THE EUCLIDEAN K-DISTANCE TRANSFORMATION IN ARBITRARY DIMENSIONS: A SEPARABLE IMPLEMENTATION

Olivier Cuisenaire

Signal Processing Institute (ITS)
Ecole Polytechnique Fédérale de Lausanne (EPFL)
CH-1015 Lausanne, Switzerland

ABSTRACT

The signed k-distance transformation (k-DT) computes the k nearest prototypes from each location on a discrete regular grid within a given D dimensional volume. We propose a new k-DT algorithm that divides the problem into D 1-dimensional problems and compare its accuracy and computational complexity to the existing raster-scanning and propagation approaches.

1. INTRODUCTION

The signed k-distance transformation (k-DT) computes the k nearest prototypes from each location on a discrete regular grid within a given D dimensional volume. It was introduces by Warfield [1] as a way to speed up k-NN classification when the size of the feature space was smaller than or comparable to the size of the data to classify. A typical example of such a problem is the classification of tissues in multichannel MR images. The data is a 3-dimensional MR volume while the feature space is typically a 1-, 2- or 3-dimensional space. The algorithm proposed by Warfield relies on 2^D raster scans over the data, one from each corner of the volume to its opposite corner. Locally, it performs a fast sorting of the k nearest prototypes from the up to $k.(D+1)$ prototypes nearest to the current location or to the D direct neighbors already reached by the current scan. It 2D it uses either the approximate chamfer distance in 2 raster scans or the special raster scanning first proposed by Danielsson for the Euclidean 1-DT.

A faster implementation relying on ordered propagation was proposed by Cuisenaire and Macq [2]. The propagation fronts from the k nearest prototypes reach each location before the propagation from any other prototype, which avoids unnecessary computations and removes the need of an explicit sorting process. On the other hand, handling of the ordered propagation front adds to the computational complexity, which nevertheless remains well below that of Warfield's approach.

These are the only two k-DT algorithms we are aware of. Both rely on generalization of the Euclidean 1-DT, which has been the topic of extensive research. An detailed state of the art up to the year 1999 can be found in [3]. It includes a number of raster-scanning algorithms which inspired Warfield's k-DT and propagation algorithms that inspired our previous work [2]. But among 1-DT algorithms, a particularly efficient approach divides the D-dimensional problem into D 1-dimensional problems [4, 5, 6, 7]. This approach - among the most efficient in 2D and by far the most efficient in 3 and more dimensions - has not yet been extended to the k-DT.

This is the topic of this paper, which is organized as follows. In section 2 we recall the principles behind all the separable Euclidean 1-DT algorithms. In section 3 we show how these can be extended the k-DT and describe the resulting algorithm in details. Section 4 discusses and computational complexity and compares it to the previous approaches.

2. THE SEPARABLE 1-DT

The Euclidean 1-DT is the transformation that computes for each pixel $\mathbf{p}$ the distance to the nearest pixel from a set X, i.e.

$$D_X(\mathbf{p}) = \min_{\mathbf{x}\in X}(\|\mathbf{p}-\mathbf{x}\|) \tag{1}$$

Often, we are actually interested in the identity of this pixel instead of merely knowing its distance. This is sometimes called signed DT, Voronoi transform or feature transform. We write it

$$\mathbf{V}_X(\mathbf{p}) = \arg\min_{\mathbf{x}\in X}(\|\mathbf{p}-\mathbf{x}\|) \tag{2}$$

In [4, 5, 6, 7], the D-dimensional problem is split into D 1-dimensional problems. When processing the dth dimension, one produces an intermediate result

$$D_{X,d}(\mathbf{p}) = \min_{\mathbf{x}\in X_d(\mathbf{p})}(\|\mathbf{p}-\mathbf{x}\|) \tag{3}$$

$$X_d(\mathbf{p}) = \{\mathbf{x}=(x_1,...,x_D) : \forall i > d, x_i = p_i\} \cap X \tag{4}$$

where $X_d(\mathbf{p})$ is the set of object points with the same coordinates as $\mathbf{p}$ for $i > d$. Initially, $D_{X,0}$ is 0 for objects pixels and ∞ for background pixels. Then, the algorithms proceed iteratively by computing, for all $\mathbf{p}$,

$$D_{X,d}(\mathbf{p}) = \min_{\mathbf{q}\in L_d(\mathbf{p})}(\sqrt{D_{X,d-1}(\mathbf{q})^2 + \|\mathbf{p}-\mathbf{q}\|^2}) \tag{5}$$

$$L_d(\mathbf{p}) = \{\mathbf{q} : \forall i \neq d, p_i = q_i\} \tag{6}$$

where $L_d(\mathbf{p})$ is the line of pixels with all coordinates identical to $\mathbf{p}$'s except for the dth. After iterating over the dimensions from $d=1$ to D, we get $D_X = D_{X,D}$.

How (5) is implemented practically varies between authors. Obviously all of them get rid of the square root operation by computing $D^2_{X,d}$ instead of $D_{X,d}$. Also, they use the alignment of $\mathbf{p}$ and $\mathbf{q}$ to simplify $\|\mathbf{p}-\mathbf{q}\|$. Equation (5) becomes

$$D_{X,d}(\mathbf{p})^2 = \min_{\mathbf{q}\in L_d(\mathbf{p})}(D_{X,d-1}(\mathbf{q})^2 + (p_d - q_d)^2) \tag{7}$$

Beyond this, the faster algorithms [5, 6, 7] rely on the property that the intersection of the Voronoi partition $\mathbf{V}_X$ and the line

$L_d(\mathbf{p})$ is ordered, i.e. that if we have two pixels $\mathbf{q}$ and $\mathbf{r} \in L_d(\mathbf{p})$ such that $q_d < r_d$, then the nearest pixels $\mathbf{x} = \mathbf{V}_{X,d}(\mathbf{q})$ and $\mathbf{y} = \mathbf{V}_{X,d}(\mathbf{r})$ are such that $x_d \leq y_d$. Unfortunately, this property does not hold for the k nearest pixels, and therefore those algorithms cannot be extended to implement the k-DT.

On the other hand, Saito [4] implements (7) in two scans over $L_d(\mathbf{p})$, and relies on a couple of heuristic rules to restrict the search intervals. While these heuristic only work for the 1-DT, a similar approach can be attempted for the k-DT, as we show in the next section.

3. THE SEPARABLE K-DT

3.1. Notations

When considering the k-DT, there is no reason to restrict ourselves to an object set X where each object pixel only appears once. Instead, we use an explicit representation where each of the object pixels $\mathbf{x}(i)$ is identified by its index i with $1 \leq i \leq S_X$

We note $N(\mathbf{p}, m)$ the index of the m^{th} nearest object pixel from $\mathbf{p}$. For all m, n such that $1 \leq m < n \leq S_X$, we have

$$\|\mathbf{p} - \mathbf{x}(N(\mathbf{p}, m))\| \leq \|\mathbf{p} - \mathbf{x}(N(\mathbf{p}, n))\| \tag{8}$$

This relates to the previously defined $\mathbf{V}_X$ via

$$\mathbf{V}_X(\mathbf{p}) = \mathbf{x}(N(\mathbf{p}, 0)) \tag{9}$$

The aim of the k-DT is to compute $N(\mathbf{p}, m)$ for all $m \leq k$ and for all $\mathbf{p} \in I$ where I is a D-dimensional volume of size $S_1 \times ... \times S_D$.

3.2. Principle

Similarly to the Euclidean DT algorithms, the principle of our approach is to split this D-dimensional problem into D 1-dimensional ones. Initially, we consider $N_0(\mathbf{p}, m)$ for which the indices under consideration are only those such that $\mathbf{x}(i) = \mathbf{p}$.

Then, for $N_1(\mathbf{p}, m)$, we consider the indices such that $\mathbf{x}(i)$ is in the same line as $\mathbf{p}$, i.e. $x(i)_j = p_j$ for $1 < j \leq D$. It can be computed from the N_0 by considering the $k.S_1$ indices $N_0(\mathbf{q}, m)$ where $\mathbf{q}$ is in the same line as $\mathbf{p}$, i.e $p_j = q_j$ for $j \neq 1$.

More generally, $N_d(\mathbf{p}, m)$ considers only indices such that $x(i)_j = p_j$ for $d < j \leq D$. It can be computed from the $k.S_d$ indices $N_{d-1}(\mathbf{q}, m)$ for $\mathbf{q}$ d-aligned with $\mathbf{p}$, i.e $p_j = q_j$ for $j \neq d$. By iterating over the dimensions from $d = 1$ to D, we end up with $N(\mathbf{p}, m) = N_D(\mathbf{p}, m)$.

This idea is formalized in algorithm 1 where we use the set $I_d = \{\mathbf{p} : \mathbf{p} \in I, p_d = 0\}$ to iterate over all lines in the d^{th} dimension, of which $\mathbf{e_d}$ is the unit vector. For memory efficiency, we consider a single array N instead of one per dimension. Indeed, N has a size $k \times S_1 \times ... \times S_D$ which is typically quite large. Instead, we do consider N_{d-1} and N_d separately for the 1-dimensional sweeps. For computational efficiency, we also store Δ_{d-1} and Δ_d, the square of the distances to corresponding to the indices in N_{d-1} and N_d. These 4 arrays each have a size $k \times S_d$ which is typically negligible compared to N's.

3.3. Computing N_d from N_{d-1} in 1D

Let us now consider how to compute N_d from N_{d-1}. First, let us notice that, for all $m \leq k$, is is possible to compute $N_d(\mathbf{p}, m)$ for all $n \leq k$ by using only $N_{d-1}(\mathbf{q}, n)$ at pixels $\mathbf{q}$ d-aligned with $\mathbf{p}$, i.e. such that $p_j = q_j, \forall\ j \neq d$. Indeed, if we consider $i = N_d(\mathbf{p}, m)$, then we must have $i = N_{d-1}(\mathbf{q}, n)$ for some $n \leq m$ and for $\mathbf{q}$ such that $q_d = x(i)_d$ and $q_j = p_j, \forall\ j \neq d$, since $N_{d-1}(\mathbf{q}, .)$ considers a subset of the indices considered by $N_d(\mathbf{p}, .)$.

Similarly to Saito [4], we use a write-paradigm instead of the read-paradigm considered so far. In other words, instead of trying to compute a given $N_d(\mathbf{p}, m)$ by considering $N_{d-1}(\mathbf{q}, n)$ for all values of $\mathbf{q}$ and n, we consider a given $N_{d-1}(\mathbf{p}, m)$ and assess whether it can be propagated to $N_d(\mathbf{q}, n)$ for all values $\mathbf{q}$ and some $n \leq k$. This is performed in two scans over the line. First we scan for increasing values of p_d and propagate backward for q_d between $p_d - 1$ and 1, i.e.

for $p_d = 1 : S_d$ **do**
  for $m = 1 : k$ **do**
    for $q_d = p - 1 : -1 : 1$ **do**
      check if $N_{d-1}(\mathbf{p}, m)$ should be added to $N_d(\mathbf{q}, .)$
    end for
  end for
end for

for all $\mathbf{p} \in I$ **do** {Initialization}
  for $m = 1 : k$ **do**
    $N(\mathbf{p}, m) \leftarrow \infty$
  end for
end for

for $i = 1 : S_X$ **do** {Compute N_0}
  $m \leftarrow 1$
  while $(N(\mathbf{x}(i), m) \neq \infty) \wedge (m \leq k)$ **do**
    $m \leftarrow m + 1$
  end while
  if $m \leq k$ **then**
    $N(\mathbf{x}(i), m) \leftarrow i$
  end if
end for

for $d = 1 : D$ **do** {Compute N_d from N_{d-1}}
  for all $\mathbf{p}' \in I_d$ **do**
    for $p_d = 1 : S_d$ **do**
      $\mathbf{p} \leftarrow \mathbf{p}' + p_d.\mathbf{e_d}$
      for $m = 1 : k$ **do**
        $N_{d-1}(p_d, m) \leftarrow N(\mathbf{p}, m)$
        $\Delta_{d-1}(p_d, m) \leftarrow \|\mathbf{p} - \mathbf{x}(N(\mathbf{p}, m))\|^2$
      end for
    end for
    kdt_1d($N_{d-1}, \Delta_{d-1}, N_d$)
    for $p_d = 1 : S_d$ **do**
      $\mathbf{p} \leftarrow \mathbf{p}' + p_d.\mathbf{e_d}$
      for $m = 1 : k$ **do**
        $N(\mathbf{p}, m) \leftarrow N_d(p_d, m)$
      end for
    end for
  end for
end for

Algorithm 1: Computes the k-distance transformation $N(\mathbf{p}, m)$ for all locations $\mathbf{p} \in I$ and all values $1 \leq m \leq k$. $N(\mathbf{p}, m)$ is the index of the m^{th} nearest neighbor from $\mathbf{p}$ in the explicit set of prototypes $X = \{\mathbf{x}(i), 1 \leq i \leq S_X\}$

Then, we do the reverse, scanning for decreasing values of p_d and propagating towards q_d between $p_d + 1$ and S_d.

Checking if $N_{d-1}(\mathbf{p}, m)$ propagates to $\mathbf{q}$ is simply done by comparing the squared distance to the currently stored k^{th} nearest neighbor for $\mathbf{q}$

$$\|\mathbf{q} - \mathbf{x}(N(\mathbf{q}, k))\|^2 = \Delta_d(\mathbf{q}, k) \quad (10)$$

with the squared distance to $N_{d-1}(\mathbf{p}, m)$, i.e.

$$\|\mathbf{q} - \mathbf{x}(N_{d-1}(\mathbf{p}, m))\|^2 = \Delta_{d-1}(\mathbf{p}, m) + (p_d - q_d)^2 \quad (11)$$

Both expressions are easy to evaluate once we store the $k.S_d$ values of both Δ_{d-1} and Δ_d for the current line. If (11) is smaller than (10), then $N_{d-1}(\mathbf{p}, m)$ needs to be inserted at the proper level among the k nearest neighbors of $\mathbf{q}$.

Finally, we need to add a proper heuristic in order to restrict the range of values which q_d scans. In his 1-DT algorithm [4], Saito scans q_d in the same direction as p_d and gets an upper bound for q_d as $p_d + (\Delta_{d-1}(p_d + 1, 0) - \Delta_{d-1}(p_d, 0) + 1)/2$. Indeed, for larger values, the distance to $N(p_d, 0)$ becomes always larger than the distance for $N(p_d + 1, 0)$.

It would be possible to adapt this method directly to the k-DT and bound q_d by $p_d + (\Delta_{d-1}(p_d + 1, k - m + 1) - \Delta_{d-1}(p_d, m) + 1)/2$ when propagating $N_{d-1}(p_d, m)$. Unfortunately, this turns out to be relatively inefficient as only nearest neighbors of p_d and $p_d + 1$ can stop the propagation. Instead, it is more efficient to back-propagate for decreasing values of q_d in the scan where p_d increases and vice-versa. Then, as soon as $N_{d-1}(p_d, m)$ does not fulfill (11) smaller than (10), we can stop propagating. With such a scheme, all neighbors for $q_d \leq p_d$ can contribute to stop the propagation. The same heuristic can also be used for the return scan with decreasing values of p_d.

Algorithm 2 summarizes the considerations of this section and proved a detailed description of the resulting method. The matlab notation is used for the loops' limits and steps.

```
for p = 1 : S_d do
  for m = 1 : k do
    N_d(p, m) ← N_{d-1}(p, m)
    Δ_d(p, m) ← Δ_{d-1}(p, m)
  end for
end for
for p = 1 : S_d do
  for m = 1 : k do
    propagate(p, m, 1)
  end for
end for
for p = S_d : −1 : 1 do
  for m = 1 : k do
    propagate(p, m, −1)
  end for
end for

procedure propagate(p, m, α)
  if N_{d-1}(p, m) ≠ ∞ then
    q ← p − α
    δ ← Δ_{d-1}(p, m) + α^2
    l ← m
    while (δ < Δ_d(q, k)) ∧ (1 ≤ q ≤ S_d) do
      while δ ≥ Δ_d(q, l) do
        l ← l + 1
      end while
      for n = k : −1 : (l + 1) do
        Δ_d(q, n) ← Δ_d(q, n − 1)
        N_d(q, n) ← N_d(q, n − 1)
      end for
      Δ_d(q, l) ← δ
      N_d(q, l) ← N_{d-1}(p, m)
      q ← q − α
      δ ← Δ_{d-1}(p, m) + (p − q)^2
    end while
  end if
end procedure
```

Algorithm 2: Procedure kdt_1d($N_{d-1}, \Delta_{d-1}, N_d$), computes N_d from N_{d-1}

4. COMPUTATIONAL COMPLEXITY

A theoretical assessment of the computational complexity of this method is not an easy task. In [3], experiments showed that Saito's Euclidean 1-DT algorithm [4] - which has significant similarities with this one - has an image-dependent complexity. For a D-dimensional image of size S in each dimension, the computational cost can vary between $\circ(S^D)$ and $\circ(S^{D+1})$. Besides, even in the later case, there is a $\circ(S^D)$ term that dominates for values of S up to approximately 300.

Instead, we chose to evaluate the computational complexity of our method experimentally and compare it with the raster scanning algorithm of Warfield [1] and our previous propagation algorithm [2]. The algorithms were implemented in C and run on a 3GHz Pentium 4 computer with 2GB of RAM, 512KB of L2 and 8KB of L1 cache.

At figure 1, we compute the k-DT in 3 dimensions, finding the k nearest neighbors among a 1000 samples from two gaussian distributions located in opposite octants of the volume. The size S of the cubic volume over which we compute the k-DT is determined such that the total amount of data generated is approximately constant, i.e. $k \times S \times S \times S \approx 64.10^6$. We vary k between 2 and 40, and therefore S between 317 and 117.

The separable approach is consistently faster than the previous methods. While its computational time does increase slightly with k, it is between 2 and 4 times faster than the propagation method, and between 4.5 and 9 times faster than the raster scanning approach.

Additional experiments in 3D with different numbers or positions of the samples - as well as with different total amount of data generated - lead to similar conclusions.

5. DISCUSSION

While the previous section is focused on computational complexity, the separable k-DT offers a number of additional benefits compared to the propagation or raster scanning methods.

Firstly, it computes the exact k-DT and is the first k-DT algorithm to do so. Indeed, both propagation and raster scanning can make a few isolated errors similar the errors of similar Euclidean 1-DT algorithms such as Danielsson's [8].

Secondly, it is extremely easy to parallelize as each line can be processed independently, which provides a linear speed-up with the number of processors used.

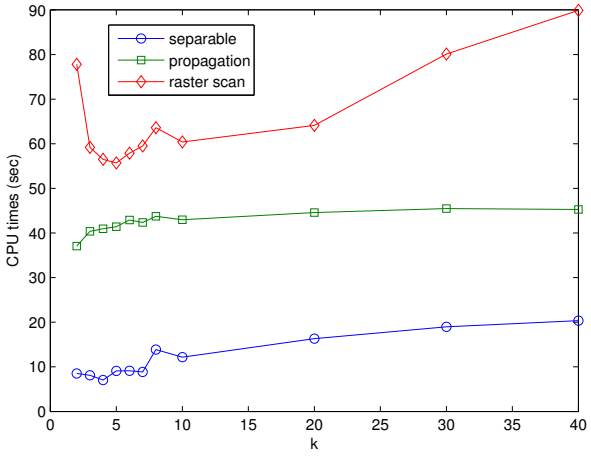

Fig. 1. Computational times for the k-DT algorithms computed with 1000 prototypes. k varies between 2 and 40 and the size S of the cubic volume in which the k-DT is computed is chosen so that $k \times S \times S \times S \approx 64.10^6$

Thirdly, compared to the propagation method, it requires a very low memory overhead compared to the amount of data it generates.

Finally, let us note that the experiments of the previous section were performed in 3 dimensions. In 2D, the results are less impressive. While the algorithm still perform efficiently, it is not always significantly outperforming the other methods. On the other hand, in $D > 3$ dimensions, the new method is both easier to implement and very significantly faster than the alternatives

6. CONCLUSION

We have developed a new algorithm to compute the Euclidean k-distance transformation, which generalizes the classical Euclidean DT by computing not only the nearest object point but the k nearest points, and can be applied to implement k-NN classification of multi-channel data.

The proposed method is both more accurate and significantly faster than previously published algorithms, especially in $D > 3$ dimensions.

7. REFERENCES

[1] S. Warfield, "Fast k-nn classification for multichannel data," *Pattern Recognition Letters*, vol. 17, pp. 713, 1996.

[2] O. Cuisenaire and B. Macq, "Fast k-nn classification with an optimal k-distance transformation algorithm," in *Proc. 10th European Signal Processing Conference*, September 2000, pp. pp. 1365–1368.

[3] O. Cuisenaire, *Distance transformations: fast algorithms and applications to medical image processing*, Ph.D. thesis, Université catholique de Louvain (UCL), Louvain-la-Neuve, Belgium, October 1999.

[4] T. Saito and J.I. Toriwaki, "New algorithms for euclidean distance transformations of an n-dimensional digitised picture with applications," *Pattern Recognition*, vol. 27, no. 11, pp. 1551–1565, 1994.

[5] T. Hirata, "A unified linear-time algorithm for computing distance maps," *Information Processing Letters*, vol. 58, no. 3, pp. 129–133, 1996.

[6] A. Meijster, J.B.T.M. Roerdink, and W.H. Hesselink, "A general algorithm for computing distance transforms in linear time," in *Mathematical Morphology and its applications to image and signal processing*, J. Goutsias, L. Vincent, and D.S. Bloomberg, Eds., 2000, pp. 331–340.

[7] C.R. Maurer Jr, R. Qi, and V. Raghavan, "A linear time algorithm for computing exact euclidean distance transforms of binary images in arbitrary dimensions," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 2, pp. 265–270, 2003.

[8] P.E. Danielsson, "Euclidean distance mapping," *Computer Graphics and Image Processing*, vol. 14, pp. 227–248, 1980.