	Number of clusters			
		SimClus		Star	
		<i>static</i>	<i>dynamic</i>	<i>static</i>	<i>dynamic</i>
Erdos-Renyi random graphs					
1000	5000	144	194	209	209
1000	10000	84	123	126	130
1000	100000	16	22	21	24
10000	50000	1424	1951	2066	2068
10000	100000	840	1227	1323	1311
10000	1000000	147	225	247	245
Power Law graphs					
1000	5000	547	564	579	579
1000	10000	525	539	550	549
1000	100000	496	502	502	503
10000	50000	3285	3498	4015	4024
10000	100000	2734	2928	3340	3340
10000	1000000	1991	2094	2156	2156

Table 1. Performance on random graphs

of clusters in comparison to Star. Specifically, the static version significantly outperforms the static (and dynamic) version of Star. As expected, dynamic

SimClus does not perform as good as the static version, yet it performs better than Star.

5.2 Newsgroup dataset

For a real-life dataset, we chose the Twenty-newsgroup data from the UCI Repository (www.ics.uci.edu/~mllearn). This dataset is interesting for our experiments, as it has a set of documents that have multiple labels. We used the rainbow package (www.cs.cmu.edu/~mccallum/bow) to convert the documents into word vectors and then used the best 100 words (in terms of mutual information gain statistics) as feature vectors. We discarded any document that did not have any of the top 100 words. The final dataset had 16701 documents. Out of those, 16199 had a unique label, 497 documents had 2 labels, and only 5 documents had more than 2 labels. There were 20 different labels in total.

In this experiment, we cluster the documents using different clustering algorithms and compare the clustering performance using supervised performance metrics, like precision, recall and F-measure. For k -Medoids and Hierarchical (UPGMA) we set the k value to be 20 to obtain 20 different clusters; then for every cluster, we use majority voting to label the cluster. For any object with multiple labels, its vote is counted for all its labels. In case of SimClus and Star, we cannot use k , so we cluster the documents by two different similarity thresholds (we used, $\beta = 0.5$ and $\beta = 0.75$). Number of clusters obtained are generally higher than 20 (exact values are shown in the k column). As the similarity threshold increases, the number of clusters also increases. Then we classify each of these clusters as the label of the cluster representative. Note that many representatives (thus many clusters) can have the same class-label; all of those clusters are collectively classified with that label. If the representative object has multiple labels, all the objects adjacent to that representative objects gets multiple labels. An object can also get multiple labels by being adjacent to multiple representatives with different class-labels. While computing precision/recall for multi-label objects, if any of the predicted labels match with any of the actual labels, the object is considered as a true positive.

Table 2 shows the results. Compared to k -Medoids and Hierarchical, both Star and SimClus achieve much better performance using F-score measure, because of the very high recall that these two achieve. The possible reason is that, instead of choosing exactly 20 centers (which

algorithm	parameters		average	average	F-Measure
	β	k	precision	recall	
k -Medoids	-	20	59.80	40.21	48.09
UPGMA	-	20	48.37	37.33	42.14
Star	0.50	125	53.50	85.96	65.95
	0.75	1060	66.83	83.79	74.36
SimClus	0.50	77	51.83	82.38	62.06
	0.75	687	70.00	82.79	75.86

Table 2. Comparison on newsgroup dataset

may not be enough for the dataset, since the cluster boundary is not regular), they sample enough representatives to cover the entire cluster. So, a document has much higher chance to be a neighbor of one of the many centers that matches with its class label. For example, if a document has a label "comp.graphics", out

of, say 500 representatives, roughly $500/20 = 25$ representatives may have a label "comp.graphics". So, the document has a much higher probability to be a neighbor of any of these. Comparing Star and SimClus, their F-values are similar; for smaller β , the former and for larger β , the latter has marginally better F-values. However, SimClus chooses 40% less centers compared to the Star algorithm.

Our third set of experiments compares the ability of SimClus and Star to predict multiple labels. For this, we find the predicted labels of the objects that has more than one actual labels (502 documents qualify). An actual multi-label is considered as recalled if at least two predicted labels matches with two of the actual labels. Table 3 shows the result for three different values of similarity threshold. The multi-recall columns of the corresponding algorithms show their respective recalls in percentage. The recall values for both the algorithms drop with increasing threshold values. The reason is with large threshold β -similarity graphs become more and more sparse, so an object is not connected to many representatives. Thus the ability to predict multiple labels diminishes.

In comparison to Star, **SimClus** performs substantially better than Star. For example, for a β value of 0.60, even with 47% less centers. As β increases, it suffers somewhat because it optimizes the number of centers. In fact, as we investigated we found that Star achieved better multi-recall, by actually selecting many of the multi-labeled nodes as the center objects.

similarity threshold (β)	SimClus		Star	
	center count	multi- recall	center count	multi- recall
0.60	144	76.49	273	48.41
0.70	382	41.63	550	45.80
0.80	1274	28.88	1811	31.47

Table 3. Overlapping clustering performance

We also compare the timing performance of SimClus in comparison to other algorithms. The result is shown in Table 4. The table shows only the execution time (in seconds) of the clustering task; the I/O cost of loading the similarity matrix is excluded for the sake of fair comparison. The timings were obtained using a 2.1 GHz machine with

algorithm	k or β	Size			
		4000	8000	12000	16701
k -Medoids	20	33.36	301.21	1200.12	3100.38
UPGMA	20	6.71	33.48	75.76	216.96
RB	20	25.64	116.68	297.88	677.28
Star (static)	0.50 0.75	0.02 0.02	0.04 0.04	0.06 0.05	0.09 0.08
SimClus (static)	0.50 0.75	5.09 0.50	27.62 2.53	68.68 6.78	197.49 15.22

Table 4. Execution time comparison

2GB RAM with Linux OS. For this experiment, we randomly select documents from the newsgroup dataset to make 3 different smaller-size datasets (4000, 8000, 12000). Besides k -Medoids, and UPGMA we also use another algorithm, RB (from the Cluto software) which is a fast partitional algorithm that uses repeated bisection. For k -Medoids, UPGMA and RB, which require a k value to cluster, $k = 20$ is used as it is the natural number of clusters for the newsgroup dataset. For Star and SimClus, timing for two different similarity thresholds (0.5 and 0.75) are reported. Different thresholds account for different number of edges in the β -similarity graphs. From Table 4, we see that Star is the fastest among all the algorithms as it just needs to sort the vertex-set only once based

on degree and its timing varies negligibly based on the number of edges. SimClus is the second fastest and its timing varies based on the number of edges in the similarity graph since the time complexity has a term related to $|E|$. The execution time of **SimClus** for 0.5 threshold (which is very relaxed for LBSC) with the entire newsgroup dataset is 1.1, 3.4, and 15.7 times better than UP-GMA, RB, and k -Medoids respectively. We also compared the execution time of our dynamic algorithm to evaluate its utility over the static algorithm. For the newsgroup dataset, average insertion time of one document is 0.13 second, which is much faster in comparison to the re-clustering time (15.22 seconds) using the static algorithm.

6 Conclusions

In this paper we proposed a clustering algorithm that uses lower bound on similarity to cluster a set of objects from the similarity matrix. Experiments on real-life and synthetic datasets show that the algorithm is faster and produces higher quality clustering in comparison to existing popular algorithms. Furthermore, it provides representative centers for every cluster; hence, it is effective in summarization or semi-supervised classification. It is also suitable for multi-label or dynamic clustering.

References

1. C. C. Aggarwal, J. Han, J. Wang, and P. S Yu. A framework for clustering evolving data streams. In *VLDB Proceedings*, August 2003.
2. J. Aslam, J. E. Pelehov, and D. Rus. The star clustering algorithm for static and dynamic information organization. *Graph Algorithms and Application*, 8(1):95–129, 2004.
3. K. S. Azoury and M. K. Warmuth. Relative loss bounds for on-line density estimation with the exponential family of distributions. *Machine Learning*, 43(3):211–246, 2001.
4. B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data streams. In *ACM SIGMOD-SIGACT-SIGART symposium on Principles of Database Systems*. ACM.
5. A. Banerjee and S. Basu. Topic models over text streams: A study of batch and online unsupervised learning. In *SIAM Data Mining*, 2007.
6. R. G-Garcia, J. Badia-Contelles, and A. Pons-Porrata. *Extended Star Clustering*, LNCS 2905. Springer, 2003.
7. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
8. G. King and W. Tzeng. On-line algorithm for the dominating set problem. *Information Processing Letters*, 61:11–14, 1997.
9. C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. *Journal of the ACM*, 41(5):960–981, 1994.
10. V. V. Vazirani. *Approximation Algorithms*. Springer-Verlag.
11. D. Zuckerman. Np-complete problems have a version that’s hard to approximate. In *Proc. of Eighth Annual Structure in Complexity Theory*, pages 305–312. IEEE Computer Society, 1993.