Distance Transforms of Sampled Functions

Pedro F. Felzenszwalb and Daniel P. Huttenlocher
{pff,dph}@cs.cornell.edu
Department of Computer Science
Cornell University
Ithaca, NY 14853

1 Introduction

Distance transforms are an important tool in computer vision, image processing and pattern recognition. A distance transform of a binary image specifies the distance from each pixel to the nearest non-zero pixel. Distance transforms play a central role in the comparison of binary images, particularly for images resulting from local feature detection techniques such as edge or corner detection. For example, both the Chamfer [3] and Hausdorff [8] matching approaches use distance transforms for comparing binary images. Distance transforms have also been used to compute the medial axis of a digital shape [1].

In this paper we consider a generalization of distance transforms to arbitrary functions on a grid rather than binary-valued ones (i.e., arbitrary images rather than binary images). There is a simple intuition underlying this generalization. Rather than a binary feature map that specifies the presence or absence of a feature at each pixel, it can be useful to have a map that specifies a cost for a feature at each pixel. One can think of a binary edge map as a restricted case, where the costs are restricted to be 0 (at edge pixels) or infinite (at non-edge pixels). For the more general feature maps it is again useful to compute a type of distance transform.

Let $\mathcal{G}$ be a regular grid and $f : \mathcal{G} \rightarrow \mathbb{R}$ a function on the grid. We define the distance transform of f to be

$$\mathcal{D}_f(p) = \min_{q \in \mathcal{G}} (d(p,q) + f(q)) \ , \tag{1}$$

where $d(p, q)$ is some measure of the distance between p and q. Intuitively, for each point p we find a point q that is close to p, and for which $f(q)$ is small. Note that if f has a small value at some location, $\mathcal{D}_f$ will have small value at that location and any nearby point, where nearness is measured by the underling distance $d(p, q)$.

The definition in equation (1) is closely related to the traditional distance transform of a set of points on a grid $P \subseteq \mathcal{G}$, which associates to each grid location the distance to the nearest point in P,

$$\mathcal{D}_P(p) = \min_{q \in P} d(p - q).$$

Many algorithms for computing the distance transform of point sets use the following alternative definition

$$\mathcal{D}_P(p) = \min_{q \in \mathcal{G}} (d(p - q) + 1(q)),$$

where $1(q)$ is an indicator function for membership in P,

$$1(q) = \begin{cases} 0 & \text{if } q \in P \\ \infty & \text{otherwise} \end{cases}$$

This alternative definition is almost the same as the definition for the distance transform of a function in equation (1), except that it uses the indicator function $1(q)$ rather than an arbitrary function $f(q)$.

When distance is measured using the l_1 norm the distance transform of a function can be computed using existing algorithms for computing the distance transform of a point set. When distance is measured by the squared Euclidean norm we introduce a new linear time algorithm for computing the distance transform of a function. This in turn provides a new technique for computing the exact Euclidean distance transform of a binary image by using indicator functions.

Efficient algorithms for computing the distance transform of a binary image using the l_1 and l_∞ distances were developed by Rosenfeld and Pfaltz [12]. Similar methods described in [2] have been widely used to efficiently compute approximations to the Euclidean distance transform. There are a number of techniques for computing the exact Euclidean distance transform of a binary image in linear time (e.g., [9, 4, 10]), however these methods are quite involved and are not widely used in practice. In contrast, our algorithm is relatively simple, easy to implement and very fast in practice.

We use the terminology distance transform of a *sampled function* rather than of an *image* for two reasons. First, we want to stress that the input is generally some kind of cost function that is being transformed so as to incorporate spatial (or distance) information. Second, there already are methods for computing distance transforms of grey level images based on minimum distances along paths, where the cost of a path is the sum of gray level values along the path [13]. We want to avoid confusion with these methods which compute something quite different from what we consider here.

1.1 Energy Minimization Problems

In addition to extending the applicability of distance transforms from binary feature detectors to "soft" multi-valued feature quality measures, distance transforms of functions arise in a number of optimization problems that are related to the Viterbi recurrence. Recently we have used these methods to develop improved algorithms for recognition of articulated objects [5], inference with large state-space hidden Markov models [7], and the solution of low-level vision problems such as stereo, image restoration and optical flow using loopy belief propagation [6].

The standard Viterbi recurrence that is used for inference using hidden Markov models can be written as

$$\delta_{t+1}(q) = b_{t+1}(q) + \min_p(\delta_t(p) + a(p,q)), \tag{2}$$

where $b_{t+1}(q)$ is the cost of state (or label) q at time $t+1$ and $a(p,q)$ is the cost of transitioning from state p to state q at two successive time steps (see [11] for a good tutorial on hidden Markov models). For our purposes a detailed understanding of this equation is not as important as observing its form. The minimization in the second term on the right hand side of the equation is closely related to the distance transform of a function. In particular, if the states are embedded in a grid and the transition costs $a(p,q)$ can be viewed as a distance between the corresponding grid locations, then the Viterbi recurrence can be written in terms of a distance transform,

$$\delta_{t+1}(q) = b_{t+1}(q) + \mathcal{D}_{\delta_t}(q) \ .$$

Thus efficient algorithms for computing the distance transform of a function apply to certain problems that can be characterized by equations of the

form in (2). For example in the case of a hidden Markov model with n states the standard computation of the Viterbi recurrence takes $O(n^2)$ time which is not practical for large values of n, while the computation using distance transform techniques takes $O(n)$ time. Similarly in applying belief propagation for solving low-level vision problems, the running time can be reduced from quadratic to linear in the number of possible labels for each pixel. This makes belief propagation practical for problems such as motion estimation, where the number of labels is relatively large.

2 Squared Euclidean Distance

2.1 One Dimension

Let $\mathcal{G} = \{0, \ldots, n-1\}$ be a one dimensional grid, and $f : \mathcal{G} \rightarrow \mathbb{R}$ an arbitrary function on the grid. The squared Euclidean (or quadratic) one-dimensional distance transform of f defined by equation (1) is given by,

$$\mathcal{D}_f(p) = \min_{q \in \mathcal{G}}((p-q)^2 + f(q)).$$

Note that for each point $q \in \mathcal{G}$ there is a constraint that the distance transform of f be bounded by a parabola rooted at $(q, f(q))$. In fact the distance transform is defined by the lower envelope of these parabolas, as shown in Figure 1. The value of the distance transform at p is simply the height of the lower envelope at that point.

We give a simple linear time algorithm for computing the distance transform under the squared Euclidean distance. Our algorithm has two distinct steps. First we compute the lower envelope of the n parabolas just mentioned. We then fill in the values of $\mathcal{D}_f(p)$ by checking the height of the lower envelope at each grid location. This is a unique approach because we start with something defined on a grid (the values of f), move to a combinatorial structure defined over the whole domain (the lower envelope of the parabolas) and move back to values on the grid by sampling the lower envelope. Pseudocode for the whole procedure is shown in Algorithm 1.

The main part of the algorithm is the lower envelope computation. Note that any two parabolas defining the distance transform intersect at exactly one point. Simple algebra yields the horizontal position of the intersection

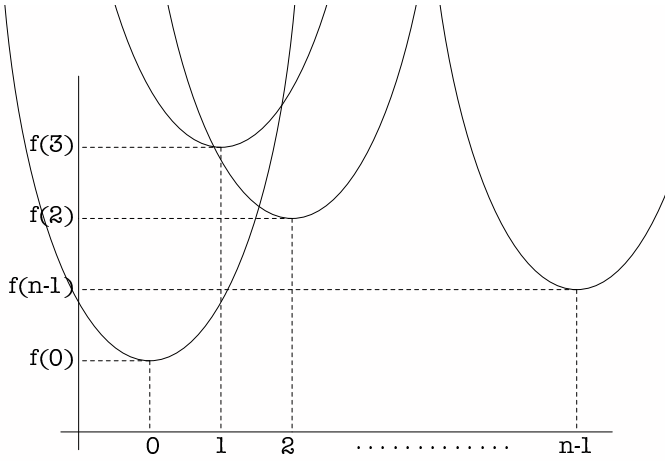


Figure 1: The distance transform as the lower envelope of n parabolas.

between the parabola coming from grid position q and the one from p as,

$$x = \frac{(f(p) + p^2) - (f(q) + q^2)}{2p - 2q}.$$

If $q < p$ then the parabola coming from q is below the one from p to the left of the intersection x, and above it to the right of x.

We can compute the lower envelope by sequentially computing the lower envelope of the first i parabolas, where the parabolas are ordered according to their corresponding horizontal grid locations. The algorithm works by computing the combinatorial structure of the lower envelope defining the distance transform. We keep track of this structure by using two arrays. The horizontal grid location of the j-th parabola in the lower envelope is stored in $v[j]$. The range in which the j-th parabola of the lower envelope is below the others is given by $z[j]$ and $z[j+1]$. The variable k keeps track of the number of parabolas in the lower envelope.

When considering the parabola from q, we find its intersection with the parabola from $v[k]$ (the rightmost parabola in the lower envelope computed so far). There are two possible cases. If the intersection is after $z[k]$, then the lower envelope is modified to indicate that the parabola from q is below the other parabolas in the lower envelope starting at the intersection point. If the intersection is before $z[k]$ then the parabola from $v[k]$ should not be part of the new lower envelope. In this case we decrease k to delete that parabola and repeat this procedure.

Algorithm $DT(f)$

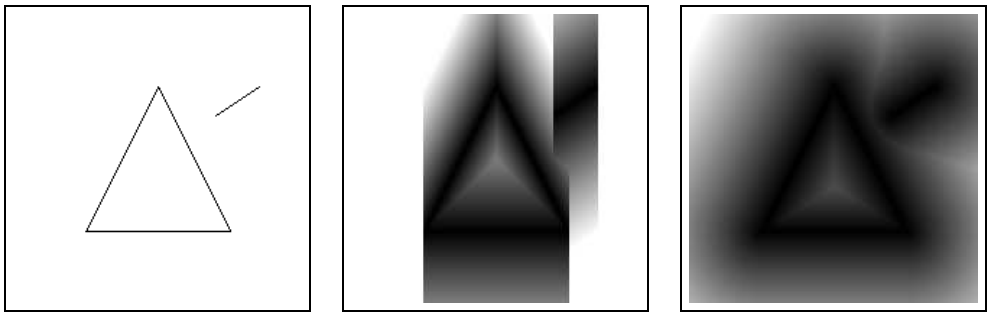
```
1.  k ← 0                          (* Index of rightmost parabola in lower envelope *)
2.  v[0] ← 0                       (* Locations of parabolas in lower envelope *)
3.  z[0] ← −∞                      (* Locations of boundaries between parabolas *)
4.  z[1] ← +∞
5.  for q = 1 to n − 1                                  (* Compute lower envelope *)
6.      s ← ((f(q) + q^2) − (f(v[k]) + v[k]^2))/(2q − 2v[k])
7.      if s ≤ z[k]
8.        then k ← k − 1
9.             goto 6
10.       else k ← k + 1
11.            v[k] ← q
12.            z[k] ← s
13.            z[k + 1] ← +∞
14. k ← 0
15. for q = 0 to n − 1                  (* Fill in values of distance transform *)
16.     while z[k + 1] < q
17.           k ← k + 1
18.     D_f(q) ← (q − v[k])^2 + f(v[k])
```

Algorithm 1: The distance transform algorithm for the the squared Euclidean distance in one-dimension.

To understand the running time of the algorithm note that we consider adding each parabola to the lower envelope exactly once. The addition of one parabola may involve the deletion of many others, but each parabola is deleted at most once. So the overall computation of the lower envelope in lines 1 through 13 takes $O(n)$ time. The computation of the distance transform values from the lower envelope in lines 14 through 18 considers each grid position and each parabola in the lower envelope at most once, so the second part of the algorithm also takes $O(n)$ time.

2.2 Arbitrary Dimensions

Let $\mathcal{G} = \{0, \ldots, n-1\} \times \{0, \ldots, m-1\}$ be a two dimensional grid, and $f : \mathcal{G} \to \mathbb{R}$ an arbitrary function on the grid. The two dimensional distance



(a) (b) (c)

Figure 2: An input function $f(x, y)$ corresponding to a binary image is shown in (a). The transform of the input along each column is shown in (b). The final distance transform is shown in (c). Dark pixels correspond to low values of the sampled functions while bright pixels correspond to high values.

transform of f under the Euclidean distance squared is given by,

$$\mathcal{D}_f(x, y) = \min_{(x',y')\in\mathcal{G}}((x - x')^2 + (y - y')^2 + f(x', y')).$$

The first term does not depend on y' so we can rewrite this equation as,

$$\mathcal{D}_f(x, y) = \min_{x'}((x - x')^2 + \min_{y'}((y - y')^2 + f(x', y'))).$$

Thus the two dimensional transform can be computed by first performing one dimensional transforms along each column of the grid, and then performing one dimensional transforms along each row of the result. This argument extends to arbitrary dimensions, resulting in the composition of transforms along each dimension of the underlying grid. Note that changing the order of these transforms yields the same result, as can be seen readily for the two-dimensional case above.

Figure 2 illustrates the computation of the two-dimensional transform of a binary picture using this method, where we start with the indicator function for the points in the grid. The notion of a distance transform for arbitrary sampled functions is important to be able to compute the two-dimensional transform by sequentially applying the one-dimensional algorithm.

3 l_1 Norm

For a set of points on a one-dimensional grid, the distance transform under the l_1 norm can be computed using a simple two-pass algorithm (e.g., [12]). Essentially the same algorithm can be used to compute the distance transform of a one-dimensional sampled function under the l_1 norm. Pseudocode for the method is shown in Algorithm 2.

Algorithm $DT(f)$
1. $\mathcal{D}_f \leftarrow f$ ($*$ Initialize $\mathcal{D}_f$ with f $*$)
2. **for** $q = 1$ **to** $n - 1$ ($*$ Forward pass $*$)
3. $\quad \mathcal{D}_f(q) \leftarrow \min(\mathcal{D}_f(q), \mathcal{D}_f(q-1) + 1)$
4. **for** $q = n - 2$ **to** 0 ($*$ Backward pass $*$)
5. $\quad \mathcal{D}_f(q) \leftarrow \min(\mathcal{D}_f(q), \mathcal{D}_f(q+1) + 1)$

Algorithm 2: The distance transform algorithm for the the l_1 norm in one-dimension.

It is straightforward to verify that this algorithm correctly computes the one-dimensional distance transform by local propagation of values. The values of the distance transform are initialized to the values of f itself. In the forward pass, each successive element of $\mathcal{D}_f(q)$ is set to the minimum of its own value and one plus the value of the previous element (this is done "in place" so that updates affect one another). In the backward pass each item is analogously set to the minimum of its own value and one plus the value of the next element. For example given the input $(4, 2, 8, 6, 1)$, after the first pass $\mathcal{D}_f$ be $(4, 2, 3, 4, 1)$, and after the second pass it will be $(3, 2, 3, 2, 1)$.

As with the squared Euclidean distance considered in the previous section, the two-dimensional transform can be computed by first performing one-dimensional transforms along each column of the grid, and then performing one-dimensional transforms along each row of the result (or vice versa). Higher dimensional transforms can analogously be computed by successive transforms along each coordinate axis.

4 Other distances

There are two simple relationships that can be used to compute distance transforms of functions under other distances not discussed so far. For ex-

ample, the distance $d(p,q) = \min(c(p-q)^2, a|p-q|+b)$ is commonly used in robust estimation and is very important in practice. Intuitively this distance is robust because it grows slowly after a certain point. We can compute the distance transform of a function under the robust distance by using the relationships described below.

First we consider the case where the distance between two points is given by the minimum of two other distances.

Theorem 1. $d(p,q) = \min(d^1(p,q), d^2(p,q)) \Rightarrow \mathcal{D}_f(p) = \min(\mathcal{D}_f^1(p), \mathcal{D}_f^2(p))$.

Proof. The result is straightforward,

$$\begin{aligned}
\mathcal{D}_f(p) &= \min_{q \in \mathcal{G}}(\min(d^1(p,q), d^2(p,q)) + f(q)) \\
&= \min_{q \in \mathcal{G}}(\min(d^1(p,q) + f(q), d^2(p,q) + f(q))) \\
&= \min(\min_{q \in \mathcal{G}}(d^1(p,q) + f(q)), \min_{q \in \mathcal{G}}(d^2(p,q) + f(q))) \\
&= \min(\mathcal{D}_f^1(p), \mathcal{D}_f^2(p)).
\end{aligned}$$

□

Now lets consider the case where the distance between two points is a scalar multiple of another distance plus a constant.

Theorem 2. $d^{(a,b)}(p,q) = ad(p,q) + b \Rightarrow \mathcal{D}_f^{(a,b)}(p) = a\mathcal{D}_{f/a}(p) + b$.

Proof. This is another simple result,

$$\begin{aligned}
\mathcal{D}_f^{(a,b)}(p) &= \min_{q \in \mathcal{G}}((ad(p,q) + b) + f(q)) \\
&= \min_{q \in \mathcal{G}}(ad(p,q) + f(q)) + b \\
&= a \min_{q \in \mathcal{G}}(d(p,q) + f(q)/a) + b \\
&= a\mathcal{D}_{f/a}(p) + b.
\end{aligned}$$

□

References

[1] H. Blum. Biological shape and visual science (part 1). *Theoretical Biology*, 38:205–287, 1973.

[2] G. Borgefors. Distance transformations in digital images. *Computer Vision, Graphics and Image Processing*, 34(3):344–371, 1986.

[3] G. Borgefors. Hierarchical chamfer matching: A parametric edge matching algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(6):849–865, November 1988.

[4] H. Breu, J. Gil, D. Kirkpatrick, and M. Werman. Linear-time euclidean distance transform algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(5):529–533, May 1995.

[5] P.F. Felzenszwalb and D.P. Huttenlocher. Pictorial structures for object recognition. To appear in the International Journal of Computer Vision.

[6] P.F. Felzenszwalb and D.P. Huttenlocher. Efficient belief propagation for early vision. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2004.

[7] P.F. Felzenszwalb, D.P. Huttenlocher, and J.M. Kleinberg. Fast algorithms for large-state-space HMMs with applications to web usage analysis. In *Advances in Neural Information Processing Systems*, 2003.

[8] D. Huttenlocher, G. Klanderman, and W. Rucklidge. Comparing images using the hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(9):850–863, September 1993.

[9] A.V. Karzanov. Quick algorithm for determining the distances from the points of the given subset of an integer lattice to the points of its complement. *Cybernetics and System Analysis*, pages 177–181, April-May 1992. Translation from the Russian by Julia Komissarchik.

[10] C.R. Maurer, 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*, 25(2):265–270, February 2003.

[11] L. Rabiner. A tutorial on hidden Markov odels and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–289, 1989.

[12] A. Rosenfeld and J.L. Pfaltz. Sequential operations in digital picture processing. *Journal of the Association for Computing Machinery*, 13(4):471–494, October 1966.

[13] D. Rutovitz. Data structures for operations on digital images. In Cheng et al., editor, *Pictorial Pattern Recognition*, pages 105–133. Thomson Book, WA, 1968.