Dynamic Connectivity in Digital Images

David Eppstein*

Department of Information and Computer Science
University of California, Irvine, CA 92717
http://www.ics.uci.edu/~eppstein/

Tech. Report 96-13

April 18, 1996

Abstract

We show that any algorithm that maintains the connected components of a digital image must take $\Omega(\log n/\log\log n)$ time per change to the image. The problem can be solved in $O(\log n)$ time per change using dynamic planar graph techniques. We discuss applications to computer Go and other games.

Keywords: *dynamic planar connectivity, percolation, lower bounds, image processing, go, lines of action.*

*Work supported in part by NSF grant CCR-9258355 and by matching funds from Xerox Corp.

1 Introduction

A basic problem in image processing consists of finding the connected components of a bitmap image (where each component consists of pixels of a common color adjacent vertically, horizontally, or sometimes diagonally; the same problem applies also to finding regions of an image separated from each other by an edge detection operator). This is a special case of graph connectivity, and can easily be solved in linear time by depth-first search, but there has been some research on fast union-find methods that allow the image to be processed in a more systematic order [3]. A large body of research, *percolation theory*, concerns itself with similar questions of connectivity in random grid subgraphs.

We are interested here in a *dynamic* version of the problem, in which individual pixels may be changed, and the connected components must be maintained as this happens. We envision the following applications for this dynamic connectivity problem:

- **Image segmentation**. The problem here is to divide a (static) image into regions, presumably belonging to different objects; for a recent paper on the subject, with further references, see Asano et al. [1]. There are various methods of performing such division including partitions by pixel value as well as more sophisticated edge detection schemes. These methods can involve various parameters that can be tuned by a local search process to form a better partition. As one changes the parameters, individual pixels will shift from one side of the partition to the other, or more or fewer pixels will be recognized by the edge detector.

- **Connectivity in video data**. In some image sequences, such as those of cells undergoing division, we may want to track the connectivity of individual objects [12]. This can be viewed as a dynamic form of image segmentation, where now changes in the input come from updates to the image rather than modification of segmentation parameters.

- **Game playing software**. The game *Lines of Action* [4, 11] involves moving stones on a checkerboard in an attempt to make them connected. Several software implementations of this game are available; it has been reported that the most difficult part of writing such a program is evaluating connectivity [4]. Connectivity also plays a large part in the evaluation of positions in Go (in which good computer play has proven surprisingly hard to attain; see [10] for many papers on the subject), and in Hex.

As our problem is a special case of dynamic planar graph connectivity (if one takes care to keep the problem planar when diagonal connections are allowed) it can be solved by dynamic graph algorithm techniques [5, 6]. In particular, an algorithm of Eppstein et al. [6] solves dynamic planar connectivity problems (assuming a fixed planar embedding, which is the case here) in time $O(\log n)$ per update.

Fredman and Henzinger [7, 9] have shown a lower bound close to this, $\Omega(\log n/\log\log n)$. However, the problem we are solving is a special case, so this lower bound does not automatically apply. Further, the lower bound proof involves the insertion and deletion of long twisted edges that seem hard to fit into the digital image framework considered here.

Since this lower bound seems not to apply, perhaps it is possible to improve the upper bounds to sublogarithmic? An analogous one-dimensional connectivity problem is equivalent to integer searching, and can be solved in time $O(\log\log n)$. D. Dyer (personal communication) has proposed a simple constant time heuristic, based on computing the total curvature of a region's boundary; is it possible to extend this to a complete algorithm?

Our main result is to dash these hopes: the $\Omega(\log n/\log\log n)$ lower bound from the general dynamic planar graph problem applies as well to our digital image connectivity problem. (This is true whether we want to test which component a point belongs to, or whether we merely want to know whether the overall image is connected.) The method involves a reduction similar to the one by Fredman and Henzinger, from a problem of maintaining the parity of prefix sums; however we simplify their method greatly in order to apply it to our problem.

We also describe a simplified method of maintaining the connected components in logarithmic time, avoiding the complicated data structures of Eppstein et al. [6], and instead combining any balanced binary tree structure with Dyer's curvature idea. This method has the further advantage for computer Go applications that it can maintain additional topological information about the configuration (number of eyes).

2 The Lower Bound

Like Fredman and Henzinger, we base our lower bound on a reduction to the *parity prefix sum* problem: given a sequence of bits, answer queries on

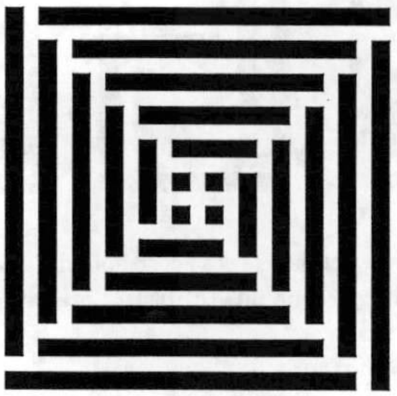

Figure 1. Sides of nested squares form layers of terminals in lower bound construction.

the total parity of initial subsequences of the bits, and allow updates that change one bit at a time. It is known that this problem has $\Omega(\log n/\log\log n)$ lower bounds per operation in the cell probe and integer RAM models of computation [2, 8], where n denotes the length of the bit sequence.

The basic idea of the lower bound is to translate an instance of the parity prefix sum problem into a planar graph as follows. We form a structure consisting of n *levels*, each consisting of four terminals. The terminals are connected at the bottom level into two pairs. The terminals at each successive level are connected to those at the previous level, in a way depending on the corresponding bit in the parity prefix problem, so that at each level the four terminals are grouped into two components in a way depending on the parity of the corresponding initial subsequence of bits. We can then test this parity at a given level ℓ by connecting the terminals at level $\ell+1$ to each other and to two of the terminals at level ℓ in such a way that the overall graph is connected if and only if the parity of the subsequence is even.

We now describe these components in more detail in the context of the image component problem we are considering. We first describe the set of terminals. Each terminal forms the side of a collection of concentric squares, each square four pixels wider than the previous, with four pixels removed from each square so that no two terminals adjoin each other. This construction is shown in Figure 1. Note that, as drawn in the figure, each terminal adjoins two others in the next larger square, one on the corresponding side of the larger square, and one on the side 90° clockwise.

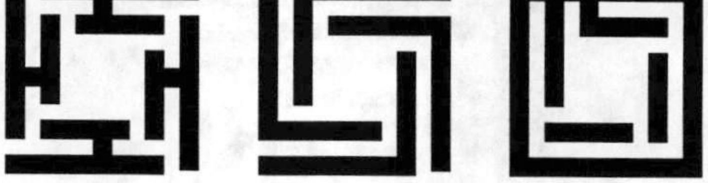

Figure 2. Connections between layers: (a) corresponding to zero bit in parity prefix problem; (b) corresponding to one bit; (c) test of prefix parity.

We define the *parity* of a subsequence of these terminals to be *even* if the upper and lower terminals are connected (indirectly, via a chain of pixels involving smaller squares) to their clockwise neighbors on the same square, and *odd* if these terminals are connected to their counterclockwise neighbors. We now describe how to add additional pixels to the image of Figure 1 to ensure that the parity of each subsequence of terminals corresponds to the parity prefix problem instance we started with. At the innermost level, this is easy: we can just add two pixels, connecting the appropriate pairs of terminals, depending on whether the first bit of the parity prefix problem is even or odd. At each subsequent level, if the parity prefix problem has an even bit, we connect each terminal with the corresponding side of the larger square (Figure 2(a)). if the parity prefix problem has an odd bit, we connect each terminal with the other adjacent side of the larger square, clockwise 90° (Figure 2(b)).

We omit the easy proof of the following lemma.

Lemma 1. *The construction above causes the parity of any initial subsequence of terminals to equal the parity of the corresponding subsequence of the prefix problem.*

It remains to show how to answer queries in the parity prefix problem. To query the parity of the first ℓ bits of the problem, we perform the following updates to the corresponding image connectivity problem. We remove all four pixels connecting terminals at level ℓ to those at level $\ell + 1$. We then connect the four level $\ell+1$ terminals to each other and to two level ℓ terminals at the top and right of their square (Figure 2(c)). As a consequence, all terminals at greater levels will be connected (via a path through the level $\ell + 1$ terminals) and these will be connected to the lower level terminals as well only if the two connections to level ℓ go to both of the two components formed by the lower level terminals.

Lemma 2. *The construction above forms a connected graph if and only if the parity of the subsequence of terminals up to level ℓ is even.*

Theorem 1. *It requires $\Omega(\log N/\log\log N)$ time per operation to add and remove pixels from a subset of an $N \times N$ grid and test whether the resulting image is connected.*

Proof: We have shown that we can use these operations to simulate a solution to the parity prefix problem with $\Omega(N)$ bits, using $O(1)$ changes to the grid per operation in the prefix problem. The result follows from known bounds on the parity prefix problem. □

Finally, we note that the same idea simplifies the lower bound method of Fredman and Henzinger for general planar graphs. Fredman and Henzinger give two versions of their proof, one using an integer searching data structure (along with the connectivity algorithm itself) as part of the simulation of the parity prefix problem, and the other based on a more complicated restriction of the prefix problem; we need neither complication. For this planar graph result, our overall framework again applies, but the terminals can be assumed to be single vertices and the connections between vertices can be single edges.

3 The Upper Bound

We now sketch a method for maintaining the number of connected components in logarithmic time per update, more simply than the algorithm of Eppstein et al. [6] which involves expanding the input planar graph to one of degree at most three, maintaining a partition of the expanded graph's edges into two trees, storing each tree as a collection of disjoint paths, and representing each path with a balanced binary tree. Instead we skip directly to the balanced binary tree stage. The method we describe has some advantages in maintaining more topological information about the components; however unlike that of Eppstein et al. it is unable to determine the identity of the component containing any particular pixel. For many applications this inability will not be a problem, as we are only interested in the number of components or in their boundaries.

Suppose we wish to find the connected components of the black pixels, in a black and white image. We begin by constructing the polygons forming the boundary between black and white regions of the image. Each pixel is involved in at most four such boundary polygons. Each connected component is bounded on the outside by one polygon, and may have other polygons bounding holes internal to the component. Thus if we can distinguish the interior boundaries from the exterior ones, we can count components simply by counting exterior boundaries. (Note also that in the application to Go evaluation, interior boundaries correspond to "eyes" and are very important for understanding the long-term stability of a component.)

To distinguish these two types of boundary polygons, we turn to an idea of D. Dyer. Recall that the total curvature of a polygon (the sum of the angles between vectors parallel to successive edges), as one traverses the polygon counterclockwise, is 360°. Dyer (personal communication) proposed simply to maintain the total curvature of all boundary regions, and test whether this number equals 360°. However this technique fails in the presence of interior boundaries. If we orient each boundary polygon so that, at positions where black pixels are below white pixels, the boundary goes leftwards, then this orientation will be clockwise for interior boundaries and counterclockwise for exterior boundaries. Thus the total curvature will be 360° or -360 respectively if the boundary is exterior or interior.

Thus we need only keep track of the total curvature of each boundary polygon to distinguish which polygons are exterior and find the total number of components of the input. One way of doing this is to represent boundary polygons using any of various balanced binary tree structures that allow lists to be split and joined in logarithmic time per update. If we store at each node of the tree the total curvature in the subtree under the node, we can read off the curvature of a boundary polygon by looking at the number stored at the root of the corresponding binary tree. A further simplification is possible: since we are only interested in distinguishing 360 from -360, and since each angle in the polygon is either 0 or 90°, we can represent the curvature θ as the values of $\theta/90°$ (modulo 3); then 360° corresponds to a value of 1 (mod 3) and $-360°$ corresponds to a value of -1 (mod 3). In this way we need store only two additional bits per binary tree node.

Theorem 2. *The method described above maintains the orientation of each boundary polygon of the input, and the number of components of the input, in time $O(\log n)$ per update.*

4 Conclusions

We have described upper and lower bounds for dynamic connectivity in bitmap images. The same problem remains open here as for planar graphs: to close the $O(\log\log n)$ gap between the upper and lower bounds. Perhaps this will prove easier for bitmap images than for the general planar graph problem.

References

[1] T. Asano, D. Z. Chen, N. Katoh, and T. Tokuyama. Polynomial-time solutions to image segmentation. *7th ACM-SIAM Symp. Discrete Algorithms* (1996) 104–113.

[2] A. M. Ben-Amram. *On the Power of Random Access Machines.* Ph.D. thesis, Tel Aviv U., School of Mathematical Sciences, 1994.

[3] M. B. Dillencourt, H. Samet, and M. Tamminen. A general approach to connected-component labeling for arbitrary image representations. *J. Assoc. Comput. Mach.* 39 (1992) 253–280.

[4] D. Dyer. Lines of Action. Online document, available at http://www.andromeda.com/people/ddyer/loa/loa.html

[5] D. Eppstein, Z. Galil, G. F. Italiano, and T. Spencer. Separator-based sparsification for dynamic planar graph algorithms. *25th ACM Symp. Theory of Computing* (1993) 208–217.

[6] D. Eppstein, G. F. Italiano, R. Tamassia, R. E. Tarjan, J. Westbrook, and M. Yung. Maintenance of a minimum spanning forest in a dynamic plane graph. *J. Algorithms* 13 (1992) 33–54.

[7] M. L. Fredman and M. R. Henzinger. Lower bounds for fully dynamic connectivity problems in graphs. *Algorithmica*, to appear.

[8] M. L. Fredman and M. E. Saks. The cell probe complexity of dynamic data structures. *Proc. 19th ACM Symp. Theory of Computing* (1989) 345–354.

[9] M. R. Henzinger. Improved data structures for fully dynamic biconnectivity. *Proc. 26th ACM Symp. Theory of Computing* (1994) 686–695.

[10] D. N. L. Levy, ed. *Computer Games II.* Springer, 1988.

[11] S. Sackson. *A Gamut of Games.* Dover, 1992.

[12] R. Samadani. Changes in connectivity in active contour models. *IEEE Worksh. Visual Motion* (1989) 337–343.