The complexity of the edge disjoint multiple paths problem when constructed over uniformly directed mesh graphs

Hajime Nagashima and C. S. James Wong
Department of Computer Science, San Francisco State University, CA 94122

Abstract : A disjoint multiple paths problem asks if there exist paths between a given set of vertices. Constraints are applied so that paths are not allowed to share vertices (vertex disjoint multiple paths) or share edges (edge disjoint multiple paths). The vertex disjoint multiple paths problem is one of the classic NP complete problems presented by Karp[1]. The edge disjoint multiple paths problem is also NP complete since it is easily transformed from the vertex disjoint multiple paths problem. Because of its importance in electronic circuit design, studies are done for restricted cases. The edge disjoint multiple paths problem remains NP complete for acyclic graphs and planar graphs. Furthermore, the edge disjoint multiple paths problem remains NP complete if the graph is limited to an undirected mesh.

In this paper, the edge disjoint multiple paths problem when constructed over a directed mesh is discussed. We found that the multiple paths problem remains NP complete in this special case. Three polynomial time algorithms are presented in which the following restrictions are made: (i) disjoint paths with the same origin row, the same destination row, distinct origin columns, and distinct destination columns, (ii) disjoint paths with the same origin column, the same destination column, distinct origin rows, and distinct destination rows and (iii) disjoint paths with the same origin row, distinct origin columns, and distinct destination rows.

JCMCC 72 (2010), pp. 145-161

1. Introduction

Both the vertex disjoint multiple paths problem and the edge disjoint multiple paths problem were proven to be NP complete during the 1970s[1][2]. The importance of these problems comes from their usefulness in the field of electronic circuit design[4]. Creating paths between a set of origins and destinations is an important part of printed circuit board design and LSI design. In order to avoid signal interferences, paths must be disjoint. Further studies have been done for special cases that include multiple paths constructed over planar graphs and acyclic graphs. The edge disjoint multiple paths problem remains NP complete when the graph is limited to an undirected mesh.

Our interest in the multiple paths problem [6] is derived from different fields of research. We found that certain scheduling problems [5] have a close relationship with multiple paths problems. In the case of semiconductor design, electric currents flow in either direction between two connected terminals. By contrast, when graphs are used to represent state changes, the possibility of a transition from one state to another state does not guarantee the possibility of reverse state transition. Accordingly, edges need to be directed for such applications. This is the reason we need to study a directed version of the edge disjoint multiple paths problem.

The edge disjoint multiple paths problem over a uniformly directed mesh (MPUDM) is defined as the following decision problem.

INSTANCE: A graph $G = (V,E)$, a collection of vertex pairs $\{(s_1,t_1),(s_2,t_2),.....(s_k,t_k)\}$, $s_i \in V, t_i \in V$ and vertices $v_i \in V$ form a matrix. All vertices except for the ones with the highest column address have directed edges to their east adjacent vertex. All vertices except for the ones with the highest row address have directed edges to their south adjacent vertex.
QUESTION: Does G contain k mutually edge disjoint paths $P = \{p_1,p_2,p_3,...p_k\}$ where the origin of p_i is s_i and the destination of p_i is t_i?

Example: An example of the MPUDM is exhibited in Figure 1. Does G contain three mutually edge disjoint paths between s_i and t_i, $(1 \le i \le 3)$?

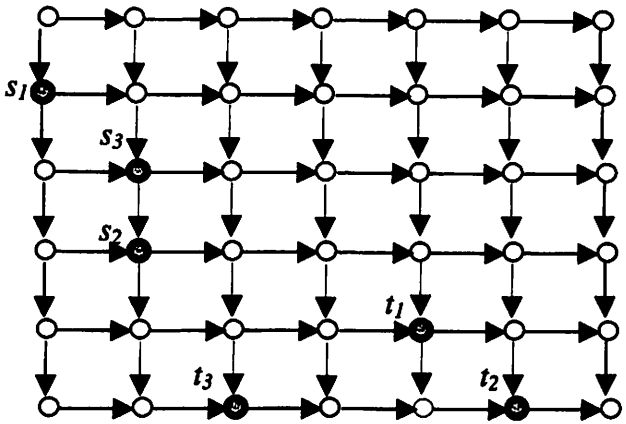


Figure 1. An instance of edge disjoint multiple paths over mesh.

In the next section, we present the proof of NP completeness of the MPUDM problem. Several polynomial time algorithms for restricted cases are presented in Section 3. Finally, the conclusion is presented in Section 4.

2. The proof of NP completeness

The proof is done by reducing an instance of the Satisfiability (SAT) problem [3] to the MPUDM. The SAT problem is defined in the following manner.

INSTANCE: A set U of Boolean variables and a collection of clauses C over U.
QUESTION: Is there a satisfying truth assignment for U such that the values of all members of C are true?

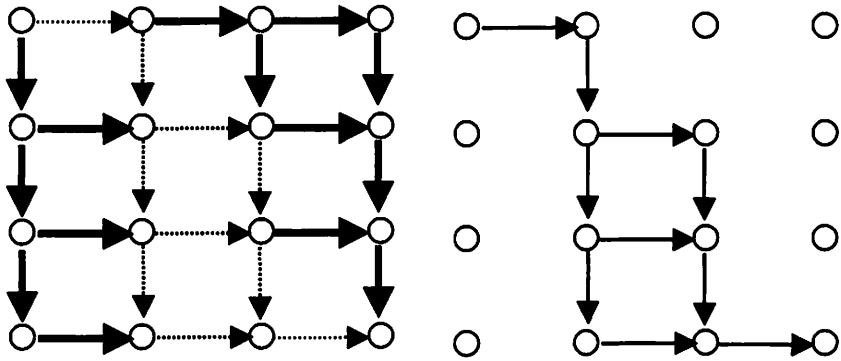
Our goal is to create an instance of MPUDM which is transformed from an arbitrary instance of the SAT problem in polynomial time. We use different kinds of paths to construct such an instance of the MPUDM.

Road blocks and routes

Consider a path from a given vertex to its adjacent vertex. This path has to occupy the only edge between the two vertices. There is no alternative way of reaching the destination vertex. Such short paths are useful in directing other paths to certain edges by blocking some edges. We call short paths used to direct other paths *road blocks*. We use road blocks to construct structures that represent Boolean variables as well as clauses. Paths that represent the state of variables and clauses are called *structure paths*. The majority of edges in the

mesh graph are blocked by road blocks and some edges are left unblocked allowing structure paths to go through them. These unblocked edges form *routes*.

A route is a series of connected unblocked edges between one origin vertex and one destination vertex. Edges in a route may be allocated to a path. Figure 2 shows an example of the usage of road blocks and a route.



(a) Road blocks restricting available edges (edges drawn bold are blocked).

(b) A route in the same graph.

Figure 2. Road blocks and a route.

A route has the following properties.

Lemma 1: If two routes have at most one common vertex, then their paths will not interfere with each other.

Proof: The assumption implies that there is no shared edge between two routes. The assumption also implies that a path which starts from one route, enters another route, and comes back to the first route is not feasible. Therefore, the set of edges available for one path is unaffected by the existence of another route.

For further discussion, we use routes to show structures that represent Boolean variables and clauses omitting edges blocked by road blocks.

Boolean variable structure

Figure 3 is a structure representing a Boolean variable.

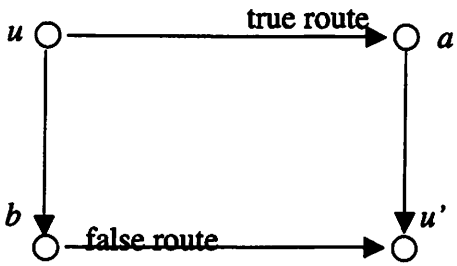


Figure 3. Boolean variable structure.

This structure has two routes, u a u' and u b u'. The former is called the true route and the latter is called the false route. A path from u to u' either occupies the true route or the false route since all other edges between u and u' are blocked. Thus, the path from u to u' takes one of two states, emulating the state of a Boolean variable. A Boolean variable structure is constructed for each variable in the SAT problem. A variable path denotes the path over a Boolean variable structure.

Multiple variable structures can be arranged so that one includes another, as shown in Figure 4. By lemma 1, routes consisting of one variable structure do not interfere with routes consisting of another variable. By convention, the outermost structure corresponds to the first Boolean variable (with the lowest index) and the innermost structure corresponds to the last Boolean variable (with the highest index).

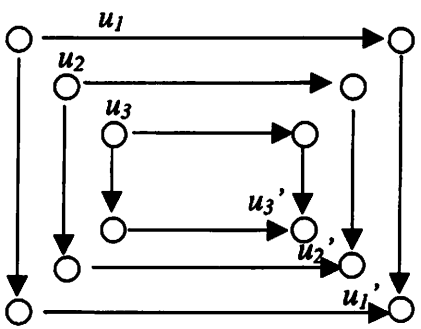


Figure 4. Arranging multiple Boolean variable structures.

True/false testing structure and clause structure

The state of a variable structure is tested and translated to be one of feasibility using the path structure shown in Figure 5.

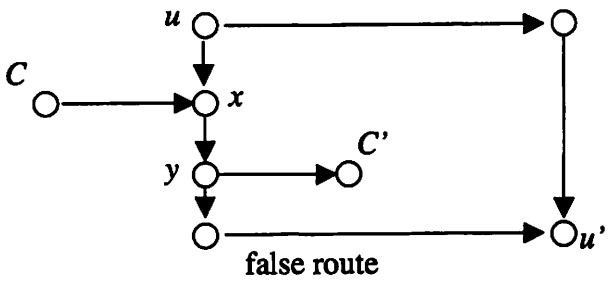


Figure 5. Simple true testing path.

The route from C to C' shares the edge between x and y with the false route of the variable structure for u. As a result, the path from C to C' is feasible if and only if the variable structure for u follows the true route. The same mechanism can be used to test if the variable structure for u follows the false route. Since this structure can test the state of one variable, these are called a simple true testing structure and a simple false testing structure.

In order to construct a structure which emulates a clause over multiple variables, we extend the structure in Figure 5 and introduce a true testing structure and false testing structure. These structures reflect the interaction between a clause and multiple variables. The true testing structure is constructed in the following manner.

1) For each Boolean variable u_i, processing in ascending order of its index, create a simple true testing structure if the clause contains u_i.
2) Connect the origins of all simple true testing structures.
3) Connect the destinations of all simple true testing structures.

Likewise, the false testing structure is constructed in the following manner.

1) For each Boolean variable u_i, processing in ascending order of its index, create a simple true testing structure if the clause contains $\overline{u_i}$.
2) Connect the origins of all simple false testing structures.

Connect the destinations of all simple false testing structures.

A clause structure is constructed by connecting the true testing structure and the false testing structure. The clause path denotes the path from the northwest edge to the southeast edge of a clause structure. The clause path is feasible if at least one of the variable structures positively participating in the clause is in a true state or at least one of variable structures negatively participating in the clause is in a false state. The clause path is otherwise infeasible. To summarize, the clause path is feasible if and only if the states of variable structures represent an assignment of Boolean variables that satisfies the clause they are configured for. The clause structure for $(u_1, \overline{u_2}, u_3)$ is shown in Figure 6.

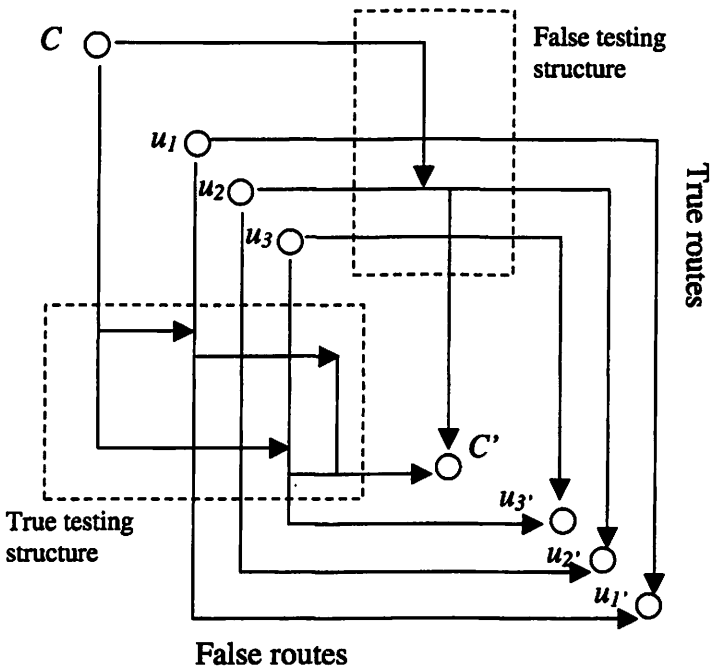


Figure 6. Clause structure for $(u_1, \overline{u_2}, u_3)$.

Building the MPUDM for an arbitrary instance of the SAT problem

A set of variable structures and a clause structure can express an instance of the SAT problem with only one clause. They can be further generalized by introducing multiple clause structures. Multiple structures are arranged so that

one surrounds the other. Per lemma 1, there is no interaction between clause structures. An example of structures that express an arbitrary instance of the SAT is shown in Figure 7.

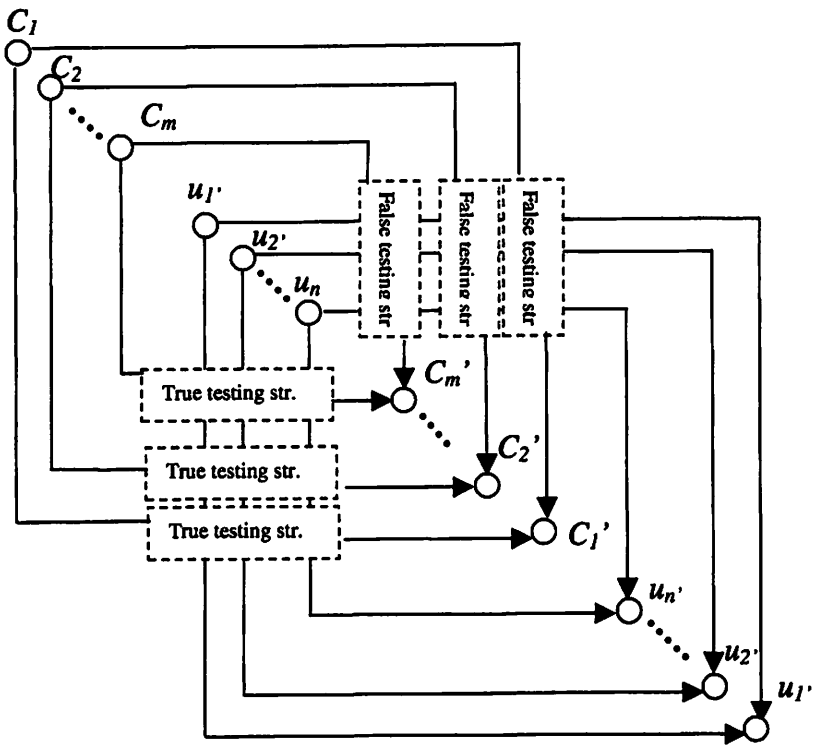


Figure 7. The MPUDM for the SAT instance with m clauses over n variables.

Lemma 2: All paths in the MPUDM built for an instance of the SAT are simultaneously feasible if and only if the corresponding SAT instance is satisfiable.

Proof : From the property of clause structures, if all paths including road blocks, variable paths, and clause paths in the MPUDM are feasible, there exists a set of states of variable structures that allow all clause paths. The assignment of variables that satisfy all clauses in the SAT instance is obtained by examining the states of variable structures.

If there is an assignment of variables which satisfies all clauses in the SAT instance, all clause paths in the MPUDM become feasible by setting variable structures reflecting the assignments of variables.

Polynomial transformation

The size of the innermost variable structure is determined by the sum of clause sizes in terms of the number of variables that they include. In the worst case, the horizontal and vertical sizes of the innermost variable structure are both upper bounded by $|C| \times |V|$. Therefore, it is constructed in polynomial time as a function of the complexity of the original SAT problem. Neither the size of the outermost variable structure nor the size of the outermost clause structure are much greater than the size of the innermost variable structure. Both are constructed in polynomial time.

The following theorem is derived from lemma 2 and polynomial transformation.

Theorem 1

The edge disjoint multiple paths problem over a uniformly directed mesh graph is NP-complete.

3. Polynomial time algorithms for restricted cases

Although the MPUDM is NP complete as discussed in the previous section, polynomial time solutions exist if additional restrictions are applied to it. We present three restricted cases and the solutions in this section.

3.1 The edge disjoint multiple paths problem with the same origin row, the same destination row, distinct origin columns, and distinct destination columns

Figure 8 illustrates an example of a restricted MPUDM problem with a 4 by 9 directed mesh graph G and a collection of 5 vertex pairs {((1,1),(4,5)), ((1,2), (4,8)), ((1,3),(4,6)), ((1,4),(4,7)), ((1,7),(4,9))}. Notice that all source vertices are from the same row and distinct columns, and all destination vertices are from the same row and distinct columns. We assume that source vertices are indexed in ascending order by their column numbers.

Each path (s_i, t_i) occupies the column interval $[o_i,d_i]$ where the column address of the s_i is o_i and the column address of t_i is d_i. We define *min_row_requirement* as the maximum number of overlapped column intervals. For Figure 8, the column intervals of all 5 vertex pairs are [1,5], [2,8], [3,6], [4,7]

and [7,9]. The maximum number of overlapped intervals is 4 which occur at interval [4,5].

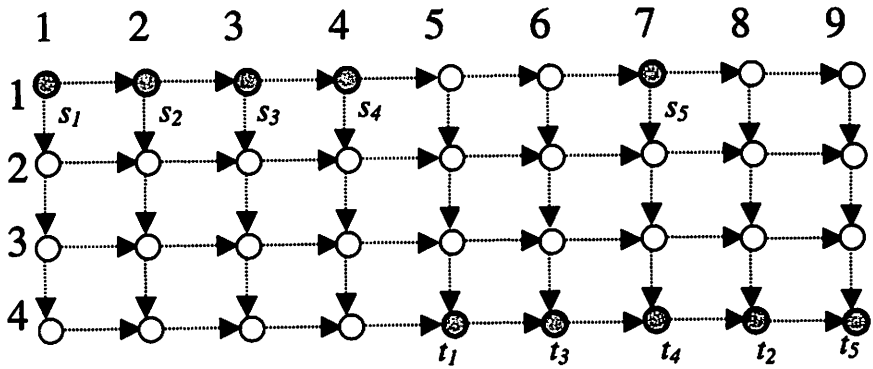


Figure 8. An example illustrating the restricted MDUMP problem with the same origin row, same destination row, distinct origin columns, and distinct destination columns.

Lemma 3: If the number of rows in graph G is less than *min_row_requirement* then there do not exist edge disjoint paths for all vertex pairs.

Proof: Assume that *min_row_requirement* is x in column interval $[c, c+1]$. This implies that x edge disjoint paths need to go from column c to column $c+1$. If the number of rows in G is less than x, then there are insufficient edges between column c and column $c+1$ for x edge disjoint paths.

If the number of rows is equal to or greater than *min_row_requirement*, then there are enough horizontal rows to be allocated to paths at any given column address. If we are able to distribute paths to rows so that the paths do not conflict over vertical edges, then all paths are constructed without conflicts. This process is accomplished by the Origin Column Order algorithm.

Origin column order algorithm: Assume source vertices are indexed in ascending order by their column numbers.

Step 0. Compute *min_row_requirement.* If *min_row_requirement* > number of rows in the graph G, then return "Not all of the vertex pairs have disjoint paths".

For each vertex pair (s_i, t_i), for i = 1 to k : // k is the index of last pair.

> Step 1. Find the first unoccupied row—Starting from s_i, traverse the vertical edges until the path reaches the unoccupied row. An unoccupied row is a row such that the edge between it and its east adjacent vertex is not assigned to any path.
>
> Step 2. Travel horizontally to the destination column—Starting from the vertex where Step 1 ends, traverse horizontal edges until the path reaches the destination column address.
>
> Step 3. Travel vertically to the destination vertex—Starting from the vertex where Step 2 ends, traverse vertical edges until the path reaches the destination.

A sample execution of the algorithm for Figure 8 is explained below and exhibited in Figure 9.

- p_1 starts at column 1 and occupies row 1 until it reaches column 5.
- p_2 starts at column 2 and occupies row 2 until it reaches column 8.
- p_3 starts at column 3 and occupies row 3 until it reaches column 6.
- p_4 starts at column 4 and occupies row 4 until it reaches column 7.
- p_5 starts at column 7 and occupies row 1 until it reaches column 9.

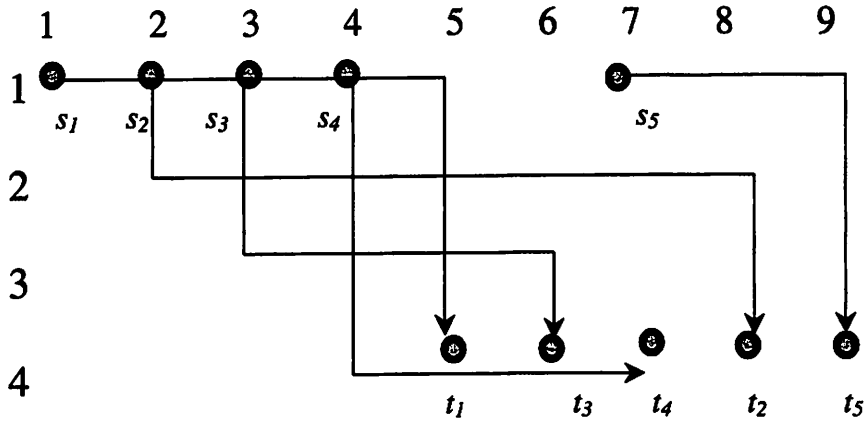


Figure 9. Solution by origin column order algorithm for the example in Figure 8.

In the remaining section, we will show that the origin column order algorithm is always able to construct all paths as long as the number of rows is equal to or greater than *min_row_requirement*. The main theorem will be based on the next few lemmas.

Suppose $p_1, p_2, p_3 \ldots, p_{i-1}$ are already constructed by the origin column order algorithm and the algorithm is attempting to construct p_i.

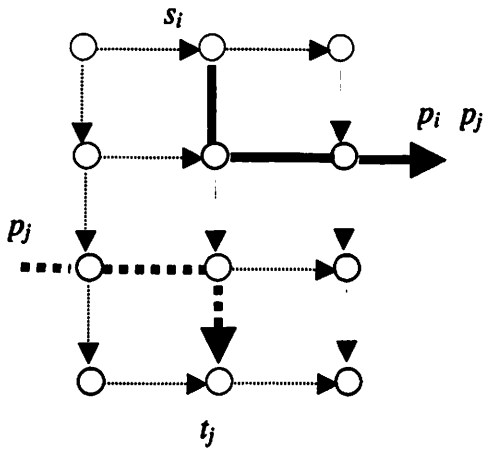
Lemma 4: The algorithm always finds an unoccupied row in p_i's origin column.

Proof: If all horizontal edges between p_i's origin column and the next column are occupied, it implies that the *min_row_requirement* exceeds the number of rows in graph G contradicting the assumption.

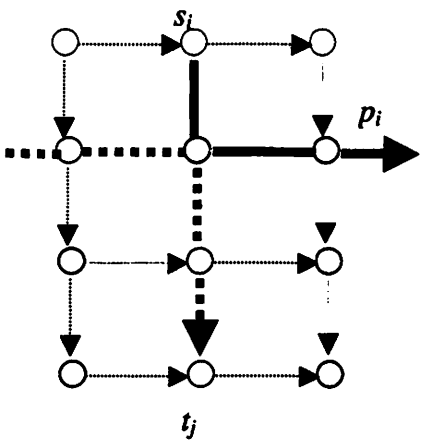
Lemma 5: The algorithm is always able to allocate the vertical edges until it reaches the first unoccupied rows.

Proof: With the origin column order algorithm, a path occupies vertical edges in its origin column or its destination column. Since two paths cannot have the same origin column address, the only possibility of two paths occupying vertical edges in the same column is the case where the origin column of a path is identical to the destination column of another path. Let p_j $(1 \leq j < i)$ be the path whose destination column is identical to the origin column of p_i, and x be the row p_j occupies to reach its destination column.

If p_i finds an unoccupied row before reaching x, there is no conflict over vertical edges as illustrated in Figure 10 (a). If p_i does not find an unoccupied row before x, p_i can occupy vertical edges until it reaches row x and p_j can occupy the vertical edges between row x and the destination as illustrated in Figure 10 (b). There is no conflict over vertical edges in this case, either.



(a) The case in which p_j's row is larger then p_i's row



(b) The case in which p_j's row is equal to p_i's row

Figure 10. The vertical edge allocation for p_i at its origin column and previously constructed paths.

Lemma 6: Once the algorithm finds an unoccupied row, it is always able to allocate horizontal edges to the path until the path reaches the destination column.

Proof: The origin column order algorithm does not allow paths to change the row they occupy until they reach their destination column. Therefore, any path $p_j (1 \leq j < i)$ does not obstruct p_i in allocating horizontal edges.

Lemma 7: The algorithm is always able to allocate vertical edges from the row it occupies to the destination.

Proof: Let $p_j (1 \leq j < i)$ be one of the previously constructed paths. p_j's origin column is less than p_i's origin column since paths are constructed in the order of their origin column. Consequently, p_j's origin column cannot be p_i's exit column. p_j's exit column cannot be p_i's exit column since two paths cannot share a destination.

Theorem 2 : If the number of rows is equal to or greater than *min_row_requirement*, and all paths have destination column addresses greater than their origin column, the origin column order algorithm always finds the solution to a MPUDM with the same origin row, the same destination row, distinct origin columns and distinct destination columns.

Proof: Lemmas 5, 6, and 7 show that it is always possible to construct the i^{th} path after $(i - 1)$ paths are constructed by the algorithm. Since it is obviously possible to construct the first path, the algorithm is always able to build all paths.

3.2 The edge disjoint multiple paths problem with the same origin column, the same destination column, distinct origin rows and distinct destination rows

Since rows and columns are symmetric in the MPUDM, we can compute *max_column_requirement* to determine the feasibility of the problem. All of the logic in Section 3.1 works if all references to rows are replaced by columns and vise versa. The paths are constructed by the origin row order algorithm.

Theorem 3 : If the number of columns is equal to or greater than *max_column_requirement*, and all paths have destination row addresses greater than their origin rows, a polynomial time algorithm exists to find the solution to

the MPUDM with the same origin column, the same destination column, distinct origin rows, and distinct destination rows.

3.3 The edge disjoint multiple paths problem with the same origin row, distinct origin columns, and distinct destination rows

To solve this case we apply the minimum turn algorithm described below.

The minimum turn algorithm: Assume source vertices are indexed in ascending order by their column numbers.

For each vertex pair (s_i, t_i), for i = 1 to k : // k is the index of the last pair

Step 1. Starting from its origin, traverse vertical edges to the path until the path reaches the destination row.

Step 2. Traverse horizontal edges until the path reaches the destination.

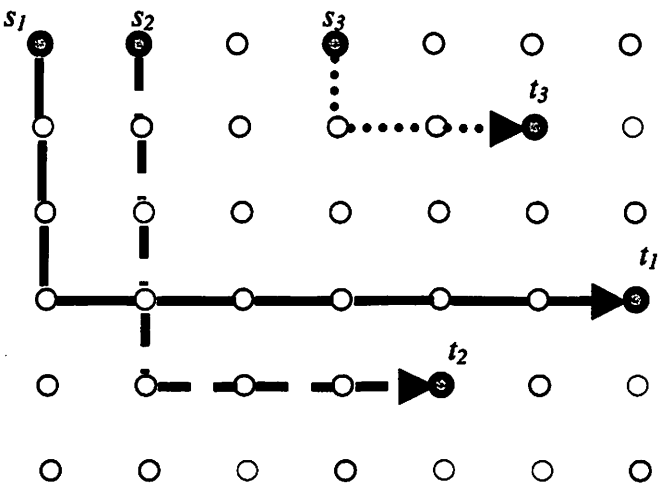


Figure 11. Sample execution of the minimum turn algorithm.

Figure 11 above shows an example and the solution obtained by the minimum turn algorithm.

Lemma 8: Considering the MPUDM with the same origin row, distinct origin columns, and distinct destination rows, paths constructed by the minimum turn algorithm do not conflict with each other.

Proof: The vertical edges allocated to a path by the minimum turn algorithm all have the same column address as the path's origin. Since two paths cannot share the same origin, multiple paths do not occupy vertical edges in the same column. Likewise, the horizontal edges allocated to a path by the minimum turn algorithm all have the same row address as the path's destination. Since two paths cannot have the same destination row, multiple paths do not occupy horizontal edges in the same row. Thus, there is no conflict over vertical edges or horizontal edges. The solution that the minimum turn algorithm finds is a valid solution for the multiple paths problem.

Theorem 4: Considering the MPUDM with the same origin row, distinct origin columns, and distinct destination rows, the minimum turn algorithm finds the solution whenever a solution exists.

Proof: The minimum turn algorithm finds a path for any given pair of origins and destinations whenever the destination is reachable from the origin. From the previous lemma, paths created by this algorithm don't conflict with each other. Consequently, the minimum turn algorithm finds all paths if all destinations are reachable from their origins. The algorithm does not find the solution if one or more destinations are unreachable from the origin.

4. Conclusions

In this paper we have studied the edge disjoint multiple paths problem when constructed over a directed mesh. We have shown that the general problem is NP complete. We presented polynomial time algorithms for three restricted cases : (i) disjoint paths with the same origin row, the same destination row, distinct origin columns, and distinct destination columns, (ii) disjoint paths with the same origin column, the same destination column, distinct origin rows and distinct destination rows, and (iii) disjoint paths with the same origin row, distinct origin columns, and distinct destination rows.

For future research, it would be interesting to investigate the complexity of the MPUDM with fewer restrictions such as "same origin row to destinations with an arbitrary row or column", "arbitrary origin row or column to the same destination row", as well as the case allowing two paths to share origins and/or destinations.

References :

[1] R. M. Karp, "On the complexity of combinatorial problems," *Networks*, 5, January 1975, pp. 45-68.

[2] M. R. Garey and D. S. Johnson, *Computers and Intractability:A Guide to the Theory of NP-Completeness*, W. H. Freeman, NY, 1979.

[3] S. A. Cook, "The complexity of theorem-proving procedures," *Proceedings Third Annual ACM Symposium on Theory of Computing*, May 1971, pp. 151-158.

[4] M. R. Kramer and J. Leeuwen, "The complexity of wire routing and finding minimum area layouts for arbitrary VLSI circuits," *Advances in Computing Research*, Vol. 2, 1984, pp. 129-46.

[5] J. Y-T. Leung, T. W. Tam, and G. H. Young, "On Line Routing of Real-Time Messages," *Journal of Parallel and Distributed Computing*, 34, 1996, pp. 211-217.

[6] H. Nagashima, "The complexity of the edge disjoint multiple paths problem when constructed over uniformly directed mesh graphs," *MS Project Report TR-05.07 2005,* Department of Computer Science, San Francisco State University, CA 94122.