JOURNAL OF ALGORITHMS 13, 636–656 (1992)

On Finding Non-intersecting Straightline Connections of Grid Points to the Boundary

YITZHAK BIRK AND JEFFREY B. LOTSPIECH

IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, California 95120-6099

Received February 1990; revised August 1991

We consider the problem of determining whether it is possible to connect a given set of N points in an $(m \times n)$ rectangular 2D 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. We provide an algorithm with either $O(m + n)$ or $O(N \log N)$ complexity. In higher dimensions, the problem is NP-complete. We then extend our results to accommodate an additional constraint, namely forbidding connections in opposite directions that run next to one another. A solution to this problem can be used to provide 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. © 1992 Academic Press, Inc.

1. INTRODUCTION

1.1. *Problem Statement*

Consider an $(m \times n)$ rectangular grid, and a given subset of N grid points. We study two variants of the connection problem: (See Fig. 1)

Problem 1. Determine whether it is possible to connect each point to the grid boundary using a straight (horizontal or vertical) line such that different lines do not intersect. If this is possible, provide such a set of connections.

Problem 2. The same problem, with the additional constraint that there be *no "near misses"* [1]; i.e., if point (i, j) is connected to the left and point $(i + 1, k)$ is to be connected to the right then $k \geq j$. (Similarly for vertical connections.)

0196-6774/92 $5.00
Copyright © 1992 by Academic Press, Inc.
All rights of reproduction in any form reserved.

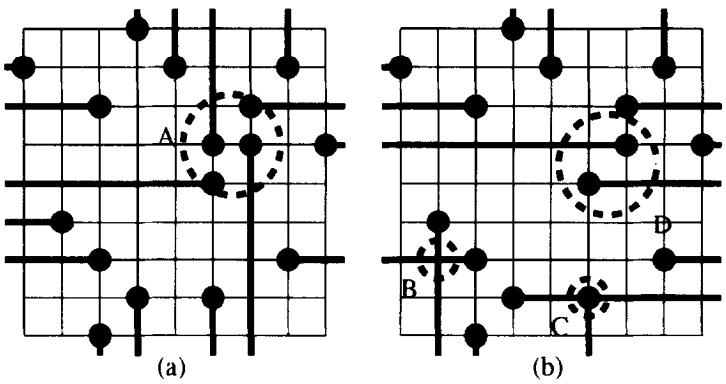

FIG. 1. (a) A legal set of assignments; note that the connections jointly marked "A" do not constitute a near miss. (b) Violations of the connection rules. B: intersection; C: connection through another point; D: near miss.

1.2. *An Application: Reconfiguring Arrays of Processors*

Rectangular arrays of processing elements (PEs) are an attractive configuration for massively parallel machines, since both the number of connections per PE and their lengths are fixed. Also, such arrays lend themselves to modular construction. This is critical for very large scale integration (VLSI) and for wafer-scale integration (WSI). Moreover, this configuration is useful for many applications (e.g., pixel processors in graphics displays).

The reliability of an array with many thousands or even millions of PEs can be disturbingly low even if individual PEs are highly reliable. However, the problem can be mitigated by including spare PEs, which can be substituted for faulty ones. In fact, the overall reliability of a multiprocessor machine with spare processors can even exceed that of a uniprocessor machine without spares. Multiprocessor programs are very often sensitive to the structure of the machine, not merely to the number of non-faulty PEs. Ideally, a PE that is substituted for a faulty one should therefore simply assume the latter's identity as well as its direct connections to its neighbors. Moreover, this should not result in a substantial increase in the communication delay.

The problems considered in this paper directly apply to rectangular arrays with single-track switches [1] (only one communication path is allowed along each horizontal/vertical channel), spare PEs along the perimeter, and an ability to convert faulty PEs into connecting elements. Figure 2 depicts such an array along with the possible switch settings. As illustrated in the figure, a faulty PE may be replaced with its healthy neighbor, say to its right; this PE, in turn, is replaced with the one to its

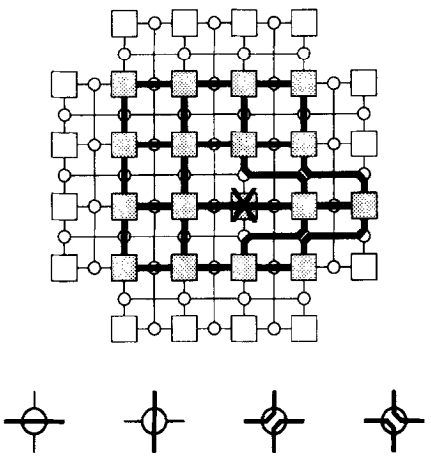

FIG. 2. A rectangular grid of PEs with single-track switches and with spare PEs along the perimeter. A compensation path for a faulty processor is also shown, as are the permissible settings of the switches.

right, and so on, until the end of the row, at which a spare PE replaces a nonspare one. The sequence of PEs in the replacement chain is referred to as the *compensation path* [1].

In [2, 1] it is shown that a legal reconfiguration is possible if one can find a set of straight, continuous, nonintersecting compensation paths for all the faulty PEs without near misses. In fact, this is the motivation for Problem 2. In [2] it is also claimed that the above is necessary, but there are exceptions [3]. Thus, the problems we address are equivalent to finding a reconfiguration that satisfies the sufficient condition for a given set of faulty PEs. Other models, based on multitrack switches, are discussed in [4, 5].

If spare PEs are located only along the perimeter of the grid, the fraction of PEs that are spares is inversely proportional to the square root of the total number of PEs. Thus, if the probability of failure of a PE remains fixed and the array is made larger, the effective yield of the array decreases. One way to overcome this is to also place rows and columns of spare PEs in the middle of the array, thus partitioning it into subarrays [1]. Here, a compensation path would end on a subgrid boundary. However, since any given spare can only substitute for one other processor, the reconfigurations of the subgrids are coupled.

1.3. *Other Applications*

Problem 1 (especially the 3D case) maps directly to the problem of concurrent reachability of a set of objects by straight arms that are parallel

to the axes. This may be useful in automated manufacturing, automated warehouses, etc. Problem 2 applies when the arm movement is not precise and extra caution is required.

The solutions to the 2D problems may be useful as a building block in placement-and-routing algorithms, especially when bends are undesirable or even prohibited. One example may be very high speed circuitry, in which the runs must be designed carefully as electromagnetic transmission lines.

1.4. *Previous Work on Problems 1 and 2*

Problems 1 and 2 for 2D grids were introduced and studied by Kung *et al.* [1] in the context of processor arrays with single-track switches. They also provided an algorithm for solving the two problems, which is based on mapping the problem to that of finding a *maximum independent set of vertices in a graph*. The idea is to represent each of the possible connections of a point as a vertex, with an edge between vertices that represent mutually-exclusive connections. The question then translates to finding a maximum independent set of vertices (MIS) with cardinality N, where N is the number of faulty PEs. The algorithm proposed in [1] is an adaptation of one for the MIS problem, which is NP-complete, and thus has exponential worst-case complexity. Nevertheless, the authors present simulation results which indicate that its average run time is quite reasonable for (20×20) grids. This approach also applies to partitioned grids and to 3D grids.

More recently, an $O(N^2)$ algorithm has been developed for the non-partitioned 2D grid by Roychowdhury and Bruck [3]. This algorithm is based on a greedy layer-peeling approach, whereby connections are first made for the outermost points and then progressively inward. A similar approach was taken by Ozawa in [6], who provided an $O(N^3)$ algorithm.

In [7], we studied Problems 1 and 2 for partitioned 2D grids and for 3D grids, and showed them to be NP-complete.

The remainder of this paper is organized as follows. In Sections 2 and 3 we develop algorithms to solve Problems 1 and 2, respectively. 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 and develop $O(N)$ algorithms to solve the problems. We will then adapt the results to the case of noncompact instances.

DEFINITIONS. 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 using L, R, T, and B to denote left, right, top and bottom, respectively. 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. Last, 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 [3], 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 nonpartitionable solutions.

2.1.1. *Maintaining the Blocking Information*

Since our goal is an $O(N)$ algorithm, efficient maintenance of connectability information is critical. This section describes our scheme. Its integration into the actual algorithms will become apparent as the algorithms are developed.

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. These are referred to as *point blocking* and *connection blocking*, respectively. Information pertaining to the two forms of blocking is kept in separate data structures.

Point blocking. This information is stored per point. 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 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, would also reflect blocking by points that are not part of the subproblem; as such, it would be overly restrictive. However, such points are always to the left of p or above it, and connections in those directions are not permitted anyhow. □

Connection blocking. Vertical connections will only be allowed in the rightmost or leftmost active column. Similarly, horizontal connections will only be allowed in the uppermost and lowest active rows. An upward connection for point (i, j) in the rightmost active column prevents any further rightward connections for points in rows i and higher (in the geographical sense). Similarly, a downward connection prevents any further rightward connections for points in rows i and lower. Thus, the rows in which rightward connections are permissible (barring point blocking) form a contiguous set which can be specified by two numbers. These are referred to as the *blocked intervals* for the right side. Similarly for all other directions. Consequently, when determining the connectability of a point in a given direction we only need to check the point's point-blocking information and the two interval delimiters for one side on the grid. Once a connection is made, we need to update one delimiter. The complexity of this test and update is thus $O(1)$.

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 its static connectability (point blocking) information, 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.

2.1.2. *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. *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. By inspection. □

LEMMA 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 4. *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 subproblem. (If a connection of such "later" points is blocked by an earlier one and this prevents a solution then there is no solution.) Last, 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 3). Since point blocking is determined only by the problem instance, there is indeed no solution. The second claim follows from Lemmas 2 and 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 observation that each point is visited only once. (The fact that an LB-BSP can be solved in linear time was established in Lemma 3 of [3].) □

2.2. *Searching for a Partitionable Solution*

ALGORITHM 1 (Partitionable solution). 1. Determine and solve the largest solvable R-BSP consisting of a set of contiguous columns, beginning with the leftmost one, and similarly for the largest L-BSP beginning with the rightmost column (see Algorithm 2 below).

2. If the largest L-BSP and R-BSP cover all columns, we have a vertically-partitionable solution and are done.

3. Else do steps 1 and 2 with appropriate modifications for T-BSP and B-BSP to look for a horizontally-partitionable solution.

Without loss of generality, we next describe the determination of the largest solvable L-BSP and the construction of a solution for it.

Lemma 4 in [3] states that any **given** L-BSP is solvable in linear time, based on an algorithm which starts at the blocked side and progresses away from it. Combined with binary search for the location of the partition, this algorithm could be used to find the largest solvable L-BSP in $O(N \log N)$ steps. This is not good enough. Instead, we construct the largest L-BSP incrementally.

A greedy 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. Our incremental construction 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.

ALGORITHM 2. Incremental construction of the largest solvable 1-BSP (L-BSP used as an example; see Fig. 3).

1. Set the right side blocking intervals to nil.
2. Sort points by column (right to left).
3. REPEAT:
   - 3.1. Find next point that is interior in its column (determined from the point's point blocking information; this point must be connected to the right).
   - 3.2. If possible, connect the point to the right; else go to 4.
   - 3.3. Solve the resulting LB − BSP and LT-BSP from right to left with preference to rightward connections. (Before making a connection, check the right side blocked intervals as well as the

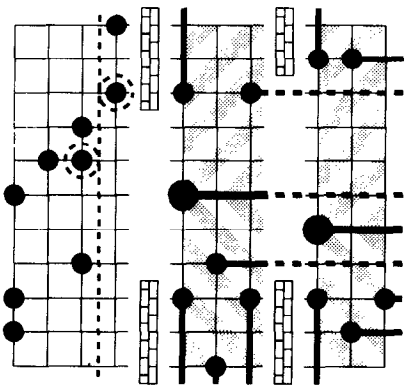

FIG. 3. Incrementally finding and solving the largest solvable L-BSP. We proceed from right to left and solve "bands," each of which consists of an (LB-BSP, LT-BSP) pair. Note the passing of the blocked intervals from one band to the next.

point blocking; whenever a vertical connection is made, update the appropriate right-side blocked interval.)

UNTIL there are no more points or a subproblem cannot be solved or the interior point could not be connected to the right.

4. If all points consumed then we have a solution.

5. Else 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 of step 3.

Note. In step 3.3, the subproblems consist only of those points in the *band* of columns beginning to the left of the previous column containing an interior point and ending with the one containing the one picked in step 3.1 of the current iteration.

THEOREM 5. *The above algorithm finds the largest solvable L-BSP and solves it with linear time and space complexity (linear in the number of points).*

Proof. The partitions into LB-BSPs and LT-BSPs due to connections of interior points are clearly correct and unavoidable, and the use of the original point-blocking information is correct (Lemma 1). This, along with Theorem 4, 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.

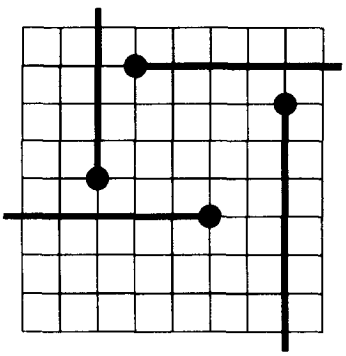

FIG. 4. A clockwise pinwheel configuration.

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 1. Finding the columns with more than two points is also linear. Last, 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.

2.3. *Searching for a Non-partitionable Solution*

2.3.1. *Implications of the Absence of a Partitionable Solution*

DEFINITION. 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 counterclockwise pinwheel can be defined similarly.

LEMMA 6. *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. □

Non-partitionable solutions are found in two primary steps: (i) construction of a set of candidate pinwheels such that any solution must contain at least one of them, and (ii) for every pinwheel in the set, attempting to find a solution containing this pinwheel. (A solution containing any one of the pinwheels suffices.)

2.3.2. *Finding Candidate Pinwheels*

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. Since there is no partitionable solution, the violating row cannot be part of a solvable B-BSP consisting of it and the higher rows. 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.

ALGORITHM 3 (Candidate pinwheels). 1. Discard all connections made in solving the 1-BSPs; retain only the identity of the row immediately above the largest solvable T-BSP, referred to as the *violating row.*

2. Pick one (arbitrary) interior point, say p_1, in the violating row.

3. Find candidate pairs of pinwheel points:

3.1. Connect p_1 downward. (This connection partitions p_1's row and those below it into an L-BSP and an R-BSP.)

3.2. Begin solving the L-BSP from left to right, with preference to downward connections. Continue until solved or some point (p_2) cannot be connected downward. (As long as we are able to make downward connections, Lemma 2 applies, assuring us that these connections do not interfere with subsequent ones. Moreover, these connections cannot interfere with (intersect) 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.)

3.3. If solution completed, repeat 3.2 for the R-BSP (left subproblem). (It is impossible that both subproblems be solvable with only downward connections.)

3.4. Having found p_2, proceed as follows:

1. p_2 *cannot be connected in any direction.* Go to 3.5. (There can be no solution in which p_1 is connected downward.)

2. p_2 *can only be connected upward.* Declare (p_1, p_2) to be a *candidate pair*. (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.)
3. p_2 *can only be connected rightward.* Make the connection, thereby partitioning the right-hand subproblem into an LB-BSP and an LT-BSP. Attempt to solve the L-BSP consisting of this subproblem-pair from right to left with preference to rightward connections until encountering a point, say p_2', which must be connected upward or cannot be connected at all. If p_2' cannot be connected at all, go to 3.5. (There is no solution to the original problem in which p_1 is connected downward.) If p_2' must be connected upward, declare (p_1, p_2') to be a candidate pair. If the L-BSP can be solved without finding a point p_2 with the above constraints, repeat 3.2–3.4 for the left-hand subproblem, and R-BSP.
4. p_2 *can be connected either upward or rightward.* Connect p_2 to the right and try to solve the L-BSP from the right with preference to rightward connections until encountering a point, say p_2', which either must be connected upward or cannot be connected at all. If there is no such point, declare no candidate pairs and repeat 3.2–3.4 for the left-hand subproblem. If p_2' cannot be connected at all (there is no solution to the original problem in which p_1 is connected downward and p_2 is connected to the right), connect p_2 upward and declare (p_1, p_2) to be a candidate pair. If p_2' must be connected upward, 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 either p_2 or p_2' or both of them must be connected upward in that solution.)

3.5. Retain candidate pairs; discard all connections; connect p_1 upward and repeat 3.2–3.4 with the appropriate modifications. (Both sets of candidate pairs are retained.)

Note. 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

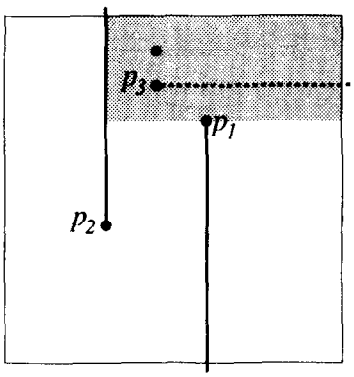

FIG. 5. Finding the top point of a candidate clockwise pinwheel.

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.

4. Complete the pinwheel for every candidate pair. We illustrate the process for 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.

4.1. Find the third pinwheel point: begin solving 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) from left to right with a preference for upward connections. (Lemma 2 guarantees that the upward connections do not interfere with any subsequent connections.) The first point that cannot be connected upward is the third pinwheel point (Lemma 7).

4.2. Find the fourth member of the pinwheel in a similar manner.

LEMMA 7. *The first point, say p_3, that cannot be connected upward, must be connected to the right in any solution which includes 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. □

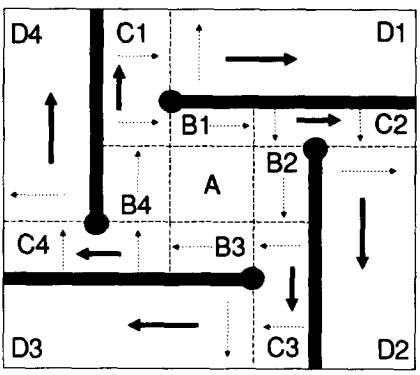

FIG. 6. A "pinwheel" solution partitioned into 13 subproblems of four types, each with at least two adjacent blocked sides. The solid arrows denote the preferred directions. Two connections in different subproblems can only intersect if both are in the respective nonpreferred directions.

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.

ALGORITHM 4. Solution with a given pinwheel.

1. Identify 13 subproblems of four types and mark the preferred directions for each one as illustrated by the thick arrows in Fig. 6 for a clockwise pinwheel. (Note that each subproblem has at least two adjacent blocked sides. In some degenerate pinwheels, subproblem A and some of the type B subproblems do not exist, but this makes no difference. The nonpreferred direction is marked with a dashed arrow, as is the only possible direction for each of the 3-BSPs, which 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.)

2. Solve the subproblems in the following order of types: D, C, B, (A). Within each type, 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, consider the blocked intervals presented to it by earlier subproblems and update the blocked interval for later ones.

THEOREM 8. *Algorithm* 4 *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 4, Lemma 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 4 for each subproblem, given the blocked intervals. In examining Fig. 6 we observe that for any subproblem, only connections in the nonpreferred direction can interfere with those of other subproblems. Lemma 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 bother to 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 $O(N)$. □

2.4. *Summary of Problem 1*

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. The main steps of our algorithm are thus as follows.

1. Determine the maximum solvable 1-BSPs and solve them. If a pair of opposite-side 1-BSPs covers the entire grid, this is the solution (Algorithms 1, 2).
2. Else look for a pinwheel solution:
   2.1. find candidate pinwheels (Algorithm 3);
   2.2. attempt to solve the problem with the different candidate pinwheels until a solution is found or all pinwheels are exhausted (Algorithm 4).

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 (in the sense of Problem 2).

LEMMA 9. *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. *Last, 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 impossible in Problem 1 remains impossible in 2. A pinwheel and a near miss are mutually exclusive. □

LEMMA 10. *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 11. *If a Problem* 1 *instance is solvable with a partition, the largest subproblems overlap, and the overlapping region contains a row (column) with more than two 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. Last, we make the actual connections for the points in this row per their assignments in the T-BSP. The claim follows from Lemma 10. □

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, present an algorithm to detect such configurations, and use them to

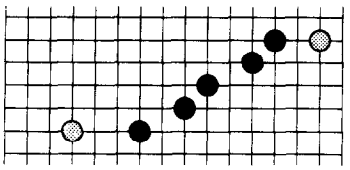

FIG. 7. A horizontal near-miss sequence (H-NMS).

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. A horizontal near-miss sequence, H-NMS for short, 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.

LEMMA 12. *The leftmost and rightmost points of an H-NMS, p_L and p_R, cannot both be connected horizontally.*

Proof. By inspection. This would force a near miss. □

LEMMA 13. *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 14. *An H-NMS is minimal if and only if all but the extreme rows contain exactly one point.*

DEFINITION. The *lowest-roof H-NMS with respect to row R* is the minimal H-NMS with the lowest possible top row and a bottom row that is not lower than R. Let p_B (p_T) denote the lowest (highest) point in an H-NMS.

LEMMA 15. *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. 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 10 and Theorem 4). 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 to 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 16. *The foregoing 4-step algorithm is correct and has time complexity* $O(N)$.

Proof. Correctness of the claim that there is no horizontally-partitionable solution (step 3). Given the situation, it follows from Lemma 15 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. Last, it follows from Lemma 10 and Theorem 4 that our construction of the T-BSP as we cycled through steps 1 and 2, discovering new minimal H-NMSs, 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 10, 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 two 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.

Complexity. Based on Lemma 13 and Corollary 14, finding a lowest-roof NMS requires $O(1)$ per row. Since each row is visited at most once, the total effort in step 1 is $O(N)$. The effort in the other steps is clearly $O(N)$. □

3.3. *Constructing a Pinwheel*

Consider the H-NMS that stopped the construction of the largest solvable T-BSP. From the proof of Theorem 16 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 of Problem 2*

THEOREM 17. *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 then proceed as in Problem 1. □

4. CONCLUSIONS

We presented an efficient algorithm for checking whether a given subset of points in a 2D rectangular grid can be connected to the grid's perimeter using straight, nonintersecting lines and without near misses. For 3D grids as well as for most variants of partitioned 2D grids, the problem is NP-complete [7].

Our algorithm can be used to reconfigure a faulty rectangular array of processing elements, thereby reducing the repair time of operational systems or increasing the effective yield of multiprocessor chips and wafers. The solution is equally applicable to arrays of arbitrary cells and possibly even to other application domains, such as concurrent-accessibility problems in automated assembly lines.

The algorithm has $O(N)$ complexity for compact instances of the problem, i.e., $O(N)$ rows and columns. Noncompact instances can be solved directly with time and space complexity $O(m + n)$; alternatively, they can first be compacted in $O(N \log N)$ steps with space complexity $O(N)$. Since N, m, and n are all given, the more desirable option can be chosen at the outset.

The $O(N \log_2 N)$ algorithm is optimal in the algebraic computation tree model. This is shown through a linear time and space reduction from element distinctness [8]. The reduction, a refinement of one by Sarrafzadeh [9], is as follows: given N elements with integer values in the range $[1, V]$, element i with integer value v_i is represented by grid point $(i + 5, v_i + 3)$. We then add three 4-point clusters: $((2,2), (1,2), (2,1), (3,2))$, $((4,2), (4,1), (3,2), (4,3))$ and $((4, V + 5), (4, V + 4), (3, V + 5), (4, V + 6))$. Each cluster consists of a primary point whose connections are blocked on three sides by the remaining points. ((3, 2) is shared by two clusters.) Any connection of the cluster points thus prevents connections of the original N points in all but a single (vertical) direction (3-BSP). If there are identical elements, there will be two points in the same column and thus no solution. Otherwise, there is a solution since the original points and the cluster points do not use common columns. A

remaining open problem is whether, given unlimited space, one can do better than $O(N \log N)$ for the case $N \ll m, n$. (Note that element-distinctness can be solved in $O(N)$ steps with V space.)

One key idea in developing an efficient algorithm was to add constraints, try to solve the constrained problem and, if unsuccessful, use the additional information (that the constrained problem has no solution) to constrain the solution if there is one. Another important idea was to "look ahead" in order to constrain the solution, rather than try to solve incrementally with small steps. This permitted us to always have a definite order of preference among options. We applied this idea in the construction of the largest possible 1-BSPs in Problem 1, where we looked for the rows with more than two points, and again in Problem 2 in the construction of the minimal H-NMS. These ideas may be useful in other problems as well.

Acknowledgment

J. Bruck told us about the 2D connection problem and advised us of the existence of an $O(N^2)$ algorithm.

References

1. S. Y. Kung, S. N. Jean, and C. W. Chang, Fault-tolerant array processors using single-track switches, *IEEE Trans. Comput.* **38**, No. 4 (1989), 501–514.
2. S. N. Jean and S. Y. Kung, Necessary and sufficient conditions for reconfigurability in single-track switch WSI arrays, *in* "Proceedings, Int. Conf. on Wafer Scale Integration, 1989."
3. V. P. Roychowdhury and J. Bruck, On finding non-intersecting paths in a grid and its application in reconfiguration of VLSI/WSI arrays, *in* "Proceedings, Symp. on Discrete Algorithms, San Francisco, CA, 1990," pp. 454–461.
4. M. Sami and R. Stefanelli, Reconfigurable architectures for VLSI processing arrays, *Proc. IEEE* **74**, No. 5 (1986), 712–722.
5. D. P. Siewiorek and R. S. Swarz, "The Theory and Practice of Reliable System Design," Digital Press, Bedford, MA, 1982.
6. T. Ozawa, An efficient algorithm for constructing systolic arrays from VLSI/WSI containing faulty elements, *in* "Proceedings, IEEE Int. Symp. on Circuits and Systems, May 1990."
7. Y. Birk and J. B. Lotspiech, "On Finding Non-intersecting Straight-Line Connections of Grid Points to the Boundary," IBM Research Report RJ 7217 (67984), Dec. 1989.
8. A. C. Yao, Lower bounds for algebraic computation trees with integer inputs, *in* "Proceedings, 30th IEEE Symp. on Foundations Of Computer Science, 1989." pp. 308–313.
9. M. Sarrafzadeh, personal communications, 1990.