Distance Transforms of Sampled Functions

Pedro F. Felzenszwalb
The University of Chicago
pff@cs.uchicago.edu

Daniel P. Huttenlocher
Cornell University
dph@cs.cornell.edu

Keywords: Euclidean distance transform, Minimum convolution, Dynamic Programming.

Abstract

This paper provides linear-time algorithms for solving a class of minimization problems involving a cost function with both local and spatial terms. These problems can be viewed as a generalization of classical distance transforms of binary images, where the binary image is replaced by an arbitrary sampled function. Alternatively they can be viewed in terms of the minimum convolution of two functions, which is an important operation in grayscale morphology. A useful consequence of our techniques is a simple, fast method for computing the Euclidean distance transform of a binary image. The methods are also applicable to Viterbi decoding, belief propagation and optimal control.

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 [5] and Hausdorff [12]

matching approaches make use of distance transforms in comparing binary images. Distance transforms are also used to compute the medial axis of digital shapes [3].

In this paper we consider a generalization of distance transforms to arbitrary functions on a grid rather than binary-valued ones (i.e., real valued 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. For example, one can think of a binary edge map as a restricted case, where the costs are 0 (at edge pixels) or infinite (at non-edge pixels). For such more general feature maps it is again useful to compute a type of distance transform, which in this case should reflect a combination of distances and feature costs.

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 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 $f(q)$.

Efficient algorithms for computing the distance transform of a binary image using the l_1 and l_∞ distances were developed by Rosenfeld and Pfaltz [18]. Similar methods described in [4] have been widely used to efficiently compute approximations to the Euclidean distance transform. These algorithms can be easily adapted to compute the distance transform of a function, as we shown in Section 3 for the case of the l_1 distance.

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. There are a number of algorithms for computing the Euclidean distance transform of a binary image in linear time (e.g., [13, 6, 15]), 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 gray level images based on minimum distances along paths, where the cost of a path is the sum of gray level values along the path [19]. We want to avoid confusion with these methods which compute something quite different from what we consider here.

1.1 Minimum Convolution

The distance transform of a sampled function is closely related to the *minimum convolution* operation. This operation and its continuous counterpart play an important role in grayscale morphology [14]. The minimum convolution of two discrete signals f and g is defined as,

$$(f \otimes h)(p) = \min_q (f(q) + h(p - q)).$$

Just like standard convolution this operation is commutative and associative,

$$\begin{aligned} f \otimes h &= h \otimes f, \\ (f \otimes h) \otimes g &= f \otimes (h \otimes g). \end{aligned}$$

When $d(p, q) = g(p-q)$ the distance transform of f is exactly the minimum convolution of f and g. Thus the algorithms described here for distance transforms of sampled functions can be seen as minimum convolution algorithms. In the case of the squared Euclidean distance we are computing the minimum convolution of a sampled function and a parabola. This is a problem that has been studied before (see [11]) but our algorithm is more efficient than previous techniques. In the case of the l_1 distance we compute the minimum convolution of a function and a diamond shaped cone using the classical algorithm from [18].

1.2 Energy Minimization Problems

In addition to extending the applicability of distance transforms from binary feature maps to "soft" ones that reflect multi-valued feature quality measures, distance transforms of functions arise in the solution of a number of optimization problems. For instance in the widely used Viterbi algorithm for hidden Markov models [17], in max-product belief propagation [16], in optimal control methods [2] and in resource allocation [1]. In these problems there is a discrete state space S, a cost $b(p)$ for each state $p \in S$, a transition cost $a(p, q)$ for changing from state p to state q, and a dynamic programming equation,

$$\delta'(q) = b(q) + \min_p(\delta(p) + a(p, q)). \tag{2}$$

For our purposes a detailed understanding of this equation is not as important as observing its form. The minimization in the second term of the right hand side is closely related to the distance transform of a function. In particular, if the set of states S can be embedded in a grid and if the transition costs $a(p, q)$ form a distance between corresponding locations in that grid, then this equation can be rewritten in terms of a distance transform,

$$\delta'(q) = b(q) + \mathcal{D}_\delta(q).$$

Thus efficient algorithms for computing distance transforms of functions apply to certain problems of the form in (2). We have recently used these methods to develop improved algorithms for recognition of articulated objects [7], for inference using large state-space hidden Markov models [9], and for the solution of low-level vision problems such as stereo, image restoration and optical flow using loopy belief propagation [8]. For instance, 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.

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.

Our algorithm for computing this distance transform 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 p. 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.

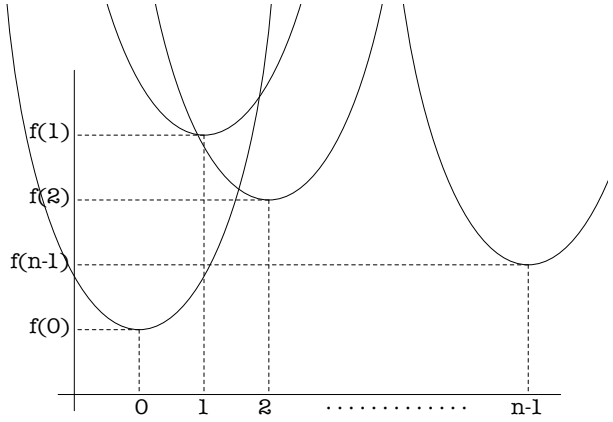


Figure 1: The distance transform as the lower envelope of n parabolas.

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

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

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

We compute the lower envelope by sequentially computing the lower envelope of the first q parabolas, where the parabolas are ordered according to their corresponding horizontal grid locations. The algorithm works by computing the combinatorial structure of this lower envelope. We keep track of the structure by using two arrays. The horizontal grid location of the i-th parabola in the lower envelope is stored in $v[i]$. The range in which the i-th parabola of the lower envelope is below the others is given by $z[i]$ and $z[i+1]$. 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, as illustrated in Figure 2. If the intersection is after $z[k]$, then the lower envelope is modified to indicate that the parabola from q is below all others 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, sowe decrease k to delete that parabola and repeat the procedure.

Algorithm $DT(f)$

1. $k \leftarrow 0$ (∗ Index of rightmost parabola in lower envelope ∗)
2. $v[0] \leftarrow 0$ (∗ Locations of parabolas in lower envelope ∗)
3. $z[0] \leftarrow -\infty$ (∗ Locations of boundaries between parabolas ∗)
4. $z[1] \leftarrow +\infty$
5. **for** $q = 1$ **to** $n - 1$ (∗ Compute lower envelope ∗)
6. $\quad s \leftarrow ((f(q) + q^2) - (f(v[k]) + v[k]^2))/(2q - 2v[k])$
7. $\quad$ **if** $s \leq z[k]$
8. $\quad\quad$ **then** $k \leftarrow k - 1$
9. $\quad\quad\quad$ **goto** 6
10. $\quad\quad$ **else** $k \leftarrow k + 1$
11. $\quad\quad\quad v[k] \leftarrow q$
12. $\quad\quad\quad z[k] \leftarrow s$
13. $\quad\quad\quad z[k + 1] \leftarrow +\infty$
14. $k \leftarrow 0$
15. **for** $q = 0$ **to** $n - 1$ (∗ Fill in values of distance transform ∗)
16. $\quad$ **while** $z[k + 1] < q$
17. $\quad\quad k \leftarrow k + 1$
18. $\quad \mathcal{D}_f(q) \leftarrow (q - v[k])^2 + f(v[k])$

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

Theorem 1. *Algorithm 1 correctly computes the distance transform of f under the squared Euclidean distance in $O(n)$ time.*

Proof. We start by showing that the algorithm correctly computes the lower envelope of the first q parabolas by induction. The base case is trivial. The lower envelope of the first

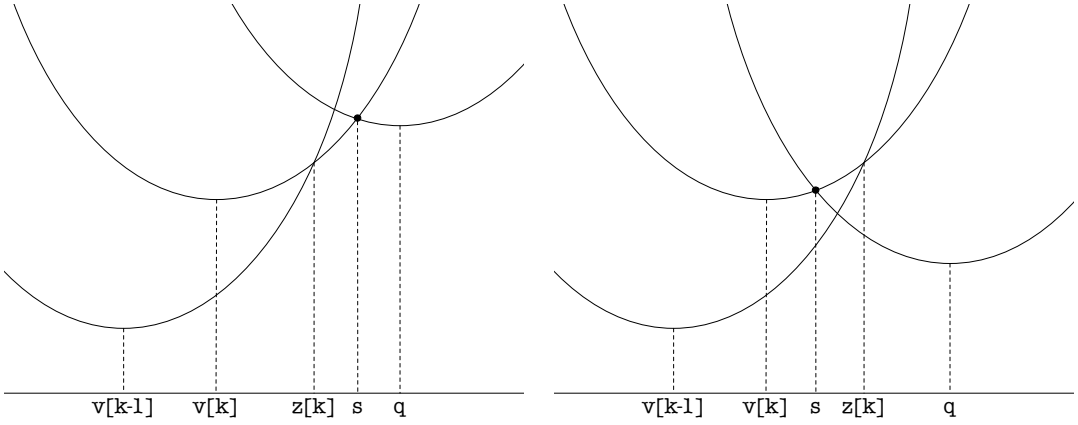
parabola is represented by a single interval from $-\infty$ to $+\infty$ dominated by the parabola from grid position 0.

Let s be the horizontal position of the intersection between the q-th parabola and the one from $v[k]$ as computed in line 6. The parabola from q is above the one from $v[k]$ to the left of s, and below it to the right of s. By the induction hypothesis the parabola from $v[k]$ is above at least one other parabola in the lower envelope to the left of $z[k]$ and below all of them to the right of $z[k]$.

Suppose $s > z[k]$ (as in Figure 2(a)). The parabola from q is below the one from $v[k]$ to the right of s, which in turn is below all others everywhere after $z[k]$. We see that the parabola from q dominates the lower envelope after s. The parabola from $v[k]$ continues to be the lowest between $z[k]$ and s. The algorithm updates the lower envelope to reflect these changes by creating a new interval from s to ∞ dominated by the q-th parabola.

Suppose $s \leq z[k]$ (as in Figure 2(b)). Now the parabola from q is below the parabola from $v[k]$ in the whole interval previously dominated by the parabola from $v[k]$. This means that the parabola from $v[k]$ is not part of the lower envelope of the first q parabolas. The algorithms modifies the lower envelope to remove the parabola from $v[k]$ and proceeds to add the parabola from q to the remanining structure. This process eventually terminates since $z[0] = -\infty$.

Once the lower envelope is computed it remains to fill in the distance transform values by sampling the height of the lower envelope at each grid location. This is done from left to right on lines 14 throgh 18. 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. $\square$



(a) (b)

Figure 2: The two possible cases considered by the algorithm when adding the parabola from q to the lower envelope constructed so far. In (a) $s > z[k]$ while in (b) $s \leq z[k]$.

2.2 Arbitrary Dimensions

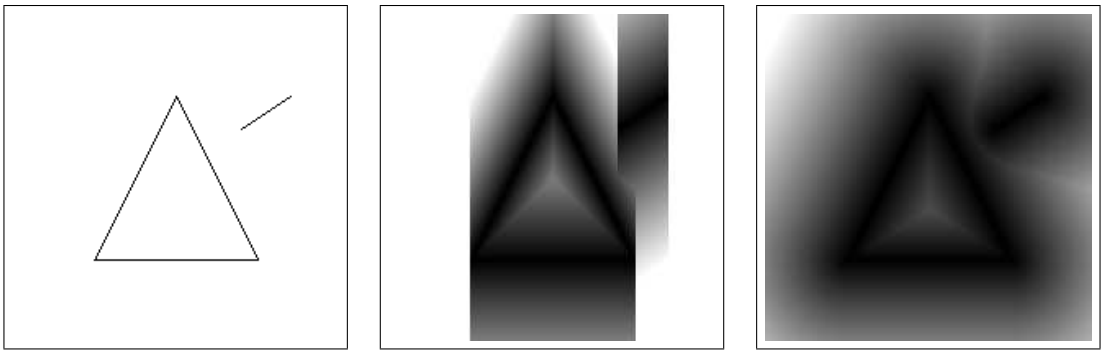
Let $\mathcal{G} = \{0, \ldots, n-1\} \times \{0, \ldots, m-1\}$ be a two dimensional grid, and $f : \mathcal{G} \rightarrow \mathbb{R}$ an arbitrary function on the grid. The two dimensional distance transform of f under the squared Euclidean distance is given by,

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

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

$$\begin{aligned} \mathcal{D}_f(x, y) &= \min_{x'}((x - x')^2 + \min_{y'}((y - y')^2 + f(x', y'))), & (3) \\ &= \min_{x'}((x - x')^2 + \mathcal{D}_{f|_{x'}}(y)), & (4) \end{aligned}$$

where $\mathcal{D}_{f|_{x'}}(y)$ is the one-dimensional distance transform of f restricted to the column indexed by x'. 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



(a) (b) (c)

Figure 3: 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.

for the two-dimensional case above. The multi-dimensional algorithm runs in $O(dk)$ time, where d is the dimension of the grid and k is the overall number of grid locations ($d = 2$ and $k = nm$ for the grid defined above).

Figure 3 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 on the grid. In equation (4) the function that must be transformed along the second dimension is not a binary function. Thus the notion of a distance transform for arbitrary sampled functions is important here. The separation of the multi-dimensional transform into multiple one-dimensional ones makes our method substantially simpler than previous techniques for computing Euclidean distance transforms.

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., [18]). 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.

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 $\mathcal{D}_f$ will be $(4, 2, 3, 4, 1)$, and after the second pass it will be $(3, 2, 3, 2, 1)$.

Theorem 2. *Algorithm 2 correctly computes the distance transform of f under the l_1 norm in $O(n)$ time.*

Proof. Let

$$a(p) = \begin{cases} |p| & \text{if } p \geq 0 \\ \infty & \text{otherwise} \end{cases} \quad \text{and} \quad b(p) = \begin{cases} |p| & \text{if } p \leq 0 \\ \infty & \text{otherwise} \end{cases}$$

Its not hard to check that $(a \otimes b)(p) = |p|$. We claim that the forward pass of the algorithm computes the minimum convolution of f with a while the backward pass computes the minimum convolution of the result with b. Since minimum convolution is an associative operation the algorithm computes $f \otimes (a \otimes b)$, which as discussed in Section 1.1 is the distance tranform of f under the l_1 norm.

Now we show that the forward pass of the algorithm does indeed compute the minimum convolution of f and a.

$$\begin{aligned}
(f \otimes a)(p) &= \min_q(f(q) + a(p-q)), && \text{[definition of } \otimes\text{]} \\
&= \min_{q \le p}(f(q) + a(p-q)), && \text{[}a(x) = \infty \text{ when } x < 0\text{]} \\
&= \min_{q \le p}(f(q) + p - q), && \text{[definition of } a\text{]} \\
&= \min\left(\min_{q \le p-1}(f(q) + p - q), f(p)\right), && \text{[}q \le p-1 \text{ or } q = p\text{]} \\
&= \min\left(\min_{q \le p-1}(f(q) + (p-1) - q) + 1, f(p)\right), && \text{[simple algebra]} \\
&= \min\left((f \otimes a)(p-1) + 1, f(p)\right). && \text{[recursive substitution]}
\end{aligned}$$

The last equation is exactly the computation performed by the algorithm. The convolution of the resulting function with b in the backward pass is analogous. Note that both the forward and backward passes take a constant number of operations per grid position, yielding a $O(n)$ algorithm overall. □

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

We can compute distance transforms of functions under other distances not discussed so far. An important case is the distance transform under the box distance defined by $d(p, q) = 0$ when $|p - q| < a$ and ∞ otherwise. This transform can be computed using a linear time algorithm for the min-filter described in [10].

Another way to obtain fast algorithms is to use the relationships described below. For example, 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 computing both a linear and a quadratic distance transform and combining the results.

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

Theorem 3. $d(p,q) = \min(d^1(p,q), d^2(p,q)) \Rightarrow \mathcal{D}_f(p) = \min(\mathcal{D}_f^1(p), \mathcal{D}_f^2(p))$.

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

Proof. This is another simple result,

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

□

Many state transition costs which appear in dynamic programming equations similar to (2) can be written as a combination of a small number of linear and quadratic and box distances using these relations. In such cases our algorithms can be used to compute the dynamic programming equation in time linear in the number of possible states.

References

[1] R. Bellman and W. Karush. Functional equations in the theorey of dynamic programming XII: An application of the maximum transform. *Journal of Mathematical Analysis and Applications*, 6:155–157, 1963.

[2] D. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 2001.

[3] H. Blum. Biological shape and visual science. *Theorerical Biology*, 38:205–287, 1973.

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

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

[6] 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.

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

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

[9] 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.

[10] Y. Gil and M. Werman. Computing 2d min, max and median filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15:504–507, 1993.

[11] C.T. Huang and O.R. Mitchell. A euclidean distance transform using grayscale morphology decomposition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(4):443–448, April 1994.

[12] 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.

[13] 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.

[14] P. Maragos. Differential morphology. In Mitra and Sicuranza, editors, *Nonlinear Image Processing*, pages 289–329. Academic Press, 2000.

[15] 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.

[16] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

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

[18] 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.

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