Performance of Two Algorithms In Minimum Sum of Diameters Clustering

Zubair Noman
Department of Computer Science
St. Cloud State University
znoman@eeyore.stcloudstate.edu

Mynul Khan
Department of Computer Science
St. Cloud State University
mkhan@eeyore.stcloudstate.edu

Abstract:

Minimum sum of diameters clustering can be solved by reduction to determination of the satisfiability of a 2- Conjunctive Normal Form or 2-SAT statement. Hansen provided an algorithm that solved $O(nlogn)$ 2-SAT instances and ran with time complexity $O(n^3logn)$ on a graph of size n. Sarnath provided an improved dynamic digraph algorithm that solved $O(m)$ 2-SAT instances on a graph with m edges and ran with time complexity $O(n^3)$. Algorithms were implemented to partition instances of a complete graph having n=50 to n=500 vertices where all the vertices are connected with each other. In tests of the two algorithms, the Sarnath algorithm consistently performed better to find the edges of largest length of two clusters such that the sum is minimized.

1 Introduction:

A general question facing researchers in many areas of inquiry is how to organize observed data into meaningful structures, that is, to develop taxonomies. Clustering of data helps to solve this problem by partitioning large sets of data into clusters of smaller sets of similar data. Clustering of data can be done in different ways. Most often the difference between a pair of entities is considered to partition the entire data set [1]. The maximum dissimilarity between any two entities within one cluster is called the Diameter of the cluster and the minimum dissimilarity between entities in one cluster with any other outside it is called the Split of the cluster. The idea of using Diameter and Split as a measure of clustering was suggested by Delattre and Hansen [2]. There are different algorithms that differ in how the distance between two clusters is computed. Average linkage clustering uses the average similarity of observations between two groups as the measure between the two groups. Complete linkage clustering uses the furthest pair of observations between two groups to determine the similarity of the two groups. Single linkage clustering, on the other hand, computes the similarity between two groups as the similarity of the closest pair of observations between the two groups. But it is well known that minimum diameter partitions suffer from the dissection effect. In which case, very similar entities may be assigned to different clusters [1, 2, 3]. But instead of minimizing the diameter of a cluster, if the sum of diameters of clusters is minimized, the dissection effect is much less damaging [1].

That is why, where homogeneity of the clusters are desirable, partitioning the entities in clusters such that the sum of the diameters of the clusters are minimized is much more effective. Brucker showed that the problem of partition a set of entities in more than two clusters such that the sum of the diameters is minimized is NP-complete [4]. Hansen provided a $O(n^3 logn)$ algorithm for the problem when a set of entities are partitioned into two clusters [1]. Sarnath provided another algorithm based on dynamic digraph connectivity that improves the time complexity of Hansen algorithm by a factor of $O(logn)$. Both Hansen and Sarnath algorithm performs operation on graphs where the entities are represented by the vertices and the dissimilarity between any pair of entities are represented by the weight of the edge that connects the vertices representing the entities. In this paper, the performance of Hansen and Sarnath algorithm on a set of graphs of size 50 to 100 are presented where Sarnath algorithm consistently performed better. The rest of the paper is organized as follows: a description of the problem is given in section two. A generic procedure to solve the problem in which Hansen and Sarnath algorithm works is presented in section three. Hansen and Sarnath algorithm and their implementation is described in section four. In section five, the performance of these two algorithms on a set of graph is compared. Conclusions are drawn in section six.

2 The Problem:

A set of n entities and the dissimilarity between a pair of entities can be represented by a graph G=(V, E) with n vertices with the length of edges representing the dissimilarity between the vertices it connects. In a complete graph, dissimilarity between each pair of vertices is represented by the length of the edge connecting them. We want to partition the set of vertices into two clusters C_0 and C_1 such the diameter of cluster C_0 is equal to some value d_0 and the diameter of cluster C_1 is equal to d_1. By definition of diameter, there is no pair of vertex in C_0 the connecting edge between them has length greater than d_0 and there is no pair of vertices in C_1 for which the connecting edge between them has length greater than d_1. Without loss of generality it can be assumed that $d_0 >= d_1$. So, C_0 is the cluster with bigger diameter and C_1 is the cluster with smaller diameter. Minimum sum of diameters clustering problem finds optimal d_0 and d_1 such that (d_0+d_1) is minimum for which there is a partition of the vertices in two clusters. Such a partition is called optimal.

3 A Generic Algorithm:

Before explaining the steps of a generic algorithm to find the optimal diameters, let us formulate a relation between the diameters, length of edges and the vertices of the graph. Assume we have a graph with a set of vertices $V=\{x_1, x_2, ..., x_n\}$ and a set of edges $E=\{e_1, e_2, ..., e_m\}$. We want to partition the vertices in two clusters C_0 and C_1 such that the diameter of cluster C_0 is some edge length d_0 and the diameter of cluster C_1 is some edge length d_1. The length of some edge 'e_{ij}' between vertices i and j is denoted d_{ij}. Let's associate a boolean variable x_i to each vertex such that:

$x_i = 0$ if x_i is in C_0
$x_i = 1$ if x_i is in C_1

For any d_0, d_1 pair, the partition needs to satisfy the following three invariants:

1) If for some edge e_{ij}, $d_{ij}>d_0$, i and j both cannot be in the same cluster. For the vertices i and j, we can assign the following boolean values: $x_i=0$, $x_j=1$ or $x_i=1$, $x_j=0$.
2) If for some edge e_{ij}, $d_0>=d_{ij}>d_1$, both i and j cannot be in cluster C_1. Either $x_i=0$, $x_j=0$, or $x_i=0$, $x_j=1$ or $x_i=1$, $x_j=0$.
3) If for some edge e_{ij}, $d_1>=d_{ij}$, the vertices can both be in C_0, or C_1 or they can be in separate clusters.

To hold the first invariant true, apply a boolean conjunct: $(x_i \vee x_j)$ to every edge e_{ij}, wherever $d_{ij}>d_0$. Refer the conjunct as Type0 constraint.

To hold the second invariant true, apply a boolean conjunct: $(x_i \vee x_j) \wedge (\neg x_i \vee \neg x_j)$ to every edge e_{ij}, wherever $d_0>=d_{ij}>d_1$. Refer the conjunct $(\neg x_i \vee \neg x_j)$ as Type1 constraint.

To hold the third invariant true, we need not to apply any constraint for edges e_{ij}, if $d_{ij}<= d_1$. There is no restriction on which cluster i or j would belong.

The following boolean table expresses this relationship:

Table 1: Boolean Conjuncts for Diameter Restriction

b	**c**	**Øb**	**Øc**	**b v c**	**Øb v Øc**	**(b v c) Ù (Øb v Øc)**
0	0	1	1	0	1	0
0	1	1	0	1	1	1
1	0	0	1	1	1	1
1	1	0	0	1	0	0

Writing for each pair (x_i, x_j) of vertices the binary relation which is implied by the value of d_{ij} with respect to d_0 and d_1, we obtain a quadratic boolean equation (E) which is in the form of 2-Conjunctive Normal Form or 2CNF expression. From the definition of (E), it follows that the equation (E) is satisfiable or has a true assignment if there is a partition of vertices such that the diameter of the bigger cluster is d_0 and the diameter of the smaller cluster is d_1. If we can find all (d_0, d_1) pair of edges for which there is a partition of vertices into two clusters satisfying the invariants, we can find the optimal (d_0, d_1) pair in linear time.

The previous results yield the following generic algorithm, which receives a connected, undirected graph G=(V, E) with a weight function w: $E \rightarrow R$ as parameter:

Generic Algorithm (G, w)

1 Identify all edge lengths that are possible candidates for d_0 and d_1

2 For each candidate edge d_0, identify the smallest value d_1 such that there exists a partitioning of the graph into two clusters

3 Find the smallest pair of (d_0, d_1) for which the sum of diameters (d_0+d_1) is minimum

4 Algorithms of Hansen and Sarnath:

The two algorithms to find minimum sum of diameters follows the structure of the generic algorithm. They each use similar process to identify the set of possible candidates for d_0. They use different approach when it comes to search for smallest d_1 for each d_0 in step 2.

It follows that the set of edges that will be the candidates for d_0 can be found in the process of growing a maximum spanning tree. The only edges whose lengths are candidates for d_0 are the edges that completed the first odd cycle in the spanning tree and the edges included in the spanning forest before the first odd cycle was encountered [1]. All candidates for d_1 are lesser than or equal to the edge length that constitutes the first odd cycle [1].

Assume, the edges are sorted in non-increasing order having lengths {d_m,d_{m-1},...d_{min},....,d_2,d_1}. The first odd cycle is found in d_{min} in the process of growing a maximum spanning tree. The set of candidate d_0 lies on the left of d_{min} value, d_{min} included. The set of d_1 lies on the right of d_{min}, d_{min} included. Hansen performs a binary search on the set of d_1 to find the smallest d_1 for which the boolean expression (E) is true. Sarnath starts with the biggest d_0 value and performs an incremental search on the set of d_1 to find a satisfiable boolean expression (E) beginning with the smallest d_1. In both algorithms, sum of all (d_0, d_1) pair is stored for each d_0 in the set. The list is scanned to find the minimum sum of diameters. The (d_0, d_1) pair for which the sum is minimum is called optimal.

Hansen Algorithm:

Hansen algorithm as described in [1] operates in the following way. First, sort all the edges in non–increasing order and find the first odd cycle that occurs in growing the maximum spanning tree. Find the set of candidate d_0 and d_1 values. The edges that are equal to or bigger than d_{min} belongs to the set S_0. The edges that are equal to or smaller than d_{min} belongs to the set S_1. For each edge d_0 in S_0, proceed to a binary search on the ordered set of S_1 to find a d_1 value. For each (d_0, d_1) pair, scan the list of edges to construct a 2CNF expression (E) and check for the satisfiability of the 2CNF expression (E). If satisfiable, find next edge that is smaller than the current d_1 from the set S_1 by applying binary search, construct (E) and check satisfiability. If not, find next edge bigger than current d_1 by applying binary search, construct (E) and check satisfiability. Store the smallest d_1 for each d_0 for which the constructed boolean expression is satisfiable. Perform these operations for every d_0 in S_0. The minimum sum (d_0, d_1) can be found by searching the list of (d_0, d_1) pair for which the boolean expression was found satisfiable.

Finding d_{min}:

The first odd cycle in the growing of a maximum spanning tree can be found in $O(n^2)$ time [1]. Apply Kruskal's algorithm to grow maximum spanning tree. If the edge in question, forms an even cycle, the edge is ignored. If the edge forms an odd cycle, then we have found d_{min}.

Checking satisfiability:

Hansen uses the algorithm described by Aspvall to check satisfiability of the boolean expression constructed for some (d_0, d_1) pair [6]. Checking the consistency of quadratic boolean equation (E) defined on a set of n vertices can be done in polynomial time [1]. The algorithm adds a directed graph to represent a 2CNF expression referred as constraint graph [7]. The constraint graph is constructed as follows: For each vertex value i in the original graph, add two vertices i and ¬i where i and ¬i are complements of each other. For each constraint $(u \lor v)$, add edges $\neg u \rightarrow v$ and $\neg v \rightarrow u$ in the constraint graph. So, for each Type0 constraint, we add two edges that are directed from negated literals to non-negated ones. For each Type1 constraint, the algorithm adds two edges directed from non-negated literals to negated ones. The algorithm to check satisfiability of (E) relies upon identifying the strong components of the constraint graph. A 2CNF expression is satisfiable if and only if no vertex *i* is in the same strong component as its complement $\bar{i}$ [6]. Strongly connected components of a directed graph G=(V, E) can be computed in linear time using two depth–first search (DFS); one on the graph itself, and the other one on the transpose of G, which is defined as $G^T = (V, E^T)$, where $E^T = \{(u, v) : (v, u) \text{ is in } E\}$ [8]. That is, E^T consists of the edges of G with their directions reversed.

Hansen (G, w)

```
1    Grow maximum spanning tree to find $d_{min}$ which is the first odd cycle in growing
     the spanning tree
2    Identify $S_0$, the set of all candidate $d_0$ edges and $S_1$, the set of all candidate $d_1$
     edges
3    $d_1 \leftarrow$ nil
4    for each $d_0$ in $S_0$
5        $d_0$ find a candidate $d_1$ from $S_1$ in binary search fashion
6            construct a boolean expression (E) for $d_0$ and $d_1$
7            Check satisfiability of the expression
8            if satisfiable
9                then search for a lower value of $d_1$
10           else    search for a higher value of $d_1$
11           store smallest $d_1$ for each $d_0$
12   Choose ($d_0$, $d_1$) pair such that the sum of $d_0$ and $d_1$ is minimum
```

Sarnath Algorithm:

Sarnath algorithm uses properties from Dynamic Digraph Connectivity to solve Minimum Sum of Diameters Clustering. The algorithm differs from Hansen in two ways:

1) Instead of carrying out a binary search to find d_1, it is found from a sequential search.
2) Instead of solving each 2CNF instance from scratch, it is done incrementally/decrementally for each new value of d_0 and d_1 [7].

The algorithm makes improvements on how the satisfiability of 2-SAT boolean expression is solved. The algorithms choose d_0 from the edge with largest length from S_0, and choose d_1 from the edge with smallest length from S_1. Each time a boolean expression (E) is found not satisfiable the algorithm chooses the next bigger edge from S_1 for d_1 until a satisfiable expression is found. Each time the expression (E) is found true, the next smaller edge is selected from S_0 for d_0. But instead of constructing the constraint graph from scratch for each 2CNF expression and solving for satisfiablity, the constraint graph is updated dynamically based on the following change in 2CNF expression:

1) Each time a bigger e_{ij} is chosen from S_1 for d_1 value, remove the Type1 constraint from e_{ij}.
2) Each time a smaller d_0 is chosen from S_0 for d_0 value, add Type0 constraint to the last edge considered for d_0.

For an initial graph of *n* vertices, the constraint graph will contain *2n* vertices. The algorithm operates on a data structure which is a matrix of size *(n x n)* that represents the constraint graph. The algorithm maintains transitive closure of the constraint graph at all time and checks for cycles in the constraint graph to find satisfiability. So, for each change in 2CNF expression (E), the matrix is updated and satisfiability is checked by querying the matrix.

5 Performance:

Big- O Analysis:

Hansen algorithm ranks the edges of the initial graph in decreasing order. For a complete graph, sorting the edges takes $O(n^2 logn)$ time for a graph with n vertices and $O(n^2)$ edges. The construction of the maximum spanning tree from the initial graph takes $O(n^2)$ time. Hansen algorithm checks at most *logn* d_1 instance and there can be at most *n* d_0. So, in the worst case, the algorithm checks *O(nlogn)* pair of (d_0, d_1). The entire algorithm takes $O(n^2logn)+O(nlogn)*O(n^2)=O(n^3logn)$ time. This is the overall time complexity of Hansen algorithm.

On the other hand, Sarnath algorithm pre-computes the matrix in $O(n^3)$ time [7]. It computes the most persistence path between all pair of vertices and considers all other vertices as intermediate vertices on the path. The calculation is analogous to Floyd-Warshall's all pair shortest path algorithm [8]. The data structure cleverly maintains the matrix so that every time the boolean expression is updated, the edges need to be deleted can be identified in constant time. Each time a edge is inserted, it however takes $O(n^2)$ time to update the matrix [7]. At every insertion, the algorithm calculates any new path between all pair of vertices that might have been created considering the new vertices connected between the new edge as intermediate vertices. The overall time complexity of Sarnath algorithm is $O(n^3)$.

CPU Performance:

The algorithms were implemented in C++. The programs were compiled using GNU g++ compiler on a Digital AlphaServer 1000A 4/266 running Digital UNIX v4.0D with 256 MB of RAM. In comparing the performance of two programs, CPU cycle used by each program is counted. The CPU cycle used by Hansen algorithm is almost three times than what was used by Sarnath algorithm when tested on a complete graph with 30 vertices.

The cycle used by Hansen algorithm was 2 ½ times when testing on a graph with 40 vertices. Both algorithms run in polynomial time and the CPU cycle used increases very rapidly even for a very small graph.

Table 2: CPU cycle used by Two Algorithms

Number Of Vertices	**Hansen's Algorithm (In Millions)**	**Sarnath's Algorithm (In Millions)**
10	1	.08
20	10	1.6
30	30	7.5
40	98	38

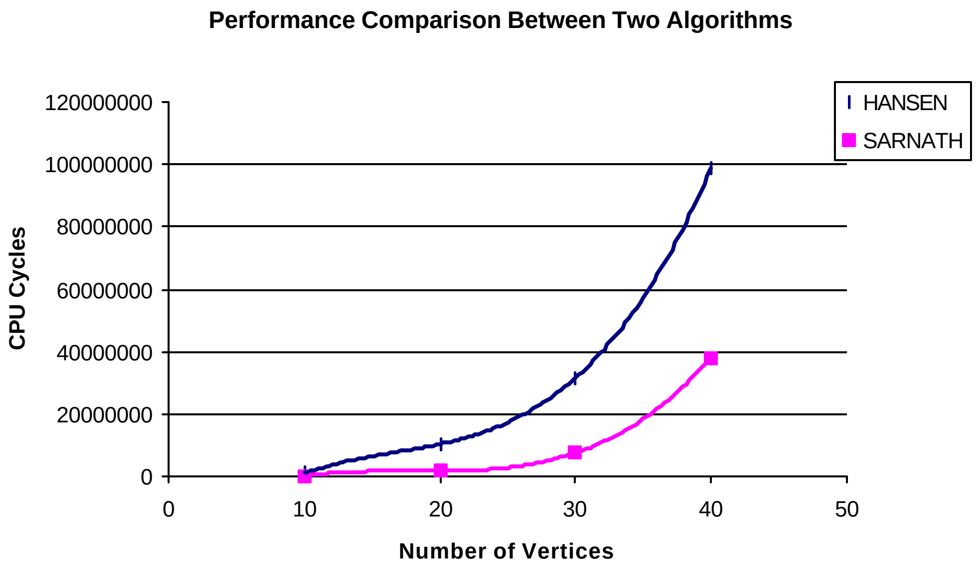

Figure 1: Graph of CPU Performance

6 Conclusion:

Both Hansen and Sarnath algorithm runs in polynomial time. Even though, Sarnath algorithm improves Hansen algorithm by a factor of *O(logn)*, the run time complexity of both the algorithms is a polynomial degree of three which makes it quite impossible to test them even on small graphs of size 100 on the machine where the algorithms were tested on. The time and space requirement of both the algorithms are quite high. The hidden constant factor of the big-O analysis for Hansen algorithm is higher than that of Sarnath due to the fact that it constructs constraint graph and finds satisfiability of the boolean expression every time a new (d_0, d_1) pair is chosen. Whereas, Sarnath does this incrementally/decrementally and only operates on one single matrix of size *(n x n)*. The resource limitation also prohibited testing on instances of large graphs. However, the algorithms were not tested against any heuristics and the performance of these two algorithms against any such heuristic is yet to be compared.

References:

1. Hansen, P and Jaumard, B. (1987). Minimum Sum of Diameters Clustering. *Journal of Classification, 4,* 215-226.

2. Delattre, M. and Hansen, P. (1980). Bicriterion Cluster Analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI 2(4)*, 277-291.

3. Cormick, R. (1971). A Review of Classification. *Journal of the Royal Statistical Society, A(134)*, 321 - 367.

4. Brucker, P. (1978). On the Complexity of Clustering Problems. *Optimization and Operations Research, Lecture Notes in Economics and Mathematical Systems*, 45-54.

5. Hartigan, John. (1975). Clustering Algorithms. New York: John Wiley and Sons.

6. Aspvall, B. et al. (1979). A Linear-Time Algorithm for Testing the Truth of Certain Quantified Boolean Formulas. *Information Processing Letters, Vol 8*, 3.

7. Sarnath, R. Dynamic digraph connectivity hastens minimum sum-of-diameters clustering. To Appear in SWAT 2002.

8. Cormen, T. et al. (2000). Introduction to Algorithms. Massachussetts: MIT Press.