Chapter 14
Clustering Social Networks Using Distance-Preserving Subgraphs

Ronald Nussbaum, Abdol-Hossein Esfahanian, and Pang-Ning Tan

Abstract Cluster analysis describes the division of a dataset into subsets of related objects, which are usually disjoint. There is considerable variety among the different types of clustering algorithms. Some of these clustering algorithms represent the dataset as a graph, and use graph-based properties to generate the clusters. However, many graph properties have not been explored as the basis for a clustering algorithm. In graph theory, a subgraph of a graph is distance-preserving if the distances (lengths of shortest paths) between every pair of vertices in the subgraph are the same as the corresponding distances in the original graph. In this paper, we consider the question of finding proper distance-preserving subgraphs, and the problem of partitioning a simple graph into an arbitrary number of distance-preserving subgraphs for clustering purposes. We then present a clustering algorithm called DP-Cluster, based on the notion of distance-preserving subgraphs. We also introduce the concept of relaxation values to the distance-preserving subgraph finding heuristic embedded in DP-Cluster, and investigate this and other variations of the algorithm. One area of research that makes considerable use of graph theory is the analysis of social networks. For this reason we evaluate the performance of DP-Cluster on two real-world social network datasets.

14.1 Introduction

Cluster analysis is a technique for partitioning a dataset into groups of similar objects. It is often used as an exploratory data analysis tool to determine the underlying structure of a data set. A recent area of application is in the realm of social networks, where the goal is to find tightly-knit groups of people, also

R. Nussbaum (✉) · A.-H. Esfahanian · P.-N. Tan
Michigan State University, East Lansing, MI 48824-1226, USA
e-mail: ronald@cse.msu.edu; esfahanian@cse.msu.edu; ptan@cse.msu.edu

T. Özyer et al. (eds.), *The Influence of Technology on Social Network Analysis and Mining*, Lecture Notes in Social Networks 6, DOI 10.1007/978-3-7091-1346-2_14,

known as communities, based on their social relations. Communities are detected by partitioning the network into subgraphs in such a way that optimizes certain graph properties (e.g., minimizing the cut between clusters [9], or maximizing an edge-based modularity function [16]). In addition to these criteria, there are other graph invariants that are potentially applicable to clustering social networks. The motivation for our work is to explore the effect of alternative graph invariants on the process of community finding.

One of the difficulties with cluster analysis is that there is no strict definition of what a cluster is. This makes it more difficult to conjecture what sort of concepts might be usable in creating a clustering algorithm. In a social network, a vertex usually represents a person or small group of persons, while the edges represent the ties between them. We might wish to cluster participants according to family ties, peer groups, business relationships, and other personal attributes or common interests. For the graph of a social network, we expect to find higher connectivity between similar vertices than dissimilar ones. More importantly, we expect the centers of social groups to be quite dense, with fewer edges between communities. Since social networks tend to follow a scale-free distribution, we will not find large cliques. Still, we expect the shortest path between any two members of the same community in the network to involve other members from the same community rather than members from other communities. It is this last premise that gives us reason to explore the use of distance-preserving subgraphs as the basis of a clustering algorithm.

A distance-preserving subgraph is an induced proper subgraph that maintains the distances (lengths of shortest paths) as in the original graph. Consider the toy example of a social network shown in Fig. 14.1 which has 8 vertices and 13 edges. There are two natural communities in the network, one involving vertices (A, B, C, D) and the other (E, F, G, H). Clearly, both subgraphs are distance-preserving because the shortest path distance between any two vertices within each subgraph is identical to the one computed from the entire network. However, adding the vertex E to the subgraph (A, B, C, D) no longer preserves the distance between the vertex pair (E, B). Similarly, the addition of vertex F to (A, B, C, D) would make the subgraph non-distance-preserving because the distance between (F, C) is not the same as that in the original graph.

We face a number of challenges along the way to creating a useful clustering algorithm. As previously mentioned, we need to find distance-preserving subgraphs within a graph that represent clusters. While we generally cannot divide a given graph into an arbitrary number of distance-preserving subgraphs, we can always divide it into at least as many as desired, since isolated vertices are distance-preserving subgraphs. So we must devise a method of merging extra clusters such that we can always reach the desired number of clusters. We might also encounter graphs that can be divided into less than the desired number of distance-preserving clusters, in which case we will have to split them. Once we have addressed these challenges, we can use the heuristics found to develop a clustering algorithm.

This work is an extension of a previous paper of ours, *Clustering Social Networks Using Distance-Preserving Subgraphs* [17]. The bulk of the new material

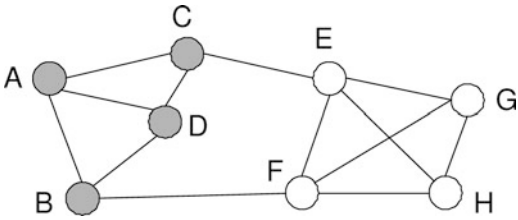

Fig. 14.1 A toy example of a social network

may be found in Sect. 14.4.4 and throughout Sects. 14.5 and 14.6. In the next section, we review related work in cluster analysis and give further background on distance-preserving subgraphs. In Sect. 14.3, we give a formal problem statement. In Sect. 14.4, we describe our algorithm, DP-Cluster, and variations on the basic algorithm. In Sect. 14.5, we present the results of applying the DP-Cluster algorithm on two social network datasets. We state our conclusions in Sect. 14.6.

14.2 Related Work

Many different clustering algorithms exist. Here we cover some of the more common ones, along with those based on the use of graph properties. We also provide some background into the topic of community finding. Finally, we give a more thorough explanation of distance-preserving subgraphs.

14.2.1 Clustering Algorithms

One of the better known methods of cluster analysis is k-means, whose name dates back to a paper by MacQueen [15]. k-means is a partitional algorithm that iteratively assigns objects to their nearest cluster centers, then recomputes the centers until they converge. Another important class of clustering methods is hierarchical clustering, which can be done in agglomerative or divisive fashion. Hierarchical clustering produces a dendrogram, which represents a series of clustering solutions, from all objects in one cluster to each object in a cluster of its own. Many variations of hierarchical clustering algorithms exist, including single-link, complete-link, group average, and Ward's method. An alternate approach is to define clusters as regions of high density objects separated by low density regions often populated by noisy observations. Two such algorithms are DBSCAN [8] and OPTICS [1].

A number of graph theoretic clustering algorithms have been developed. These algorithms are designed to optimize certain graph properties. For example, spectral clustering methods were developed to minimize variants of the graph cut property

using the eigenvectors of the graph Laplacian matrix. Other heuristics have also been proposed based on minimizing diameter, p-centers, and other graph properties [4, 9, 18].

14.2.2 Community Finding

The technique of community finding, also called group detection [11], positional analysis [7, 21] or blockmodelling [21], is the process of placing vertices into groups in such a way that the vertices within a group are "similar" to each other and "dissimilar" to vertices in other groups. For example, Newman and Girvan [16] proposed an approach for discovering community structure in networks using modularity function as a measure of cluster quality. Scripps et al. [19] presented an overlapping clustering algorithm, taking into account the ill-effects of bridge vertices in a network. Liu et al. [14] gave an algorithm for building hierarchical communities using client-side web browsing history. More recently, there has been considerable interest in discovering dynamic communities in an evolving social network. For example, Zhou et al. [23] presented an algorithm for finding dynamic communities by combining the statically derived communities subject to certain constraints to ensure consistency of the clusters at different time periods. Tantipathananandh et al. [20] developed a community finding algorithm for dynamic networks using a graph coloring method.

14.2.3 Distance-Preserving Subgraphs

The topic of distance-preserving subgraphs is relatively unexplored. A subgraph of a graph G of order n is called a distance-preserving subgraph if the distances between every pair of vertices in the subgraph are the same as the corresponding distances between them in G, and the subgraph is a proper subgraph of G. A distance-preserving subgraph is necessarily induced if all edge weights in G are equal, since the omission of any edge would increase the distance between the endpoints of that edge. One can determine whether a given subgraph is distance-preserving by computing all-pairs shortest paths for the subgraph, and for G. If there are no negative cycles in G, then Floyd Warshall's algorithm [10] may be used to accomplish this, which has a time complexity of $O(n^3)$. If instead we have a subgraph $H \subseteq G$ that is known to be distance-preserving, and another vertex $v \in G$, determining whether $H \cup v$ is a distance-preserving subgraph of G is a single-source shortest path problem. This can be computed in $O(n^2)$ using Dijkstra's algorithm [6], or the Bellman-Ford algorithm [3] if G has negative edge weights.

A related concept is the distance-hereditary graph, first proposed by Howorka [13]. A graph G is distance-hereditary if every connected induced subgraph is

distance-preserving. Distance-hereditary graphs are a subset of perfect graphs, and have been studied extensively [2, 5, 12].

14.3 Problem Statement

Let $G = (V, E)$ be a graph representing a connected social network of order $|V| = n$. Order refers to the number of vertices in a graph, and size refers to the number of edges. Our goal is to use the concept of distance-preserving subgraphs to partition $V(G)$ into k pairwise disjoint subsets $V_1, \ldots, V_k$, such that $V_1 \cup \cdots \cup V_k = V$, where V_i induces a distance-preserving subgraph. With n vertices and k clusters, on average a cluster must contain n/k vertices. Of course, the purpose of using cluster analysis is to find good clusters, not just ones of equal order. However, this fact hints at the fundamental question facing us: Given a graph G and integer m, does G contain a distance-preserving subgraph of order m? Answering this question raises several immediate issues.

14.3.1 Challenges

Our foremost concern is attaining clusters that are distance-preserving. Naturally, G will not always contain a distance-preserving subgraph of order m. So it will not always be possible to partition G into k distance-preserving clusters. Instead, we will seek to find clusters that are as close to being distance-preserving as possible. In the next section, we offer a way to quantitatively measure such almost distance-preserving subgraphs.

Even if G does have a distance-preserving subgraph of order m, we still have to find it. We suspect that this cannot be done for the general case in polynomial time. It definitely is computationally infeasible to do so in a naive fashion, so we resort to heuristics.

Since any distance-preserving algorithm is based off of distances between pairs of vertices, we have a problem if G is not connected. Our distance-preserving clustering algorithm assumes that G is a connected network. If the graph G is disconnected, we could apply our algorithm to each component separately.

14.3.2 Algorithmic Approach

Here we are at a crossroads. In the DP-Cluster algorithm presented in the next section, we partition G into as few properly distance-preserving clusters as we can find according to a heuristic, and proceed to merge them until we have k clusters. However, once we define a measure for how distance-preserving a subgraph

is, we could instead construct an agglomerative hierarchical clustering algorithm. Specifically, we could start with G partitioned into n distance-preserving clusters, and repeatedly merge the two clusters whose union was most distance-preserving according to some metric, until we had k clusters left. It is less apparent how we might construct an efficient divisive hierarchical clustering algorithm. We proceed with our hybrid method in the interest of keeping larger parts of the clusters distance-preserving. It is not clear which path might lead to a better performing or less computationally expensive algorithm, although our algorithm seeks to minimize the number of times an all-pairs shortest paths algorithm is invoked.

14.4 DP-Cluster

Here we describe our heuristic for finding distance-preserving subgraphs, give a definition for an almost distance-preserving subgraph that we can use as a metric to merge these clusters, and present a simple clustering algorithm that combines the two, which we call DP-Cluster. The basic algorithm is covered in Subsect. C, and variations on the basic algorithm are discussed in Subsect. D. In the next section we compare DP-Cluster to the hierarchical clustering algorithm in Matlab's Statistics Toolbox (see *linkage* and *cluster*), along with random clustering for a baseline.

14.4.1 Finding Distance-Preserving Subgraphs

Finding large distance-preserving proper subgraphs in a graph is not an easy task. Clearly we cannot test each of the almost 2^n induced subgraphs of G to see whether or not it is distance-preserving. The best heuristic we have found so far is to start with a single vertex as a cluster C, which is trivially distance-preserving. At each step, we attempt to add each neighbor not in C to C in turn. If there is some non-empty set of neighbors such that the union of a single element with C leaves C distance-preserving, we choose one of them to permanently add to C according to some criteria, and continue. Once we reach a step where no neighbors can be added to C and have it remain distance-preserving, we are done.

14.4.2 Almost Distance-Preserving Subgraphs

We define the *average distance increase* for a subgraph of G as the sum of the distance increases between the subgraph and G, divided by the number of vertices in the subgraph. If the subgraph is distance-preserving, then this value is 0. The less distance-preserving the subgraph is, the higher this number will be.

14.4.3 The Algorithm

Our algorithm begins with each vertex from G in a separate cluster, all of which are trivially distance-preserving subgraphs of G. These clusters must be combined until there are only k clusters left. We pick one of these singleton clusters at random and use it as the start vertex to find a distance-preserving subgraph as described in the first part of this section. Random selection is used when there is more than one choice of vertices to add to the current cluster. Once this has finished, we set these vertices aside, pick another vertex at random, and repeat the process. Eventually, this will leave us with G partitioned into some number of distance-preserving clusters.

If we do not stop cluster growth at n/k vertices, we may end up with less that k distance-preserving clusters. In this case, we would have to break distance-preserving clusters apart to achieve k clusters. Since this has not happened with an actual dataset, we do not concern ourselves further with this possibility. Instead, we end up with a number of distance-preserving clusters larger than k. We then take each pair of clusters, and calculate how close to distance-preserving each potential merger is, according to the metric given in the second part of this section. The merge with the lowest value, i.e., the most distance-preserving, is made. This process is repeated until only k clusters remain.

Below is pseudocode for the basic versions of DP-Cluster. *clusters* is the set of clusters, whose members are sets of vertices. *unused* is the set of vertices that have not been placed into a cluster. *neighbors* is the set available vertices adjacent to some vertex in the current cluster. *usable* is the set of available vertices which can be added to the current cluster, and still have it be distance-preserving. RANDOM returns a random element from a set. NEIGHBORS returns the set of neighbors for a given vertex of a graph. RANDOMIZE returns a random permutation of a set. COUNT returns the number of elements in a set. ALMOST-DP returns how close a cluster is from being distance-preserving, according to the method described in the previous subsection.

14.4.4 Variations

Very often our distance-preserving subgraph finding heuristic will detect more than one neighbor whose addition leaves the resulting subgraph distance-preserving. In these cases, we must choose between them according to some criteria. Our default method is to do so randomly. This approach has the benefit that the search may be terminated after the first suitable vertex is found. The use of other metrics will affect subgraph generation in different ways. For reasons of efficiency, we prefer simple local measures that can be precomputed from G, rather than non-hereditary graph invariants that must be computed during execution. Unfortunately, this prevents us from using potentially useful graph invariants, such as diameter or girth, as tiebreak methods. Instead, we use the degree and clustering coefficient (CC) of vertices as

Algorithm 1 DP-Cluster

input : A graph $G = (V, E)$, and integer k.
output: A partition of V into k disjoint *clusters*.

```
unused ⟵ V
for i ⟵ 1 to n do
    v ⟵RANDOM(unused)
    unused ⟵ unused \ {v}
    clusters[i] ⟵ {v}
    neighbors ⟵NEIGHBORS(G, v) ∩ unused
    while unused do
        neighbors ⟵RANDOMIZE(neighbors)
        usable ⟵ {}
        foreach v ∈ neighbors do
            if IS_DP(G, clusters[i] ∪ v) then
                usable ⟵ usable ∪ v
            end
        end
        if usable ≠ {} then
            v ⟵RANDOM(usable)
            unused ⟵ unused \ {v}
            clusters[i] ⟵ clusters[i] ∪ v
            neighbors ⟵ neighbors ∪NEIGHBORS(G, v) ∩ unused
        else
            break
        end
    end
    c1, c2 ⟵ 0
    best ⟵ ∞
    while COUNT(clusters) > k do
        foreach i ∈ clusters do
            foreach j ∈ clusters \ {i} do
                if ALMOST_DP(clusters, i, j) < best then
                    c1 ⟵ i
                    c2 ⟵ j
                    best ⟵ALMOST_DP(clusters, i, j)
                end
            end
        end
        clusters ⟵ clusters ∪ {C1 ∪ C2} \ {C1, C2}
    end
end
return clusters
```

computed from the original graph. The clustering coefficient of a vertex v is the number of edges found between the neighbors of v divided by the total possible number of edges between the neighbors of v [22].

The DP-Cluster algorithm consists of two fairly independent parts, the distance-preserving subgraph finding heuristic, and the metric used to merge subgraphs. The first part partitions all vertices into distance-preserving subgraphs, and the second

merges them as best it can. If we want more work done in the first part, we can relax the requirement that each subgraph found must be distance-preserving. Instead, we require that each distance in the subgraph found must be less than or equal to the corresponding distance in G, plus some constant. We call this constant the *relaxation value*. A relaxation value of 0 (and random tiebreaks) is equivalent to the basic version of DP-Cluster, i.e., each subgraph found in the first part of the algorithm must be distance-preserving.

14.4.5 Efficiency

Using distance-preserving subgraphs in any manner requires running an all-pairs shortest paths algorithm on G. Regardless of how we formulate our clustering algorithm, our running time must be at least $O(n^3)$. However, in generating distance-preserving subgraphs by adding one vertex at a time, we can use a single-source shortest paths algorithm in our distance-preserving subgraph finding heuristic. Variations that increase the size of the distance-preserving subgraphs found decrease the number of costly merges that must be performed.

14.5 Experimental Evaluation

In this section we compare and contrast DP-Cluster against a few existing hierarchical clustering algorithms, as well a random clustering algorithm, using different social network datasets. In each of our experiments, we test the basic algorithm, and examine the effects of using different tiebreaking methods and nonzero relaxation values. For clarity, we will continue to refer to the unmodified DP-Cluster algorithm as presented in Sect. 14.4.3, with random tiebreaks, and a relaxation value of 0, by using the adjective *basic*.

14.5.1 Datasets

The two datasets we use for testing are CiteSeer and Cora. They can be downloaded from the University of Maryland website at http://www.cs.umd.edu/~sen/lbc-proj/LBC.html. Vertices in these datasets are scientific publications, and edges represent citations, i.e., if the graph contains an edge (A, B), then either paper A cites paper B or paper B cites paper A. Although the papers represent authors, clustering in these datasets is closer to document classification than community finding in an ordinary social network. CiteSeer, which is composed of general topic computer science papers, contains 3,312 vertices, 4,536 edges, and 6 class labels: Agents, Artificial Intelligence, Database, Information Retrieval, Machine Learning, and Human-Computer Interaction. Cora, which consists entirely of

Table 14.1 Dataset properties – largest component

Dataset	Order	Size	Diameter	Number of classes
CiteSeer	2,110	3,668	28	6
Cora	2,485	5,069	19	7

machine learning papers, contains 2,708 vertices, 5,278 edges, and 7 class labels: Cased Based, Genetic Algorithms, Neural Networks, Probabilistic Methods, Reinforcement Learning, Rule Learning, and Theory. Since we want connected social networks for testing purposes, we extract the largest component from each dataset. For the remainder of this paper, we are referring to the largest component whenever we reference CiteSeer and Cora (Table 14.1).

14.5.2 Experiments

Before attempting to evaluate the full DP-Cluster algorithm, we want to look at results for each of the intermediate steps. Our first experiment is simply to find a single distance-preserving subgraph, using the incremental approach described in Sect. 14.4.1. When multiple vertices can be added to a cluster such that the cluster is still distance-preserving, we must choose between them in some fashion. We consider random selection, as well as the degree and clustering coefficient of the vertex as found in the original graph G. For vertex degree and clustering coefficient, we run two trials for each metric, one using ascending order, and the other with descending order. The goal in this experiment is to determine the effect of using different tiebreak methods and relaxation values on the clusters found, and to help choose which tiebreak methods to focus on going forward. We run 100 trials for each combination of dataset and method of breaking ties, and present statistics for the distance-preserving subgraphs found. We also run 100 trials for each combination of dataset and method of breaking ties with a relaxation value of 1. For each trial, we calculate the entropy of the final distance-preserving subgraph using the actual class labels. If G has k labels, then the entropy of the subgraph is given by the equation

$$entropy = -\sum_{i=1}^{k} \frac{m_i}{m} \log_k \frac{m_i}{m},$$

where m is the order of the subgraph, m_i is the number of vertices in the subgraph with label i, and $m_1 + \ldots + m_k = m$. Over all trials, we calculate the average order, standard deviation (σ) of the order, and the average diameter and entropy of the distance-preserving subgraphs found.

In the second experiment we examine the first part of the DP-Cluster algorithm, using random tiebreaks, as well as vertex degree and clustering coefficient in descending order only. That is, we run the algorithm as normal, but discard any clusters in excess of the number of class labels instead of merging them. As a result,

each run of the DP-Cluster algorithm gives us a different subset of the vertices in G. The goal of this experiment is to give us some idea of the applicability of distance-preserving subgraphs to cluster analysis, without testing our definition of almost distance-preserving subgraphs as defined in Sect. 14.4.2. Here we run 100 trials on each dataset for each of the specified tiebreak methods, and calculate the average order and entropy over all trials. We run additional trials using random tiebreaks and nonzero relaxation values. In this experiment, the entropy for a given trial is the weighted average of the individual cluster entropies. For comparison, we calculate these same values using MATLAB's hierarchical clustering (HC) with various metrics, described in more detail below, and random clustering. Random clustering is exactly that, assigning each vertex to one of the k clusters with equal probability. As a different subset of vertices from G are used in each trial, the inputs to the hierarchical clustering and random clustering algorithms are only those vertices found by DP-Cluster in that particular trial.

The third experiment tests the full DP-Cluster algorithm as presented in Sect. 14.4.3, again using random tiebreaks, as well as vertex degree and clustering coefficient in descending order only. For each trial G is fully partitioned into distance-preserving subgraphs, which are then merged into k almost distance-preserving subgraphs. For this experiment, we run 20 trials on each dataset for each tiebreak method, and present the average entropy over all runs. Again, we run additional trials using random tiebreaks and nonzero relaxation values. We also consider cluster stability. For each run, we create an $n \times n$ incidence matrix, where entry i, j is 1 if i and j belong to the same cluster, and 0 if they do not. We then calculate the average correlation between these matrices for each pair of trials. Again, we calculate these same values using hierarchical clustering, and random clustering algorithms.

For evaluating MATLAB's hierarchical clustering algorithm, we use three different distance metrics. *Single linkage* (nearest neighbor) uses the minimum of all the distances between pairs of vertices in the two clusters. *Complete linkage* (furthest neighbor) uses the maximum of all the distances between pairs of vertices in the two clusters. *Average linkage* uses the average unweighted distance between all pairs of vertices in the two clusters. For each of these distance metrics, the pair of clusters with the minimum distance between clusters are merged at each step.

14.5.3 Results

In this subsection we give the results for our three experiments.

14.5.3.1 Finding a Single Distance-Preserving Subgraph

In Tables 14.2 and 14.3 we see that the average distance-preserving subgraph found was reasonably large, with considerable variation depending on the tiebreak method used. Even so, the average order is still much smaller than the average cluster needs

Table 14.2 Finding a single distance-preserving subgraph – CiteSeer

Tiebreak method	Relaxation value	Average order	σ(order)	Average diameter	Average entropy
Random	0	101	58	14	0.636
Degree, incr.	0	56	35	12	0.664
Degree, decr.	0	176	87	16	0.645
CC, incr.	0	86	51	14	0.657
CC, decr.	0	131	69	15	0.652
Random	1	223	112	17	0.684
Degree, incr.	1	131	87	16	0.666
Degree, decr.	1	416	184	19	0.720
CC, incr.	1	149	85	16	0.684
CC, decr.	1	334	182	19	0.667

Table 14.3 Finding a single distance-preserving subgraph – Cora

Tiebreak method	Relaxation value	Average order	σ(order)	Average diameter	Average entropy
Random	0	110	86	10	0.618
Degree, incr.	0	47	52	8	0.562
Degree, decr.	0	169	109	11	0.634
CC, incr.	0	64	57	9	0.590
CC, decr.	0	150	106	10	0.626
Random	1	188	140	11	0.642
Degree, incr.	1	91	99	11	0.624
Degree, decr.	1	339	166	13	0.707
CC, incr.	1	124	103	11	0.615
CC, decr.	1	298	189	12	0.666

to be, so we do not have to consider the problem of splitting excessively large distance-preserving subgraphs. The standard deviation (σ) of the order is also quite high, indicating that the heuristic is sensitive to the choice of initial vertex. More intelligent selection of starting vertices might not only reduce this, but improve the performance of the DP-Cluster algorithm as well. The average diameter found with the CiteSeer dataset is much higher than the one found for Cora, which corresponds to the fact that the original CiteSeer graph had a diameter of 28 compared to a diameter of 19 for Cora (Table 14.1).

It is clear from our results that the method of tiebreaking used has a significant effect on the order of the subgraph found. Specifically, using vertex degree and clustering coefficient in descending order both bias our heuristic towards using higher degree vertices earlier. This yields much larger subgraphs compared to selecting a vertex at random, along with an expected increase in diameter and entropy. We suspect what is happening here is that if higher degree vertices are not chosen sooner, they are likely never able to be used at all, limiting the pool of unused neighbors, and ultimately the size of the distance-preserving subgraph found. Notably, although using these metrics in ascending order produces clusters smaller than those found using random tiebreaks, entropy does not decrease. With

Table 14.4 Basic DP-Cluster, discarding excess clusters – CiteSeer

Algorithm	Average entropy
DP-Cluster	0.656
HC, single	0.815
HC, complete	0.627
HC, average	0.660
Random	0.888

Table 14.5 Basic DP-Cluster, discarding excess clusters – Cora

Algorithm	Average entropy
DP-Cluster	0.618
HC, single	0.795
HC, complete	0.589
HC, average	0.637
Random	0.811

the CiteSeer dataset, average entropy actually increases when using vertex degree and clustering coefficient in ascending order instead of random tiebreaks. Other methods of breaking ties might produce even larger subgraphs, or subgraphs of lower entropy. As previously mentioned, many graph invariants we would like to use are unfeasible for complexity reasons. Given the results here, subsequent experiments will focus on the use of random tiebreaks, as well as vertex degree and clustering coefficient in descending order only.

The effect of relaxing the distance-preserving requirement even by 1 is significant. Average cluster size is slightly less than double in most cases compared to the corresponding trial with a relaxation value of 0. Results using higher relaxation values continue the trend towards larger and larger clusters, and are omitted for brevity.

14.5.3.2 Partial Clustering With DP-Cluster

The basic DP-Cluster algorithm used approximately one quarter of the vertices from the largest component of the corresponding dataset on average. The hierarchical and random clustering algorithms used the same subset of vertices found by DP-Cluster in each trial as inputs, which is why the average order of these clustering algorithms is the same. We had some concern that the order of successive distance-preserving subgraphs would decrease rapidly after the first one. This was not observed in practice, and the average order for these trials was only slightly less than the average order for the corresponding trials in our first experiment times the number of class labels for that dataset (Tables 14.4 and 14.5).

The average entropy for all the clusters found by the basic DP-Cluster algorithm, while not as low as we might desire, did not increase appreciably from the previous

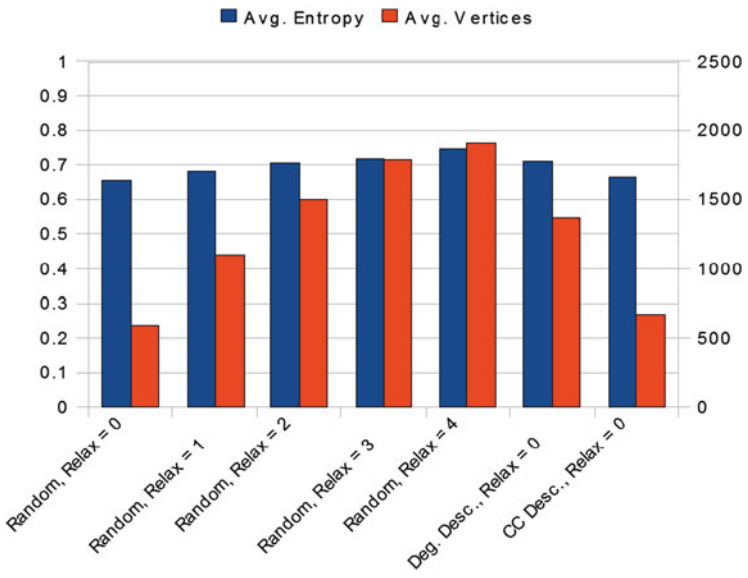

Fig. 14.2 DP-Cluster variations, discarding excess clusters – CiteSeer

experiment. For the CiteSeer dataset, DP-Cluster resulted in an average cluster order of 581, and it outperformed two of the three hierarchical clustering algorithms. With the Cora dataset, DP-Cluster resulted in an average cluster order of 633, and it outperformed the same two hierarchical clustering algorithms. Since the set of vertices used here varies from trial to trial, it is meaningless to consider correlation (Tables 14.4 and 14.5).

As seen in Figs. 14.2 and 14.3, variations in DP-Cluster drastically affect our results. Here we examine alternate tiebreak methods and nonzero relaxation values, without comparing each one back to the hierarchical clustering algorithm. In line with what we saw in the first experiment, the number of vertices used increases dramatically with each successive relaxation value. As we increase the relaxation value from 0 to 4, we go from using a quarter of the vertices in the dataset in each trial, to using nearly all of them.

The alternate tiebreak methods tested performed somewhat differently than we anticipated from the results of the first experiment. Using vertex degree resulted in much larger clusters as expected, and the average entropy of these trials did not show an improvement over random tiebreaks after taking into account the number of vertices used. With the clustering coefficient, the average number of vertices used in all clusters found was not that much higher than using random tiebreaks. This stands in contrast to the first experiment, where the average subgraph order found using the clustering coefficient metric was much higher than when choosing randomly. As with vertex degree, the entropy is again roughly in line with random tiebreaks after adjusting for the number of vertices used.

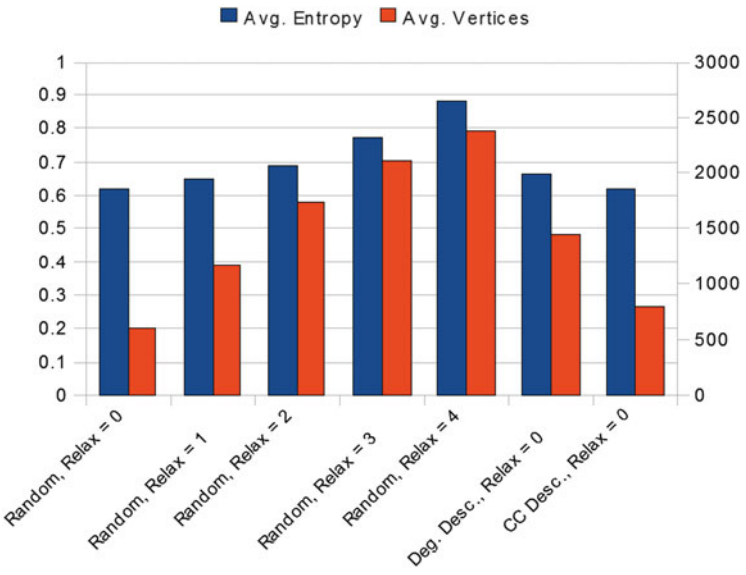

Fig. 14.3 DP-Cluster variations, discarding excess clusters – Cora

14.5.3.3 Full Clustering With DP-Cluster

For the CiteSeer dataset, performance of the hierarchical clustering algorithms range from somewhat better to much worse than basic DP-Cluster, depending on the distance metric used to create the hierarchical cluster tree. Unsurprisingly, random clustering led to much higher average entropies than all other clustering algorithms (Tables 14.6 and 14.7).

With the Cora dataset, basic DP-Cluster performs better than all but one of the distance metrics. It is not immediately clear why complete-link hierarchical clustering, which merges clusters according to the furthest distance between them, performs so well here. What is certain is that cluster entropy has significantly increased from the previous experiment. Also of concern is the relatively low correlation found between trials using DP-Cluster (Table 14.7).

We were less optimistic after the previous experiment that nonzero relaxation values would improve results using the full DP-Cluster algorithm. Here we see that for both datasets entropy decreases up to a relaxation value of 2, after which it increases again (Figs. 14.4 and 14.5). Correlation also increases along with the relaxation value, although this is probably due to the increase in variance of cluster order.

As with the previous experiment, using alternate tiebreak methods had mixed performance. Results using vertex degree and the clustering coefficient had entropies similar to DP-Cluster with a relaxation value of around 2. With both datasets, results using the clustering coefficient had higher correlations than those using vertex degree.

Table 14.6 Basic DP-Cluster, merging excess clusters – CiteSeer

Algorithm	Average entropy	Average correlation	Average clusters
DP-Cluster	0.778	0.264	102
HC, single	0.952	1.000	N/A
HC, complete	0.727	1.000	N/A
HC, average	0.898	1.000	N/A
Random	0.951	0.091	N/A

Table 14.7 Basic DP-Cluster, merging excess clusters – Cora

Algorithm	Average entropy	Average correlation	Average clusters
DP-Cluster	0.783	0.205	181
HC, single	0.937	1.000	N/A
HC, complete	0.748	1.000	N/A
HC, average	0.883	1.000	N/A
Random	0.937	0.077	N/A

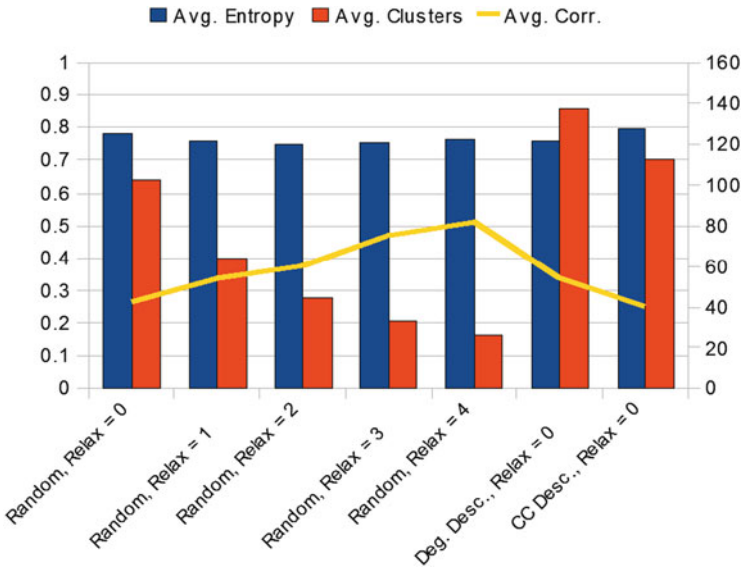

Fig. 14.4 DP-Cluster variations, merging excess clusters – CiteSeer

14.5.4 Discussion

Overall, the results of DP-Cluster compare favorably to hierarchical clustering methods. The algorithm has some obvious weaknesses, some of which may be due to the current implementation. Like many incremental algorithms, DP-Cluster is order dependent. It does find clusters of reasonably low entropy, even after "leftover" clusters are taken into account. However, complete-link hierarchical

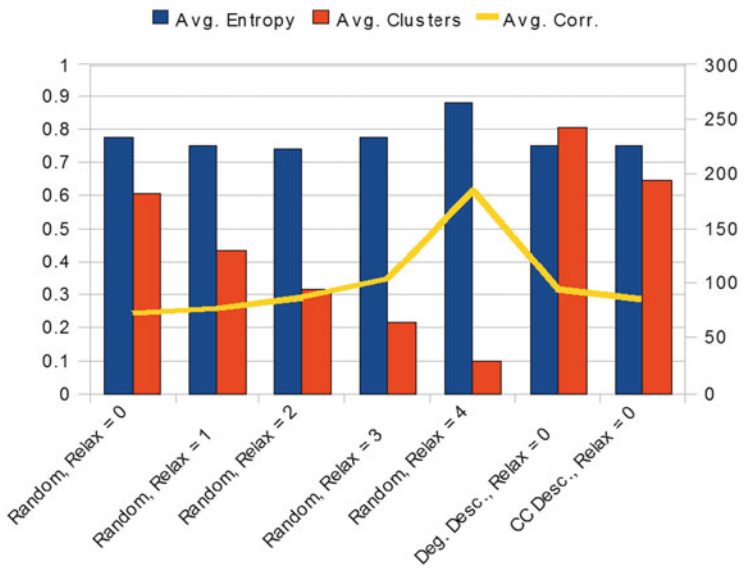

Fig. 14.5 DP-Cluster variations, merging excess clusters – Cora

clustering outperforms DP-Cluster on the CiteSeer and Cora datasets, although the gap is smaller with some of the DP-Cluster variations. This outperformance might disappear on other datasets depending on their structure. Also, DP-Cluster found clusters of fairly low stability given their low entropy. A better distance-preserving subgraph finding heuristic should reduce this effect. An algorithm with better time complexity would allow us to explore larger datasets as well. So far we have only tested DP-Cluster on datasets of modest size.

In this paper, we treat paper citations as undirected links, rather than directed arcs. One avenue of exploration would be to treat them as arcs instead. Alternately, we could investigate social networks in which links indicate a bidirectional relationship, rather than a unidirectional relationship like a citation or following another profile. An entirely different approach with the paper citation networks would be to treat authors as links instead of papers.

Our experimental results lead us to make the following conjecture.

Conjecture 1. Almost all graphs are distance-preserving, i.e., an arbitrary graph on n vertices has at least one distance-preserving subgraph for each order $m = 1, \ldots, n$.

If true, it underscores the need to develop heuristics for identifying distance-preserving subgraphs that better reflect actual communities in social networks. An exhaustive search of all connected graphs on up to 11 vertices provides some support for our conjecture, as seen in the table below (Table 14.8).

Table 14.8 A survey of distance-preserving graphs

n	# connected graphs	# connected, non-DP graphs	Proportion connected, non-DP graphs
5	21	1	0.04762
6	112	1	0.00892
7	853	4	0.00469
8	11,117	19	0.00171
9	261,080	183	0.00070
10	11,716,571	2,474	0.00021
11	1,006,700,565	107,176	0.00011

14.6 Conclusions and Future Work

The performance of the fairly simple DP-Cluster algorithm shows promise for the use of distance-preserving subgraphs in community finding. Nonetheless, further exploration is needed to develop a better performing algorithm. Along with finding better clusters, finding more consistent clusters is an issue. Possibilities here include improving the heuristic for finding distance-preserving clusters through intelligent selection of initial vertices, different tiebreak methods, or other variations in the same vein as our concept of relaxation values. Changing the manner in which the distance-preserving clusters are merged is also an option. Another approach is to create an agglomerative distance-preserving based hierarchical algorithm, which would combine the "finding" and "merging" processes into a single step. Our previously stated efficiency concerns regarding the use of all-pairs shortest paths versus single-source shortest path algorithms apply here.

References

1. Ankerst, M., Breunig, M., Kriegel, H., Sander, J.: OPTICS: ordering points to identify the clustering structure. ACM SIGMOD Rec. **28**(2), 49–60 (1999)
2. Bandelt, H., Mulder, H.: Distance-hereditary graphs. J. Comb. Theory B **41**(2), 182–208 (1986)
3. Bellman, R.: On a routing problem. Q. Appl. Math. **16**(1), 87–90 (1958)
4. Charikar, M., Chekuri, C., Feder, T., Motwani, R.: Incremental clustering and dynamic information retrieval. In: Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, pp. 626–635. ACM, New York (1997)
5. Damiand, G., Habib, M., Paul, C.: A simple paradigm for graph recognition: application to cographs and distance hereditary graphs. Theor. Comput. Sci. **263**(1–2), 99–111 (2001)
6. Dijkstra, E.: A note on two problems in connexion with graphs. Numer. Math. **1**(1), 269–271 (1959)
7. Doreian, P., Batagelj, V., Ferligoj, A.: Positional analyses of sociometric data. Models and Methods in Social Network Analysis, pp. 77–97. Cambridge University Press, New York (2005)
8. Ester, M., Kriegel, H., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proceedings of KDD, vol. 96, pp. 226–231. AAAI, Menlo Park (1996)

9. Flake, G., Tarjan, R., Tsioutsiouliklis, K.: Graph clustering and minimum cut trees. Int. Math. **1**(4), 385–408 (2004)
10. Floyd, R.: Algorithm 97: shortest path. Commun. ACM **5**(6), 345 (1962)
11. Getoor, L., Diehl, C.: Link mining: a survey. ACM SIGKDD Explor. Newsl. **7**(2), 12 (2005)
12. Hammer, P., Maffray, F.: Completely separable graphs. Discret. Appl. Math. **27**(1–2), 85–99 (1990)
13. Howorka, E.: A characterization of distance-hereditary graphs. Q. J. Math. Oxf. Ser. **2**(28), 417–420 (1977)
14. Liu, K., Bhaduri, K., Das, K., Nguyen, P., Kargupta, H.: Client-side web mining for community formation in peer-to-peer environments. ACM SIGKDD Explor. Newsl. **8**(2), 20 (2006)
15. MacQueen, J.: Some methods for classification and analysis of multivariate observations. Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability, vol. 1, pp. 281–297. Defense Technical Information Center, Ft. Belvoir (1966)
16. Newman, M., Girvan, M.: Finding and evaluating community structure in networks. Phys. Rev. E **69**(2), 26113 (2004)
17. Nussbaum, R., Esfahanian, A., Tan, P.: Clustering social networks using distance-preserving subgraphs. In: 2010 International Conference on Advances in Social Networks Analysis and Mining (ASONAM), pp. 380–385. IEEE, Washington, DC (2010)
18. Plesnik, J.: A heuristic for the p-center problem in graphs. Discret. Appl. Math. **17**(3), 263–268 (1987)
19. Scripps, J., Tan, P.: Constrained overlapping clusters: minimizing the negative effects of bridge-nodes. Stat. Anal. Data Min. **3**(1), 20–37 (2010)
20. Tantipathananandh, C., Berger-Wolf, T., Kempe, D.: A framework for community identification in dynamic social networks. In: Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, p. 726. ACM, New York (2007)
21. Wasserman, S., Faust, K.: Social Network Analysis: Methods and Applications. Cambridge University Press, Cambridge (1994)
22. Watts, D., Strogatz, S.: Collective dynamics of 'small-world' networks. Nature **393**(6684), 440–442 (1998)
23. Zhou, D., Councill, I., Zha, H., Giles, C.: Discovering temporal communities from social network documents. In: Proceedings of the 2007 Seventh IEEE International Conference on Data Mining, pp. 745–750. IEEE, Washington, DC (2007)