Two Birds With One Stone: A Similarity-Guaranteed Clustering Algorithm and Its Search Tree

Sanpawat Kantabutra and Chumphol Bunkhumpornpat

***Abstract*—A new data structure called "clustering tree" is presented in this paper. With this new data structure, a new clustering algorithm producing a set of clusters with guaranteed similarity is described. We also show that the same clustering tree can be used efficiently as a *d*-dimensional search tree.**

***Index Terms*—Clustering methods, Data clustering, Data mining, Search methods, Search trees**

I. Introduction

CLUSTERING, or partitioning into dissimilar groups of similar objects, is a problem with many variants in mathematics and applied sciences. With the advent of Internet technology comes the availability of vast amounts of data. In the past decade, several clustering methods have been invented. In general major clustering methods can be classified into 5 categories: *partitioning* [1]-[6], *hierarchical* [7]-[9], *density-based* [10]-[12], *grid-based* [13]-[15], and *model-based* methods [16]-[17]. In addition, some clustering algorithms integrate the ideas of these clustering methods. All of these methods are either heuristic-based or experiential or both. The nature of heuristic approach cannot guarantee quality of the clustering. The typical justification of the clustering quality given by practitioners and experimentalists is case-by-case. In other words, we can always find a case in which the outcome of the clustering is bad. Moreover, experimental results also leave the burden of interpretation of the clustering results to the users. Consequently, one user may interpret the results differently from another. Theoreticians, on the other hand, have been busy studying quality measures that are simple to define (e.g. k-medians, minimum sum, minimum diameter, etc.) and these methods most often use optimization on certain criteria or objective functions. It is well known in clustering literature that there are simple examples in which the *right* clustering is obvious but optimizing these measures produces undesirable solutions.

Thus far, there have been some quantitative measures for the *goodness* of clustering results [1], [18]. For instance, the popular k-means algorithm uses a squared-error function to indicate the clustering quality. Even though this function is quantitative, we still do not have a clear idea how good the clustering is. A clustering that produces the squared-error, say, of 145.6 does not tell us the similarity relationship between any pair of points in a cluster. The lack of this information can put us at a disadvantage. In several applications such as web-based search engines it is very useful to know this relationship between pairs of points in the clusters.

All existing clustering algorithms require a number of parameters as their inputs and these parameters can significantly affect the cluster quality. These parameters are also *case-specific*. That is, some values are good for some certain cases and appropriate values can be found only through trial-and-error and we have no way to accurately predict the appropriate values of these parameters. In this paper we want to avoid experiential methods and advocate the idea of *need-specific* as opposed to case-specific because users always know the needs of their applications. We believe it is a good idea to allow users to define their desired similarity within a cluster and allow them to have some flexibility to adjust the similarity if the adjustment is needed.

Presently the problem of clustering is usually not associated with the problem of searching points in *d*-dimensional space. Even though there are some clustering algorithms as in [19]-[20] that use multidimensional indexing structures such as *R*-trees and *k-d* trees, we usually consider the two problems independently. Our clustering method, however, simultaneously produces both a search tree of *d*-dimensional points and clusters (hence, the title!). This salient feature is useful in several applications. For example, in Internet-based search engines, we often want to find out a cluster (i.e., a set of pages) to which a given point (i.e., a desired page) belongs.

In the following sections our clustering algorithm is first presented and proofs of some clustering properties are given. We subsequently discuss our search tree and provide proofs of its properties. Finally, we end our paper with a conclusion to summarize strengths of our methods and possible

Manuscript received May 12, 2004.

Sanpawat Kantabutra is with the Theory of Computation Group, Department of Computer Science, Chiang Mai University, Chiang Mai, 50200, THAILAND (corresponding author to provide phone: 053-943409; fax: 053-943433; e-mail: sanpawat@alumni.tufts.edu).

Chumphol Bunkhumpornpat is also a member of the Theory of Computation Group, Department of Computer Science, Chiang Mai University, Chiang Mai, 50200, THAILAND (e-mail: chumphol@chiangmai.ac.th).

improvements.

II. MST-BASED CLUSTERING ALGORITHM

A. Minimum Spanning Trees

Given a connected, undirected graph $G = (V,E)$, where V is the set of nodes, E is the set of edges between pairs of nodes, and a weight $w(u,v)$ specifying the weight of the edge (u,v) for each edge $(u,v) \in E$, the minimum spanning tree problem is to find an acyclic subset $T \subseteq E$ that connects all of the vertices and whose total weight $w(T) = \sum_{(u,v) \in T} w(u,v)$ is minimized. Several well-established algorithms exist to solve the minimum spanning tree problem [21]-[23].

B. Clustering Problem

A *clustering* of a set is a partition of its elements into subsets (clusters) according to some measure of optimality [24]. Although there are many variations of optimization criteria, they can be divided into two broad classes depending on whether some *inner-cluster* measure or *intra-cluster* measure is used. The main emphasis, however, is that elements in the same cluster should be more similar to one another than they are to elements in the other clusters. Several empirical and heuristic clustering algorithms exist to solve the clustering problem as in [25].

C. MST Clustering Model

It is obvious to us that a minimum spanning tree can be used to model a clustering of d-dimensional points. A point in the clustering problem can be represented by a corresponding node in the graph. A distance between a pair of points can be represented by a corresponding weighted edge. In [26] clustering problems under the two optimization criteria were considered and both maximum and minimum spanning trees were used in their algorithms. However, only points in two dimensions were considered in this paper. In addition, each of the optimization criteria was used separately. In other words, there were two kinds of clustering problems; one that minimizes the maximum intra-cluster distance (diameter) and the other maximizes the minimum inter-cluster distance.

Our algorithm is also based on the minimum spanning tree but is *not* limited to two-dimensional points. Moreover, our algorithm produces clusters of d-dimensional points with a guaranteed diameter and a *naturally* appropriate inter-cluster distance.

D. The Algorithm

Our MST-based clustering algorithm works on a data structure called "*clustering tree*". A clustering tree is a binary tree that assists the algorithm in the clustering process. We give the definition of a clustering tree and the recursive algorithm to build it.

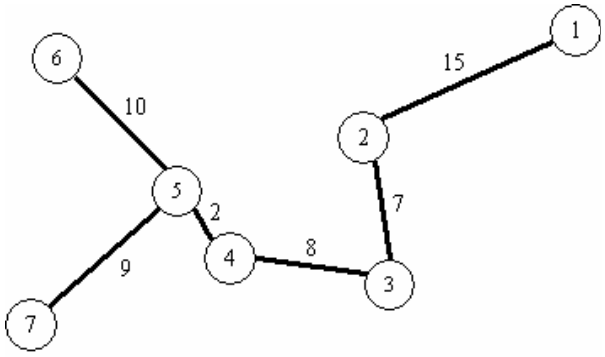

Figure 1: Minimum Spanning Tree

Definition 1: Given a minimum spanning tree T, a clustering tree is a binary tree in which each node contains, among other things, a distance between a pair of points a, b in T and the corresponding point identification numbers of a and b. Each parent node always contains a larger distance than that of its children.

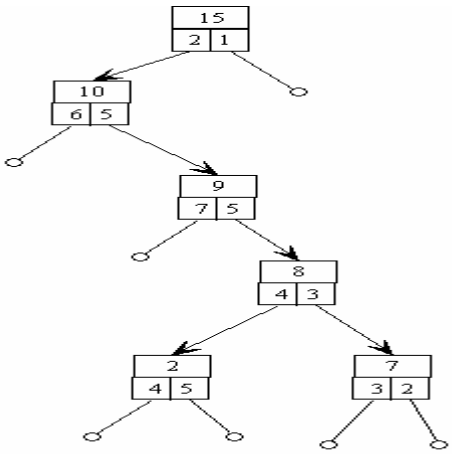

Figure 2: Clustering Tree

Figures 1 and 2 illustrate a minimum spanning tree and its corresponding clustering tree respectively.

Algorithm Clustering Tree
Input: a minimum spanning tree T and its root r.
Output: a clustering tree S.

```
Tree(T, r)
1.  if (T has some edge)
2.      e = get-max-edge(T)
3.      rs = make-blank-node()
4.      fill-info(rs, e)
5.      return connect(Tree(left-subtree(r), r.left-node-id),
                       Tree(right-subtree(r), r.right-node-id), rs)
6.  else
7.      rs = make-blank-node()
8.      return rs
```

The Clustering Tree algorithm recursively transforms a minimum spanning tree T to a corresponding clustering tree S.

It first checks to see if T still contains some edge (line 1). If so, it finds a maximum edge e and creates a blank root node rs of the tree S (lines 2-4). It then recursively solves the left and right minimum spanning subtrees (line 5) until T has no edge and S is returned (lines 7-8).

Theorem 1: The Clustering Tree algorithm takes $O(|E|^2)$ time.
Proof: We can derive a recurrence from the algorithm as follows. Let $T(|E|)$ be the number of steps of the algorithm.

$$T(|E|) = T(J) + T(|E| - (1 + J)) + |E|$$

where $1 \le J \le |E| - 2$ and $T(1) = 1$. From this recurrence, we can see that $T(|E|)$ is maximized when $J = 1$ or $J = |E| - 2$. Without loss of generality, let $J = 1$. We use the method of iteration to derive the following equation.

$$T(|E|) = I|E| - I(I-2) + T(|E| - 2I)$$

where I is an integer. This equation stops when $T(|E|-2I) = T(1)$ or $I = \frac{|E|-1}{2}$. Substitute $\frac{|E|-1}{2}$ for I in the equation. We have $T(|E|) = O(|E|^2)$. □

Once a clustering tree is constructed, the Clustering algorithm is used to get clusters of points. The Clustering algorithm can be described as follows.

Algorithm Clustering
Input: a clustering tree S and a desired diameter d.
Output: K clusters where K is also an output.
Clustering(S, d)
1. if (diameter(S) > d)
2. Clustering(left-subtree(root(S)), d)
3. Clustering(right-subtree(root(S)), d)
4. else
5. root(S).cluster-number = cluster-number()
6. print-cluster-number()
7. traverse-and-print(S)
8. return

The Clustering algorithm firstly checks to see if the diameter of all points in S is greater than the given d (line 1). If so, it recursively solves the subproblems at both left and right subtrees (lines 2-3) until the diameter becomes less than or equal to d. It then proceeds to record the cluster number in the root node of the current subtree for the current cluster (line 5). Note that function *cluster-number()* in line 5 generates a sequence of numbers and is a global function. The algorithm reports the cluster number and its members in lines 6-7.

Theorem 2: The Clustering algorithm produces K clusters, each of which has a diameter ≤ d.
Proof: In lines 1-3 the algorithm will continue to separate a larger tree into two smaller subtrees if the diameter of points in the larger tree is greater than d. In lines 4-6 the algorithm outputs points in a cluster only when the diameter of points in the cluster is smaller than or equal to d. □

Theorem 3: The Clustering algorithm takes $O(n^3)$ time.
Proof: Let n be the number of nodes in the clustering tree. In the worst case, the clustering tree S is recursively divided into two subtrees of sizes $n - 1$ and 1 until each point becomes a singleton cluster (i.e., d is smaller than the least weight of all edges in the minimum spanning tree). Because the cost of checking the diameter of the tree is n^2, we can derive a recurrence as follows. Let $T(n)$ be the number of steps in the algorithm.

$$T(n) = T(n - J) + T(J) + n^2$$

$T(n)$ is maximized when $J = 1$ or $J = n - 1$. Without loss of generality, let $J = 1$. Using the method of iteration, we can derive the following summation.

$$T(n) = \sum_{i=0}^{n-2} [(n-i)^2] + n$$
$$= \frac{1}{6}[2n^3 + 3n^2 + 7n - 6]$$
$$T(n) = O(n^3)$$

Thus, the Clustering algorithm takes $O(n^3)$ time. □

In section II we mentioned that our algorithm generates a naturally appropriate inter-cluster distance. The inter-cluster distance is the distance in the *parent* node of the root in line 5 in the Clustering algorithm. This distance tells us the *least* amount of similarity that points between the two clusters differ. This piece of information can be very useful in several applications.

III. Search Tree

In some clustering applications, given a point, users often need to query its cluster membership. A clustering tree according to definition 1 can also be a search tree for this purpose. Currently, most clustering algorithms do not provide a search mechanism for cluster membership. The ones that do provide are rather simplistic. K-means algorithm and its variants, for instance, use the nearest distance between the query point and the cluster center to indicate the point's cluster membership. This can be misleading since a cluster in K-means scheme does not have a well-defined structure. The cluster membership identification in K-means and its variants only works well with spherical clusters with an equal size. This lack of a well-defined structure forces a search process in K-means clusters to have a linear time with respect to the number of points in a cluster.

Our Clustering Tree algorithm, on the other hand, produces

a search tree with a well-defined structure that enables the search time to be proportional to the height of the tree. If the tree is balanced, the height will be logarithmic. The following is the definition of the search algorithm.

Algorithm Search
Input: a clustering tree *S*, its root node *r*, and a query point *p*.
Output: cluster membership number of the point *p*.
Search(*S*, *r*, *p*)
1. if (*r*.cluster-number = nil)
2. if (*d*(*r*.left-point, *p*) > *d*(*r*.right-point, *p*))
3. Search(right-subtree(*r*), *r*.right-node, *p*)
4. else
5. if (*d*(*r*.left-point, *p*) < *d*(*r*.right-point, *p*))
6. Search(left-subtree(*r*), *r*.left-node, *p*)
7. else
8. print(*r*.cluster-number)
9. return

In this algorithm $d(x, y)$ is an Euclidean function that returns the distance between points x, y. We also assume there is *no case* in which $d(r.\text{left-point}, p) = d(r.\text{right-point}, p)$. The algorithm begins by checking the cluster number of r (line 1). If it is *nil*, it recursively searches one of the two subtrees, depending upon the nearer distance to p (lines 2-6). The algorithm continues until it finds a cluster number in r, which indicates the cluster membership of p (line 8). Please note that the algorithm returns a cluster membership number, even though in actuality the point p may *not* exist in the cluster. In other words, it returns the cluster number to which the point p should belong.

Lemma 1: Let S be a clustering tree, r the root node of S, and p a given point. If $d(y, p) < d(z, p)$ where y is one point in root node r and z is the other point in root node r, then $d(x_1, p) < d(x_2, p)$ for all points x_1 in the same subtree as y and for all points x_2 in the same subtree as z.
Proof: We will prove it by contradiction. Suppose there exist some point x_1 in the same subtree as y and some point x_2 in the same subtree as z such that $d(x_1, p) > d(x_2, p)$ (i.e., we do not consider the equal case). Because $d(x_1, p) > d(x_2, p)$, p must be in the same subtree as x_2 for otherwise the edge incident with p in the corresponding minimum spanning tree would not be minimized. It follows that there must exist a *minimum* edge (m, p) in the corresponding minimum spanning tree for some point m in the same subtree as z. If $m = z$, $d(y, p) < d(z, p)$. If $m \neq z$, we know that $d(y, p) < d(m, p) = d(z, p) + \varepsilon$ for some constant ε. Hence, this is a contradiction since $d(m, p)$ is not minimized. Lemma 1 is true. □

Theorem 4: The Search algorithm is correct.
Proof: The proof is an induction on the depth of the clustering tree with the application of lemma 1. □

IV. Conclusion

We have presented a data structure called the clustering tree. As the title of this paper suggests, the benefit of this clustering tree is twofold. This clustering tree can be used with our clustering algorithm to produce a set of similarity-guaranteed clusters, which is a prominent feature of this algorithm. The same clustering tree can also be used efficiently as a search tree. All of these look nice from theoretical perspectives. However, from practical points of view, there is still some room for improvement for the running time of the clustering algorithm. This could perhaps be accomplished by using some appropriate data structure. In our search algorithm we impose that there is no case in which $d(r.\text{left-point}, p) = d(r.\text{right-point}, p)$. In practice this may not be realistic but this problem can be easily solved by little modification. If the equality case occurs, users can reenter a new point q where q is the old point $p + \iota$ where ι is some very small vector that only increases the old point p a bit. Since the original point p is changed only a little, the cluster membership number should be roughly the same as that of the original p. Even though this does not work in all cases, most clustering applications do not require exact cluster membership.

References

[1] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. 5th Berkeley Symp. Math. Statist, Prob*., 1967, vol. 1, pp. 281-297.
[2] L. Kaufman and P.J.Rousseeuw, Finding Groups in Data: An Introduction to Cluster Analysis. New York: John Wiley & Sons, 1990.
[3] Z. Huang. "Extensions to the k-means algorithms for clustering large data sets with categorical values," *Data Mining and Knowledge Discovery*, vol. 2, pp. 283-304, 1998.
[4] S. L. Lauritzen. "The EM algorithm for graphical association models with missing data" *Computational Statistics and Data Analysis*, vol. 19, pp. 191-201, 1995.
[5] P. Bradley, U. Fayyad, and C. Reina. "Scaling clustering algorithms to large databases," in *Proc. 1998 Int. Conf. Knowledge Discovery and Data Mining*, New York, Aug. 1998, pp. 9-15.
[6] R. Ng and J. Han. "Efficient and effective clustering method for spatial data mining," in *Proc. 1994 Int. Conf. Very Large Data Bases*, Santiago, Chile, Sept. 1994, pp. 144-155.
[7] T. Zhang, R. Ramakrishnan, and M. Livny. "BIRCH: An efficient data clustering method for very large databases," in *Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data*, Montreal, Canada, June 1996, pp.103-114.
[8] S. Guha, R. Rastogi, and K. Shim. "CURE: An efficient clustering algorithm for large databases," in *Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data*, Seattle, WA, June 1998, pp.73-84.
[9] G. Karypis, E.-H. Han, and V. Kumar. "CHAMELEON: A hierarchical clustering algorithm using dynamic modeling" *COMPUTER*, vol. 32, pp. 68-75, 1999.
[10] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. "A density-based algorithm for discovering clusters in large spatial databases," in *Proc. 1996 Int. Conf. Knowledge Discovery and Data Mining*, Portland, OR, Aug. 1996, pp. 226-231.
[11] M. Ankerst, M. Breunig, H.-P. Kriegel, and J. Sander. "OPTICS: Ordering points to identify the cluster structure," in *Proc. 1999 ACM-SIGMOD Int. Conf. Management of Data*, Philadelphia, PA, June 1999, pp.49-60.
[12] A. Hinneburg and D. A. Keim. "An efficient approach to clustering in large multimedia databases with noise," in *Proc. 1998 Int. Conf. Knowledge Discovery and Data Mining*, New York, Aug. 1998, pp. 58-65.

[13] W. Wang, J. Yang, and R. Muntz. "STING: A statistical information grid approach to spatial data mining," in *Proc. 1997 Int. Conf. Very Large Data Bases*, Athens, Greece, Aug. 1997, pp. 186-195.
[14] G. Sheikholeslami, S. Chatterjee, and A. Zhang. "WaveCluster: A multi-resolution clustering approach for very large spatial databases," in *Proc. 1998 Int. Conf. Very Large Data Bases*, New York, Aug. 1998, pp. 428-439.
[15] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. "Automatic subspace clustering of high dimensional data for data mining applications," in *Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data*, Seattle, WA, June 1998, pp.94-105.
[16] D. Fisher. "Improving inference through conceptual clustering," in *Proc. 1987 AAAI Conf.*, Seattle, WA, July 1987, pp. 461-465.
[17] T. Kohonen. "Self-organized formation of topologically correct feature maps," *Biological Cybernetics*, vol. 43, pp. 59-69, 1982.
[18] R. Kannan, S. Vempala, and Adrian Vetta. "On Clusterings: Good, Bad, Spectral," in *Proc. Of the 41st Foundations of Computer Science*, Redondo Beach, 2000.
[19] E. Knorr and R. Ng. "A unified notion of outliers: Properties and computation," in *Proc. 1997 Int. Conf. Knowledge Discovery and Data Mining*, Newport Beach, CA, Aug. 1997, pp. 219-222.
[20] E. Knorr and R. Ng. "Algorithms for mining distance-based outliers in large datasets," in *Proc. 1998 Int. Conf. Very Large Data Bases*, New York, Aug. 1998, pp. 392-403.
[21] J. B. Kruskal. "On the shortest spanning subtree of a graph and the traveling salesman problem," in *Proc. Of the American Mathematical Society*, 1956, vol. 7, pp. 48-50.
[22] R. C. Prim. "Shortest connection networks and some generalizations," *Bell System Technical Journal*, vol. 36, pp. 1389-1401, 1957.
[23] Robert E. Tarjan. *Data Structures and Network Algorithms*. Society for Industrial and Applied Mathematics, 1983.
[24] J. A. Hartigan, *Clustering Algorithms*, John-Wiley, New York, 1975.
[25] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*, San Francisco: Morgan Kaufmann Publishers, 2001, ch. 8.
[26] T. Asano, B. Bhattacharya, M. Keil, and F. Yao. "Clustering algorithms based on minimum and maximum spanning trees," in *Proc. Of the 4th Annual Symposium on Computational Geometry*, Urbana Champaign, IL, Jan. 1988, pp. 252-257.