JOURNAL OF ALGORITHMS 12, 516–529 (1991)

How to Play Bowling in Parallel on the Grid

JEHOSHUA BRUCK

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

AND

VWANI P. ROYCHOWDHURY*

Information Systems Laboratory, Stanford University, California 94305

Received February 26, 1990; accepted May 30, 1990

Suppose that there are m players each situated at a node of a two-dimensional grid. Every player would like to play bowling by rolling the ball towards one of the boundaries of the grid. Clearly, the paths used by the players should be perpendicular to the boundaries and nonintersecting. We would like to maximize the number of players that are able to play, namely, to find an assignment, of maximum cardinality, of paths to players. Hence, we are interested in solving the following geometrical problem: given a set of m distinguished nodes in a two-dimensional grid, determine a set of non-intersecting straight lines of maximum cardinality such that every line starts at one of the distinguished nodes and connects it to one of the boundaries of the grid. Our main result is an $O(m^3)$ algorithm for solving the problem which is the first known polynomial time algorithm for this problem. © 1991 Academic Press, Inc.

1. INTRODUCTION

In a recent paper [6] an efficient $O(m^2)$ algorithm has been presented for the following related geometrical problem: Given a set of m nodes in a two-dimensional grid, does there exist a set of m non-intersecting straight lines, such that *every given node* is connected to one of the four boundaries of the grid by one of the lines in the set? The following problem, however, was left as an open problem: Suppose that *every node* cannot be connected to the boundary of the grid by a set of nonintersecting line segments; then what is the maximum number of such nodes that can be connected to the boundary? In this paper we present an $O(m^3)$ algorithm for solving this open problem.

*Supported in part by the SDIO/IST and managed by the Army Research Office under Contract DAAL 03-87-K-0033.

The motivation for this kind of problems (besides bowling...) arises in the context of developing efficient algorithms for reconfiguring processor arrays in the presence of faulty processors. The study of reconfigurable arrays is especially important in the case of wafer scale integration (WSI) technology where, for example, a large number of processors, configured in the form of a grid, can be put on a single wafer. Due to yield problems, some of the processors are invariably going to be faulty. In such a case, instead of treating the whole wafer as defective, one can work around the faulty processors and reconfigure the rest in the form of a grid by using spare processors.

It was shown in [3] that for certain models of processor arrays, called the single track models, a sufficient condition for replacing the faulty nodes by spare ones is the existence of a set of nonintersecting straight lines that connect each of the faulty processors to one of the spares along the boundary of the array. Thus, the algorithm presented in [6] provides a polynomial time algorithm for checking whether *all the faulty processors* in a given array can be replaced by spare ones. In this context, the algorithm presented in this paper can be applied to determine the *maximum number of faulty processors* that can be replaced.

As a further comment, we might observe that while the problem solved in [6] can be regarded as a *satisfiability* problem, the problem tackled in this paper is a maximum-computing problem. There are several instances (e.g., 2-SAT [1]), where determining satisfiability is in the class P, whereas calculating the maximum is NP-complete. An example that is more geometrical in nature is the Manhattan routing problem [2, 5]. There are several efficient algorithms [2] for determining whether given pairs of nodes can be connected in the Manhattan routing scheme by non-intersecting wires; however, the corresponding problem of calculating the maximum number of pairs of nodes that can be so connected is NP-complete [5]. Thus from an algorithmic-complexity point of view, we consider our polynomial time algorithm to be very interesting.

The rest of this paper is organized as follows: Section 1.1 contains a precise definition of the geometrical problem. Section 2 contains results for some special cases, which are then used in Section 3 to develop a polynomial time algorithm for Problem 1 stated below. Finally Section 4 contains some concluding remarks.

1.1. *The Geometrical Problem*

Our geometrical problem can be stated as follows:

Problem 1. *Let V be the set of grid points in an $p \times n$ two-dimensional grid, and let $M \subset V$. Determine a set of straight lines of maximum cardinal-*

ity such that

1. *Each straight line in the set originates at some node $i \in M$ and connects it to one of the four boundaries of the grid.*
2. *Each node $i \in M$ is assigned at most one straight line in the set.*
3. *The straight lines are non-intersecting.*

Let $m = |M|$; we can now make the following observations:

1. If there is a row (column) in the grid that contains none of the nodes in M, then it is clear from the definition of our problem that the row (column) has no role. Hence, without loss of generality, we can delete such rows (columns) from the description of our problem and assume that $1 \leq p, n \leq m$.
2. If a straight line originating at any node $i \in M$ passes through another node $j \in M$, then it would be considered as intersecting with any line originating at j and would be disallowed.
3. Each vertex $i \in M$ can be assigned to at most one of four possible line segments, where each segment is along one of the four grid lines intersecting at i. Hence, instead of talking in terms of assigning line segments we can talk in terms of assigning directions; e.g., assigning a segment that connects a vertex i to the left side of the grid can be interpreted as assigning the direction *Left* to the node v. In the rest of this paper we shall interchangeably use the two equivalent descriptions.

Definition 1. An *assignment* for Problem 1 is a mapping of every node in the set M to the set of four possible directions $D = \{Left, Right, Up, Down\}$ or to the null set implying that the node under consideration is not assigned any line segment. An assignment is a *valid assignment* if the corresponding line segments do not intersect.

Thus Problem 1 involves calculating a valid assignment of maximum cardinality.

2. Solving Special Cases

In this section we consider special cases of our problem; in particular, we are going to restrict the possible directions that a node $i \in M$ can be assigned to two or three (as opposed to four in Problem 1). We shall then use the results as building blocks to solve the general problem. The two special cases are as follows:

Case 1. The line segments assigned to the nodes in M can be along only two directions, and the permitted directions are at right angles, *e.g.*, $\{Left, Down\}$ (see Fig. 1a).

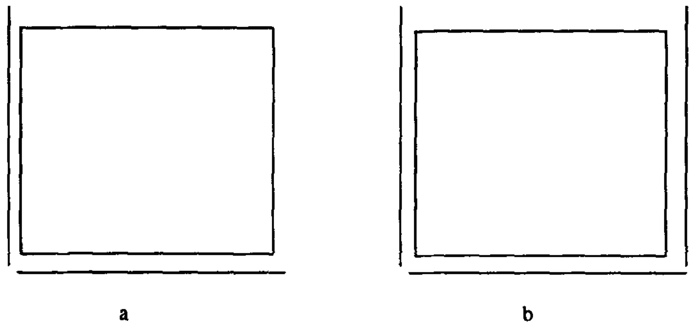

FIG. 1. Special Cases 1 and 2; permissible directions are shown by solid lines along the corresponding side.

Case 2. The line segments assigned to the nodes in M can be along only three directions, *e.g.*, $\{Left, Right, Down\}$ (see Fig. 1b).

Note that a possible assignment of a direction to a node in M can be blocked by another node. For every node $i \in M$ we shall define S_i as the set of possible directions (after taking into consideration any blocking by other nodes in M) that can be assigned to it.

The key idea for solving Case 1 is to reduce it to a problem of determining a maximum independent set in a bipartite graph.

The reduction: In Case 1, for every $i \in M$, $S_i \subseteq \{L_i, D_i\}$, where L_i, D_i correspond to the direction *Left*, *Down*, respectively. Our problem is to find a subset of $\bigcup_{i \in M} S_i$ of maximum size, such that (i) every $i \in M$ is assigned at most one line segment and (ii) the line segments are non-intersecting. It is possible to formulate the above problem as a problem of finding a maximum independent set in a bipartite graph $G(V \cup U, E)$ that is defined as follows: (1) For every S_i, if the direction $L_i \in S_i$ then define a node $v_i \in V$; similarly if $D_i \in S_i$ then define a node $u_i \in U$. That is, the set V has one node for every possible assignment of direction *Left*, and U has one node for every possible assignment of direction *Down*. (2) There exists an edge $e_k = (v_l, u_j) \in E$ if and only if either (a) line segments corresponding to v_l and u_j correspond to the same node $i \in M$, or (b) the line segments corresponding to nodes v_l and u_j intersect.

LEMMA 1. *The maximum can be computed for Case* 1 *by determining a maximum independent set in the bipartite graph* $G = (V \cup U, E)$, *as defined above.*

Proof. The proof follows directly from the construction. In particular, if we examine any independent set of G and consider the assignments that the nodes in the set correspond to, then the set of assignments satisfies the following two required properties: (a) every $i \in M$ is assigned at most one line segment (because there is an edge in G if two nodes correspond to assignments to the same node in M); and (b) the line segments corresponding to different nodes do not intersect (because if two line segments intersect then there is an edge connecting the corresponding nodes in G). Since Problem 1 requires one to determine a set of maximum cardinality that satisfies the above two properties, it can be solved by determining a maximum independent set in G. □

The maximum independent set problem is in general NP-complete [1]. Fortunately, however, there is a polynomial time algorithm for the problem when the graph is bipartite [2, 4]. Hence, for Case 1 we obtain

LEMMA 2. *There is an $O(m^{1.5} \log m)$ algorithm for computing the maximum in Case* 1.

Proof. Solving the maximum independent set problem in a bipartite graph that corresponds to intersections between line segments takes $O(m^{1.5} \log m)$ [2]. □

2.1. *A Solution for Case* 2

Can one also use the above approach to solve Case 2? One can construct a corresponding graph G for Case 2; i.e., one can define a node for every possible line segment and then define an edge between two nodes in G if the corresponding line segments intersect or if the corresponding line segments belong to the same node in M. The graph G, however, need not necessarily be bipartite and one can easily demonstrate so by constructing odd cycles in it. It suffices to mention here that cycles of odd lengths appear in G because in Case 2 it is possible to have line segments that are parallel but start at the same node in M, one to the *Right* and one to the *Left*. Since G is no longer bipartite, a polynomial time algorithm for determining a maximum independent set in G is no longer apparent.

The general idea behind solving for the maximum in Case 2 is to partition the grid into subproblems, each of which corresponds to Case 1, and then combine the results to get a solution. Before we proceed further we shall introduce certain concepts and terminologies that will be used extensively in the rest of this paper.

Consider any node $i \in M$ and the four *quadrants* defined by the node (as illustrated in Fig. 2). Let us further put the restriction that nodes belonging to M in any quadrant can only be assigned segments that do not cross the inside boundaries of the quadrant; e.g., in quadrant 1, the permissible directions are *Left* and *Down*, and in quadrant 3, the permissible directions are *Up* and *Right*. Each of the quadrants then correspond

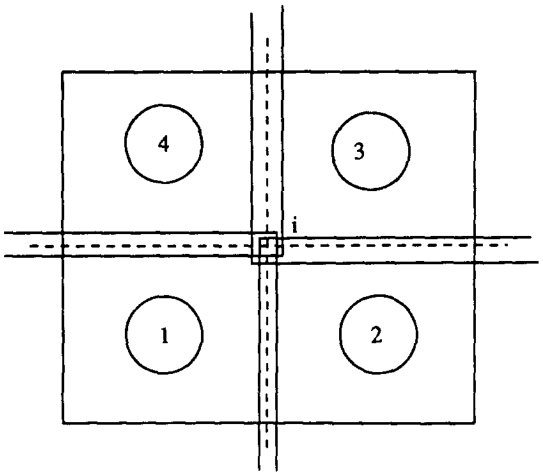

FIG. 2. The quadrants defined by a node in the grid.

to special Case 1, and the maximum number of nodes belonging to M that can be assigned compatible directions can be determined using the algorithm described above.

DEFINITION 2. For every $i \in M$, we define $q_{i1}, q_{i2}, q_{i3}, q_{i4}$ as the maximum obtained by solving Problem 1 for the respective quadrants, as defined above.

LEMMA 3. *The set of values* $\{q_{ij}\}$, $1 \leq i \leq m$ *and* $1 \leq j \leq 4$, *can be computed in time* $O(m^{2.5} \log m)$.

Proof. In order to calculate $q_{i1}, \ldots, q_{i4}$ for every $i \in M$, one has to solve Problem 1 for four instances of Case 1. Hence by Lemma 1, the complexity for calculating all the q_{ij}'s is $O(m^{2.5} \log m)$. □

We shall assume for the rest of this paper that the set of values $\{q_{ij}\}$, $1 \leq i \leq m$ *and* $1 \leq j \leq 4$, *has been precomputed.*

Before we state the result for Case 2, it will be useful to consider the following situation: Let $i \in M$, and let us assume that i can be assigned the direction *Down*. Let us also define a_i as the maximum number of nodes (in M) that can be assigned compatible line segments in the part of the grid below (and including) the row containing node i, assuming that node i is assigned *Down*. Then it follows trivially that $a_i = q_{i1} + q_{i2} - 1$ (-1 appears because the assignment of i is counted in both q_{i1} and q_{i2}). If $i \in M$ is blocked by some node in the same column but below it then we shall define $a_i = 0$.

LEMMA 4. *If we assume that the set of values $\{q_{ij}\}$, $1 \le i \le m$ and $1 \le j \le 4$, is given, then there is an $O(m)$ time algorithm for computing the maximum in Case 2. Hence, there is an $O(m^{2.5} \log m)$ algorithm for determining the maximum in Case 2.*

Proof. Consider the rows containing three or more nodes of M. Let us define r_i as the maximum number of line segments we can assign assuming that (i) in all the rows above row i none of the nodes in M is assigned direction *Down*, and (ii) in row i we assign one of the inner nodes the direction *Down*. In the case where a row has two or fewer nodes of M, define r_i to be the maximum number of horizontal (direction *Down* is not allowed) line segments we can assign in all the rows above and including the row i. *Clearly, the solution to Case 2 is obtained by taking the maximum r_i over all the rows.* Next we outline a procedure to calculate the r_i's.

In the first step, compute the set $\{a_i | i \in M\}$ from the q_{ij}'s as discussed above. The rest of the algorithm can be described as follows:

Sequentially examine the rows of the grid starting with the topmost row. Each such row may contain one, two, or more than two nodes belonging to M. The result of this scan is the set of r_i that we defined above; the maximum of the r_i's gives the desired answer. We also have a global variable that we call t that we update each time we visit a row.

1. If the row under consideration contains one or two nodes from M, then assign those nodes the directions *Left* to the leftmost node and *Right* to the rightmost node (for a single node assign one of *Left*, *Right*). Add 2 to t, if the row contains two nodes from M, otherwise add 1. Let $r_i = t$.
2. If the row under consideration contains three or more nodes from M then let $r_i = t + \hat{a}_i$, where $\hat{a}_i$ is the maximum of the a's that correspond to the nodes in row i that are blocked from both the left and the right sides (inner nodes). Add 2 to t.

It is possible to perform the above steps in $O(m)$ time (by using bucket sort). Hence, the complexity of solving Case 2 of our problem is dominated by the complexity of computing $(q_{ij}\}$ (which is $O(m^{2.5} \log m)$). It is also possible to recover the assignments of directions to the nodes that leads to the maximum. □

In fact one can show that in the same time complexity ($O(m^{2.5} \log m)$), one can compute much more than the maximum for a single instance of Case 2. Given a grid, define a *vertical partitioning* (*horizontal partitioning*) i, as a restriction that none of the line segments can cross the column (row) i. Thus a vertical (horizontal) partitioning divides the grid into two instances of Case 2. The following lemma states the time complexity for determining the maximum for all possible vertical or horizontal partitioning of the grid.

LEMMA 5. *The maximum for all the vertical and horizontal partitionings can be computed in time $O(m^{2.5} \log m)$.*

Proof. First all the q_{ij}'s are computed in time $O(m^{2.5} \log m)$. Now given the q_{ij}'s, the maximum for any particular vertical or horizontal partitioning involves calculating the maximum for two instances of Case 2. By the previous lemma, it can be done in time $O(m)$. There are $2m$ vertical and horizontal partitionings; hence, the time required after computing the q_{ij}'s is $O(m^2)$. Hence, the total time complexity is $O(m^{2.5} \log m)$ (dominated by the computations of q_{ij}'s). □

We shall assume for the next section that the maximum for all horizontal and vertical partitionings have been calculated.

3. A Polynomial Time Algorithm for Problem 1

In this section we shall outline an $O(m^3)$ time algorithm for solving Problem 1 in the general case when all the four sides are permitted. However, the discussion can be made modular if we first consider three special configurations as shown in Figs. 3a, b, and c; we shall refer to these configurations as *Type* A, *Type* B, *and Type* C, respectively. In what follows we define the configurations and describe the algorithms associated with them.

3.1. *Algorithms for the Special Configurations*

First consider the configuration *Type* A (as illustrated in Fig. 3a): it has two nodes $i, j \in M$ that are assigned the direction *Right*; moreover, it is assumed that any node of M in the region A1 can only be assigned the direction *Right*. It is easily observed that given any pair of nodes there are at most four such configurations (one for each of the directions).

We are interested in computing the maximum in the region A1 of the configuration of *Type* A; when we refer to maximum later, we shall mean the maximum in the specified region A1. The maximum can be computed easily in linear time ($O(m)$) as follows: determine the rows in the region A1 that has one or more nodes of M. Since the nodes can only be assigned the direction *Right*, the maximum is obtained by counting the total number of such rows.

DEFINITION 3. Define b_{ij}^k, $i, j \in M$, $1 \leq k \leq 4$, as the maximum for the kth *Type* A configuration of the (i, j) pair. If the kth *Type* A configuration of the (i, j) pair does not exist (because of blocking), then set $b_{ij}^k = 0$.

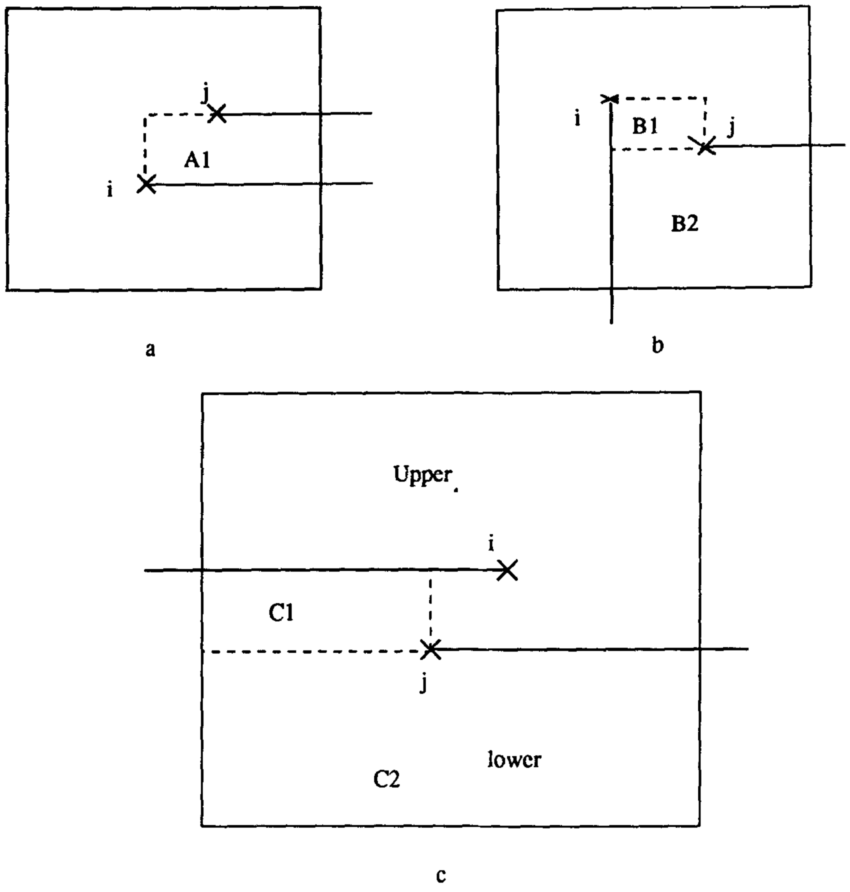

FIG. 3. Different types of configurations considered in Section 3.

LEMMA 6. *The set of values* $\{b_{ij}^k\}$, $i, j \in M$ *and* $1 \leq k \leq 4$, *can be calculated in time* $O(m^3)$.

Proof. From the above discussions we know that a particular b_{ij}^k can be computed in time $O(m)$. Since there are $O(m^2)$ such b_{ij}^k's, the total time complexity is $O(m^3)$. □

From now on, we shall assume that the set of values $\{b_{ij}^k\}$, $i, j \in M$ and $1 \leq k \leq 4$, has been precomputed.

Next consider the configuration *Type* B: it has a node $i \in M$ going *Down* and a node $j \in M$ going to the *Right*. Moreover, the only permissible direction in the region B1 is *Down* and the permissible directions in

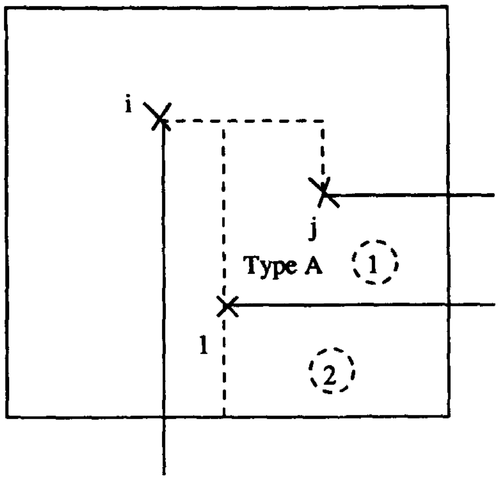

FIG. 4. An assignment within a configuration of Type B.

the region B2 are *Down* and *Right*. It is again easily observed that given any pair $i, j \in M$, there are at most four possible *Type* B configurations.

The key to solve for the maximum in *Type* B (*only in the regions* B1 *and* B2) is to judiciously use the precomputed values of q_{ij}'s and b_{ij}^{k}'s. In particular, consider any node k in the region B2 that is going to the right (as shown in Fig. 4); it divides the region (to the right of node k) in two. However, the maximum in region 1 is given by b_{kj}^{1} and the maximum in the region 2 is given by q_{k2}. Since both of these values are precomputed, we can define a new quantity $d_k = b_{kj}^{1} + q_{k2}$; thus d_k is the maximum (in the region to the right of the node k), and it can be defined to be 0 if the node k cannot be assigned the direction *Right*.

LEMMA 7. *Given the set of values* $\{q_{ij}\}$ *and* $\{b_{ij}^{k}\}$, *the maximum in any Type* B *configuration can be computed in time* $O(m)$.

Proof. Calculate d_k (as defined above) for every k in the region B2. This requires time $O(m)$; next perform the following:

Set a global variable $t = 0$. Sequentially examine the columns starting with the leftmost column (in the combined region B1, B2). Each such column may contain one, or more than one, node belonging to M. The result of this scan is a set of numbers each corresponding to a column. Let r_i be the number that corresponds to column i.

1. If the column under consideration contains only one node (belonging to M) then assign it the direction *Down* and add 1 to t.

2. If the column under consideration has more than one node (in the region B2) then let $r_i = t + \hat{d}_i$, where $\hat{d}_i$ is the maximum of the d's that correspond to the nodes in row i that are blocked from below. Add 1 to t.

Note that the r_i's, for column i with more than one node, are basically the maximum number of line segments we can assign, assuming that (i) in all the columns to the left of i we did not use the direction *Right*, and (ii) in column i we assign one of the inner nodes the direction *Right*. Clearly, the maximum is obtained by taking the maximum r_i over all the columns. □

DEFINITION 4. Define c_{ij}^k, $i, j \in M$, $1 \le k \le 4$, as the maximum for the kth *Type* B configuration of the (i, j) pair. If the kth *Type* B configuration of the (i, j) pair does not exist (because of blocking), then set $c_{ij}^k = 0$.

LEMMA 8. *The set of values* $\{c_{ij}^k\}$, $i, j \in M$, $1 \le k \le 4$, *can be calculated in time* $O(m^3)$.

Proof. One requires at most $O(m^3)$ time for precomputing q_{ij}'s and b_{ij}^k's. Moreover, given the precomputed values, we know (from the previous lemma) that a particular c_{ij}^k can be computed in time $O(m)$. Since there are $O(m^2)$ such c_{ij}^k's, the total added time complexity is $O(m^3)$. Hence, the total time complexity is $O(m^3)$. □

Next consider the configuration *Type* C (as illustrated in Fig. 3c): it has a node $i \in M$ going to the *Left* and a node $j \in M$ going to the *Right*. Moreover, as shown in the figure, the whole grid is partitioned in two non-interacting parts, and without loss of generality, let us consider the bottom part. It has two regions: (a) C1, where the permissible directions are *Down* and *Left*; and (b) C2, where the permissible directions are *Down*, *Left*, and *Right*. It is again easily observed that given any pair $i, j \in M$, there are at most two possible *Type* C configurations (assuming that we are interested in only horizontal partitions). Next, we shall discuss how to calculate the maximum in the lower part of a *Type* C configuration (the region that consists of C1 and C2); the same can be repeated for the upper part.

The idea for calculating the maximum in a *Type* C configuration is very similar to that for Case 2 (discussed in the previous section). Consider a node $l \in M$ in the region C2 that is assigned the direction *Down*. Then as defined before, a_l gives the maximum number of nodes (in M) that can be assigned compatible line segments in the part of the grid below node i. If, however, one has a node $l \in M$ in the region C1 that is going down (see Fig. 5) then the left region is an instance of Case 1; however, the right region is a configuration of *Type* B. We can thus define a quantity

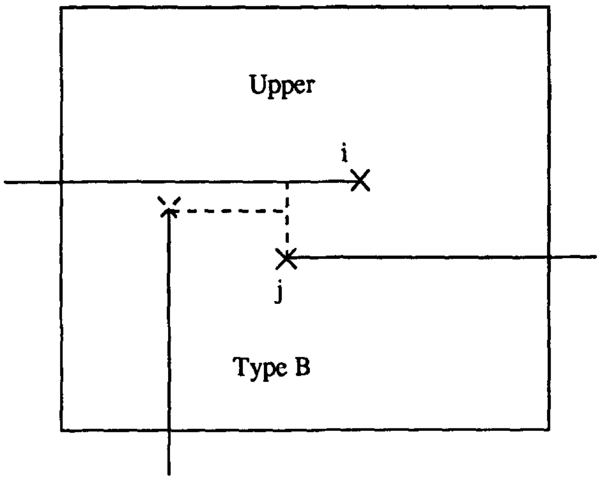

FIG. 5. An assignment within a configuration of Type C.

$f_l = q_{l1} + c^1_{lj}$ for every node $l \in M$ in the region C1; if l is blocked from below, then set $f_l = 0$. Next, we can present the following algorithm for calculating the maximum for configuration of *Type* C.

LEMMA 9. *Given the set of values* $\{q_{ij}\}$ *and* $\{c^k_{ij}\}$, *the maximum in any Type* C *configuration can be computed in time* $O(m)$.

Proof. Consider only the lower part. Calculate f_k (as defined above) for every $l \in M$ in the region C1. This requires at most time $O(m)$; next perform the following:

Set a global variable $t = 0$. Sequentially examine the rows starting with the topmost row (in the region C1). Each such row may contain one, or more than one, node belonging to M. The result of this scan is a set of numbers each corresponding to a column. Let r_i be the number that corresponds to row i.

1. If the row under consideration contains only one node (belonging to M) then assign it the direction *Left* and add 1 to t.

2. If the column under consideration has more than one node (in the region B2) then let $r_i = t + \hat{f}_i$, where $\hat{f}_i$ is the maximum of the f's that correspond to the nodes in row i that are blocked from *Left*. Add 1 to t.

3. Stop when the whole region C1 is examined; from now on, the region C_2 is the same as a special Case 3 and one can avoid repetition of computations that were done before. In particular, we showed in Lemma 5

how to precompute the maximum for all possible instances of Case 3. Set $r_{\text{last}} = t +$ the precomputed value.

Note that the r_i's, for column i with more than one node, are basically the maximum number of line segments we can assign assuming that (i) in all the rows above i we did not use the direction *Down*; and (ii) in row i we assign one of the inner nodes the direction *Down*. Clearly, the maximum is obtained by taking the maximum r_i over all the rows and also r_{last}. One can do the same procedure for the upper part of the configuration. □

LEMMA 10. *The maximum for all possible instances of Type* C *configurations can be computed in time $O(m^3)$.*

Proof. There are $O(m^2)$ possible instances of *Type* C configurations. From the previous lemma we know that given the set of values $\{q_{ij}\}$ and $\{c_{ij}^k\}$, the maximum in any (*Type* C) configuration can be computed in time $O(m)$. However, the set of values $\{q_{ij}\}$ and $\{c_{ij}^k\}$ can be precomputed in time $O(m^3)$; hence, the total complexity is again $O(m^3)$.

3.2. *The General Algorithm*

The general algorithm will just make use of the precomputed values.

LEMMA 11. *Any assignment of line segments to the nodes in M satisfies one of the following two conditions:*

1. *There is an instance of (Type* C*) configuration, or*

2. *there exists a vertical partitioning of the grid such that no line segment cuts across the partition.*

Proof. The proof follows quite obviously from the observation that if there is a vertical partitioning then a configuration of *Type* C cannot exist, and *vice versa*. Moreover, note that if there are pairs of nodes that are assigned horizontal line segments in opposite directions that overlap along some column(s), then one can always find a pair with minimum separation such that the assumptions in the configuration of *Type* C are always satisfied. □

THEOREM 1. *There is an $O(m^3)$ algorithm for solving Problem* 1.

Proof. The maximum for all possible vertical partitionings can be computed in time $O(m^{2.5m} \log m)$ (see Lemma 5). Moreover, the maximum for all possible configurations of *Type* C can be computed in time $O(m^3)$ (see Lemma 10). Hence by the previous lemma, Problem 1 can be solved by choosing the maximum over all vertical partitionings and over all configurations of *Type* C. The whole procedure takes time $O(m^3)$. □

4. Concluding Remarks

The main result in the paper is an $O(m^3)$ algorithm for solving the following problem: given a set of m distinguished nodes in a two-dimensional grid, determine a set of non-intersecting straight lines of maximum cardinality such that every line starts at one of the distinguished nodes and connects it to one of the boundaries of the grid. The motivation for this problem comes from the area of VLSI reconfiguration. There are two key ideas in our algorithm: (i) *precomputing* certain subproblems (by the reduction to the maximum independent set in a bipartite graph problem) that are useful for finding the final solution; (ii) *partitioning* the grid and solving the subproblems (using the precomputed data) and then combining the results.

The obvious open problem is: can we do better than $O(m^3)$? Another open problem is can we do better than $O(m^{2.5} \log m)$ when only three sides are permitted (Case 2 in Section 2)?

We note here that recently the result of [6] was improved from $O(m^2)$ to $O(m \log m)$ [7].

References

1. M. R. Garey and D. S. Johnson, "Computers and Intractability—A Guide to the Theory of NP-Completeness," Freeman, New York, 1979.
2. H. Imai and T. Asano, Efficient algorithms for geometric graph search problems, *SIAM J. Comput.* **15**, No. 2 (1986), 478–494.
3. 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.
4. C. H. Papadimitriou and K. Steiglitz, "Combinatorial Optimization: Algorithms and Complexity," Prentice-Hall, Englewood Cliffs, NJ, 1982.
5. R. Raghavan, J. Cohoon, and S. Sahni, "Manhattan and Rectilinear Wiring," Technical Report 81-5, Computer Science Dept., University of Minnesota, Minneapolis, 1981.
6. V. P. Roychowdhury and J. Bruck, On finding non-intersecting paths in grids and its application in reconfiguring VLSI/WSI arrays, *in* "First Annual Symposium on Discrete Algorithms, San Francisco, CA, January 1990."
7. Y. Birk and J. B. Lotspiech, A fast algorithm for connecting grid points to the boundary with nonintersecting straight lines, *in* "Second Annual Symposium on Discrete Algorithms, San Francisco, CA, January 1991."