Chapter 52
A Fast Algorithm for Connecting Grid Points to the Boundary with Nonintersecting Straight Lines

Yitzhak Birk* Jeffrey B. Lotspiech*

Abstract

We consider the problem of determining whether it is possible to connect a given set of N points in an $(m \times n)$ rectangular grid to the grid's boundary using N disjoint straight (horizontal or vertical) lines. If this is possible, we find such a set of lines. Our yalgorithm can have either $O(m+n)$ or $O(N\log N)$ complexity. We then extend our algorithm to accommodate an additional constraint, namely forbidding connections in opposite directions that run next to one another. A solution to this problem is equivalent to providing a set of processor substitutions which reconfigure a fault-tolerant rectangular array of processing elements to avoid the faulty processors while retaining its important properties. We have also shown that the problem is NP-complete for 3-D grids as well as for partitioned 2-D grids.

1 Introduction.

Problem Statement and Main Results.

Consider an $(m \times n)$ rectangular grid, and a given subset of N grid points. (See Fig. 1.)

Problem 1. Connect each point to the grid boundary using a straight (horizontal or vertical) line such that different lines do not intersect, or indicate that there is no solution.

Problem 2. The same problem, with the additional constraint that connections in opposite directions in adjacent rows (columns) may have at most one common column (row) position. For example, if point (i,j) is connected to the left and point $(i+1,k)$ is connected to the right then $k \geq j$. A violation of this constraint is called a *near miss.*

We provide an efficient algorithm for solving both problems. It is linear in N once the points are sorted by row and by column.

We have also considered a variety of partitioned grids. Here, the connections are made to subgrid boundaries, but the boundaries are shared among neighboring subgrids, so any given position on the boundary can accept a connection from a point in either of two subgrids but not from both. We showed that the problem is NP-Complete for most types of partitions, as well as for 3-D grids [1].

Applications.

Problems 1 and 2 for 2-D grids were introduced by Kung et al [2][3]. They considered a rectangular processor-array which has spare processors along its boundary. A faulty processor is replaced by its neighbor, say on its right, which in turn is replaced by its right neighbor; this substitution continues until a processor in the rightmost column is replaced by a spare processor. The advantage of such a reconfiguration is that the logical structure of the array is preserved and the connections remain short. They proposed a simple hardware design for the switches which helps implement this substitution strategy, and showed how a solution to Problem 2 can be mapped directly to switch settings that implement a legal reconfiguration of the array. (The points to be connected correspond to the faulty processors, and the direction of the connection for a point corresponds to the choice of substituting neighbor.) See Fig. 2.

An actual array may consist of interconnected discrete processors, VLSI or Wafer-Scale Integration (WSI). In the two latter cases, reconfiguration is an important way of increasing the manufacturing yield, since simple replacement is essentially impossible. The importance of efficient reconfiguration algorithms is in reducing the mean time to repair a failing operational system and increasing the cost-effectiveness of the reconfiguration stage of a manufacturing process (in VLSI/WSI).

Applications of large rectangular arrays include pixel processors for high-performance displays, which can have in excess of 1M processors, systolic arrays, large multiprocessors (e.g. the intel Touchstone), etc.

The interest in partitioned grids stems from the fact that, as the number of processors increases, a decreasing fraction of them may fail and still permit reconfiguration. (Circumference/Area.)

*IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, CA 95120-6099. birk@ibm.com, lotspiech@ibm.com.

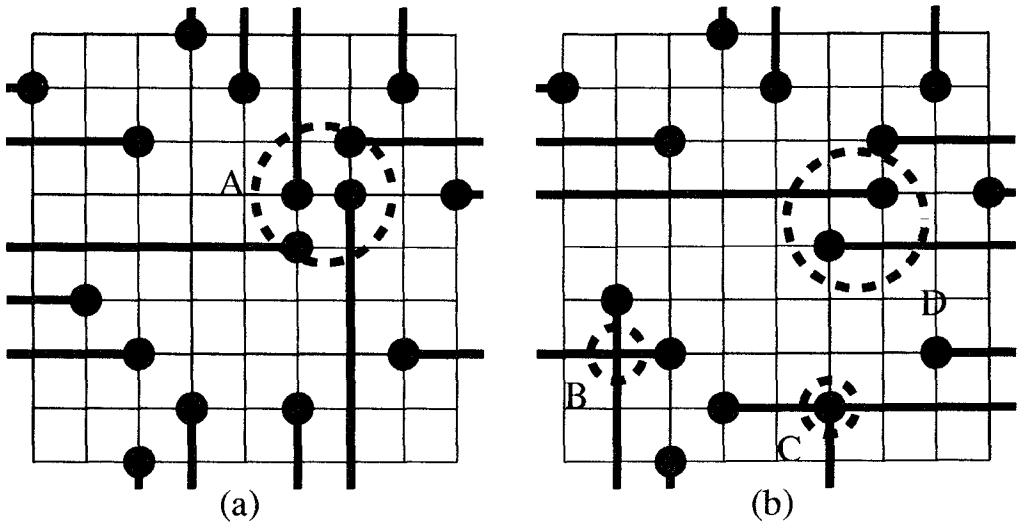

Figure 1: (a) A legal set of assignments; note that the connections jointly marked "A" do not constitute a near miss. (b) Violations of the connection rules. B: intersection; C: connection through another point; D: near miss.

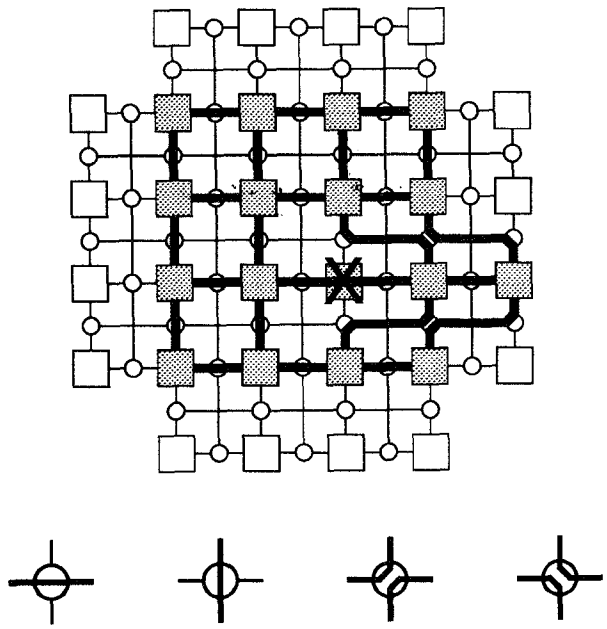

Figure 2: A rectangular grid of PEs with single-track switches and with spare PEs along the perimeter. A compensation path for a faulty processor is also shown, as are the permissible settings of the switches.

Previous Work.

A number of researchers have attempted to find efficient algorithms for problems 1 and 2: in [3], the problem was translated into that of finding a maximum independent set of vertices in a graph, which is NP-Complete, and a heuristic algorithm was provided. Ozawa [4] provided an $O(N^3)$ algorithm. Roychowhury and Bruck, who also studied related problems [6][7], developed an $O(N^2)$ algorithm [5][6].

The remainder of the paper is organized as follows. In section 2 we present an algorithm for solving problem 1. In section 3, this is extended to solve problem 2, and section 4 concludes the paper.

2 Finding Non-Intersecting Straight Paths in a Rectangular Grid (Problem 1).

2.1 Preliminaries. Every instance of Problem 1 is equivalent to one in which there are no empty rows or columns in the grid. Similarly, every Problem 2 instance is equivalent to one in which consecutive empty rows (columns) are represented by a single empty row (column). Throughout the remainder of the paper, we will therefore assume that the problem instance is provided in compact form. Once all the results are derived, we will adapt them to the case of non-compact instances.

DEFINITION 2.1. *A* partitionable solution *is one in which it is possible to pass a straight (horizontal or vertical) line through the grid without intersecting any of the connections. A* k-Blocked-Side Problem *(k-BSP) is one in which connections to k sides of the grid are forbidden. When necessary, we identify the blocked sides. For example, an LB-BSP is a 2-BSP wherein the left and bottom sides are blocked. An* active point *is one which has yet to be connected; initially, all points are active. An* active row (column) *is one that contains an active point(s). An* extremal point *in a column is the lowest or highest one; other points are* interior*; similarly for rows. Lastly, we say that a problem instance is* solvable *if and only if all its points can be connected.*

A direct, greedy approach to solving the problem, such as the one used in [5], appears to lead to quadratic

complexity. Instead, we adopt a two-step approach.

1. We attempt to find a partitionable solution.
2. If no partitionable solutions exist, we use this knowledge to facilitate the search for non-partitionable solutions.

2.2 Maintaining the Blocking Information. Our goal is to develop an algorithm with $O(N)$ complexity. In developing such an algorithm, nothing can be taken for granted. In particular, the data structures holding the blocking information and the algorithms for updating them must be carefully integrated into the main algorithm. In this section, we explain how the information is created and maintained. The integration will become apparent as we develop the algorithms.

The connection of a point in a given direction can be prevented either by another point residing on the prospective connecting line or by an earlier connection whose line intersects it. Information pertaining to the two forms of blocking is kept in separate data structures.

Point Blocking.

This information is stored per point, and is computed as follows. Initially, all points are considered blocked in all directions. We sort them by column (bucket sort), find the highest and lowest point in each column, and mark them unblocked for upward and downward connections, respectively. Similarly for rows and right/left connections. Since the problem is compact, this takes $O(N)$ steps.

LEMMA 2.1. *Point-blocking information that is based on the entire grid may be used in determining a point's connectability in an LB-BSP containing only the points to the right and above a given grid position. (Similarly for other k-BSPs.)*

Proof. The point-blocking information for a given point, say p, also reflects blocking by points that are not part of the subproblem; as such, it is incorrect. However, such points are always to the left of p or above it, and connections in those directions are not permitted anyhow.

Connection Blocking.

For each side of the grid, we keep the blocked row (for left and right sides) and column (for top and bottom sides) positions. Initially, all positions are unblocked. This information is updated as we assign connections to points, and applies to the remaining active points. (We always treat entire rows or columns, moving away from a side into the grid.)

The only connections that can prevent a remaining active point from being connected to a given side are parallel to that side and always reach the perimeter of the grid. Therefore, for each side of the grid, at most two sets of contiguous positions are blocked by connections, and each set includes one of the ends of the side. We consequently only need to keep two numbers per side. Whenever connectability to a side is tested, two numbers must be checked in addition to the local point-blocking information, and only one (different) number needs to be updated when a connection is made. This observation is critical to the construction of an efficient algorithm.

Data Organization.

We maintain three doubly-linked lists of the points: sorted by row, sorted by column, and arbitrarily-ordered "master copies" of the points. There are bidirectional pointers among the different copies of each point. The master copy of a point also contains the connectability information (point blocking), as well as the direction in which the point has been connected (initially nil). The delimiters of the two blocked intervals for each side are kept separately.

2.3 A 2-BSP with Adjacent Blocked Sides (A Corner Problem). Throughout the development of our algorithm, we rely heavily on properties of 2-BSPs with adjacent blocked sides. We now establish these properties. Without loss of generality, we consider an LB-BSP, but all the results derived here apply to other such BSPs and will be used without further comments.

LEMMA 2.2. *In an LB-BSP, connecting a point in the rightmost active column or one in the lowest active row to the right can never interfere with subsequent connections.*

Proof. Obvious.

LEMMA 2.3. *Any upward connection that is made in solving an LB-BSP from right to left or from bottom to top with preference to rightward connections must be part of every solution to the LB-BSP.*

Proof. Consider a point, say p, that is connected upward. Since there is preference to rightward connections, it follows that p could not be connected to the right. This, in turn, could be due to the existence of another point to the right of p in the same row. Alternatively, a point to the right of p and lower than it, say p', was connected upward. We proceed recursively to determine why p' could not be connected to the right. However, this recursive argument chain must end with

point-blocking, since we keep moving to the right and points in the rightmost column can all be connected to the right. Thus, the inability to connect p to the right can always be traced to point-blocking, which is determined strictly by the problem instance.

THEOREM 2.1. *Solving an LB-BSP from right to left or from bottom to top with preference to rightward connections yields a solution if and only if there is one. Moreover, such a solution minimizes the restrictions on connections of points that may be added in columns to the left of the current grid. Lastly, the problem can be solved with O(N) complexity.*

Proof. Consider a point, say p, which cannot be connected. Since this is an LB-BSP and we are solving from right to left, the inability to connect p upward can only be due to point blocking. The inability to connect it to the right is also due to point blocking (Lemma 2.3). Since point-blocking is determined only by the problem instance, there is indeed no solution. The second claim follows from Lemmas 2.2,2.3 and the fact that upward connections are the only ones that affect the connectability of the new points. The complexity follows directly from the description of the algorithm. [1]

2.4 Searching for a Partitionable Solution.

Initially, we attempt to construct a vertically-partitionable solution by constructing the largest solvable R-BSP and L-BSP. If these overlap, we are done; otherwise, we attempt to construct a horizontally-partitionable solution in a similar way. Without loss of generality, we describe the determination of the largest solvable L-BSP and the construction of a solution for it. A straightforward incremental approach (column by column) fails, since the connection of a point in a single-point column is undecidable if it can be connected upward or downward but not to the right. Instead, we take a different approach, which is based on two observations:

1. In an L-BSP, interior points in a column may only be connected to the right. Making such a connection partitions this column and the ones to the right of it into an LB-BSP and an LT-BSP.

2. In an L-BSP solved from the right, there is always a solution if every remaining active column has at most two points.

As illustrated in Fig. 3, we begin by setting the right side blocking intervals to nil. Next, we use the list of points sorted by column and scan it from right to left in search of a point that is interior in its column (this is easily determined from the point's point-blocking information); this point must be connected to the right. We make the connection and go on to solve the resulting LB-BSP and LT-BSP from right to left with preference to rightward connections. Before making a connection, we always check the right side blocked intervals as well as the point-blocking; whenever a vertical connection is made, we update the appropriate right side blocked interval. Once this is completed, we repeat the process beginning at the column to the left of the current one and without resetting the blocked intervals; in each iteration, we only connect the points in the current "band" (no backtracking). If we fail to find a legal connection, then the largest solvable L-BSP is the one whose leftmost column is immediately to the right of the one whose interior point was used for partitioning in the current iteration.

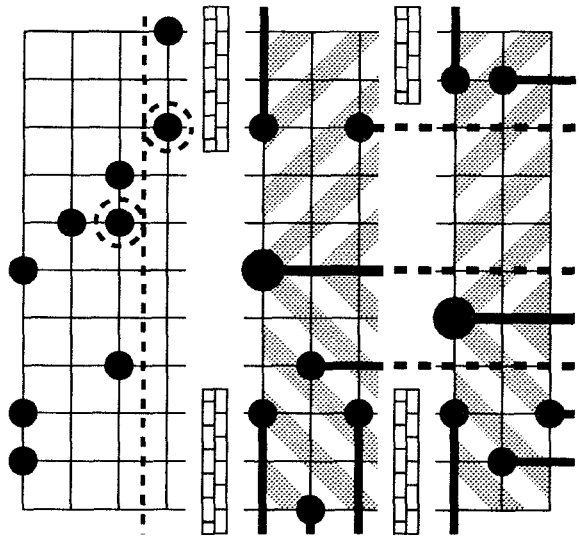

Figure 3: Incrementally finding and solving the largest solvable L-BSP. We proceed from right to left and solve "bands", each of which consists of an (LB-BSP, LT-BSP) pair. Note the passing of the blocked intervals from one band to the next.

THEOREM 2.2. *The above algorithm finds the largest solvable L-BSP and solves it with linear time and space complexity (linear in the number of points).* [2]

Proof. The partitions into LB-BSPs and LT-BSPs due to connections of interior points are clearly correct

[1] The fact that an LB-BSP can be solved in linear time was established in [5] (Lemma 3).

[2] Lemma 4 in [5] states that any given L-BSP is solvable in linear time. However, the algorithm presented there starts at the blocked side and progresses away from it. It therefore does not lend itself to an incremental determination of the largest solvable L-BSP. If that algorithm were used, this determination would require $O(N\log N)$ steps (binary search). Since we use a different algorithm, we also furnish a proof for the complexity claim.

and unavoidable, and the use of the original point-blocking information is correct (Lemma 2.1). This, along with Theorem 2.1, guarantees that each (LB-BSP,LT-BSP) pair is solved correctly. Applying this theorem in a simple induction on subproblem pairs also proves that any interference of connections made for an early pair with connections desired in later pairs is unavoidable.

The complexity of the solution to a subproblem pair is linear in the number of points in that pair; this follows from the algorithm and Lemma 2.1. Finding the columns with more than 2 points is also linear. Lastly, we have not created any new data structures. Thus, the L-BSP has linear time and space complexity.

The remaining largest solvable 1-BSPs are found and solved in a similar manner. If the union of the largest solvable L-BSP and R-BSP or that of the largest solvable T-BSP and B-BSP cover all points, this is our solution. Otherwise, we must continue.

2.5 Searching for a Non-Partitionable Solution. We use the fact that there is no partitionable solution to characterize any solutions that may exist. All the connections made in solving the 1-BSPs are discarded; we retain only the identity of the row immediately above the largest solvable T-BSP, and refer to it as the *violating row.* This is used in the search for a solution.

2.5.1 Implications of the Absence of a Partitionable Solution.

DEFINITION 2.2. *A* Clockwise Pinwheel *is a set of four points, each connected in a different direction, such that the connections resemble a clockwise pinwheel (see Fig. 4).*

Formally: the point that is connected to the right is above and not to the right of the one connected to the left; the point connected upward is to the left of and not higher than the one connected downward. A counter-clockwise pinwheel can be defined similarly.

LEMMA 2.4. *If there is no partitionable solution then any solution must contain a pinwheel.*

Proof. The inability to place a horizontal (vertical) partition implies that any solution must include at least one pair of points, one point connected downward (leftward) and the other connected upward (rightward), such that the union of the row (column) positions spanned by the two connections is the entire range of rows (columns). If neither a horizontal partition nor a vertical one is permissible, then both types of pairs must be part of any solution, but such a pair of horizontally-connected points and a pair of vertically-connected ones can only coexist if they form a pinwheel.

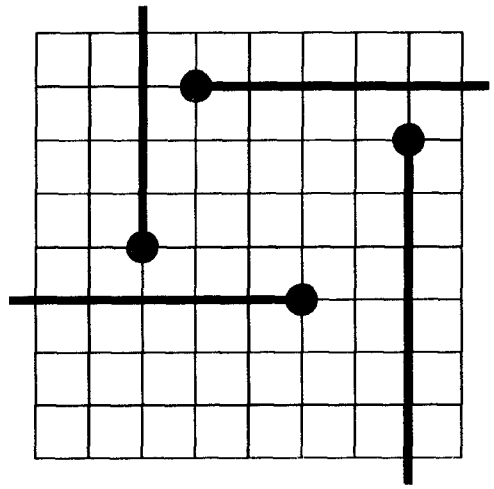

Figure 4: A clockwise pinwheel configuration.

2.5.2 Finding Candidate Pinwheels. In this section, we find a small number of candidate pinwheels, such that any solution must contain at least one of them. We begin by considering the violating row (the one immediately above the largest solvable T-BSP), and note that it must contain at least three points. Clearly, if there is no partitionable solution, then the violating row cannot be part of a solvable B-BSP either. Therefore, if an interior point in the violating row is connected downward (upward) in a solution to the problem, then there must be a point in this row or in a lower (higher) one which must be connected upward (downward) in that solution.

We pick one (arbitrary) interior point, say p_1, in the violating row. Without loss of generality, we initially assume that it is connected downward. The process will be repeated with an upward connection. The downward connection of p_1 partitions its row and the ones below it into two similar 1-BSPs. Without loss of generality, we examine the right-hand one, which is an L-BSP, and try to find a point that *must* be connected upward.

We begin solving this L-BSP from left to right, with preference to downward connections. As long as we are able to make downward connections, Lemma 2.2 applies, assuring us that these connections do not interfere with subsequent ones. Moreover, these connections cannot interfere with those of points in higher rows than the violating row. Finally, the left subproblem is isolated from the right one by the downward connection of p_1. If we can connect all points downward, we conclude that the right subproblem does not yield the sought-after mandatory upward connection and repeat the process for the left subproblem (which must therefore yield a forced upward connection). If we do find a point, say p_2, which cannot be connected downward, we must consider several cases:

1. *p_2 cannot be connected in any direction.* We conclude that there can be no solution in which p_1 is connected downward, and proceed to look for a solution in which it is connected upward.

2. *p_2 can only be connected upward.* Since we are solving from left to right and p_2 is the first point that cannot be connected downward, this can only be due to point-blocking. It follows that if p_1 is connected downward as part of a solution to the entire original problem then p_2 must be connected upward in that solution. We declare (p_1, p_2) to be a *candidate pair.*

3. *p_2 can only be connected rightward.* We make the connection, thereby partitioning the right-hand subproblem into an LB-BSP and an LT-BSP. We attempt to solve the L-BSP consisting of this subproblem-pair from right to left with preference to rightward connections until we discover a point, say p_2', which must be connected upward or cannot be connected at all. If p_2' cannot be connected at all, we conclude that there is no solution to the original problem in which p_1 is connected downward. If p_2' must be connected upward, we declare (p_1, p_2') to be a candidate pair. If we can solve the L-BSP without finding a point with the above constraints, we conclude that the right-hand subproblem does not yield the sought-after mandatory upward connection and proceed to examine the left-hand subproblem.

4. *p_2 can be connected either upward or rightward.* We again connect p_2 to the right and try to solve the L-BSP from the right with preference to rightward connections until we discover a point, say p_2', which either must be connected upward or cannot be connected at all. If there is no such point, we declare no candidate pairs and move on the left-hand subproblem. If p_2' cannot be connected at all, we conclude that there is no solution to the original problem in which p_1 is connected downward and p_2 is connected to the right. We therefore pick the upward connection for p_2 and declare (p_1, p_2) to be a candidate pair. If p_2' must be connected upward, we declare (p_1, p_2) as well as (p_1, p_2') to be candidate pairs. The meaning of two candidate pairs is that if p_1 is connected downward as part of a solution to the original problem then p_2, p_2' or both of them must be connected upward in that solution.

If the right-hand problem and the left-hand one both yield candidate pairs, it follows that there is no solution (conflicting pinwheels are required to accommodate a pair from each subproblem). Once the right-hand problem yields a pair, we therefore need not examine the left-hand one; if there is no solution, we will discover this later.

The above process thus provides us with up to four candidate pairs (including those for an upward connection of p_1). If there is a solution, it must contain at least one of those connection pairs. We discard the tentative connections made in the process of discovering the candidate pairs and, for each pair, we attempt to complete the pinwheel and solve. If we fail at any point, we discard the connections made and begin again on the next pair. We now describe the process for a single pair.

Consider a candidate pair (p_1, p_2) with p_1 connected downward and p_2 connected upward so as to form part of a clockwise pinwheel. Let us focus on the L-BSP bounded on the left by the upward connection of p_2 and on the bottom by the row containing p_1 (the shaded area in Fig. 5). We begin at the leftmost column and solve

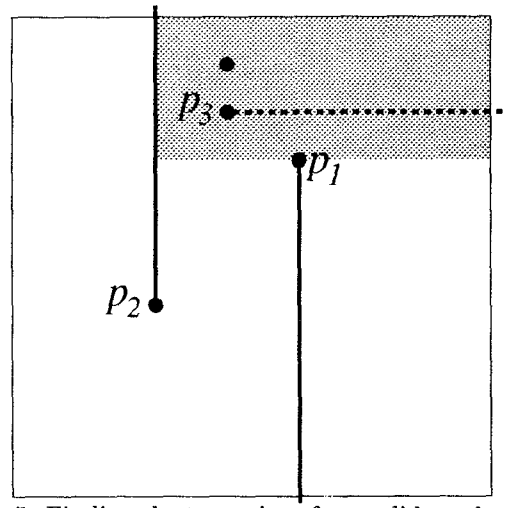

Figure 5: Finding the top point of a candidate clockwise pinwheel.

rightward with a preference for upward connections. (Lemma 2.2 guarantees that the upward connections do not interfere with any subsequent connections.)

LEMMA 2.5. *The first point, say p_3, that cannot be connected upward, must be connected to the right if there is a solution with this candidate pair; i.e., p_3 is the top pinwheel point.*

Proof. The only alternative is downward. However, if p_3 is connected downward then no points to the right of p_3 can be connected to the left. (They are blocked by the vertical connection of p_2 or by that of p_3.) Thus, any solution would permit a vertical partition immediately to the right of p_3's column, which contradicts our knowledge that there is no partitionable solution.

The fourth member of the pinwheel is found simi-

larly. The computational complexity of the pinwheel-finding step is clearly linear in the number of points. The arguments are similar to ones already used, and are not repeated.

2.5.3 Solving a Problem with a Specific Pinwheel. As illustrated in Fig. 6 for a clockwise pinwheel, we identify 13 subproblems of four types and note that they all have at least two blocked sides. (In some degenerate pinwheels, subproblem A and some of the type B subproblems do not exist, but this makes no difference.) We begin by picking a preferred direction for each 2-BSP, as depicted by the solid arrows in the figure. The other direction is marked with a dashed arrow, as is the only possible direction for each of the 3-BSPs. (The latter choice is consistent with the fact that any connection in this direction is mandatory.) The choice of preferred directions is solely a function of the orientation of the pinwheel. Next, we solve the subproblems in the following order of types: D,C,B,(A). Within each type, we solve in increasing index order. Each subproblem is solved from the appropriate side with preference to connections in the direction of the solid arrow, as was done for 2-BSPs in earlier sections. In solving a subproblem, we consider the blocked intervals presented to it by earlier subproblems and update the blocked interval for later ones.

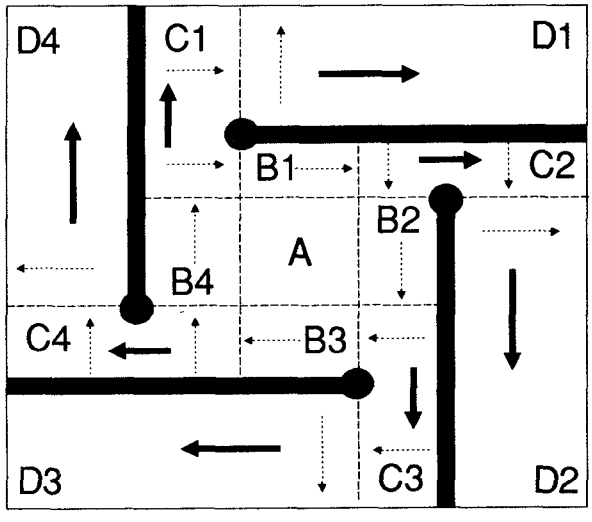

Figure 6: A "pinwheel" solution partitioned into 13 subproblems of 4 types, each with at least two adjacent blocked sides. The solid arrows denote the preferred directions. Two connections in different subproblems can only intersect if both are in the respective non-preferred directions.

THEOREM 2.3. *The algorithm will find a (correct) solution containing a given pinwheel if and only if there is one, and has linear (in the number of points to be connected) time and space complexity.*

Proof.

Correctness. The solution to each subproblem with given blocked intervals is correct (Theorem 2.1, Lemma 2.1). The problems are solved in sequence, and the blocked intervals are updated. Consequently, no blocking is overlooked.

Finding a solution if there is one. This is guaranteed by Theorem 2.1 for each subproblem given the blocked intervals. In examining Fig. 6, we observe that two (permissible) connections in different subproblems can interfere with each other only if both are in the non-preferred directions of the respective subproblems. Lemma 2.3 states that any such connections made by our algorithm are mandatory; consequently, any resulting interference could not be avoided.

Complexity. In each subproblem, we examine each point once. Since the number of subproblems is fixed, the time complexity is $O(N)$ steps even if we do not sort the points by subproblem to which they belong. Since we did not create any new data structure, other than a fixed number of blocked-interval delimiters, the space complexity is also $O(N)$.

2.6 Summary. We first attempt to solve the problem assuming that the solution can be partitioned by a straight line. If this fails, we have gained the knowledge that any solution must contain a pinwheel configuration. Furthermore, we have identified one point that must be part of such a pinwheel. Although the exact direction in which this point is connected and the identity of the remaining three pinwheel points are not yet revealed, there are at most four possibilities. We try every one. All steps have linear time and space complexity.

3 Solutions Without Near Misses (Problem 2).

3.1 Relating the Solutions of Problem 1 and Problem 2. In relating the two problems, we assume that the problem is presented as a compact instance. (Compactness in the sense of Problem 2.)

LEMMA 3.1. *If there is no solution to Problem 1 for a given instance, then there is no solution to Problem 2. Similarly, if there is no partitionable solution to 1 then there is no such solution to 2. Lastly, any solution to Problem 1 which contains a pinwheel is also a solution to Problem 2.*

Proof. Adding the "no near miss" requirement only constrains the connections, so anything that was impos-

sible in Problem 1 remains impossible in 2. A pinwheel and a near miss are mutually exclusive.

LEMMA 3.2. *If a 1-BSP instance of Problem 1 is solvable with one of the points in the row (column) closest to the blocked side connected away from that side, then the solution is also valid for Problem 2.*

Proof. Without loss of generality, let us consider a T-BSP. A near miss of vertical connections is impossible, since there are no upward connections. The downward connection of a point in the top row partitions the problem into an LT-BSP and an RT-BSP. In each of those, only one horizontal direction is permitted, so there can be no horizontal near miss.

LEMMA 3.3. *If a Problem 1 instance is solvable with a partition and the overlapping region of the relevant 1-BSPs contains a row (column) with more than 2 points, then this instance of Problem 2 also has a solution.*

Proof. By construction. Without loss of generality, we assume a horizontal partition. We pick a row with more than two points in the overlapping region, and solve the T-BSP and the B-BSP with this as the top and bottom row, respectively. Lastly, we make the actual connections for the points in this row per their assignments in the T-BSP. The claim follows from Lemma 3.2.

From the above, it follows that the only case that must be considered is an instance for which Problem 1 has a partitionable solution, but the two largest 1-BSPs have no common row (column) with interior points.

3.2 Finding a Partitionable Solution. We begin by attempting to construct a horizontally-partitionable solution. If this fails, we try to construct a vertically partitionable solution in a similar manner. We will only describe the search for a horizontally-partitionable solution. In Problem 1, we could always enlarge a T-BSP to include rows that contain at most two points by simply connecting those points horizontally. However, such haphazard connections may lead to near misses, so this approach cannot be used for Problem 2. Instead, we characterize configurations of points that are troublesome in this respect, detect such configurations, and use them to restrict and prioritize the connections in a similar manner to the use of rows with interior points in Problem 1.

3.2.1 The Horizontal Near-Miss Sequence (H-NMS).

DEFINITION 3.1. *A "Horizontal Near-Miss Sequence", H-NMS, is a sequence of points, one per row and ordered by row, such that the column positions of its members constitute either a monotonically increasing or a monotonically decreasing sequence. Moreover, the only possible horizontal connection of the rightmost (leftmost) member of the sequence is to the left (right). We use p_L (p_R) to denote the leftmost (rightmost) member (See Fig. 7). A V-NMS is defined similarly.*

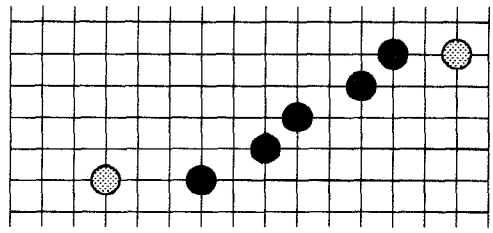

Figure 7: A horizontal near-miss sequence (H-NMS).

LEMMA 3.4. *The leftmost and rightmost points of an H-NMS, p_L and p_R, cannot both be connected horizontally.*

Proof. This would lead to a near miss.

LEMMA 3.5. *If there is a 2-point row between that of p_L and that of p_R, then either its left point (p_l) forms an H-NMS with p_L or its right point (p_r) forms one with p_R or both.*

Proof. At least one of the two points in this row must be a member of any given p_L-p_R H-NMS. If p_l can be a member of the sequence, it immediately follows that the p_L-p_l subsequence is an H-NMS. Similarly, if p_r can be a member, it immediately follows that p_r-p_R is an H-NMS.

COROLLARY 3.1. *An H-NMS is minimal if and only if all but the extreme rows contain exactly one point. (Fig. 7 depicts a minimal H-NMS.)*

DEFINITION 3.2. *The* lowest-roof H-NMS with respect to row R *is the minimal H-NMS with the lowest top row and a bottom row that is not lower than R.*

Based on Lemma 3.5 and Corollary 3.1, the lowest-roof H-NMS with respect to row R can easily be found in linear time [1].

Let p_B (p_T) denote the lowest (highest) point in an H-NMS.

LEMMA 3.6. *If p_B cannot be connected downward as part of a solution to the T-BSP containing its row and the ones below it, then the largest T-BSP cannot include the row of p_T.*

Proof. By contradiction. If it did, p_B would also be part of the T-BSP and would thus have to be connected horizontally. However, this would prevent p_T from being connected downward (intersection) or horizontally (would lead to a near miss).

The H-NMS thus plays a role similar to that played by the rows with interior points in Problem 1.

3.2.2 Constructing the Largest Solvable T-BSP.

DEFINITION 3.3. *The* upper (lower) fence *is the lowest (highest) row which is part of the largest B-BSP (T-BSP) that is known to be solvable for Problem 2. Initially, the upper (lower) fence is the lowest (highest) row with interior points which is part of the largest solvable B-BSP (T-BSP) for Problem 1. The two fences are separated by a band of rows. Since there is a horizontally-partitionable solution to Problem 1, each row in the band contains at most two points.*

The solution obtained in the construction of the largest T-BSP for Problem 1, except for rows above the uppermost one with interior points, is also a partial solution for Problem 2; moreover, it constrains the connections of points in higher rows to a minimum extent (Lemma 3.2 and Theorem 2.1). We therefore start out with the solution obtained in Problem 1 for the T-BSP whose top row is the initial lower fence, and try to grow this T-BSP upward, i.e., to raise the lower fence. Since we are dealing with a horizontal partition, we are only concerned with "horizontal" near misses. We proceed as follows:

1. Find the lowest-roof H-NMS with respect to the lower fence. If one is not found before reaching the top fence, go to step 4; otherwise,

2. Try to connect the lowest member of the H-NMS downward and solve the T-BSP consisting of its row and the ones below it. (This is an incremental solution, by bands, as in Problem 1.) If successful, move the lower fence to the roof of the H-NMS and go to step 1. Otherwise,

3. Try to connect the H-NMS member in the roof of the H-NMS upward and to solve the B-BSP whose lowest row is the roof of the H-NMS. If successful, there is a solution; go to step 4 to complete it. Otherwise, there is no solution to Problem 2 which permits a horizontal partition.

4. Connect points in the remaining band using only horizontal connections, as follows: make a pass from bottom to top (of the band), connecting points in 2-point rows to the appropriate side; points in subsequent single-point rows are connected so as not to create a near miss with the previous row; if there is a choice, they are not connected at this stage. Then, make a pass from top to bottom of the band, connecting the remaining points so as not to create a near miss with the row immediately above them; if there is a choice, pick an arbitrary direction.

THEOREM 3.1. *The foregoing 4-step algorithm is correct.*

Proof.

Correctness of the claim that there is no horizontally-partitionable solution (step 3). Given the situation, it follows from Lemma 3.6 that a partition can only be located between the extreme rows of the H-NMS. (We apply the lemma twice, reversing the roles of top and bottom.) However, if the partition is between those rows it follows that p_B and p_T must both be connected horizontally, which would lead to a near miss. Lastly, it follows from Lemma 3.2 and Theorem 2.1 that our construction of the T-BSP as we cycled through steps 1 and 2, discovering new minimal H-NMS's, could not have unnecessarily prevented us from connecting the lowest H-NMS member downward.

Correctness of the claim that we are done. As we loop through steps 1 and 2, we keep solving additional bands of a T-BSP. By Lemma 3.2, we are guaranteed that the T-BSP below the lowest row of the most recent H-NMS (this is the last one we solved) is also a valid solution to Problem 2. The remaining active rows, which contain no more than 2 points per row, are connected as described in step 4. The fact that there is no H-NMS involving those or the current fences guarantees that step 4 is completed successfully without near misses.

3.3 Constructing a Pinwheel. Consider the H-NMS that stopped the construction of the largest solvable T-BSP. From the proof of Theorem 3.1 it follows that at least one of the two extreme members of this NMS must have a vertical connection; moreover, this connection may not lead to a solvable T-BSP or B-BSP containing this member's row. If such a point is connected downward (upward), some point in the same row or in lower (higher) one must therefore be connected upward (downward) as part of any solution. This brings us back to the situation we had in Problem 1, except that we have up to eight candidate pairs since there are two potential anchors for the pinwheel. The near miss constraint no longer comes to play, since a pinwheel precludes a near miss.

3.4 Summary.

THEOREM 3.2. *For a compact instance, we can solve the connection problem with the near miss constraint (Problem 2) requiring time and space which are*

linear in the number of points to be connected.

Proof. We find a partitionable solution or decide that there are none in linear time. If there is no partitionable solution, we proceed as in Problem 1.

4 Conclusions.

We have presented a practical, efficient algorithm for connecting a given subset of points in a rectangular grid to its boundary using straight, nonintersecting lines and without near misses whenever possible. This algorithm was recently shown by M. Sarrafzadeh to be optimal (reduction from Element Uniqueness [9]). For other configurations, we showed in [1] that the problem is NP-Complete.

This algorithm can be used to provide a reconfiguration assignment for a rectangular array of processing elements in the event of processor failures. This is useful both for failures of previously functioning processors and for increasing the yield of multi-processor wafers by reconfiguring them to avoid defective parts. Our algorithm constitutes a significant improvement over the previous ones. For example, in the case of a square array of pixel processors with one million PEs, which is very realistic, our algorithm is up to 1000 times faster than the $O(N^2)$ algorithm.

Throughout the discussion, we assumed compact instances of the problem, i.e., $O(N)$ rows and columns. For those, the time and space complexities are both $O(N)$. Non-compact instances can be solved directly, with time and space complexity $O(m+n)$; alternatively, the given instance can be compacted by sorting in $O(N\log N)$ steps with space complexity $O(N)$. Since N,m and n are all known, we can choose the more desirable option. (An asymptotically more efficient $(O(N\log N/\log\log N))$ sorting algorithm is available [8], but the constants make it impractical.)

We have thus far implicitly assumed that there is a spare processor at both ends of every row and column. However, the algorithm extends easily to the case of an arbitrary subset of those locations containing spares (or faulty spares). This is done by simply updating the point-blocking information to reflect the inability to connect to the unavailable spares.

Taking a non-greedy approach in developing an almost linear algorithm is somewhat counterintuitive, yet was key to our success. We used this in two ways: (i) added a "dummy" constraint to the original problem, looking for solutions that adhere to this constraint; whenever such solutions were not found, we used this knowledge to establish a true additional constraint to which any solution (if there is one) must adhere; (ii) handled undecidable situations by deferring the decision and continuing to scan the input until an additional constraint was established which decided the situation. We recommend that this approach be tried whenever a greedy one fails.

Acknowledgment. J. Bruck told us about the problem and advised us of the existence of a quadratic-complexity algorithm.

References

[1] Y. Birk and J.B. Lotspiech, *On Finding Non-Intersecting Straight-Line Connections of Grid Points to the Boundary*, Tech. Rep. RJ 7217 (67984), IBM, Almaden Research Center, San Jose, CA Dec. 1989.

[2] S.N. Jean and S.Y. Kung, *Necessary and Sufficient Conditions for Reconfigurability in Single-Track Switch WSI Arrays*, in Proc. Int. Conf. on Wafer Scale Integration, Jan. 1989.

[3] S.Y. Kung, S.N. Jean and C.W. Chang, *Fault-Tolerant Array Processors Using Single-Track Switches*, IEEE Trans. Computers, C-38:4, pp. 501–514, Apr. 1989.

[4] T. Ozawa, *An Efficient Algorithm for Constructing Systolic Arrays from VLSI/WSI Containing Faulty Elements*, Proc. 1990 IEEE Intern. Symp. on Circuits and Systems, May 1990.

[5] V.P. Roychowdhury and J. Bruck, *On Finding Non-Intersecting Paths in a Grid and its Application in Reconfiguration of VLSI/WSI Arrays*, in Proc. Symp. On Discrete Algorithms, San Francisco, CA, Jan. 1990.

[6] V.P. Roychowdhury, J. Bruck and T. Kailath, *Efficient Algorithms for Reconfiguration in VLSI/WSI Arrays*, IEEE Trans. Computers, C-39:4, pp. 480–489, Apr. 1990.

[7] J. Bruck and V.P. Roychowdhury, *How to Play Bowling in Parallel on the Grid???*, Tech. Rep. RJ 7209 (67688), IBM, Almaden Research Center, San Jose, CA, Dec. 1989.

[8] M.L. Fredman and D.E. Willard, *Blasting Through the Information Theoretic Bound with Fusion Trees* , in Proc. ACM Symposium on Theory of Computing, Baltimore, MD, May 1990.

[9] M. Sarrafzadeh, *A Lower Bound of the Point Connection Problem*, unpublished note, Sep. 1990.