MINIMUM SPANNING TREES

Project INF442

7 octobre 2019

Luiz Bezerra Mathews Lucas

Table des matières

1 Motivation

Spanning trees and Minimum Spanning Trees (MSTs) are fundamental objects in graph theory, and are used in many different applications. In computer science, for instance, MSTs are useful to perform flooding in (wired) computer networks with a minimum number of transmissions. Protocols such as the Spanning Tree Protocol (STP)[8] or the standard IEEE 802.1d build a spanning tree overlay on top of an Ethernet network (layer-2), in order to avoid routing loops. Multicast protocols such as DVMRP $(DistanceVectorMulticastRoutingProtocol)$[1] also rely on spanning trees on top of the network graph (layer-3) to deliver multicast flows over a set of destinations.

MST is also used for data science purposes. This project explores the use of MST as a basis for clustering algorithms, and the performance of these algorithms with respect to other heuristics (e.g., k-means).

The project is organized in sections and tasks. These are provided to give you hints on aspects that can be interesting to look at. Tasks 1 and 2 are basic and required for the others; the rest can be explored depending on your interest and will be graded accordingly. You are encouraged to use the provided references and resources as pointers to develop your work.

2 Formal definitions

Given a graph G = (V, E), a spanning tree for G, denoted T, is a tree (i.e., a graph in which every pair of vertices is connected by way of one and only one path) such that every vertex v ∈ V is part of it. In other words, it is a subgraph T ⊂ G that contains all vertices in G, is connected and is acyclic.

Given a weighted graph G = (V, E, w), where w(e) is the weight of edge e ∈ E, the minimum spanning tree is the spanning tree of G with minimum weight.

3 Sequential implementation of classic algorithms

Several algorithms have been proposed to compute, if exists, the MST of a given graph. The main ones are $Kruskal$[7], $Prim$[6] and $Boruvska$[5].

3.1 Task 1

Describe and implement the Boruvska and Prim algorithms to compute the MST for a given general graph. Discuss their complexity. Test and discuss their performance with networks of different sizes (n).

3.1.1 • Prim

Prim's algorithm is a greedy algorithm that finds a minimum spanning tree for a weighted undirected graph.

The algorithm may informally be described as performing the following steps :

Initialize a tree with a single vertex, chosen arbitrarily from the graph. Grow the tree by one edge : of the edges that connect the tree to vertices not yet in the tree, find the minimum-weight edge, and transfer it to the tree. Repeat step 2 (until all vertices are in the tree).

It has a complexity of O((V+E)log V), for a graph represented as a adjacency *list*.[14]

Execution time of our prim implementation :

Number of Vertices	Execution time (ms)
50	0.288
100	1.184
200	4.923
500	33.744
1000	146.620
2000	667.502
10000	20707.972

3.1.2 • Boruvska

Borůvka's algorithm is a greedy algorithm for finding a minimum spanning tree in a graph for which all edge weights are distinct. The algorithm begins by finding the minimum-weight edge incident to each vertex of the graph, and adding all of those edges to the forest. Then, it repeats a similar process of finding the minimum-weight edge from each tree constructed so far to a different tree, and adding all of those edges to the forest. Each repetition of this process reduces the number of trees, within each connected component of the graph, to at most half of this former value, so after logarithmically many repetitions the process finishes. When it does, the set of edges it has added forms the minimum spanning forest. This way it has a complexity O(E log V).[15]

Execution time of our prim implementation :

Number of Vertices	Execution time (ms)
50	0.201
100	0.747
200	2.618
500	23.172
1000	105.106
2000	501.676
10000	13388.421

3.2 Task 2

Describe and implement Kruskal algorithm for a given general graph. As in the previous case, discuss its complexity, test it and evaluate it performance with networks of different sizes (n).
The Kruskal's algorithm is a minimum-spanning-tree algorithm which finds an edge of the least possible weight that connects any two trees in the forest, it is a greedy algorithm in graph theory as it finds a minimum spanning tree for a connected weighted graph adding increasing cost arcs at each step.
It's complexity is $O(E * log(E))$ since its most expensive task is to sort the array of edges by weight.[9]
The implementation of this task can be found in the file *graph.cpp* in the function $graph :: Kruskal()$.
Execution time of our kruskal implementation :

Number of Vertices	Execution time (ms)
50	0.326
100	1.266
200	5.734
500	34.845
1000	150.976
2000	726.445
10000	23818.124

4 Parallel implementation of MST algorithms, and fully distributed computation of MST

For parallel implementations, you can use the Message Passing Interface (MPI) standard and, in particular, the MPI library for C2.

Prim is the most intuitive approach to MST construction, but it is not easily parallelizable ; Boruvska algorithm is less intuitive, but presents better parallel characteristics : most of the research on parallel construction of MSTs relies on this work. Different parallelizations of these algorithms have been proposed in the literature ; we propose to explore some of them, e.g. the parallelized versions of Prim and Boruvska described by $Kumar$[2] **and** $Guang - rongetal.$[3]**, respectively (students may explore other possibilities from the existing literature).**

4.1 Task 3

Implement parallelized versions of Prim and Boruvska with MPI, and evaluate their communication cost and time performance for different computer networks

These algorithms (Prim, Boruvska) assume that a central instance has a complete view of the graph and can thus compute the MST. In a real computer network, the MST computation would require a full knowledge of the network topology, the resulting forwarding decisions would need to be implemented afterwards in each of the involved computers. Note that explored parallelized versions of the algorithms rely on the same setup – a central instance able to perform its computation by using several concurrent processors.

In practice, central computation with full knowledge of the graph is not feasible in many computer networks. Instead, distributed algorithms that can be used independently in each computer of the network are preferred. $Gallager$[4] **proposed a distributed computation of the Minimum Spanning Tree (MST) on a network, in which nodes only need to know about their neighbors in order to run the heuristic. Neighbor sensing is typically achieved in computer networks by allowing every node to periodically sending presence messages (denominated Hello messages) to neighbors reachable through all its interfaces (outgoing edges).**

The Prim's implementation with MPI(Message Passing interface) consists in dividing all the vertices among the processors, each processor should send the minimum distance between each vertex at his domain and the MST. The implementation of Boruvka algorithm is similar, but we divide the edges. Here some results for Prim's parrallel implementation, at first for 1000 vertices :

Processors	Execution time (ms)
2	0.004242
5	15.003981
10	47.443992

As we can see the algorithm can be very efficient for small values of Number of processors. This happens because we have just one computer in this network and so he can't manage to execute a lot of process in parallel. Now we are going to test this algorithm for bigger entries :

Number of vertices	Processors	Execution time (s)
2000	2	0.0.014741
2000	4	0.013466
10000	2	0.360749
10000	4	0.347698

Once again we can see that the algorithm is very powerful. We can compare him with Prim's classical, which resolve the same graph in 20.7079 s.

4.2 Task 4

Implement Gallager's algorithm using MPI. You can emulate neighbor sensing by enabling the root to distribute neighborhood information to every node in the network. Discuss the complexity of this algorithm and the communication costs that it incurs for a network with n nodes.

5 MST for clustering

MST can be used for data clustering. This requires that data items are connected through a graph. Given a d-multi-dimensional dataset S with distance δ (e.g. the Euclidian distance), in which items s ∈ S are vectors in $\mathbb{R}^d$, a graph G = (V, E, c) can be induced, in which V is the set of vertices, E is the set of edges, and c is the cost function for the edges.

- vertices v ∈ V are data items of S, and
- two vertices u and v are connected through an edge $e_{u,v}$ ∈ E. The cost of such an edge is c(u, v) = δ(u, v) in $\mathbb{R}^d$

Given a value k, the MST T over a graph G corresponding to a dataset S allows to determine k clusters by removing the k most "expensive" (with highest distance) edges in the MST, in a reduced graph $T^0 \subset$ T. By definition, T^0 consists of k connected components. The set of vertices of each connected component is a cluster of the MST-based clustering.

5.1 Task 5

Implement this MST-based k-clustering algorithm, by using the Kruskal algorithm for MST. Compare its complexity and performance with the k-means algorithm, with Forgy initialization, and discuss your results

The actual approach used was to remove the most "expensive" edges until we get k clusters. The difference is notorious when look into the special cases such that the most expensive edge connects the tree with a vertex of degree 1, and single vertex can not be considered as a cluster.

The implementation of this task can also be found in the file *graph.cpp* in the function $graph :: kClustering(int\ k, cloud * c)$ that receives also the cloud of points determined by the entry.

The complexity of this initial approach is $O(ElogV)$ as it is majored by Kruskal's complexity, and the k-means with its heuristic implementation (Lloyd's algorithm) have a complexity of $O(V^k * poly(V, k, d))$. Therefore, the performance of the MST-based k-clustering is better. We got some data-sets[10] to compare them, and we could

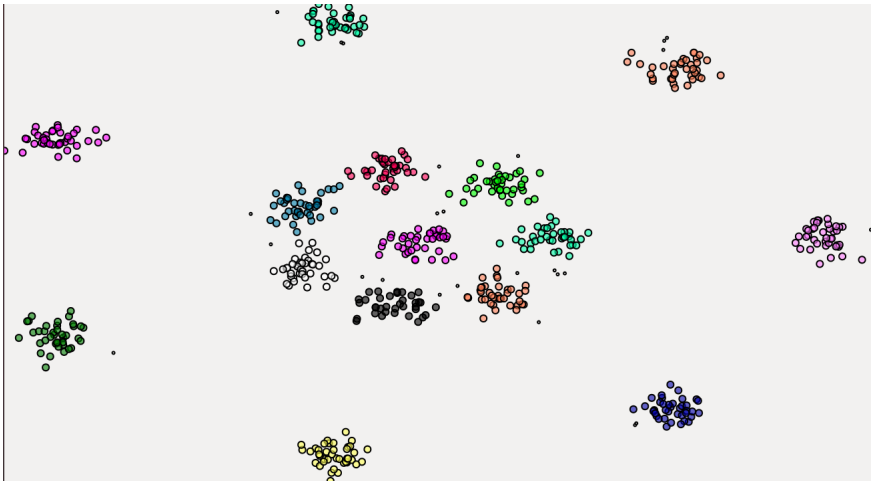

Figure 1 – MST-based k-clustering R15 data-set

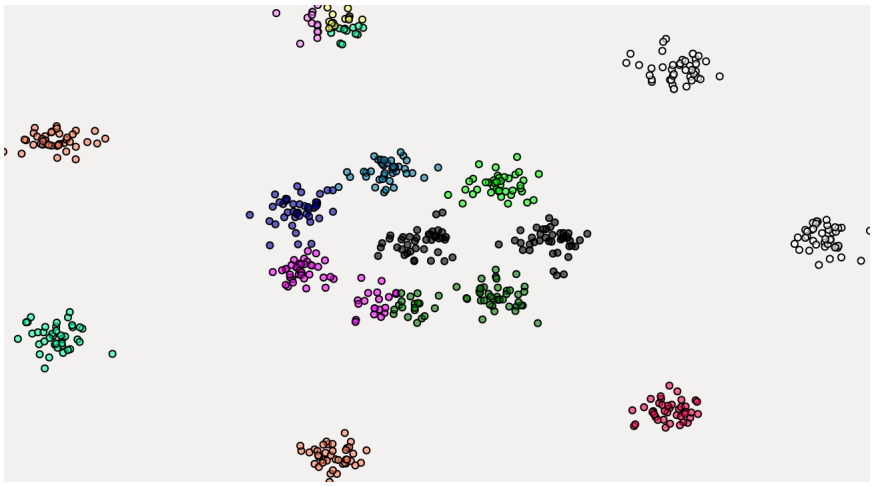

Figure 2 – k-Means R15 data-set

notice that the first one may not have an optimal performance in some cases, but in center-symmetric cases it has a really good performance, as shows the figures 1 and 2, others results can be found in the source of the project.

6 Extensions

The MST clustering algorithm sketched in section 5 takes the number of clusters, k, as input of the algorithm. Sometimes, however, the number of clusters in a dataset is not known in advance and needs to be inferred from the data itself

Different strategies can be proposed for MST-based clustering without an explicit value for k. One of the simplest ones consists of scanning the range of feasible k's, perform a MST k-clustering for each of them (as implemented in Task 5), and choose the optimal clustering according to a

given metric (e.g., smallest ratio between intra-cluster and inter-cluster distance). Others can be found in the literature : see e.g. [?].

6.1 Task 6

Explore the existing literatures for MST-based clustering strategies without explicit number of cluster (k). Implement one of these strategies and evaluate it against some of the available datasets (see section 6).
We have implemented 2 approachs :

1. A smallest ratio between intra-cluster and inter-cluster distance related
2. The Maximum Standard Deviation Reduction Clustering Algorithm[11]

For the first method we took the ratio between the maximum intra-cluster distance (the maximum distance between 2 points in a same cluster) and the minimum inter-cluster distance (defined as the minimum distance between 2 cluster, that is defined as the distance between their centers), but, as a choice, we divided this ratio for the square of the number of clusters and we aimed to minimize this result. We haven't really found this "correction" in the literature, but we found it pertinent as it gave some good results.
For the second method we made a direct implementation of the algorithm proposed by Grygorash.
We observed that the complexity of the first method is far better then the second one, and therefore the second one performs better in some contexts, we can see that in other contexts (as said, those center-symmetric examples) the first one have a really good performance. But the first one have a limitation in our algorithm, it's necessary to have some idea about the number of clusters, it is not necessary to know the exact number, but it's necessary to give a number maximum of cluster that is expected. Some results can be found in the source of the project.

7 References

[1] T. Pusateri (2003) : Distance Vector Multicast Routing Protocol. Internet-Draft.
https ://tools.ietf.org/html/draft-ietf-idmr-dvmrp-v3-11.

[2] A. Grama, A. Gupta, G. Karypis, V. Humar (2003). Introduction to Parallel Computing, Second Edition. Addison-Wesley, 2003. ISBN 0-201-64865-2.

[3] W. Guang-rong, G. Nai-jie (2000) : An efficient parallel minimum spanning tree algorithm on message passing parallel machine. J. Softw. 11(7), 889898 (2000).

[4] R. G. Gallager, P. A. Humblet, P. M. Spira (1983) : A Distributed Algorithm for Minimum-Weight Spanning Trees. ACM Transactions on Programming Languages and Systems, vol. 5, no. 1, Jan. 1983, pp. 66-77.

[5] M. Sollin (1965) : Le tracé de canalisation. Programming, Games, and Transportation Networks.

[6] R. C. Prim (1957) : Shortest connection networks and some generalizations, Bell System Technical Journal, 36 (6) : 13891401.

[7] J. B. Kruskal (1956) : On the shortest spanning subtree of a graph and the traveling salesman problem. Proceedings of the American Mathematical Society. 7 : 4850.

[8] Cisco Systems : Spanning Tree Protocol. http ://www.cisco.com/c/en/us/tech/lan-switching/spanningtree-protocol/index.html

[9] Kruskal's algorithm : https ://en.wikipedia.org/wiki/Kruskal%27s_algorithm

[10] P. Fränti and S. Sieranoja K-means properties on six clustering benchmark datasets Applied Intelligence, 48 [11] Oleksandr Grygorash, Yan Zhou, Zach Jorgensen (2006) : Minimum Spanning Tree Based Clustering Algorithms. Proc. 2006 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'06).

[12] P. Fränti and S. Sieranoja K-means properties on six clustering benchmark datasets Applied Intelligence, 48 (12), 4743-4759, December 2018 http ://cs.joensuu.fi/sipu/datasets/

[13] https ://github.com/natedomin/polyfit [14]https ://en.wikipedia.org/wiki/Prim[15]https ://en.wikipedia.org/wiki/
spanning-tree/blob/master/src/main.c