1991

Determining Maximum k-Width-Connectivity on Meshes

Susanne E. Hambrusch
Purdue University, seh@cs.purdue.edu

Frank Dehne

Report Number:
91-025

Hambrusch, Susanne E. and Dehne, Frank, "Determining Maximum k-Width-Connectivity on Meshes" (1991). *Department of Computer Science Technical Reports*. Paper 874.
<https://docs.lib.purdue.edu/cstech/874>

**DETERMINING MAXIMUM k-WIDTH
CONNECTIVITY ON MESHES**

Susanne E. Hambrusch
Frank Dehne

CSD-TR-91-025
March 1991

Determining Maximum k -Width-Connectivity on Meshes

Susanne E. Hambrusch *
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907, USA

Frank Dehne †
School of Computer Science
Carleton University
Ottawa, Canada K1S 5B6

March 29, 1991

Abstract

Let I be a $n \times n$ binary image stored in a $n \times n$ mesh of processors with one pixel per processor, and call I *k-width-connected* if, informally, between any pair of 1-pixels there exists a path of width k (composed of 1-pixels only). We consider the problem of determining the largest integer k such that I is k -width-connected, and present an optimal $O(n)$ time algorithm for the mesh architecture.

Keywords: Parallel algorithms, meshes, binary images, connected components, k -width-connectivity.

*Research supported in part by ONR under contracts N00014-84-K-0502 and N00014-86-K-0689, and by NSF under Grant MIP-87-15652.

†Research partially supported by the Natural Sciences and Engineering Research Council of Canada.

1 Introduction

Detecting forms of connectivity in binary images is a fundamental problem in image processing [12, 14]. Because of the relevance of parallel computation to image processing and computer vision, the parallel complexity of connectivity problems has been studied extensively [2, 3, 5, 6, 7, 8, 11]. In this paper we consider k -width-connectivity which is stronger, more fault-tolerant form of connectivity in images. Informally, an image is k -width-connected if between any pair of 1-pixels (i.e., entries of value '1') there exists a path of width k . In [4], Dehne and Hambrusch present a parallel mesh algorithm that, given an integer k and a binary image I stored in an $n \times n$ mesh with one pixel per processor, determines the k -width-components of I in $O(n)$ time. The problem of determining the k -width-components has a number of applications. One is in image segmentation where an image is partitioned into coherent regions that satisfy certain requirements [12]. Another application is the detection of connectivity in VLSI masks where electrical connectivity between components is maintained by a channel whose width is never less than a value λ [9]. The image might also represent the corridors of a maze, in which case the fact that a and b are in the same k -width-component implies that a robot occupying a $k \times k$ area is able move from a to b .

A natural problem that arises is that of, given a binary image I , determine the largest integer k so that image I is k -width-connected (i.e., I contains one k -width-component). For the remainder of the paper let k^* denote this largest k . Determining k^* has obvious relevance to the applications stated above. If image I represents a VLSI mask, then the value k^*

represents the robustness of the design with respect to deficiencies in the VLSI production process. If the image represents the corridors of a maze, then k^* represents the largest side length of a square-shaped robot that can move freely between any two points in the maze.

The value of k^* can easily be determined in $O(n \log k^*)$ time by using the algorithm presented in [4] for performing a binary search for k^* . In this paper we present an algorithm to determine k^* in $O(n)$ time. Our algorithm is based on a very different approach from the one used in [4]. We generate k^* in two stages. The first stage generates a preliminary estimate for k^* by having every 1-pixel perform a certain amount of “local” computation. This preliminary value represents an upper bound on the value of k^* . The final value of k^* is obtained by generating a weighted graph that models bottlenecks in image I . For this graph we determine the largest edge weight w such that removing all edges of weight at least w breaks all cycles in the graph. We then show that w equals k^* .

The paper is organized as follows. In Section 2 we state some of the necessary definitions. Section 3 contains the description of the approach used by our algorithm and Section 4 describes its implementation on the mesh. Section 5 concludes.

2 Definitions and Preliminaries

Throughout, image I is of size $n \times n$ and is stored in a mesh containing n^2 processors, with every processor containing $O(1)$ registers. We assume that the image is stored in the obvious way; i.e., the processor in row i and column j stores the pixel in the same row and column. In cases where it is

obvious, we refer to the processor storing pixel x as processor x .

We start by giving the formal definition of a k -width-connected image. Let x and y be two 1-pixels in image I . We assume, w.l.o.g., that no 1-pixels are located adjacent to the border of I . Let $P(x, y)$ be a path from x to y ; i.e., there exist 1-pixels $x = v_0, v_1, \dots, v_{m-1}, v_m = y$ such that v_i and v_{i+1} are horizontally or vertically adjacent. Unless stated otherwise, two pixels are called *adjacent* if they are horizontally or vertically adjacent. Consider two paths $P(x, y) = v_0, v_1, \dots, v_m$ and $P(x', y') = w_0, w_1, \dots, w_l$. The paths $P(x, y)$ and $P(x', y')$ are *shadow paths* if and only if (i) no pixel is contained in both paths, (ii) x and x' as well as y and y' are adjacent, and (iii) $P(x, y)$ and $P(x', y')$ are the union of a sequence $B_1, \dots, B_l$ of 2×2 blocks of 1-pixels where B_i and B_{i+1} share exactly two adjacent pixels, $1 \leq i \leq l-1$, B_i and B_{i+2} share at most one pixel, $1 \leq i \leq l-2$, and no other two blocks share a pixel.

Image I is *k -width-connected* if and only if for any two 1-pixels a and b there exist k mutually disjoint paths $P_i(x_i, y_i)$, $1 \leq i \leq k$, so that

- path $P_i(x_i, y_i)$ has length at least k ,
- paths $P_i(x_i, y_i)$ and $P_{i+1}(x_{i+1}, y_{i+1})$ are shadow paths,
- $x_1, x_2, \dots, x_k$ (resp. $y_1, y_2, \dots, y_k$) are on a common row or column, and
- $a = x_p$ and $b = y_r$ for some p and r .

Observe that a 1-pixel can belong up to k k -width-components. Figure 2 shows an image that is 3-width-connected, but not 4-width-connected. The

image consists of two 4-width-components: one is formed by the 1-pixels “enclosed” by holes H_1 and H_2 and one is formed by the remaining 1-pixels. Seven 1-pixels of this image belong to both 4-width-components.

To reduce the number of special cases that need to be considered we assume I is at least 3-width-connected. Using the algorithm presented in [4], we can determine in $O(n)$ time whether I is 3-width-connected.

We characterize 1-pixels that are adjacent (horizontally, vertically, or diagonally) to at least one 0-pixel into contour and corner pixels. We call a 1-pixel adjacent to exactly one 0-pixel a *1-contour* pixel and a 1-pixel adjacent to exactly two 0-pixels a *2-contour* pixel. Observe that, since I is at least 3-width-connected, a 1-pixel can be adjacent to at most two 0-pixels and these two 0-pixels cannot be in the same row or column. We say 1-pixel x is a *1-corner* pixel if x is diagonally adjacent to exactly one 0-pixel p with the following corner property: x and p are adjacent to two common 1-pixels (both of which are contour pixels). A 1-pixel in image I can be a 1-contour pixel as well as a 1-corner pixel. However, since I is 3-width-connected, it cannot both be a 1-contour pixel and a 2-corner pixel. For any 1-pixel x there can be two distinct 0-pixels making x a corner pixel. If 1-pixel x is diagonally adjacent to two such 0-pixels, we say x is a *2-corner* pixel. Note that the two 0-pixels making x a 2-corner pixel cannot be in the same row or column. See Figure 2 for an illustration of the different types of 1-pixels.

The *boundary graph* $G_b = (V_b, E_b)$ is an undirected, planar graph with weights on both the vertices and the edges. For any vertex x , let $w(x)$ be the weight of this vertex and for every edge (x, y) , let $w(x, y)$ be the weight of this edge. Every contour pixel x corresponds to one vertex in G_b , namely

contour vertex x , with $w(x) = \infty$. For any two adjacent contour pixels x and y , G_b contains the edge (x, y) . Let a *1-block* of size k be a subimage of I of size $k \times k$ which contains only 1-pixels. The weight of the edge (x, y) equals the size of the largest 1-block that contains both x and y . Obviously, this 1-block has x and y on its border. A contour vertex is incident to exactly two edges. Some of these edges are set by the corner vertices and are described below.

Every 1-corner pixel x induces one vertex x in G_b . Let y and z be the two 1-pixels adjacent to pixel x and which are also adjacent to the 0-pixel making x a corner pixel. The weight of vertex x , $w(x)$, equals the size of the largest 1-block that contains pixel x in one of its four corners, but does *not* contain y nor z . Graph G_b contains the edges (y, x) and (x, z) . The weight of the edge (y, x) (resp. (x, z)) equals the size of the largest 1-block containing both y and x (resp. x and z). Intuitively, $\min\{w(x), w(y, x), w(x, z)\}$ is the maximum width of a path from 1-pixel y to 1-pixel z via 1-pixel x . Clearly, the image may contain another, wider, path from y to z . When a pixel x is both a 1-contour and a 1-corner pixel it induces two distinct vertices in G_b . When pixel x is a 2-corner pixel, x also induces two vertices in G_b , one for each corner. The weights and the adjacent edges are set in a corresponding way. Figure 3 shows the boundary graph induced by the image of Figure 2. In Figure 3 corner vertices are represented by solid circles and contour vertices by empty circles. The weights are shown only for edges and vertices in the connected components induced by holes H_4 and H_5 (vertex weights of ∞ are not shown). The three pairs of vertices corresponding to the same 1-pixel are enclosed by dashed lines.

Let B_1 be the set of 1-pixels corresponding to the exterior boundary of the component in image I and let $B_2, B_3, \dots, B_m$ be the $m - 1$ interior boundaries, $m \geq 1$. The boundary graph G_b consists of m connected components, each having the form of a cycle. The two vertices induced by a pixel that is 1-contour and a 1-corner pixel or a 2-contour pixel belong to different connected components (since I is 3-width-connected, such 1-pixels are part of two different boundaries).

3 Overview of the Algorithm

As stated in the introduction, the value of k^* is determined in two stages. In the first stage we compute an upper bound on k^* by using the weights associated with the edges in the boundary graph. Let k be the value generated by the first stage, $k \geq k^*$. In the second stage we use the value of k to set up a hole graph that models the bottlenecks of size at most $k - 1$ induced by the corner pixels. We determine k^* by applying a cycle-breaking procedure to the hole graph. Our algorithm makes use of the following two properties.

Property 1 *Let x and y be two adjacent contour vertices in G_b . Then, $k^* \leq w(x, y)$.*

Proof: The largest 1-block containing both pixel x and pixel y has size $w(x, y)$. Assume image I is k' -width-connected with $k' > k^*$. Then, there must exist two disjoint 1-blocks of size k' such that one 1-block has x in one of its corners and the other 1-block has y in one of its corners. However, this implies that there exists a 1-block of size k' containing both x and y . This is not possible and thus the property follows. $\square$

For any vertex in G_b representing a contour or corner pixel x , we denote the two vertices in G_b adjacent to x (as well as the respective pixels) by $a1(x)$ and $a2(x)$.

Property 2 *Let x be a contour vertex adjacent to a corner vertex in G_b . Then, $k^* \leq \max\{w(x, a1(x)), w(x, a2(x))\}$.*

Proof: Assume x is a contour vertex adjacent to a corner vertex. If image I is k^* -width-connected, then 1-pixel x is contained in at least one 1-block of size k^* that contains x in one of its four corners. Such a 1-block may or may not contain the corner pixel adjacent to x . $\square$

STAGE 1

The first stage generates a value $k \geq k^*$ for which Properties 1 and 2 are satisfied. For the boundary graph of Figure 3, Stage 1 determines $k = 3$ (which coincides with the final answer). Using k , the contour pixels of image I can be partitioned into sets so that between any two contour pixels in the same set there exists a path of width k . Assume we have generated a minimum partition. If this partition consists of only one set, we have $k = k^*$. Otherwise, corner pixels induce bottlenecks that force us to set $k^* < k$. The following property states that for any corner pixel x , the value of $w(x)$ determines the width of the path going from pixel $a1(x)$ to $a2(x)$ via x .

Property 3 *Let x be a corner vertex and let k be the entry determined by Stage 1. If $w(x) > w(x, a1(x))$ or $w(x) > w(x, a2(x))$, then $k \leq w(x)$.*

Proof: W.l.o.g. assume $w(x) > w(x, a1(x))$. Let $y = a1(x)$ and $z = a1(y)$, $z \neq x$. Vertex y is a contour vertex in the connected component of G_b

containing vertex x . By Property 2 we have $k \leq \max\{w(y, x), w(y, z)\}$. If $w(y, x) \geq w(y, z)$, then $k \leq w(y, x) < w(x)$ and the property follows. Assume we have $w(y, x) < w(y, z)$. W.l.o.g let the pixels corresponding to vertices x , y , and z be in a common row and let x be to the left of y . Since $w(x) > w(x, y)$, there exists a 0-pixel p in the column containing pixel y so that p is adjacent to the border of the 1-block of size $w(x)$. (If such a 0-pixel p would not exist, we would have $w(x, y) \geq w(x)$.) See Figure 4 for an illustration. Pixel p effects the maximum size of a 1-block containing y and z and it makes it impossible to have $w(y, z) > w(y, x)$. Thus, the property follows. $\square$

STAGE 2

Let k be the quantity determined by Stage 1. From Property 3 it follows that, if $w(x) < k$, then the widest path from $a1(x)$ to $a2(x)$ going via corner pixel x has a width of $w(x)$. We say that x *induces a bottleneck* of size $w(x)$. If the only way to go from $a1(x)$ to $a2(x)$ is via corner pixel x , then this bottleneck cannot be avoided and we have $k^* \leq w(x) < k$. In order to determine which bottlenecks can and which cannot be avoided, we perform a cycle-breaking procedure on a hole graph induced by image I and k . We next define this hole graph.

Every boundary of the connected component in image I induces a *hole*, with the outer boundary inducing the *outer hole* and every other boundary inducing an *inner hole*. Assume we have labeled the holes so that $H(y)$ is the label of the hole containing 0-pixel y . Image I contains m boundaries and thus m holes. The *hole graph* $G_h = (V_h, E_h)$ is an undirected, planar, non-simple (i.e., it can contain multiple edges and self-loops), m -vertex graph

with costs on the edges. Vertex v_i of the hole graph corresponds to the i -th hole. We next describe how the edges of G_h are formed. Let x be a corner pixel with $w(x) < k$. Let s be the 0-pixel diagonally adjacent to x making x a corner pixel. Consider now the 1-block of size $w(x)$ that has pixel x in one of its four corners. We also refer to this 1-block as the 1-block *associated* with pixel x . There exists at least one other 0-pixel t that is adjacent to the border of the 1-block associated with x .

Property 2 allows us to restrict the position of this 0-pixel. Pixel t cannot be in the same row or column as 0-pixel s . If it would be, there would exist a contour pixel for which, due to Property 2, we would have $k \leq w(x)$. In addition, we cannot have two 0-pixels t_1 and t_2 such that t_1 and t_2 are adjacent to different borders of the 1-block associated with x . Again, if this would happen, there would exist a contour pixel causing $k \leq w(x)$.

Consider now the sequence λ formed by the $w(x)$ pixels containing pixel t and being adjacent to the side of the 1-block associated with x . List the elements in λ so that the first element corresponds to a pixel in either the same row or column containing pixel x . For the pixel labeled x in Figure 5(a), we have $\lambda = 1000$. No such sequence can contain a 1-pixel that is both to the left and the right of a 0-pixel. For example, for $w(x) = 4$, $\lambda = 0101$ is impossible. Obviously, λ contains at least one 0-pixel, and it must contain at least one 1-pixel (otherwise, $k \leq w(x)$). Should λ contain more than one 0-pixel, we choose t to correspond to the first 0-pixel in sequence λ . 0-pixel t is always diagonally adjacent to a corner pixel y so that y belongs either to the 1-block associated with x or y is in the same row or column as 0-pixel s . It is possible that there exist two choices for pixel y . In this case select

y so that it is closer to the row or column containing pixel x . Observe that we have $w(x) = w(y)$ (if we would have $w(y) < w(x)$, there would exist a contour pixel forcing $k \leq w(x)$). Furthermore, when applying the rules described above to pixel y , y chooses pixels s and z .

Having chosen x, s, t , and y according to these rules, we add to G_h the edge $(H(s), H(t))$ with cost $w(x) = w(y)$. We also say that corner pixels x and y induce the edge $(H(s), H(t))$ with cost $w(x)$. Figure 5(b) illustrates how edges are created for the portion of an image shown in Figure 5(a). In summary, the hole graph is not necessarily connected, it can contain multiple edges (even with identical costs) and self-loops, and the cost of every edge is less than k .

Lemma 1 *Let v_i be a vertex in G_h incident to a self-loop of cost c . Then, $k^* \leq c$.*

Proof: The self-loop on vertex v_i implies that there exist two corner pixels x and y such that each one is diagonally adjacent to a 0-pixel belonging to the hole associated with vertex v_i . Pixels x and y induce a bottleneck that cannot be avoided. The only path connecting contour pixels $a1(x)$ and $a2(x)$ (resp. $a1(y)$ and $a2(y)$) goes via corner pixel x (resp. y). We thus need $k^* \leq w(x)$ (resp. $k^* \leq w(y)$). Since $w(x) = w(y) = c$, the lemma follows. $\square$

Let k be the new estimate of k^* that takes the existence of self-loops in G_h into account. Let G_h be the hole graph in which edges of cost $\geq k$ have been deleted. Obviously, this new G_h contains no self-loops.

Lemma 2 *Let v_i and v_j be two vertices in G_h that have two edges, one of cost c_1 and one of cost c_2 , between them. Then, $k^* \leq \max\{c_1, c_2\}$.*

Proof: Let x_1 and y_1 be the two corner pixel that induce the edge of cost c_1 . From the way the edges of G_h are formed we know that $w(x_1) = w(y_1) = c_1$ and corner pixel y_1 (resp. x_1) is on the boundary of the 1-block of size c_1 associated with x_1 (resp. y_1). Assume that x_1 is diagonally adjacent to a 0-pixel in hole H_i and y_1 is diagonally adjacent to a 0-pixel in hole H_j . Since there exist two edges between v_i and v_j , there must exist a third corner pixel x_2 . Corner pixel x_2 can not be on the boundary of the 1-block of size c_1 associated with x_1 nor on the boundary of the 1-block of size c_1 associated with y_1 . Hence, there exist two other corner pixels x_2 and y_2 inducing an edge of cost c_2 . These four corner pixels induce two bottlenecks, one of size c_1 and one of size c_2 , such that there exist 1-pixels “surrounded” by holes H_i and H_j that can only be reached via one of the two bottlenecks. Hence we have $k^* \leq \max\{c_1, c_2\}$ and the lemma follows. $\square$

Self-loops and multiple edges represent cycles in G_h whose effect on k^* can be determined locally. Eliminating all self-loops and multiple edges turns G_h into a simple graph. Let k now be the estimate of k^* after self-loops and multiple edges have been handled. The next lemma characterizes how the remaining edges of G_h determine k^* .

Lemma 3 *If the hole graph G_h contains no cycles, then image I is k -width-connected and $k^* = k$. If the hole graph G_h contains cycles, let k' be the largest integer such that when all edges of cost $\geq k'$ are removed from G_h , the resulting graph contains no cycle. Then, image I is k' -width-connected*

and $k^* = k'$.

Proof: We first show that when G_h contains no cycles, there exists a path of width k between any two 1-pixels. The non-trivial part of the argument is to show that, given a corner pixel x with $w(x) < k$, there exists a path of width k between $a1(x)$ and $a2(x)$. We use graph G_h to show the existence of these paths.

Observe first that, given any corner pixel x with $w(x) < k$, there exists a 1-block of size k containing both x and $a1(x)$ (resp. $a2(x)$). This can be seen as follows. Property 2 implies that $a1(x)$ (resp. $a2(x)$) is contained in a 1-block of size k . If this 1-block does not contain x , then there exists a contour pixel for which Property 2 is violated.

Assume every tree of G_h is rooted with respect to some arbitrarily chosen vertex. Let x be a corner pixel diagonally adjacent to a 0-pixel in hole H_j . We prove the following invariant for every directed edge $\langle v_j, v_i \rangle$:

1. There exists a path of width k between $a1(x)$ and $a2(x)$.
2. Let y be the corner pixel diagonally adjacent to a 0-pixel in hole H_i that induced with x the edge (v_j, v_i) in G_h . Then, there exists a path of width k between $a1(y)$ and $a2(y)$.

The invariant is proven by an inductive argument. Assume vertex v_j is a leaf node. Then, exactly one 0-pixel of H_j is diagonally adjacent to a corner pixel x with $w(x) < k$. Thus, there exists a path of width k between 1-pixels $a1(x)$ and $a2(x)$ by going “around” hole H_j avoiding the bottleneck induced by x . As stated above, there exists a 1-block of size k containing both y and $a1(y)$ (resp. $a2(y)$). These 1-blocks can be used to form a path

of width k from $a1(y)$ to $a2(y)$: use one 1-block to go from $a1(y)$ to one of $a1(x)$ or $a2(x)$, use the already existing path of width k to go from $a1(x)$ to $a2(x)$, and use the second 1-block to return to $a2(y)$. Consider now an edge $\langle v_j, v_i \rangle$ where v_j does not correspond to a leaf node. Let x be a corner pixel diagonally adjacent to a 0-pixel in hole H_j . If the 1-block associated with x is not adjacent to a 0-pixel in hole H_i , the existence of a path of width k between $a1(x)$ and $a2(x)$ has already been proven. Otherwise, an argument identical to the one given for proving the second part of the invariant for the case when v_j is a leaf shows the existence of a path of width k between $a1(x)$ and $a2(x)$. The proof of the second part of the invariant for the case when v_j is not a leaf node is as for the leaf node case.

Once the invariant holds for all edges of the rooted trees, it follows that for any corner pixel x there exists a path of width k from $a1(x)$ to $a2(x)$. Having shown that every bottleneck can be avoided, it is easy to see that there exists a path of width k between two contour or corner pixels. The existence of a path of width k between any two 1-pixels now follows. From the Properties 1 and 2 and Lemmas 1 and 2 it further follows that $k = k^*$.

Assume now that G_h contains a cycle. Consider any cycle C in G_h . Let (v_i, v_j) be an edge on cycle C . Assume edge (v_i, v_j) was induced by the corner pixels x and y . Then, any path from $a1(x)$ to $a2(x)$ must go through at least one of the bottlenecks captured by cycle C . Opening the cycle by breaking the bottleneck with the maximum weight maximizes the final value of k . This is exactly what happens in the cycle-breaking problem. The existence of a path of width k' in image I uses an argument identical to the one given for the case when G_h contains no cycles. The lemma now

follows. $\square$

4 The Mesh Algorithm

In this section we describe how to perform the steps described in Stage 1 and Stage 2 on a mesh of size $n \times n$ in $O(n)$ time. Every 1-pixel of image I can determine in $O(1)$ time what type of 1-pixel it is and what type of vertices it induces in the boundary graph. The weights of the vertices and the edges in the boundary graph can be determined in $O(n)$ time using straightforward data movement techniques. Hence, Stage 1 can be executed in $O(n)$ time.

In order to set up the hole graph needed in Stage 2, assume we have labeled the holes of image I . This can be done in $O(n)$ time using a connected component labeling algorithm [2, 6]. Using the weights of the boundary graph, the hole graph is then set up in $O(n)$ time. Self-loops and multiple edges within the hole graph are eliminated in $O(n)$ time. The remainder of this section describes how to determine the largest value k' that breaks all cycles in hole graph G_h .

Assume all self-loops and multiple edges have been removed from G_h . Let k be the estimate of k^* after the removal of these edges. Assume further that the updated graph G_h contains no edges of cost $\geq k$. Our cycle-breaking algorithm uses binary search. Assume we are testing whether a given value l breaks the cycles in G_h . We remove all edges of cost $\geq l$ and check whether the resulting graph contains no cycles (i.e., whether it is a forest). If it contains a cycle, l is an upper bound for k' . If it contains no cycle, l is a lower bound for k' because it could be possible to break the cycles with a larger value. By choosing appropriate values for l each time,

we can determine the correct value of k' in $O(\log |V_h|)$ iterations. In order to obtain an efficient parallel algorithm for the mesh architecture, we combine the binary search with a data reduction technique. More precisely, after one iteration we also reduce the size of the graph by at least one half. We next describe how generate from G_h , after one cycle-breaking test, a graph of at most half the size.

Let l be the weighted median among the edge costs of graph G_h . Let $G_{h,l} = (V_{h,l}, E_{h,l})$ be the graph obtained from G_h by deleting all edges of cost $\geq l$.

Case 1: $G_{h,l}$ contains a cycle.

In this case the algorithm continues with $G_{h,l}$. Observe that $|E_{h,l}| \leq |E_h|/2$.

Case 2: $G_{h,l}$ contains no cycles.

Graph $G_{h,l}$ consists of a collection of trees, $T_1, T_2, \dots, T_r$. Let $G'_{h,l} = (V'_{h,l}, E'_{h,l})$ be the graph used by the next iteration. $G'_{h,l}$ is generated as follows. We shrink every tree T_i to a single vertex u_i and let $V'_{h,l} = \{u_1, u_2, \dots, u_r\}$. For vertex v in G_h , let $t(v)$ be the tree containing vertex v . For every edge (v_1, v_2) in G_h having edge cost c with $c \geq l$ we include in $G'_{h,l}$ the edge $(u_{t(v_1)}, u_{t(v_2)})$ with a cost of c . After all edges have been added, graph $G'_{h,l}$ contains self-loops and multiple edges. The next step is the removal of the self-loops and the multiple edges.

Let u_i be a vertex in $G'_{h,l}$ that has a self-loop of cost c . The existence of a self-loop in $G'_{h,l}$ implies that G_h contains a cycle in which all but one of the edges has cost $< l$. In order to break this cycle we need $k' \leq c$. We remove this self-loop on u_i and reduce the value of k' so that $k' \leq c$. Assume

now that there exist two edges between vertices u_i and u_j , one of cost c_1 and another of cost c_2 with $c_1 \leq c_2$. These two edges imply that graph G_h contains a cycle in which all but two edges have a weight $< l$. In order to break this cycle we need $k' \leq \max\{c_1, c_2\}$. We remove the edge with cost c_2 from $G'_{h,l}$ and reduce k' accordingly. By applying these rules to all self-loops and multiple edges, we change $G'_{h,l}$ into a simple graph. At the same time, we improve our estimate of k' .

Graph $G'_{h,l}$ may, however, not satisfy the requirement $|E'_{h,l}| \leq |E_h|/2$. When G_h contains many edges of cost l , graph $G'_{h,l}$ may contain too many edges. This problem can be fixed as follows. Assume $|E'_{h,l}| > |E_h|/2$. Let $\hat{l}$ be the smallest edge cost in G_h with $\hat{l} > l$. If no such $\hat{l}$ exists, we obviously have $k' = l$. Let $G_{h,\hat{l}}$ be the graph obtained from G_h by deleting all edges of cost $\geq \hat{l}$. When $G_{h,\hat{l}}$ contains a cycle, $k' = \hat{l}$ and we are done. If $G_{h,\hat{l}}$ contains no cycle, we apply the shrinking process described above to generate graph $G'_{h,\hat{l}}$. We now have $|E'_{h,\hat{l}}| \leq |E_h|/2$ and the next iteration uses $G'_{h,\hat{l}}$.

Figure 6 illustrates one iteration of the cycle-breaking algorithm. Figure 6(a) shows an initial graph G_h , (b) shows $G_{h,3}$, (c) shows $G'_{h,3}$ and (d) shows $G'_{h,3}$ after self-loops and multiple edges have been removed. At this point we have $3 \leq k' \leq 6$. When continuing with $G'_{h,3}$, we do not need to consider $l = 3$ again (even though 3 is the weighted median among the edge weights in $G'_{h,3}$). Using $l = 4$ does not break all cycles and thus the cycle-breaking algorithm returns $k' = 3$.

Assume every processor i of the $n \times n$ mesh contains at most one edge (u_i, u_j) of an at most n^2 -vertex planar graph G . Then, in $O(n)$ time we can determine the connected components of graph G and whether any of these

components contains a cycle. This result has been claimed for general graphs [13]. Since the algorithm for planar graphs can be simplified, we briefly sketch the main idea for an algorithm solving both problems. The algorithm uses a data reduction technique in which, in $O(n)$ time, the problem is reduced from one on at most n^2 vertices to one on at most $n^2/2$ vertices.

Every vertex u_i of G selects, among the edges adjacent to it, the smallest-indexed vertex u_j is incident to. The selected edges form a forest and we next use the algorithm described in [1] to determine the connected components of this forest. We then shrink each component representing a forest to a “supernode”, put back the edges of G not in the forest, and obtain a new graph G' . Every self-loop or pair of edges between the same pair of vertices in G' represents a cycle in G . If we are testing for cycles in G , the existence of a self-loop or a multiple edge in G' indicates the termination of the algorithm. In case we need to solve the connected component problem on G , we remove all self-loops and multiple edges in G' . It is easy to see that G' contains at most $n^2/2$ vertices. Since G is a planar graph, we have also reduced the number of edges by a constant fraction (this statement is not true for general graphs). We compress the remaining edges of G' into the top-left corner of the mesh and recursively solve the connected component or the cycle testing problem on G' . Once the connected component numbers of the vertices in G' are known, we can assign the correct component numbers to the vertices of G in $O(n)$ time. The overall running time of the algorithm determining the connecting components or testing for the existence of cycles is thus $O(n)$.

Summarizing, we obtain the following result.

Theorem 4 *Given an image I stored in an $n \times n$ mesh of processors with one pixel per processor, the largest k such that I is k -width-connected can be determined in $O(n)$ time.*

5 Conclusion

In this paper we studied the problem of determining, for a binary image I stored in a $n \times n$ mesh of processors, the largest integer k such that I is k -width-connected. We present an optimal $O(n)$ time solution to this problem. By having every pixel (i.e., the respective processor) perform local computations, our algorithm generates first a preliminary estimate of the result. The final result is then obtained by generating a graph that models bottlenecks in image I and applying a cycle-breaking algorithm to this graph.

References

- [1] Atallah, M.J., Hambrusch, S.E., "Solving Tree Problems on a Mesh-connected Processor Array", *Information and Control*, Vol. 69, pp. 168-187, 1986.
- [2] Cypher, R., Sanz, J., Snyder, L., "Algorithms for Image Component Labeling on SIMD Mesh Connected Computers", *IEEE Trans. on Computers*, Vol. 39, pp. 276-281, 1990.
- [3] Cypher, R., Sanz, J., Snyder, L., "Hypercube and shuffle-exchange algorithms for image component labeling", *Journal of Algorithms*, 1989, Vol. 10, pp. 140-150.
- [4] Dehne, F., Hambrusch, S.E., "Parallel Algorithms for Determining k -Width-Connectivity in Binary Images", *Proceedings of 2nd IEEE Symp. on Parallel and Distributed Computing*, pp 488-496, 1990 (full paper to appear in *JPDC*).
- [5] Dyer, C., Rosenfeld, A., "Parallel Image Processing by Memory-Augmented Cellular Automata", *IEEE Pami*, 1981, Vol. 3, pp 29- 41.

- [6] Hambruch, S.E., TeWinkel, L., "A Study of Connected Component Algorithms on the MPP", *Proc. of 3rd Internat. Conf. on Supercomputing*, pp 477-483, 1988.
- [7] Lim, W., Agrawal, A., Nekludova, L., "A fast parallel algorithm for labeling connected components in image arrays", *Parallel Processing for Computer and Vision Display*, Ed. P. Dew, R. Earnshaw, and T. Heywood, Addison-Wesley, pp 169-179, 1989.
- [8] Levialdi, S., "On Shrinking Binary Picture Patterns", *CACM*, 1972, Vol. 15, pp. 7-10.
- [9] Mead, C.A., Conway, L.A., *Introduction to VLSI Systems*, Addison Wesley, 1980.
- [10] Miller, R., Stout, Q., *Parallel Algorithms for Regular Architectures*, manuscript, to be published by MIT press.
- [11] Nassimi, D., Sahni, S., "Finding Connected Components and Connected Ones on a Mesh-Connected Parallel Computer", *SIAM J. on Comp.*, 1980, pp. 744-757.
- [12] Pavlidis, T., *Algorithms for Graphics and Image Processing*, CSP, 1982.
- [13] Reif, J., Stout, Q., "Optimal Component Labeling Algorithms for Mesh Computers and VLSI", manuscript.
- [14] Rosenfeld, A., "Connectivity in Digital Pictures", *JACM*, 1970, Vol. 17, pp. 146-160.

Figure 1: A path of width 3 between 1-pixels a and b

Figure 2: An image that is 3-width, but not 4-width connected

Figure 3: The boundary graph of the image shown in Figure 2

Figure 4: Illustrating Property 3 with $w(x) = 5$, $w(x, y) = 3$, 0-pixel p makes $w(y, z) > w(y, x)$ impossible

(a) assume $k = 5$ from Stage 1

(b) edges between H_1 and H_2 induced by x , x' , and x''

Figure 5: Creating edges of the hole graph