Boruvka Meets Nearest Neighbors

Mariano Tepper[1,**], Pablo Musé[2], Andrés Almansa[3], and Marta Mejail[4]

[1] Department of Electrical and Computer Engineering, Duke University
mariano.tepper@duke.edu
[2] Instituto de Ingeniería Eléctrica, Facultad de Ingeniería,
Universidad de la República
pmuse@fing.edu.uy
[3] CNRS - LTCI UMR5141, Telecom ParisTech
andres.almansa@telecom-paristech.fr
[4] Departamento de Computación, Facultad de Ciencias Exactas y Naturales,
Universidad de Buenos Aires
marta@dc.uba.ar

Abstract. Computing the minimum spanning tree (MST) is a common task in the pattern recognition and the computer vision fields. However, little work has been done on efficient general methods for solving the problem on large datasets where graphs are complete and edge weights are given implicitly by a distance between vertex attributes. In this work we propose a generic algorithm that extends the classical Boruvka's algorithm by using nearest neighbors search structures to significantly reduce time and memory consumption. The algorithm can also compute in a straightforward way approximate MSTs thus further improving speed. Experiments show that the proposed method outperforms classical algorithms on large low-dimensional datasets by several orders of magnitude.

1 Introduction

The computation of the minimum spanning tree (MST) is a classical problem in computer science. For an undirected weighted graph, it can be simply stated as finding a tree that covers all vertices, called a spanning tree, with minimum total edge cost. It is taught in every course of algorithms and data structure as an example where greedy strategies are successful and it is regarded as one of the first historical foundations of operations research.

Maybe the two most widely known algorithms to compute the MST are Prim's and Kruskal's [1]. There is a third classical algorithm by Boruvka [1] that mysteriously remained shadowed by the other two. This fact is emphasized by the

* Supported by FREEDOM (ANR07-JCJC-0048-01), CNES (R&T Echantillonnage Irregulier DCT/SI/MO - 2010.001.4673), Callisto (ANR-09-CORD-003), ECOS Sud U06E01, ARFITEC (07 MATRH), STIC Amsud (11STIC-01 - MMVPSCV), and the Uruguayan ANII (PR-POS-2008-003).

** Work partially done while M. Tepper was with the Departamento de Computación at the Universidad de Buenos Aires.

fact that Boruvka's algorithm is also known as Sollin's algorithm, despite the fact that Sollin re-discovered it independently years later.

The MST is particularly interesting for many data analysis tasks in computer vision and pattern recognition. A clear example is clustering, where the classical single-linkage hierarchical algorithm [2] can be proven equivalent to computing the MST. In a seminal work, Zahn [3] studied the benefits of using the MST for clustering. More recently, the MST received attention due to the growth in the size of clustering datasets, e.g., [4,5]. The approximate MST (AMST), suboptimal but faster, also received attention for the same reasons [6].

We now slightly change the definition of the problem to a form more suitable for data analysis (e.g., clustering). Let M be a set and $d : M \times M \to \mathbb{R}^+$ a distance function. Then d and the pair (M, d) are said to be a metric on M and a metric space, respectively. Given a data set $X \subseteq M$, the MST of X is defined as the MST of the weighted undirected graph $G = (V, E)$ where each $v_i \in V$ is identified with a feature $x_i \in X$, $E = V \times V$ (the graph is complete), and the graph's weighting function $\omega : E \to \mathbb{R}$ is defined as $\omega((v_i, v_j)) = d(x_i, x_j)$.

The problem is classically addressed by using metric spaces with exploitable specific characteristics like the Euclidean space, e.g., the Euclidean MST is contained in the Delaunay triangulation of X [7]. Recent work has aimed at building an AMST [6] through a clever use of space-filling curves.

Nearest neighbors (NNs) search structures have been used to compute the MST [8]. The approach proved successful; moreover, using such structures allows in addition to compute the AMST in a natural and straightforward way. A revision of this approach is needed, in the light of novel NNs techniques and increasing computational power. More recently, Leibe et al. [9] used NNs techniques for hierarchical clustering using the average-link criterion. Although they improved the method's performance, their algorithm is not suitable for extremely large datasets.

Classical algorithms for computing the MST run in $\mathrm{O}(n^2 \log n)$, where $n = |X|$. However, one must compute all $n(n-1)/2$ distances and thus a double-sided problem appears: (1) storing all $n(n-1)/2$ results for $n \geq 10^5$ is prohibitive; (2) even if results are not stored, for $n \geq 10^5$ the overall running-time is also prohibitive. Keep in mind that, in modern pattern recognition applications, feature sets of 10^5 or more points are becoming common [10]. In this work we address the MST problem without computing all distances in E. We build on Boruvka's approach [1] by an appropriate use of NNs search techniques.

The rest of the paper is structured as follows. In Section 2 we propose a general approach to compute the MST using NNs search structures. Section 3 shows empirical results of the proposed approach on a synthetic dataset. Finally, some final remarks and future work are presented in Section 4.

2 A Nearest Neighbors Approach

First let us explain Boruvka's algorithm: it creates a forest (i.e., a set of trees) where each isolated edge is a tree and gradually merges these trees by adding

Algorithm 1: Computation of the MST $T = (V, E_T)$ of feature set X.

1 $E_T \leftarrow \emptyset$;
2 **while** $|E_T| < |V| - 1$ **do**
3     $E' \leftarrow \emptyset$;
4   **foreach** *connected component* C *of* T **do**
5         $(u_m, v_m) \leftarrow \underset{u \in C,\ v \notin C}{\arg\min}\ d(u, v)$;
6         $\delta_m \leftarrow d(u_m, v_m)$;
7         $E' \leftarrow E' \cup \{(u_m, v_m, \delta_m)\}$;
8   **while** $E' \neq \emptyset$ **do**
9         $(u_m, v_m, \delta_m) \leftarrow \underset{(u,v,\delta) \in E'}{\arg\min}\ \delta$;
10         $E' \leftarrow E' \setminus \{(u_m, v_m, \delta_m)\}$;
11   **if** $E_T \cup \{(u_m, v_m, \delta_m)\}$ *does not contain cycles* **then**
12             $E_T \leftarrow E_T \cup \{(u_m, v_m, \delta_m)\}$

the smallest edge whose endpoints lie on different trees (see Algorithm 1). We propose to express the term in line 4 of Algorithm 1 in terms of finding NNs in the set $V \setminus C$:

$$u_m = \underset{u \in C}{\arg\min}\, d(u, \mathrm{NN}_d(V \setminus C, u)), \tag{1}$$

$$v_m = \mathrm{NN}_d(V \setminus C, u_m), \tag{2}$$

where $\mathrm{NN}_d(A, b)$ returns the NN $a \in A$ of b using metric d. We also modify the function $\mathrm{NN}_d(A, b)$ by adding an additional constraint function $\rho : X \to \{0, 1\}$ on the returned element. We denote it by $\mathrm{NN}_{d,\rho}(A, b)$. It returns the NN $a \in A$ of b using metric d such that $\rho(a) = 1$. By setting $\rho(v) = (v \notin C)$ we have

$$\mathrm{NN}_d(V \setminus C, u) = \mathrm{NN}_{d,\rho}(V, u). \tag{3}$$

This kind of problem is sometimes referred to as Foreign NNs in the literature.

We are sure that the desired node v_m is among the k NNs of u where $k = |C| + 1$. Therefore in the worst case, using a naive approach, $\mathrm{NN}_{d,\rho}$ amounts to perform a k-NNs search and then a simple check among them by using ρ. Note that k is a dynamic (growing) quantity and it is not possible to fix it in advance. The problem is thus of a different nature than finding the MST in a constrained degree graph. Of course, there is no need to compute that many NNs, since the constraint can be directly incorporated in the NN technique.

Priority queues can be used to prune the number of NNs searches performed during the algorithm [8]. We propose to use several priority queues, one for each connected component in a partial (i.e., already computed) MST. The nodes u_i of a partial MST are stored, with their foreign NNs u_j, in a priority queue where the priority of a node is the inverse of $d(x_i, x_j)$. The use of a priority queue is indeed interesting in this context, as the next edges to add to the MST are at the top of the priority queues. The top of the queues are removed and the top-priority foreign NNs are added to the MST. After merging two connected components, their priority queues are also merged.

Additionally, the priority queue must be updated, since disjoint connected components are merged and some foreign NNs might not be foreigners anymore. Note that it may not be necessary to update the entire priority queue. This is because the current priority of each of these nodes (the priority before the insertion in the MST) serves as an upper bound of its real priority (the priority after the insertion in the MST). The real priority of a node needs only to be computed when its current priority is on the top of the queue.

We omit the pseudocode of the resulting algorithm because of space constraints, see [11] for further details. Note that the space complexity is still $O(n)$. In the first iteration, there are n queues, each of length 1. In the second iteration there are roughly $n/2$ queues, each of length 2, and so on.

2.1 Approximate MST

If we simply relax the search by finding approximate NNs we end up with an AMST algorithm. Approximate NNs queries are much faster than exact ones, specially in high-dimensional spaces.

Typically, $\mathrm{ANN}_d(X, u, \eta)$ ensures that, if the true NN is at distance δ, the approximate NN is at a distance lower than $\delta(1+\eta)$. Note that AMSTs can also be obtained by using a probability bound on the NN distance [12].

Lai et al. [6] have previously studied AMSTs. Their approximation is obtained by using space-filling structures, i.e., Hilbert curves. Their work differs from ours in two central points. First, our algorithm allows to combine MSTs and AMSTs in a single framework, in which the only difference between them is a relaxation parameter. Their work is restricted to AMSTs. Second, Hilbert curves are fractal and the space-filling accuracy follows an exponential scale. It relies on a scale parameter that has a non-intuitive meaning and which is difficult to choose. It is not straightforward to set automatically a suitable scale for a given point set configuration. The relaxation parameter in our method has a clear interpretation and it is easy to monitor its effect.

3 Experimental Results

For the NN computations, choose the list-of-clusters (LOC) structure [13,14]. It is reported to be very efficient and resistant to the intrinsic dimensionality of the data set. It can also be implemented in primary and in secondary memory. See [11] for further details on how to adapt the structure for our specific purposes.

As distance computations are the dominating speed factor, we measure performance and complexity as a function of them. We sample points from a uniform distribution in the unit hyper-cube. We tested with four different dimensionalities $\mathbb{R}^2$, $\mathbb{R}^5$, $\mathbb{R}^{10}$ and $\mathbb{R}^{20}$. We compared the following methods (see Table 1): **Bvka**: the classical Boruvka's algorithm, where all distances are precomputed

Table 1. The methods compared in this work. $\overline{s}$ stands for average number of distance operations needed to complete a NNs search.

Method	Solution	Number of distances computed	Number of distances stored	Space complexity	Search speed
Bvka	MST	$n(n-1)/2$	all	$O(n^2)$	—
Bvka-O	MST	$O(n^2 \log n)$	none	$O(1)$	linear
Bvka-LOC	MST	$O(\overline{s}n \log n)$	none	$O(n)$	sub-linear
Bvka-PQ-LOC	MST	$O(\overline{s}n \log n)$	$n-1$	$O(n)$	sub-linear
Bvka-A η	AMST	$O(\overline{s}n \log n)$	$n-1$	$O(n)$	sub-linear

and stored in memory; **Bvka-O**: the proposed algorithm where an online linear search is used to compute NNs; **Bvka-LOC**: the proposed algorithm where NNs are computed online by using LOC; **Bvka-PQ-LOC**: the proposed algorithm where NNs are computed online by using LOC and priority queues; **Bvka-A** η: Bvka-PQ-LOC modified to compute the AMST by using approximate NNs. Note that the reduced memory complexity of the algorithm guarantees that we will be able to treat large datasets without memory issues.

Comparisons were made for relatively small feature sets ($|X| \leq 10^4$) to be able to compare with a classical MST implementation. A summary of our results is shown in Figure 1. Our method exhibits a very strong performance improvement in low dimensions (Fig. 1, top row). Bvka-LOC and Bvka-PQ-LOC in both cases outperforms Bvka several orders of magnitude. We can also notice a strong performance degradation of Bvka-LOC with the increase of dimensionality (Fig. 1, bottom row). The only cause is the NNs search structure. It is a well known fact that the performance of NNs search structures tends to become linear in high-dimensions. In any case, our method is generic: any NN structure can be used. Another structure may provide better results in high dimensions and we plan to explore these issues in future work.

Table 2a summarizes the results from Figure 1 by analyzing the slope of the different curves. The proposed approach lowers in practice the number of distance computations needed to solve the problem. The quadratic profiles of Bvka and Bvka-O are reduced to supralinear (e.g., $n^{1.6}$ approximately) by Bvka-LOC and Bvka-PQ-LOC. As stated, the latter shows a computational cost which is less sensitive to an increase in dimensionality.

We provide a simple example of the incidence of using the AMST, shown in Figure 2a. We use X uniformly distributed on the square $[0, 1]^2$ and Euclidean distance. Computing the MST required 9613 distance computations with our algorithm, while taking 9155, 8705 and 7840 with $\eta = 0.1$, $\eta = 0.2$, $\eta = 0.5$ respectively. There is an important improvement in performance while the number of topology changes is small. Moreover, when carefully inspected, these changes are reasonable. It is a well known fact that (even little) jitter noise in the dataset greatly affects the topology of the MST [4]: computing the AMST can be seen as perturbing the dataset with such a noise. Usually η is chosen to be quite small, and its use has more meaning in large and high-dimensional datasets. In our toy example, keeping η small does not introduce changes in the topology of the tree. We exaggerated η to show actual topology changes.

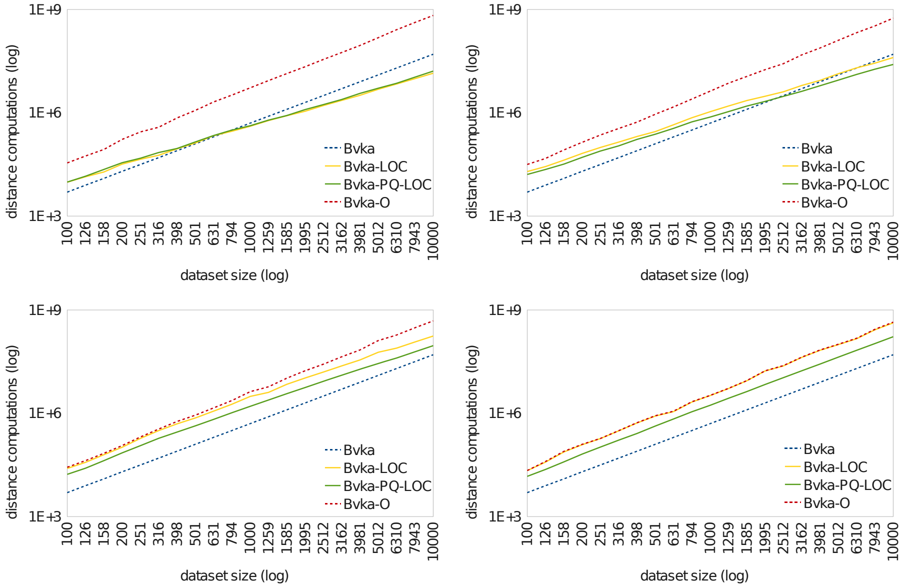
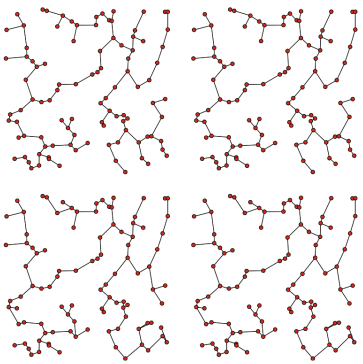
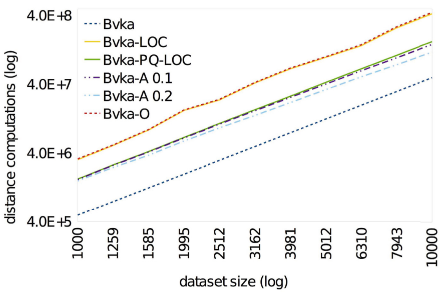


Fig. 1. Comparison in the number of distance computations as $|X|$ grows. From left to right: top row, $X \subset \mathbb{R}^2$ and $X \subset \mathbb{R}^5$; bottom row, $X \subset \mathbb{R}^{10}$ and $X \subset \mathbb{R}^{20}$. The radii in the list-of-clusters were chosen such that each bucket has $\sqrt{|X|/2}$ internal elements. Both scales are logarithmic.



(a)



(b)

Fig. 2. (a) Comparison of the MST (using Bvka) vs the AMST (using Bvka-A η) for several levels of relaxation η. From left to right: MST, AMST ($\eta = 0.1$), AMST ($\eta = 0.2$), AMST ($\eta = 0.5$). (b) Comparison in the number of distance computations of the MST and the AMST algorithms for $\eta = 0.1$ and $\eta = 0.2$ with $X \subset \mathbb{R}^{20}$.

Table 2. (a) Slopes of the different curves in Figure 1 in a log-log scale. In low dimensions, Bvka-LOC is better than any classical algorithm while Bvka-PQ-LOC resists better the dimensionality increase. (b) Running times (in seconds) on an Intel Core 2 Duo at 2.2 GHz for 10^5 uniformly distributed points using Euclidean distance.

(a)

Method	$\mathbb{R}^2$	$\mathbb{R}^5$	$\mathbb{R}^{10}$	$\mathbb{R}^{20}$
Bvka	2	2	2	2
Bvka-O	2.14	2.12	2.13	2.15
Bvka-LOC	1.58	1.66	1.92	2.15
Bvka-PQ-LOC	1.61	1.6	1.87	2.03

(b)

Dim.	Bvka-PQ-LOC	Bvka-A 0.1	Bvka-A 0.2
$\mathbb{R}^2$	32	27	23
$\mathbb{R}^5$	85	63	48

A performance comparison between MSTs and AMSTs is shown in Figure 2b. We use X uniformly distributed in the hyper-cube $[0,1]^{20}$ and Euclidean distance. As argued before Bvka-LOC's performance tends to Bvka-O's in high-dimensions. Bvka-A greatly improves the performance: it is 1.7 and 1.62 times faster than Bvka-O and Bvka-LOC respectively when $|X| = 10^4$.

Computing the MST for $|X| = 10^5$ is not possible with classical algorithms on standard computers, since approximately $5 \cdot 10^9$ distances must be computed and stored. This means more than 18.6 GB if we use 32 bits to store each computed distance. Using minimum memory (less than 20 MB), we were able to compute the MST using Euclidean distance, without, explicitly nor implicitly, exploiting the nature of the Euclidean space (i.e., without relying on Delaunay triangulations). Table 2b presents the resulting running times for all considered algorithms. Again, these results can be improved, as we did not perform any tuning of the list-of-clusters.

Finally, more efficient search algorithms can be implemented for a given NNs structure that might increase the performance of the proposed algorithms, such as the best-bin-first or an optimized depth-first [15].

4 Final Remarks

The dominating factor when computing the MST of a feature set X is the number of distance computations to be performed. We presented a method for computing the MST based on a clever use of NNs search structures. It has $\mathrm{O}(n^2)$ and $\mathrm{O}(n)$ time and space complexities respectively. However, in practice it outperforms classical algorithms for large, and low dimensional, datasets.

The same algorithm with a slight modification can also be used to compute the AMST: instead of finding NNs, one finds approximate NNs. In high-dimensional datasets, we showed the performance increase that results from using AMSTs. Moreover, the computed AMSTs exhibit a stable behavior.

There are three conceptual main lines for future work. The first consists on performing an experimental evaluation of NNs search structures and their incidence on the performance of the proposed algorithm. This includes the evaluation of different criteria in list-of-clusters for selecting the centers and the radii. Second, we did not explore other search algorithms [15] which may reduce the number of distance computations per query. Finally, when using AMSTs, the trade-off between enhanced speed and accuracy must be explored more carefully.

Last, from the implementation point of view, the proposed algorithms can be parallelized without any reformulation. Moreover, in list-of-clusters, the exhaustive search within a bucket can be implemented using vectorial processors as the bucket size is fixed.

References

1. Graham, R., Hell, P.: On the history of the minimum spanning tree problem. Annals of the History of Computing 7(1), 43–57 (1985)
2. Jain, A.K., Dubes, R.C.: Algorithms for clustering data. Prentice-Hall, Inc., Upper Saddle River (1988)
3. Zahn, C.T.: Graph-Theoretical Methods for Detecting and Describing Gestalt Clusters. Transactions on Computers C-20(1), 68–86 (1971)
4. Carreira-Perpiñán, M., Zemel, R.: Proximity graphs for clustering and manifold learning. In: NIPS (2005)
5. Felzenszwalb, P., Huttenlocher, D.: Efficient Graph-Based Image Segmentation. International Journal of Computer Vision 59(2), 167–181 (2004)
6. Lai, C., Rafa, T., Nelson, D.: Approximate minimum spanning tree clustering in high-dimensional space. Intelligent Data Analysis 13(4), 575–597 (2009)
7. Eddy, W., Mockus, A., Oue, S.: Approximate single linkage cluster analysis of large data sets in high-dimensional spaces. Computational Statistics & Data Analysis 23(1), 29–43 (1996)
8. Bentley, J., Friedman, J.: Fast Algorithms for Constructing Minimal Spanning Trees in Coordinate Spaces. Transactions on Computers 27(2), 97–105 (1978)
9. Leibe, B., Mikolajczyk, K., Schiele, B.: Efficient Clustering and Matching for Object Class Recognition. In: BMVC (2006)
10. Frank, A., Asuncion, A.: UCI machine learning repository (2010)
11. Tepper, M., Musé, P., Almansa, A., Mejail, M.: Boruvka Meets Nearest Neighbors. Technical report, HAL: hal-00583120 (2011)
12. Toyama, J., Kudo, M., Imai, H.: Probably correct k-nearest neighbor search in high dimensions. Pattern Recognition 43(4), 1361–1372 (2010)
13. Chavez, E., Navarro, G.: An Effective Clustering Algorithm to Index High Dimensional Metric Spaces. In: SPIRE (2000)
14. Chávez, E., Navarro, G.: A compact space decomposition for effective metric indexing. Pattern Recognition Letters 26(9), 1363–1376 (2005)
15. Samet, H.: Depth-First K-Nearest Neighbor Finding Using the MaxNearestDist Estimator. In: ICIAP (2003)