Boruvka Meets Nearest Neighbors

Mariano Tepper, Marta Mejail, Pablo Musé, Andrés Almansa

Boruvka Meets Nearest Neighbors

Mariano Tepper , Marta Mejail

Departamento de Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires
e-mail: mtepper@dc.uba.ar; marta@dc.uba.ar

Pablo Musé

Instituto de Ingeniería Eléctrica, Facultad de Ingeniería, Universidad de la República,
e-mail: pmuse@fing.edu.uy

Andrés Almansa

CNRS - LTCI UMR5141, Telecom ParisTech
e-mail: andres.almansa@telecom-paristech.fr

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 reduce significantly time and memory performances. 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. Finally, to illustrate the usefulness of the proposed algorithm, we focus on a classical computer vision problem: image segmentation. We modify a state-of-the-art local graph-based clustering algorithm, thus permitting a global scene analysis.

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.

The history of the MST problem up to 1985 was reviewed by Graham and Hell [1985]. Maybe the two most widely known algorithms to compute the MST are Prim's and Kruskal's [Cormen et al., 2003]. There is a third classical algorithm by Boruvka [Graham and Hell, 1985] that mysteriously remained shadowed by the other two. This fact is emphasized by the fact that Boruvka's algorithm is also known as Sollin's algorithm, despite the fact that Sollin re-discovered it independently years later.

Under a restricted random-access computational model, Karger et al. [1995] published a randomized algorithm that runs on expected linear time. Up to date, Chazelle's (2000) is the fastest pointer-based algorithm to compute the MST. Table 1 summarizes

Algorithm	Model	Time
Prim [Cormen et al., 2003]	pointer-based	$O(m \log n)$
Kruskal [Cormen et al., 2003]	pointer-based	$O(m \log n)$
Boruvka [Graham and Hell, 1985]	pointer-based	$O(m \log n)$
Karger [Karger et al., 1995]	random-access	$O(m)$
Chazelle [Chazelle, 2000]	pointer-based	$O(m\alpha(m,n))$

TABLE 1

Major MST algorithms, their computational model and their time complexity (n and m are the number of nodes and the number of edges, respectively). Karger's and Chazelle's are amortized complexities. All algorithms have a spatial complexity of $O(m)$ *at least.*

the complexity of classical and state-of-the-art algorithms. Pettie and Ramachandran [2002] proposed an optimal theoretical algorithm which runs in time $O(\mathscr{T}^*(m,n))$ where n (respectively m) is the number of vertices (respectively edges) of the graph and $\mathscr{T}^*$ is the minimum number of edge-weight comparisons needed to determine the solution.

The MST algorithm is particularly interesting for many data analysis tasks in computer vision and pattern recognition. A clear example is clustering, where the classical single-link hierarchical algorithm [Jain and Dubes, 1988] can be proved to be equivalent to computing the MST. In a seminal work, Zahn [1971] studied the benefits of using the MST for clustering, which were lately checked in psychophysical experiments by Dry et al. [2009]. More recently, the MST received much attention maybe due to the growth in the size of clustering datasets [e.g. Carreira-Perpiñán and Zemel, 2005, Felzenszwalb and Huttenlocher, 2004]. The approximate MST (AMST), suboptimal but faster, also received attention for the same reasons [Lai et al., 2009].

We now slightly change the definition of the problem to a form more suitable for feature sets analysis (e.g. clustering).

Definition 1 *Given a set* M *and a function* $d : M \times M \to \mathbb{R}$ *such that,* $\forall x, y \in M$*,*

- $d(x,y) \geq 0$ *(non-negativity),*
- $d(x,y) = 0 \Leftrightarrow x = y$ *(identity of indiscernibles),*
- $d(x,y) = d(y,x)$ *(symmetry) and*
- $d(x,z) \leq d(x,y) + d(y,z)$ *(triangle inequality).*

Then d *and the pair* (M,d) *are said to be a metric on* M *and a metric space, respectively.*

Definition 2 *Given a metric space* (M,d) *and feature 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* $x_i \in X$*,* $E = V \times V$ *and the graph's weighting function* $\omega : E \to \mathbb{R}$ *is defined as*

$$\omega((v_i, v_j)) = d(x_i, x_j). \tag{1}$$

In other terms, the problem remains the same but the graph is now complete by definition. In such a context, and as the feature set gets larger, all the previous algorithms are worthless (as explained later in Section 1.1).

The problem is classically addressed by using metric spaces with exploitable specific characteristics, i.e. the Euclidean MST is contained in the Delaunay triangulation of X [Eddy et al., 1996]. Recent work has aimed at building an AMST [Lai et al., 2009] through a clever use of space-filling curves. The fractal nature of such curves imposes the use of a scale parameter, which is not easy to set automatically.

Bentley and Friedman [1978] and Murtagh [1983] addressed the problem of using nearest neighbors search structures to compute the MST. The approach proved successful; moreover, using such structures allows in addition to compute the AMST in a natural and straightforward way. Both works are outdated and a revision in the light of novel nearest neighbors techniques and increasing computational power is much needed. More recently, Leibe et al. [2006] also use nearest neighbors 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.

In this work we address the MST problem without computing all distances in E. We build on Boruvka's approach, summarized in Algorithm 1, by an appropriate use of nearest neighbors search techniques.

Algorithm 1 Compute the minimum spanning tree of X using metric d

Require: $X \neq \emptyset$
Ensure: $T = (X, E_T)$ is the minimum spanning tree of X using metric d
$E_T \leftarrow \emptyset$
while $|E_T| < |X| - 1$ **do**
  $E' \leftarrow \emptyset$
  for each connected component C of T **do**
    $(u_m, v_m) \leftarrow \underset{u \in C,\ v \notin C}{\arg\min}\, d(u,v)$
    $\delta_m \leftarrow d(u_m, v_m)$
    $E' \leftarrow E' \cup \{(u_m, v_m, \delta_m)\}$
  end for
  while $E' \neq \emptyset$ **do**
    $(u_m, v_m, \delta_m) \leftarrow \underset{(u,v,\delta) \in E'}{\arg\min}\, d$
    $E' \leftarrow E' \setminus \{(u_m, v_m, \delta_m)\}$
    if $E_T \cup \{(u_m, v_m, d_m)\}$ does not contain cycles **then**
      $E_T \leftarrow E_T \cup \{(u_m, v_m, d_m)\}$
    end if
  end while
end while

1.1. Complexity

Let us define $n = |X|$. The tasks to be performed are:

- creation of n singletons in $O(n)$ time,
- $n-1$ unions of sets and
- n^2 operations to find whether two nodes are in the same subtree or not.

The disjoint-sets data structure [Tarjan and Van Leeuwen, 1984] gives extremely efficient operations on disjoint sets. The amortized complexity then is $O(n^2\ \alpha(n^2, n))$ per

iteration, where alpha is the extremely slow growing inverse Ackermann function, i.e. in practice $\alpha(n^2,n) \leq 4$.

The number of connected components will be reduced by at least a factor of 2 in each iteration. At most $\log n$ iterations must be performed. Therefore, the complexity is $O(n^2\ \alpha(n^2,n)\ \log n)$ which can be approximated without fear by $O(n^2\ \log n)$.

The previous paragraphs contain an implicit fact: one must perform all $n(n-1)/2$ distance computations and thus a double-sided problem appears:

in space : storing all $n(n-1)/2$ results for $n \geq 10^5$ is prohibitive,
in time : 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 points or more are becoming common [Frank and Asuncion, 2010].

The rest of the paper is structured as follows. In Section 2 we propose a general approach to compute the MST using nearest neighbors search structures. Section 3 deals with search structures and in particular with a slight modification needed to compute the MST. Section 4 shows empirical results of the proposed approach on a synthetic dataset and Section 5 shows results on real image segmentation examples. Finally, some final remarks and future work are presented in Section 6.

2. A Nearest Neighbors Approach

Let us assume that we have a function $\mathrm{NN}_d(A,b)$ that returns the nearest neighbor $a \in A$ of b using metric d. We will discuss such functions and their implementation in Section 3.

The term $\arg\min_{u \in C,\ v \notin C} d(u,v)$ in line 5 of Algorithm 1 can be straightforwardly expressed in terms of finding the nearest neighbor in the set $V \setminus C$:

$$u_m = \arg\min_{u \in C} d(u, \mathrm{NN}_d(V \setminus C, u)), \tag{2}$$

$$v_m = \mathrm{NN}_d(V \setminus C, u_m). \tag{3}$$

Let us define a constraint function $\rho : X \to \{0,1\}$. We propose to modify the function $\mathrm{NN}_d(A,b)$ by adding an additional constraint ρ on the returned element. We denote it by $\mathrm{NN}_{d,\rho}(A,b)$. In summary, it returns the nearest neighbor $a \in A$ of b using metric d such that $\rho(a) = 1$. By setting

$$\rho(v) = (v \notin C) \tag{4}$$

we have

$$\mathrm{NN}_d(V \setminus C, u) = \mathrm{NN}_{d,\rho}(V, u). \tag{5}$$

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

We are sure that the desired node v_m is among the k nearest neighbors of u where $k = |C| + 1$. Therefore in the worst case, using a naive approach, $\mathrm{NN}_{d,\rho}$ amounts to

perform a k-nearest neighbors 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 of a different nature than finding the MST in a constrained degree graph.

Rewriting Algorithm 1 in terms of these new elements results in Algorithm 2. Our work is similar to Bentley and Friedman's (1978). They showed the pertinence of using nearest neighbors search structures to compute the MST in a Kruskal-like algorithm. Although they showed that the use of nearest neighbors search structures was fruitful for the computation of MSTs, their work received little attention.

Algorithm 2 Compute the minimum spanning tree of feature set X with metric d using nearest neighbors structures.

Require: $X \neq \emptyset$
Ensure: $T = (V, E_T)$ is the minimum spanning tree of G using d
$E_T \leftarrow \emptyset$
while $|E_T| < |V| - 1$ **do**
  $E' \leftarrow \emptyset$
  for all connected components C of T **do**
    $u_m \leftarrow \arg\min_{u \in C}(u, \mathrm{NN}_{d,\rho}(V, u))$
    $v_m \leftarrow \mathrm{NN}_{d,\rho}(V, u_m)$
    $\delta_m \leftarrow d(u_m, v_m)$
    $E' \leftarrow E' \cup \{(u_m, v_m, \delta_m)\}$
  end for
  while $E' \neq \emptyset$ **do**
    $(u_m, v_m, \delta_m) \leftarrow \arg\min_{(u,v,\delta) \in E'} d$
    $E' \leftarrow E' \setminus \{(u_m, v_m, \delta_m)\}$
    if $E_T \cup \{(u_m, v_m, d_m)\}$ does not contain cycles **then**
      $E_T \leftarrow E_T \cup \{(u_m, v_m, d_m)\}$
    end if
  end while
end while

Beside using of nearest neighbors, Bentley and Friedman also use priority queues to prune the number of nearest neighbors searches performed during the algorithm. First let us explain that Kruskal's algorithm is greedy: it creates a forest (i.e. a set of trees) where each isolated edge is a tree and gradually merges these trees by adding the smallest edge whose endpoints lie on different trees. They propose to store the nodes of the partial (i.e. already computed) forest, along with their foreign nearest neighbors, in a single and global priority queue where the priority of a node is the inverse of the distance to its foreign nearest neighbor. The use of a priority queue is indeed interesting in this context, as the next edge to add to the MST is at the top of the priority queue. The top of the queue is removed and the top-priority foreign nearest neighbors is added to the MST. This node is also added to the queue, after computing its foreign nearest neighbors. Additionally, the priority queue must be updated, since disjoint connected components are merged and some foreign nearest neighbors 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. Thus, by using a global priority queue Bentley and Friedman were able to speed up a Kruskal-like algorithm.

We already stated our interest in building on Boruvka's algorithm, hence a global priority queue is not suitable in this case. Alternatively, we propose to use several priority queues, one for each connected component in the partial MST. Each queue, only holds the nodes in its respective connected component, and their foreign nearest neighbor. After an insertion in the MST, two connected components are merged and their priority queues are also merged.

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.

Algorithm 3 is the result of this modification. The operators top and pop return the top-priority element in the queue and remove it, respectively.

2.1. Approximate MST

We stated that our approach allows to compute approximate MSTs. Indeed, if we simply relax the search by finding the approximate nearest neighbors we end up with an approximate minimum spanning tree algorithm. Approximate nearest neighbors queries are much faster than exact ones, specially in high-dimensional spaces.

Typically, $\text{ANN}_d(X, u, \eta)$ ensures that, if the true nearest neighbor is at distance δ, the approximate nearest neighbor is at a distance lower than $\delta(1+\eta)$. Note that AMSTs can also be obtained by using a probability bound on the nearest neighbor distance [Toyama et al., 2010].

Lai et al. [2009] 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 chooseIt 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. Nearest Neighbors Search Structures

The problem of efficiently finding nearest neighbors has received much attention, as it is in the core of a great variety of problems in pattern recognition and classification. To name a few, non-parametric density estimation [Fukunaga, 1999], non-linear manifold learning [Tenenbaum et al., 2000] and descriptors matching [Lowe, 2004].

A plethora of nearest neighbors search structures have been proposed over the years:

- Kd-trees and randomized Kd-trees [Bentley, 1975], [Silpa-Anan and Hartley, 2008],

Algorithm 3 Compute the minimum spanning tree of X with metric d using nearest neighbors structures and multiple priority queues.

Require: $X \neq \emptyset$
Ensure: $T = (V, E_T)$ is the minimum spanning tree of G using d
for all $v \in V$ **do** {each node is a connected component}
  $v_m \leftarrow \mathrm{NN}_{d,\rho}(V, v)$
  $d_m \leftarrow d(v, v_m)$
  add (v, v_m) to the empty priority queue $Q_{\{v\}}$ with priority d_m
end for
$E_T \leftarrow \emptyset$
while $|E_T| < |V| - 1$ **do**
  $E' \leftarrow \emptyset$
  for all connected components C of T **do**
    $(v, v_m) \leftarrow \mathrm{top}(Q_C)$
    while $v_m \in C$ **do** {not a foreign nearest neighbors}
      $\mathrm{pop}(Q_C)$
      $u_m \leftarrow (v, \mathrm{NN}_{d,\rho}(V, v))$
      $\delta_m \leftarrow d(v, u_m)$
      add (v, u_m) to Q_C with priority δ_m
      $(v, v_m) \leftarrow \mathrm{top}(Q_C)$
    end while
    $\delta_m \leftarrow d(v, v_m)$
    $E' \leftarrow E' \cup \{(v, v_m, \delta_m)\}$
  end for
  for all $(u, v) \in E'$ **do**
    $C \leftarrow$ connected component of T containing u
    $C' \leftarrow$ connected component of T containing v
    merge Q_C and $Q_{C'}$ into $Q_{C \cup C'}$
  end for
  while $E' \neq \emptyset$ **do**
    $(u_m, v_m, \delta_m) \leftarrow \underset{(u,v,\delta) \in E'}{\arg\min}\ d$
    $E' \leftarrow E' \setminus \{(u_m, v_m, \delta_m)\}$
    if $E_T \cup \{(u_m, v_m, d_m)\}$ does not contain cycles **then**
      $E_T \leftarrow E_T \cup \{(u_m, v_m, d_m)\}$
    end if
  end while
end while

Size	Dimensions	List-of-clusters	kd-tree	k-means tree	Linear
10^4	2	0.04	0.22	0.1	2.76
	3	0.03	0.32	0.17	2.64
	4	0.08	0.4	0.45	2.59
	5	0.04	0.61	1.11	2.56
	10	0.19	1.72	2.84	3.23
	20	0.8	5.62	4.16	4.48
10^5	2	0.31	1.33	0.3	47.65
	3	0.49	2.04	0.91	52.84
	4	0.72	3.07	2.42	50.19
	5	0.6	4.34	5.47	49.88
	10	2.7	14.51	53.52	59.26
	20	18.39	70.68	66.07	65.21

TABLE 2

Running-time comparison of search structures in the size and in the dimensionality of the dataset. The dataset is composed by uniformly distributed points. The values are the average of 1000 random queries and are expressed in milliseconds.

- VP-trees and MVP-trees [Yianilos, 1993], [Bozkaya and Ozsoyoglu, 1997],
- M-trees and their extensions [Ciaccia and Patella, 1998], [Skopal et al., 2003], [Skopal et al., 2005], [Zhou et al., 2003],
- hashing-based methods [Andoni and Indyk, 2008],
- hierarchical k-means trees [Muja and Lowe, 2009]
- list-of-clusters [Chavez and Navarro, 2000], [Chávez and Navarro, 2005].

The above list is, of course, non-exhaustive.

All metric search structures, except hashing-based methods, exploit, one way or another, the metric's triangular inequality to reduce the number of distance computations to be performed in a search. Hashing-based methods aim at finding hash functions which ensure that the probability of collision is much higher for objects that are close to each other than for those that are far apart.

3.1. List-of-clusters

Now we turn our attention to the list-of-clusters structure [Chavez and Navarro, 2000, Chávez and Navarro, 2005]. It is reported to be very efficient and resistant to the intrinsic dimensionality of the data set. Our experiments, shown in Table 2, support this claim. It can also be implemented in primary and in secondary memory. Furthermore, it has the advantage of being easy to implement and understand, as it does not involve complex data structures. For this reason we choose list-of-clusters to explain the modifications needed to find foreign nearest neighbors in depth.

The core of the list-of-clusters construction algorithm is the choice of a center $c \in X$ and a radius r_c. The possible choices for them will be discussed later. Let us define

- **internal elements**:

$$I_{X,c,r_c} \stackrel{def}{=} \{x \in X - \{c\},\ d(c,x) \leq r_c\},$$

- **external elements**:

$$E_{X,c,r_c} \stackrel{def}{=} \{x \in X,\ d(c,x) > r_c\}.$$

A bucket is defined as the tuple (c, r_c, I_{X,c,r_c}). The process is repeated inside E_{X,c,r_c} recursively, producing at the end a list of buckets. This procedure is depicted in a recursion-free manner, in Algorithm 4. List-of-clusters, where only external elements are split, can be seen as a degenerated VP-tree, where external and internal elements are split.

Algorithm 4 Build a list-of-clusters from X using metric d

Require: $X \neq \emptyset$
Ensure: $L_{X,d}$ is the list of clusters of X
  $L_{X,d} \leftarrow \emptyset$
  while $X \neq \emptyset$ **do**
    select a center $c \in X$
    select a radius r_c
    $I \leftarrow \{x \in X - \{c\},\ d(c,x) \leq r_c\}$
    $X \leftarrow X - I - \{c\}$
    $L_{X,d} \leftarrow L_{X,d} : (c, r_c, I)$
  end while

There are two decisions to make at the core of the building algorithm: the selection of the center c and the radius r_c [Chávez and Navarro, 2005]. At iteration i, for selecting the center c_i, we chose the element farthest to c_{i-1} in the remaining set. The objective of such a choice is to minimize the overlap between regions. By using partitions of fixed size, the radius r_c is then easily deduced.

Now we focus our attention to search itself. The original search algorithm in the list-of-clusters structure iterates through the list of buckets and performs exhaustive searches only when needed (determined by triangular inequalities). An exhaustive search occurs within a bucket's internal elements. It can be written in the following terms

$$x_m \leftarrow \arg\min_{x \in I} d(x,q) \tag{6}$$

$$d_m \leftarrow d(x_m, q). \tag{7}$$

The standard list-of-clusters search algorithm [Chávez and Navarro, 2005] is shown in Algorithm 5, with the introduction of the constraint ρ in the exhaustive search (lines 10 and 9). The operators head and tail return the first element in the list and remove it, respectively. Adding constraint ρ to any other search structure is similar.

Algorithm 5 can be very easily modified to find approximate nearest neighbors. It is sufficient to rewrite lines 4 and 8 as follows

$$d_c / r \leq 1 + \eta \tag{8}$$

$$d_c / (r_c + r) \leq 1 + \eta, \tag{9}$$

where η is a relaxation parameter. Note that more complex nearest neighbors formulations can be used, such as probabilistic bounds [Toyama et al., 2010].

Algorithm 5 Search for the nearest neighbor p of q in the list-of-clusters $L_{X,d}$ with initial radius r and restriction ρ

Require: $L_{X,d}$ is not empty and $r > 0$
Ensure: $p \in X$ is the nearest neighbor of q if $\exists x \in X,\ d(x,q) \leq r$
repeat
  $(c, r_c, I) \leftarrow \text{head}(L_{X,d})$
  $d_c \leftarrow d(c,q)$
  if $d_c \leq r$ **then**
    $p \leftarrow c$
    $r \leftarrow d_c$
  end if
  if $d_c \leq r_c + r$ **then**
    $x_m \leftarrow \arg\min_{\substack{x \in I \\ \rho(x)=1}} d(x,q)$
    $d_m \leftarrow d(x, x_m)$
    if $d_m \leq r$ **then**
      $p \leftarrow x_m$
      $r \leftarrow d_m$
    end if
  end if
  $L_{X,d} \leftarrow \text{tail}(L_{X,d})$
until $L_{X,d}$ is not empty or $d_c \leq r_c - r$

4. Experimental Results

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:

Bvka: all distances are precomputed and stored in memory and then Algorithm 1 is performed.

Bvka-O: Algorithm 2 where an online linear search is used to compute nearest neighbors.

Bvka-LOC: Algorithm 2 where nearest neighbors are computed online by using the list-of-clusters search structure.

Bvka-PQ-LOC: Algorithm 3 where nearest neighbors are computed online by using list-of-clusters search structure.

Bvka-A η: Bvka-PQ-LOC modified to compute the AMST by using approximate nearest neighbors (see Equations 8 and 9) where η is the relaxation parameter.

A summary of these methods is presented in Table 3. Note that the reduced memory complexity of the algorithm warranties that we will be able to treat large datasets without "out of 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. At a first approach and as expected, all algorithms have a polynomial dependency on the dataset size, since they are linear in a log-log plot. In fact, we are not aiming at improving the worst case complexity of Boruvka's algorithm, but at improving its expected performance.

Method	Solution	Distances computed	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	$O(\bar{s}n \log n)$	none	$O(n)$	sub-linear
Bvka-PQ-LOC	MST	$O(\bar{s}n \log n)$	$n-1$	$O(n)$	sub-linear
Bvka-A η	AMST	$O(\bar{s}n \log n)$	$n-1$	$O(n)$	sub-linear

TABLE 3

The methods compared in this work. $\bar{s}$ stands for average number of distance operations needed to complete a nearest neighbors search. The space required by the nearest neighbors search structure is $O(n)$.

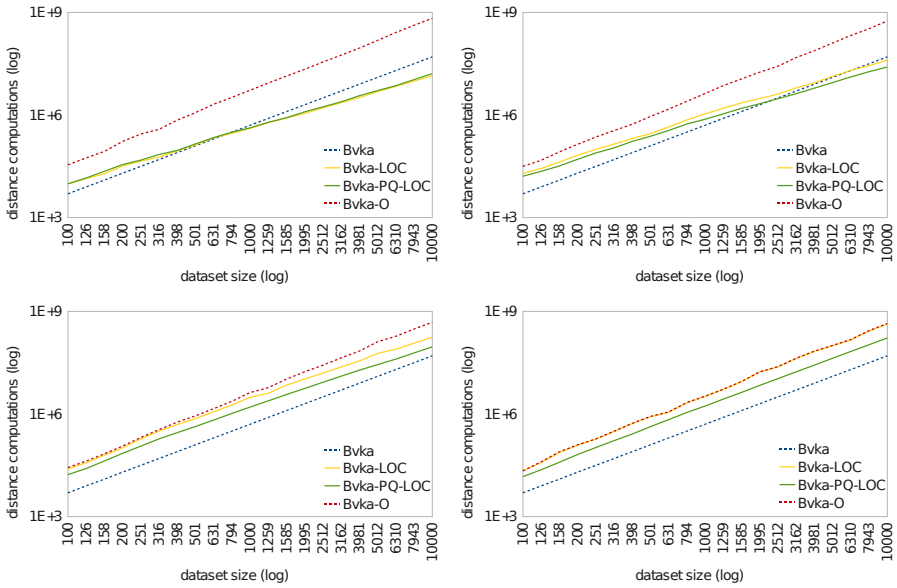


FIG 1. *Comparison in the number of distance computations as $|X|$ grows. From left to right: top row, $X \subset \mathbb{R}^2$ and $X \subset \mathbb{R}^5$; bottom row, $X \subset \mathbb{R}^{10}$ and $X \subset \mathbb{R}^{20}$. The radii in the list-of-clusters were chosen such that each bucket has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

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

TABLE 4

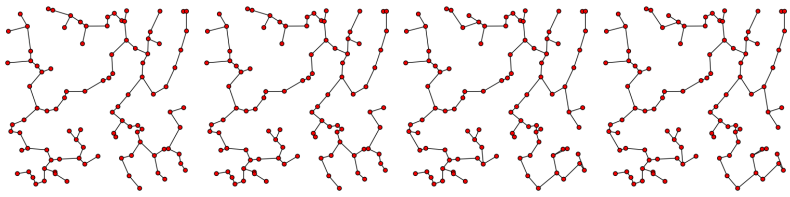
Slopes of the different curves in Figure 1 in a log-log scale. In low dimensions, Bvka-LOC is better than any classical algorithm while Bvka-PQ-LOC resists better the dimensionality increase.

Results are indeed encouraging. Our method exhibits a very strong performance improvement in low dimensions, see the top row in Figure 1. 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, see the bottom row in Figure 1. The only cause is the nearest neighbors search structure. It is a well known fact that the performance of nearest neighbors search structures tends to become linear in high-dimensions. In any case, our method is generic: any nearest neighbor structure can be used. Another structure may provide better results in high dimensions and we plan to explore these issues in future work.

Table 4 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 2. 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 [Carreira-Perpiñán and Zemel, 2005]: computing the AMST can be seen as perturbing the dataset with such a noise.



(a) MST (b) AMST ($\eta = 0.1$) (c) AMST ($\eta = 0.2$) (d) AMST ($\eta = 0.5$)

FIG 2. *Comparison of the MST (using Bvka) vs the AMST (using Bvka-A η) for several levels of relaxation η.*

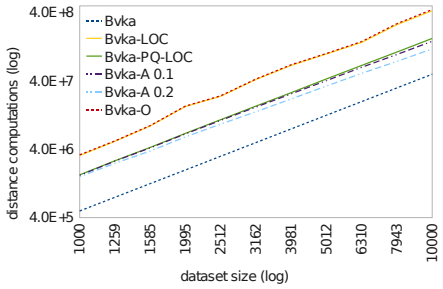


FIG 3. *Comparison in the number of distance computations of the MST and the AMST algorithms for* $\eta = 0.1$ *and* $\eta = 0.2$ *with* $X \subset \mathbb{R}^{20}$.

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.

A performance comparison between MSTs and AMSTs is shown in Figure 3. 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 5 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.

Moreover, since parallelization is straightforward, it can be exploited to boost the performance. Every iteration in both nearest neighbors-searching cycles, lines 4 to 9 in Algorithm 2 and lines 9 to 20 in Algorithm 3, can be run in parallel since it operates on a single disjoint connected component. We estimate that at least a 20X speedup can be achieved by using GPGPU.

Finally, more efficient search algorithms can be implemented for a given nearest neighbors structure that might increase the performance of the proposed algorithms, such as the best-bin-first or an optimized depth-first [Samet, 2003].

5. Application to Image Segmentation

Felzenszwalb and Huttenlocher [2004] proposed a clustering method and used it for image segmentation. The algorithm is graph-based, building a forest of partial MSTs. The authors argue that:

> "There are several possible ways of determining which feature points to connect by edges. We connect each point to a fixed number of nearest neighbors. Another possibility is to use all the

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

TABLE 5
Running times (in seconds) on an Intel Core 2 Duo at 2.2 GHz for 10^5 uniformly distributed points using Euclidean distance

> neighbors within some fixed distance δ. In any event, it is desirable to avoid considering all $O(n^2)$ pairs of feature points." [Felzenszwalb and Huttenlocher, 2004]

Felzenszwalb and Huttenlocher also test a version of their algorithm in which the graph layout is determined by the spatial neighborhood in the image. The previous citation is true in a context where all $n(n-1)/2$ distances are needed to compute the real MST.

On the one hand, it is clear that pruning the complete graph yields a faster algorithm. Nowadays a small resolution image has $640 \times 480 = 307200$ pixels. Using algorithms available in the literature, finding the MST is intractable.

On the other hand, from a conceptual point of view, pruning is dangerous. Results can be severely affected, as discussed by Fowlkes and Malik [2004] and Cour et al. [2005]. The use of k-nearest neighbors graphs or fixed radius graphs also brings a new complication: both parameters have to be carefully selected to yield a connected graph. For example, if k is fixed such that the resulting graph is not fully connected, an image where all pixels have the same color would end up being segmented!

Based on the concepts of Section 2, we propose a new version of Felzenszwalb and Huttenlocher's algorithm. This version drops the original requirements of feature locality, becoming global. It exploits the fact that the proposed algorithm makes tractable the problem of computing the MST for a set of $10^5 \sim 10^6$ features.

The classical choice to partition a feature set in hierarchical clustering, is to globally thresholding the hierarchy at a fixed level [Jain and Dubes, 1988]. Felzenszwalb and Huttenlocher proposed a similar partitioning criterion but thresholds are locally adaptive, thus providing more realistic and useful partitions. This algorithm can also link disjoint nodes across the hierarchy, thus building a forest of partial MSTs, each of which may not be a subtree of the global MST. To illustrate the usefulness of the proposed MST algorithm, we use the splitting criterion by Felzenszwalb and Huttenlocher without allowing for such links across the hierarchy. In general, this modification does not involve a negative impact on the results but allows working with larger datasets.

Let again X be a feature set and d a suitable metric. Let $T = (X, E_T)$ be the minimum spanning tree of X. We say $C \in X$ is a component if T_C is a (connected) subtree of T with node set C. For a component C we define

$$\max(C) = \max_{\substack{v_i, v_j \in C \\ (v_i, v_j) \in E_T}} d(v_i, v_j) + \tau(C) \tag{10}$$

The function τ can take many forms, but can be simply defined as $\tau(C) = s/|C|$, where s acts as a scale parameter [Felzenszwalb and Huttenlocher, 2004]. Then, given two

components C_1 and C_2 we define

$$\min(C_1, C_2) = \min\left(\max(C_1), \max(C_2)\right) \tag{11}$$

Felzenszwalb and Huttenlocher's algorithm is non other than Kruskal's with an additional criterion to avoid merging certain connected subcomponents in the MST. Our modified version, instead of constructing a MST, builds a minimum spanning forest. Given two disjoint components C_1 and C_2 they are only merged if

$$\min(C_1, C_2) \geq \min_{\substack{v_i \in C_1 \\ v_j \in C_2}} d(v_i, v_j) \tag{12}$$

Each tree in the forest is finally detected as a cluster. The process is described in Algorithm 6.

Algorithm 6 Compute the clustering of feature set X using metric d

Require: $X \neq \emptyset$
Ensure: S is a clustered partition of X
compute the MST $T = (X, E_T)$
$n \leftarrow |X|$
$m \leftarrow |E_T|$
$\tilde{E}_T \leftarrow$ sort E_T by non-decreasing edge weight.
$S \leftarrow \{\{v_1\}, \ldots, \{v_n\}\}$
while $\tilde{E}_T \neq \emptyset$ **do**
    $(v_i, v_j) \leftarrow \text{head}(\tilde{E}_T)$
    let C_i and C_j be the sets of S containing v_i and v_j respectively.
    if $(C_i \neq C_j) \wedge (d(v_i, v_j) \leq \min(C_i, C_j))$ **then**
        $S \leftarrow \left(S - C_i - C_j\right) \cup \left\{C_i \cup C_j\right\}$
    end if
    $\tilde{E}_T \leftarrow \text{tail}(\tilde{E}_T)$
end while

In the following experiments, the color value in RGB space for each pixel is used as a feature in $\mathbb{R}^3$ and Euclidean distance is used to construct the MST. As stated above, any metric could have been used without changing the algorithm hence, in this sense, the algorithm is parametric on the metric.

For our tests we compared three different versions of Felzenszwalb and Huttenlocher's algorithm:

FH-Grid: Algorithm 6 using the image grid connectivity to prune the complete graph
FH-Bvka-NN: Algorithm 6 using the complete graph and the proposed algorithm to compute the MST.
FH-Bvka-A η: Algorithm 6 using the complete graph and the proposed algorithm to compute the AMST.

We tested these algorithms on images from the Berkeley segmentation dataset [Martin et al., 2001]. Since their size is $321 \times 481 = 154401$ pixels, computing the MST with any classical (or even with Chazelle's state-of-the-art algorithm) is not possible.

The results on Figures 4, 5 and 6 (first three rows) show the importance of the role of global mechanisms in image segmentation.

Human visual system is able to capture and to use global characteristics. For example, we perceive a blue sky as a whole, even when a part appears through a hole in some object. Obviously, FH-Grid can not reproduce such behavior while global methods perform well.

By using the image connectivity, artifacts are introduced (clearly visible in all examples). Although this can be corrected by using locality directly on the color space, scale parameters are left free and must be correctly tuned to obtain satisfactory results.

Another issue can bee observed in the methods obtained with FH-Grid: the number of regions is badly overestimated. Accuracy in the localization of visually perceived region frontiers comes at the price of over-segmenting the image.

As a counter part, all global methods we tested have difficulties when segmenting images where several objects share common colors. This is also natural and is a widely adopted mechanism by animals in the from of camouflage. Animals copy colors and structures from their environment to avoid being detected by predators or preys. More careful and local inspection are necessary to detect them.

The last three rows in Figures 4, 5 and 6 illustrate the stability of using FH-Bvka-A with different levels of approximation compared to the exact FH-Bvka-NN: changing η maintains the global structures in the segmentation.

6. 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 nearest neighbors search structures. It has $O(n^2)$ and $O(n)$ 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 nearest neighbors, one finds approximate nearest neighbors. In high-dimensional datasets, we showed the performance increase that results from using AMSTs. Moreover, in our tests the AMST, as computed, has an stable behavior.

To show the pertinence of the proposed algorithm, we use it to improve a state-of-the-art image segmentation algorithm proposed by Felzenszwalb and Huttenlocher [2004]. Due to the memory and performance limitations of classical MST algorithms, Felzenszwalb and Huttenlocher's method is restricted to finding segmentations based on local connections or forced to rely on arbitrary connectivity parameters. Thanks to our input, a global segmentation can be obtained, without modifying the overall algorithm and without any extra parameter.

There are three conceptual main lines for future work. The first consists on performing an experimental evaluation of nearest neighbors 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 [Samet, 2003] which may reduce the number of distance computations per query. Finally, when using approximate MSTs, 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

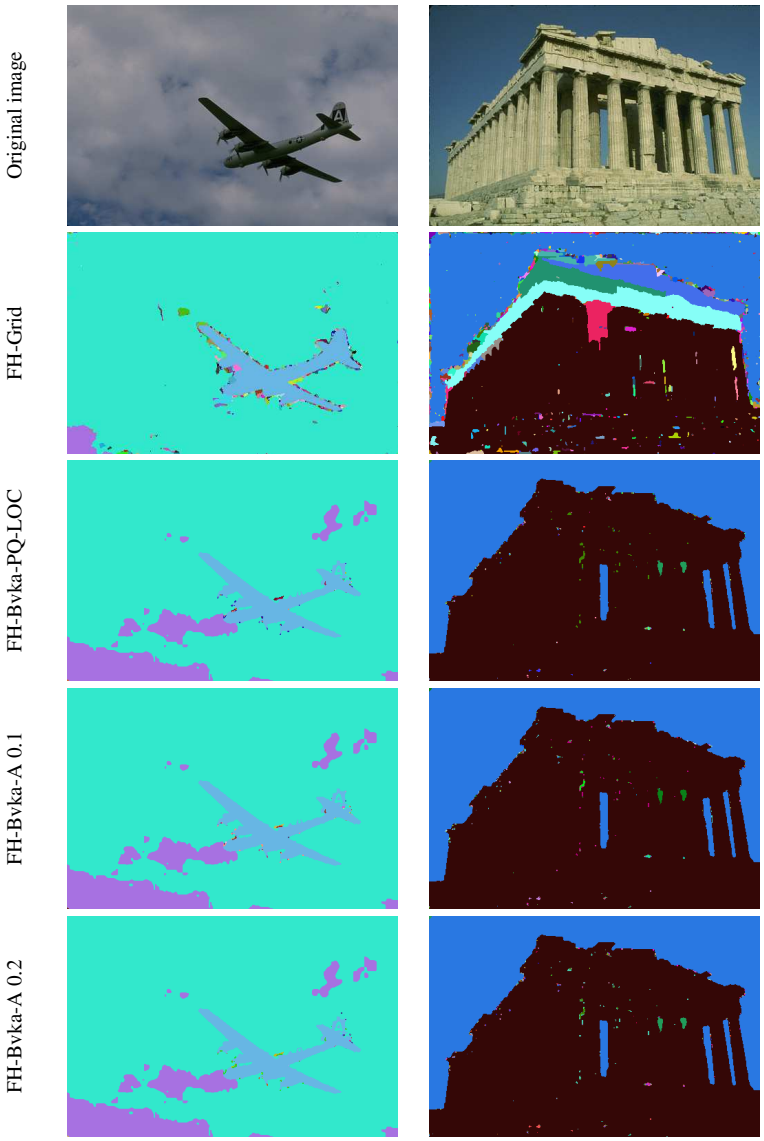


FIG 4. *Effect of using local, complete and approximate complete graphs. The oversegmentation produced by FH-Grid has been corrected by all the proposed methods. Notice also that the approximate versions of FH-Bvka-A show an stable behavior.*

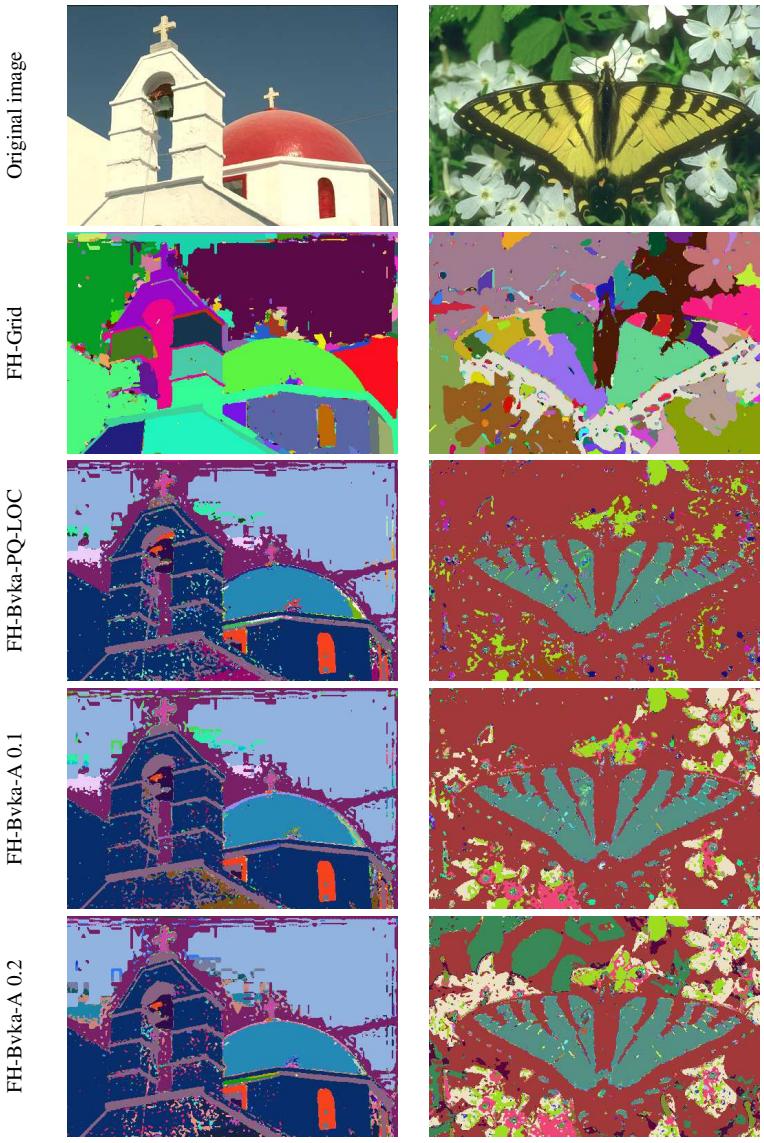


FIG 5. *Effect of using local, complete and approximate complete graphs. The oversegmentation produced by FH-Grid has been corrected by all the proposed methods. Notice also that the approximate versions of FH-Bvka-A show an stable behavior.*

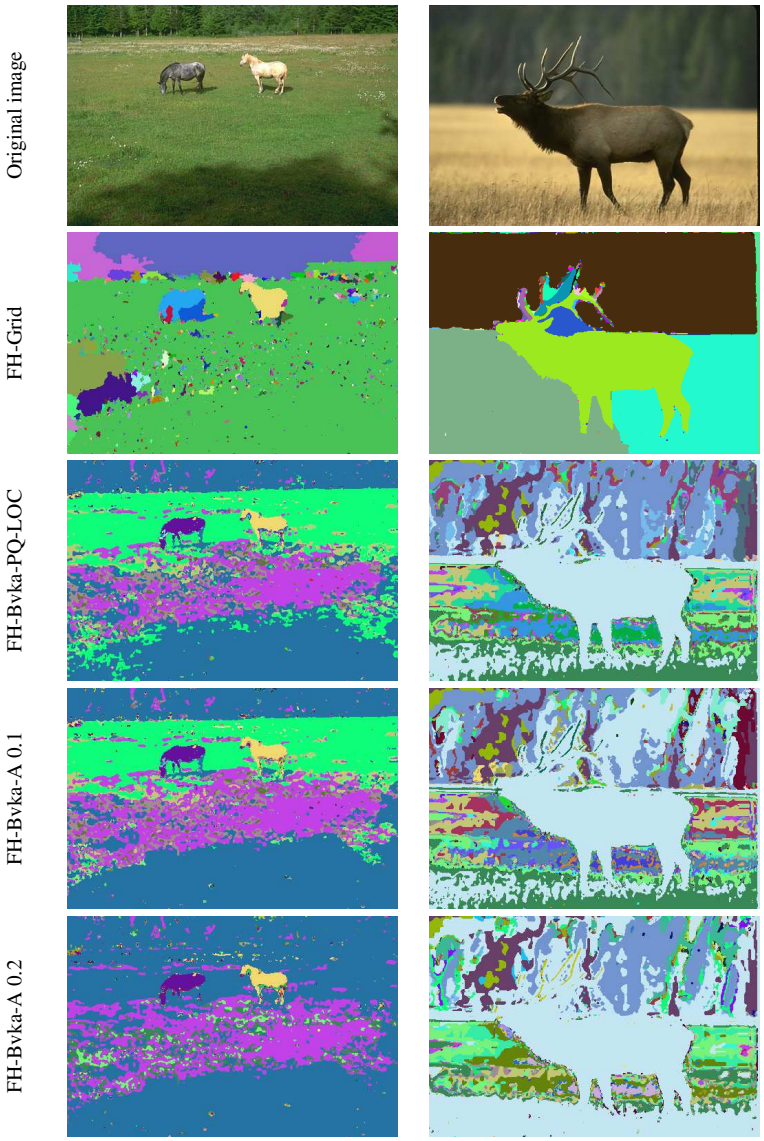


FIG 6. *Effect of using local, complete and approximate complete graphs. The oversegmentation produced by FH-Grid has been corrected by all the proposed methods. Notice also that the approximate versions of FH-Bvka-A show an stable behavior.*

within a bucket can be implemented using vectorial processors as the bucket size is fixed.

Acknowledgements

The authors wish to thanks Pablo Cancela for his fruitful comments and acknowledge financial support by CNES (R&T Echantillonnage Irregulier DCT / SI / MO - 2010.001.4673), FREEDOM (ANR07-JCJC-0048-01), Callisto (ANR-09-CORD-003), ECOS Sud U06E01, ARFITEC (07 MATRH) and STIC Amsud (11STIC-01 - MMVP-SCV) and the Uruguayan Agency for Research and Innovation (ANII) under grant PR-POS-2008-003.

References

A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Communications of the ACM*, 51(1):117–122, January 2008.

J. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, September 1975.

J. Bentley and J. Friedman. Fast algorithms for constructing minimal spanning trees in coordinate spaces. *IEEE Transactions on Computers*, 27(2):97–105, 1978.

T. Bozkaya and M. Ozsoyoglu. Distance-based indexing for high-dimensional metric spaces. In *Proceedings of the 1997 ACM SIGMOD international conference on Management of Data*, volume 26, pages 357–368, New York, NY, USA, June 1997. ACM.

M. Carreira-Perpiñán and R. Zemel. Proximity graphs for clustering and manifold learning. In *Advances in Neural Information Processing Systems*, volume 17, pages 225–232, 2005.

E. Chavez and G. Navarro. An effective clustering algorithm to index high dimensional metric spaces. In *Proceedings of the Seventh International Symposium on String Processing Information Retrieval*, pages 75+, Washington, DC, USA, 2000. IEEE Computer Society.

E. Chávez and G. Navarro. A compact space decomposition for effective metric indexing. *Pattern Recognition Letters*, 26(9):1363–1376, 2005.

B. Chazelle. A minimum spanning tree algorithm with inverse-ackermann type complexity. *Journal of the ACM*, 47(6):1028–1047, November 2000.

P. Ciaccia and M. Patella. Bulk loading the m-tree. In *Proceedings of the 9th Australasian Database Conference*, pages 15–26, 1998.

T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. McGraw-Hill Science / Engineering / Math, 2nd edition, December 2003.

T. Cour, F. Benezit, and J. Shi. Spectral segmentation with multiscale graph decomposition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 1124–1131, Washington, DC, USA, 2005. IEEE Computer Society.

M. Dry, D. Navarro, K. Preiss, and M. Lee. The perceptual organization of point constellations. In *Annual Meeting of the Cognitive Science Society*, 2009.

W. Eddy, A. Mockus, and S. Oue. Approximate single linkage cluster analysis of large data sets in high-dimensional spaces. *Computational Statistics & Data Analysis*, 23 (1):29–43, 1996.

P. Felzenszwalb and D. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(2):167–181, September 2004.

C. Fowlkes and J. Malik. How much does globalization help segmentation? Technical report, 2004.

A. Frank and A. Asuncion. UCI machine learning repository, 2010. URL http://archive.ics.uci.edu/ml.

K. Fukunaga. *Introduction to Statistical Pattern Recognition*. Academic Press, second edition, 1999.

R. Graham and P. Hell. On the history of the minimum spanning tree problem. *Annals Of The History Of Computing*, 7(1):43–57, 1985.

A. K. Jain and R. C. Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.

D. Karger, P. Klein, and R. Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *Journal of the ACM*, 42(2):321–328, 1995.

C. Lai, T. Rafa, and D. Nelson. Approximate minimum spanning tree clustering in high-dimensional space. *Intelligent Data Analysis*, 13(4):575–597, 2009.

B. Leibe, K. Mikolajczyk, and B. Schiele. Efficient clustering and matching for object class recognition. In *Proceedings of British Machine Vision Conference*, 2006.

D. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, November 2004.

D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proceedings IEEE International Conference on Computer Vision, 2001*, volume 2, pages 416–423, 2001.

M. Muja and D. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *VISAPP International Conference on Computer Vision Theory and Applications*, pages 331–340, 2009.

F. Murtagh. A survey of recent advances in hierarchical clustering algorithms. *The Computer Journal*, 26(4):354–359, November 1983.

S. Pettie and V. Ramachandran. An optimal minimum spanning tree algorithm. *Journal of the ACM*, 49(1):16–34, January 2002.

H. Samet. Depth-first k-nearest neighbor finding using the maxnearestdist estimator. In *International Conference on Image Analysis and Processing*, volume 0, pages 486+, Los Alamitos, CA, USA, 2003. IEEE Computer Society.

C. Silpa-Anan and R. Hartley. Optimised kd-trees for fast image descriptor matching. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 0, pages 1–8, Los Alamitos, CA, USA, 2008. IEEE Computer Society.

T. Skopal, J. Pokorný, M. Krátký, and V. Snášel. Revisiting m-tree building principles. In *Advances in Databases and Information Systems*, volume 2798, pages 148–162. Springer, September 2003.

T. Skopal, J. Pokorný, and V. Snášel. Nearest neighbours search using the pm-tree. In

Database Systems for Advanced Applications, pages 803–815, 2005.

R. Tarjan and J. Van Leeuwen. Worst-case analysis of set union algorithms. *Journal of the ACM*, 31(2):245–281, April 1984.

J. Tenenbaum, V. De Silva, and J. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, December 2000.

J. Toyama, M. Kudo, and H. Imai. Probably correct k-nearest neighbor search in high dimensions. *Pattern Recognition*, 43(4):1361–1372, April 2010.

P. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, pages 311–321, Philadelphia, PA, USA, 1993. Society for Industrial and Applied Mathematics.

C. T. Zahn. Graph-theoretical methods for detecting and describing gestalt clusters. *Transactions on Computers*, C-20(1):68–86, 1971.

X. Zhou, G. Wang, J. Xu Yu, and G. Yu. M^+-tree: a new dynamical multidimensional index for metric spaces. In *Proceedings of the 14th Australasian Database Conference*, pages 161–168, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc.