REDUNDANCY IN ALL PAIRS FAST MARCHING METHOD

Luca Bertelli, Baris Sumengen, B.S. Manjunath

Dept. of Electrical and Computer Engineering
University of California, Santa Barbara
Santa Barbara, CA 93106

ABSTRACT

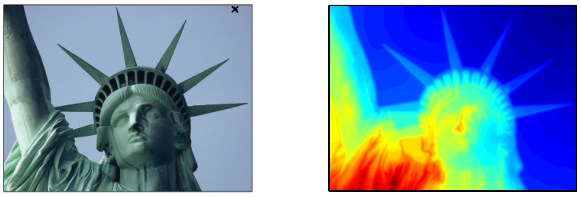
In this paper, we analyze the redundancy in calculating all pairs of geodesic distances on a rectangular grid. Fast marching method is an efficient way to estimate the geodesic distances from a point. But when calculated for all the points on the grid, this introduces certain redundancy. Our analysis shows that over 90% of the distances are actually recalculated. We propose a novel solution which exploits this redundancy to reduce the number of distances evaluated using the Fast Marching Method and enforces the symmetry of the distance matrix. Experimental results show the improved accuracy obtained with our implementation.

1. INTRODUCTION

The concept of pairwise similarity between pixels is important in several applications. Many kernel-based or graph-based image processing methods such as spectral clustering, support vector machines, etc., depend on the availability of good pairwise similarities between pixels that can be computed in an efficient manner. Building a similarity graph by calculating similarities between pixels can be ultimately reduced to a distance computation, traditionally meant as spatial or feature distance. In [1], we introduced and utilized all pairs geodesic distances for image segmentation. Values of an edge function represent the cost of crossing a pixel when connecting two pixels along the shortest path. Fig. 1 shows geodesic distances from point 10×600.

Fast Marching Method (FMM) [2] is an efficient algorithm which can be used to find geodesic distances between pixels. The distance between two pixels can be formulated as the minimum number of edges that one needs to cross while traveling from one pixel to another. To achieve this, first an edge strength function is generated from the image gradients and then this function is used as a cost function for crossing the pixels. By calculating distances from each pixel to all other pixels and expressing similarities as a monotonically decreasing function of the distances, we can obtain a similarity matrix, which can be fed to a graph partitioning or kernel based method.

This work was supported by the National Science Foundation award NSF ITR-0331697.



(a) (b)

Fig. 1. a) Original image of size 480×640 b) Geodesic distances calculated from the point (10,600) using an edge strength function as the cost function in Fast Marching algorithm. The black cross in a) is the point from which Fast Marching is run.

There is redundancy when naively calculating distances from each pixel to all other pixels. This redundancy can be exploited in order to reduce the number of distance calculations when Fast Marching Method is used. When distances from a pixel to all the other ones are calculated, we can actually obtain distances among some other points in the image. This is performed as follows: we store minimum paths from a starting point (zero distance) to the rest of image and then we evaluate the distances between the points along each path. In fact, when we calculate the minimum path between A and B, we also know the minimum path between any pair of points along the path (see Fig. 2). This information is used in the following Fast Marching runs, reducing the number of operations required.

The main observations in this paper are as follows: 1) There is high redundancy, which is introduced by naively repeating the Fast Marching Method for each point in the image. 2) Exploiting this redundancy will significantly reduce the number of distances evaluated using the Fast Marching Method. 3) We can ensure that the pairwise distance matrix is symmetric resulting in a symmetric positive definite similarity matrix required in graph partitioning or kernel based methods.

The rest of the paper is organized as follows. In Section 2 we give a brief description of the Fast Marching algorithm and our implementation, in Section 3 we describe how to ex-

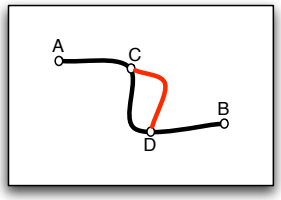


Fig. 2. The black path is the shortest path connecting A and B, which is also the shortest path between any other pair of points along the same path (in this example C and D). Any other path between C and D would result a higher distance.

ploit, using a geodesic tree, the redundancy generated by the naive All Pairs Fast Marching Method. Section 4 measures this redundancy and compares the results and the accuracy of the proposed method with the naive approach. We then conclude in Section 5.

2. FAST MARCHING METHOD FOR DISTANCE CALCULATION

Fast Marching Method (FMM) introduced by Sethian [2, 4, 6], is an efficient algorithm to estimate geodesic distances in continuous domain using a first order approximation. In general, to find shortest paths on a graph, Dijkstra's algorithm can be used [3]. A major drawback of Dijkstra's algorithm is that if we create a graph from an image using 4-neighbors of pixels, the shortest paths on the graph will calculate L_1 distance, *e.g.* the diagonal of a unit square is 2 (instead of $\sqrt{2}$) regardless of how much we refine the grid.

On the other hand, Fast Marching Method is an efficient algorithm to calculate geodesic distances by solving an *eikonal equation* in discrete domain, where 4-neighbors are used to estimate actual distances. The general expression of the eikonal equation in 2D is as follows:

$$|\nabla T(x,y)| = F(x,y). \tag{1}$$

The function $F(x,y)$ gives the local weights used in arc length calculation: $ds^2 = F^2(x,y)(dx^2+dy^2)$. If F is constant over the whole domain, the solution of the eikonal equation is exactly the Euclidean distance. If F is an edge strength function then the solution T gives us the geodesic distances based on image gradients. The boundary conditions are given as a set of points for which $T(x,y)$ is known.

In our implementation of FMM we utilize three static matrices T, S and P and one priority queue Q_{FM}. In the matrix T we store the solution of the eikonal equation (geodesic distances). The matrix S tells us the state of each point: 1 corresponds to *decided*, 0 to *far away* and -1 to *close*. The matrix P contains the positions of *close* points in the priority queue.

To initialize the algorithm we set $S(x,y) = 0$ and $T(x,y) = \infty \; \forall \; x,y$. Then for every point (x,y), whose value $T_0(x,y)$ is known, we set $T(x,y) = T_0(x,y)$, we update their state as *decided* ($S = 1$) and we add their 4-neighbors to the priority queue, updating their state to *close* ($S = -1$). For every point in the queue, we solve a quadratic equation to update their value and their position in the queue.

After this initialization phase, the main cycle of the algorithm begins, as described in Algorithm 1.

Algorithm 1 Main Loop of Fast Marching Algorithm.

while Q_{FM} is not empty **do**
  Extract q=Minimum(Q_{FM}).
  Add q to the *decided* Set ($S(x_q, y_q) = 1$).
  Set the state of the neighbors of q as *close* ($S = -1$).
  Add 4-neighbors of q to Q_{FM} or if a neighbor is already in the queue then update its distance accordingly.
end while

Depending on the type of the priority queue used, complexity of this process is $O(Nlog(N))$[2] or $O(N)$ [5].

3. DISTANCES ALONG MINIMUM PATHS

The Fast Marching algorithm described in the previous section can be used to find the geodesic distances of one point (x_0, y_0) of the image to all the other points, simply by setting the initial condition $T(x_0, y_0) = 0$ and solving (1) for T. By repeating this process for all the points naively, we can obtain distances between all pairs of points. In this section we propose a way to reduce the number of distance evaluations (number of distances obtained by extracting the minimum from the priority queue q of Section 2) by exploiting previously calculated distances. By doing this, we also guarantee the symmetry of the distance matrix. Symmetry is not available when the naive approach is used. This is because of the recalculation of the same distances but with different discretization errors.

After minimum paths from a starting point to all other points in the image are calculated, we can show that we have enough information to evaluate the distances among all the points within each minimum path. Consider for example the case depicted in Fig. 2. The point A is the starting point while B is the last point along the minimum path. It can be easily shown that

$$||C - D|| = ||A - D|| - ||A - C|| \tag{2}$$

where $||\cdot||$ represent the geodesic distance. Suppose we have another path connecting C and D that is shorter: $||C - D||_{path2} < ||C-D||_{path1}$. We can then write: $||A-C||_{path1} + ||C - D||_{path2} + ||D - B||_{path1} < ||A - B||_{path1}$, which results a contradiction and proves that $||C - D||_{path1}$ is the minimum path. This shows that (2) can be used to evaluate the distance between C and D.

Using minimum paths, we can build a distance tree where the root of the tree is the starting point (zero distance) and all

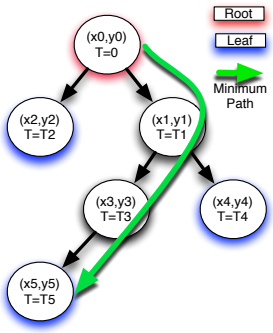


Fig. 3. Geodesic Tree: in every node we store the position of the point and the value of T, solution of the current FM iteration.

other points are nodes of the tree. The parent of a node is selected as the previous point along the minimum path. For each Fast Marching run, we build a tree containing all the points in the image. In this tree we can distinguish between three kinds of nodes (see Fig. 3): 1) The root, the starting point of the Fast Marching, which has 0 distance and has no parents. 2) The internal nodes, points, which have both parents and children. 3) The leaves, points with no children. Starting from the leaves and moving towards the root (or starting from the root and moving to the leaves), distances between points belonging to the same minimum path can be estimated. The geodesic distance between two points on a shortest path is the difference between their distances from the root node. In Fig. 3, the distance between (x_1,y_1) and (x_5,y_5) is simply $T_5 - T_1$.

4. EXPERIMENTAL RESULTS

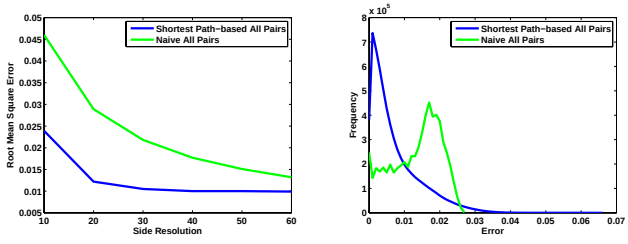
In this section we compare accuracy and number of total distance calculations using our algorithm and the naive All Pairs Fast Marching Method. In particular, we are interested in evaluating the accuracy of our implementation and in measuring the redundancy generated with the naive approach.

We calculated the all pairs euclidean distances using both the naive method and by exploting the redundancies on a unit square discretized at various grid resolutions. Parents in discrete domain are calculated as follows. At the end of each Fast Marching iteration, we take the gradient of $T(x,y)$–the solution of (1)–which represents the direction of propagation of the distance front. After discretizing the gradient direction at 45 degree increment, the parent of each pixel becomes one of the 8 neighbors at the opposite direction of the gradient. We compare these results with the ground truth represented by the exact euclidean distances. The results reported in Table 1 show that by avoiding recalculation of same distances, root mean squared error goes down significantly.

As expected, the accuracy of the two algorithms increases with increasing the grid resolution and the Shortest Path based implementation turns out to be more accurate than the naive All Pairs FM across all the different resolutions (see Fig. 4 (a)). Since full matrices are kept in memory for this calculations, we are not increase grid size more than 60×60. Ideally, sparse multi-resolution distances [1], where values are compressed as distances increase, should be used.

Table 1. Root Mean Square Error of the approximate solutions with respect to the exact euclidean distances at different grid resolutions.

Naive All Pairs	AP with Trees	Resolution
$4.60 \cdot 10^{-2}$	$\mathbf{2.39 \cdot 10^{-2}}$	10×10
$2.89 \cdot 10^{-2}$	$\mathbf{1.22 \cdot 10^{-2}}$	20×20
$2.18 \cdot 10^{-2}$	$\mathbf{1.05 \cdot 10^{-2}}$	30×30
$1.77 \cdot 10^{-2}$	$\mathbf{1.00 \cdot 10^{-2}}$	40×40
$1.51 \cdot 10^{-2}$	$\mathbf{1.00 \cdot 10^{-2}}$	50×50
$1.32 \cdot 10^{-2}$	$\mathbf{0.99 \cdot 10^{-2}}$	60×60

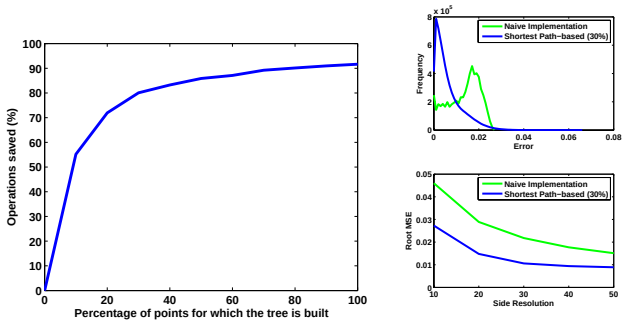


(a) (b)

Fig. 4. a) We compare the accuracy of our shortest path-based algorithm (blue) with the naive Fast Marching All Pairs Implementation (green) for different grid resolutions. b) The error curves at the 50×50 resolution. With the Shortest Path-based approach most of the errors are concentrated around zero and the number of points with error bigger than maximum error in the Naive approach is small.

Table 2 shows how many Fast Marching computations we can save at each resolution. In other words, we evaluate how many distances are actually calculated using Fast Marching and how many are reused from previous calculations. This is a measure of how much redundancy is present in the naive implementation, which recalculates the same distances again and again. The computational savings by exploiting this redundancy become more relevant as the grid becomes large (*e.g.* a big image).

The longer the shortest paths are, the more pairwise distances we can estimate ahead of time. For boundary points, shortest paths are longer than a point at the center of the image. Therefore pixels are visited by following a spiral-like order, starting from the borders of the image and going towards the center. Within each square ring, the order is randomized. This is done to distribute the inaccuracy uniformly over the image. In Fig. 6 distance maps obtained using the Shortest paths based algorithm and the ideal euclidean solution is shown. As we move close to the center point some spatial artifacts are visible more and more. As shown in Fig.



(a) (b) At 30%

Fig. 5. a) We show the computational savings if we build the tree only for a certain percentage of points. A value around 30% gives a percentage of operations saved close to 80, which means that most of the redundancy can be exploited building the tree for one pixel out of three. b) Error plots as in Fig. 4, but building the tree for 30% of the points.

Table 2.

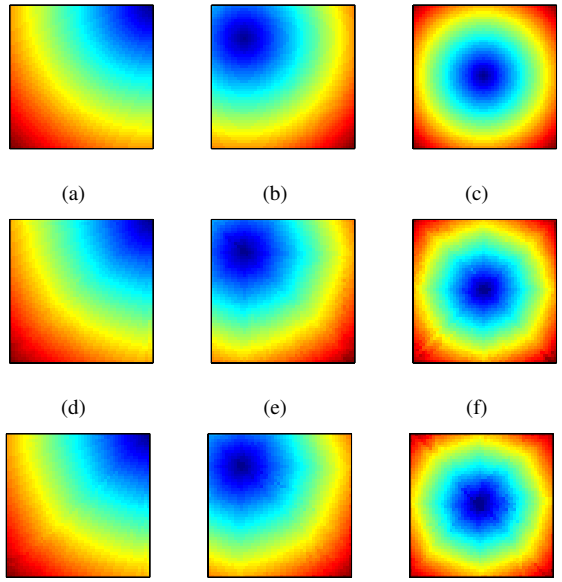
Redundancy Removal	
Grid Resolution	Operations Saved (%)
10×10	69.78
20×20	81.76
30×30	87.03
40×40	89.98
50×50	91.65
60×60	92.69

4, the mean of the root mean squared error plot is very low but at some outlier points we observe higher errors than naive method.

Building the shortest path trees for every point is also not optimal. Two neighboring pixels may have very similar shortest path trees resulting in reestimation of same distances. So, we don't need to build a tree at every point. Fig. 5 shows that for a 50×50 grid building the tree for only 30% of the points in the image reduces the number of distances calculated via Fast Marching by 80% . Therefore the cost of building and navigating the tree can be strongly reduced, still exploiting most of the redundancy present in the naive implementation. In addition, we also observe best RMSE rates when for only 30 to 40% of the pixels the geodesic trees are built. This optimal percentage goes down as the grid size increases.

5. CONCLUSIONS

In this paper we addressed the problem of all pairs distances calculation using the Fast Marching Method. We analyzed its redundancy, which is observed when repeating the Fast Marching Method for each pixel in the image. We showed that the number of distances evaluated using the Fast Marching Method can be reduced up to 90% and the symmetry of the distance matrix is enforced, which guarantees a symmetric positive definite similarity matrix needed in graph partitioning and kernel based methods. We showed that the accuracy



(a) (b) (c) (d) (e) (f) (g) (h) (i)

Fig. 6. a) b) and c) euclidean distances from the points (1,50), (12,12) and (25,25) respectively. d) e) and f) distances evaluated with the Shortest Path algorithm, building the tree for 30% of the points. g) h) and i) distances evaluated with the Shortest Path algorithm, building the tree for every point.

of our approximation is comparable to the naive case.

6. REFERENCES

[1] B. Sumengen, Luca Bertelli, and B. S. Manjunath, "Fast and adaptive pairwise similarities for graph cuts-based image segmentation," Tech. Rep., November 2005.

[2] J. A. Sethian, "A marching level set method for monotonically advancing fronts," *Proc. Nat. Acad. Sci.*, vol. 93, no. 4, 1996.

[3] E. W. Dijkstra, "A note on two problems in connection with graphs," *Numerische Mathematik*, vol. 1, pp. 269271, 1959.

[4] J A Sethian, *Level set methods and fast marching methods: evolving interfaces in computational geometry, fluid mechanics, computer vision, and materials science*, Cambridge University Press, 1999.

[5] Liron Yatziv, Alberto Bartesaghi, and Guillermo Sapiro, "O(n) implementation of the fast marching algorithm," *Journal of Computational Physics*, September 2005.

[6] L. Cohen and R. Kimmel, "Fast marching the global minimum of active contours," in *in Proc. IEEE International Conference on Image Processing (ICIP'96)*, 1996, pp. 473–476.