Available online at www.sciencedirect.com
SCIENCE DIRECT®
Theoretical Computer Science 326 (2004) 155–185
Theoretical Computer Science
www.elsevier.com/locate/tcs

Complexity of pairwise shortest path routing in the grid

Teofilo F. Gonzalez*, David Serena

Department of Computer Science, University of California, Santa Barbara, CA 95064, USA

Received 14 July 2003; received in revised form 9 June 2004; accepted 18 June 2004
Communicated by G.F. Italiano

Abstract

In parallel and distributed systems many communications take place concurrently. The efficient delivery of all the messages depends on the routing algorithms as well as the underlying interconnection network topology. The grid is a planar network topology that lends itself for efficient VLSI implementation and therefore is of interest for theoretical analysis. Frequently, networks and switches achieve high performance by delivering the messages through shortest paths. In addition, network fault tolerance improves through insuring that the traversed paths are both edge and/or node disjoint. The edge disjoint criterion is useful when network links are the predominant constraint, and the node disjoint criterion becomes important when switches are the fault tolerant bottleneck. Because the latter necessarily implies the former, it is apparent that node disjointness contributes to fault tolerance and enhanced performance. In this paper, we examine the *k*-pairwise node and edge disjoint shortest paths problem in the undirected graph topology of the grid. Herein it is shown that the *k*-pairwise node as well as the *k*-pairwise edge disjoint shortest paths decision problems are NP-hard, and remain NP-hard even for many different restrictions on the problem. We also discuss polynomial time algorithms for restricted versions of our problems.

Keywords: Grid networks; Fault-tolerance; Node disjoint shortest paths; Edge disjoint shortest paths; NP-completeness

* Corresponding author.
E-mail addresses: teo@cs.ucsb.edu (T.F. Gonzalez), david.serena@navy.mil (D. Serena).

0304-3975/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2004.06.027

1. Introduction

We establish NP-completeness or NP-hardness for routing problems defined over the mesh or grid topology, which are common to many applications. Specifically, we show that the k-pairwise node and edge disjoint shortest path problems in grid graphs are NP-hard. Before we present our results, we formally define our problems and discuss previous related work. The *k-pairwise node disjoint shortest paths problem* for the grid is given p pairs of nodes and q blocking nodes ($k = p + q$) denoted by

$$X = \{X_1, X_2, \ldots, X_p, X_{p+1}, X_{p+2}, \ldots, X_{p+q}\}$$

where $X_i = (s_i, t_i)$, for $1 \leqslant i \leqslant p$, and $X_i = (a_i)$ for $p + 1 \leqslant i \leqslant p + q$, find node disjoint shortest paths in the grid for all the pairs X_i, i.e. the paths do not include blocking nodes and no two such paths have a node in common. Each pair $X_i = (s_i, t_i)$ consists of two *endpoints* which are called the *source* and *target*, respectively. The nodes a_i are called *blocking nodes or faulty processors*. Every node in the *grid* is represented by an ordered pair of integers (j, k) and there is an edge from node (j, k) to nodes $(j + 1, k)$ and $(j, k + 1)$. The distance between the source and target nodes of pair X_i (or *pair distance*) in the grid is denoted by $d(X_i) = d(s_i, t_i)$ and it is $|j - \ell| + |k - m|$ where $s_i = (j, k)$ and $t_i = (\ell, m)$. This distance is usually called *Manhattan* distance. By a *shortest path* for the pair X_i we mean any path from s_i to t_i with length equal to $d(X_i)$, i.e. the path is the shortest path in the graph between the two nodes independent from any other blocking nodes or endpoints of pairs $X_{i'}$ for $i' \neq i$. The *edge disjoint shortest paths problem* is given X (without faulty nodes or $q = 0$) in the grid, find shortest paths connecting each s_i to t_i such that no two paths have an edge in common.

Note that in the context of undirected graphs the order of the source to target in the routing request is not important. Therefore, in this context we consider the undirected pairs $X_i = \{s_i, t_i\}$ instead of the directed pair (s_i, t_i). The directed pair problems are studied in [6]. Both the decision problem and the search problem (generate a solution for yes-instances) are important for analysis. The problem instance $\{\{(0, 0), (1, 1)\}, \{(0, 1)\}\}$ has a solution, and search algorithms would generate the path $(0, 0) \leftrightarrow (1, 0) \leftrightarrow (1, 1)$. The problem instance $\{\{(1, 1), (2, 2)\}, \{(1, 2)\}, \{(2, 1)\}\}$ in the 4×4 grid has no solution because of the blocking nodes, but it has a routing with arbitrary length paths: $(1, 1) \leftrightarrow (0, 1) \leftrightarrow (0, 2) \leftrightarrow (0, 3) \leftrightarrow (1, 3) \leftrightarrow (2, 3) \leftrightarrow (2, 2)$. For the edge disjoint shortest path problem the instance $\{\{(1, 1), (2, 2)\}\}$ has a solution; however, the problem instance $\{\{(1, 1), (2, 2)\}, \{(2, 1), (1, 2)\}\}$ does not have edge disjoint shortest paths.

Many of the message routing problems mentioned above are known to be computationally difficult for general graphs when one allows arbitrary length paths, rather than just shortest ones. Karp [10] analyzes the k-pairwise disjoint paths problem in general graphs and established NP-completeness. Shiloach [20] presented a polynomial time algorithm to construct node disjoint paths in a graph for two pairs of vertices, and Watkin [21] showed that $(2k - 1)$-connectedness is a necessary condition for a graph to admit disjoint paths for a set of k pairs of vertices.

The related problem where one seeks to find k-pairwise node disjoint paths (*arbitrary distance pairs*) from vertices in set $\{s_1, s_2, \ldots, s_k\}$ to vertices in set $\{t_1, t_2, \ldots, t_k\}$ is called the set-to-set node disjoint paths problem. In this problem one needs to find k node disjoint

paths from one set to the other such that the paths are from s_i to $t_{\phi(i)}$ where $\phi : \{1, 2, \ldots k\} \rightarrow \{1, 2, \ldots k\}$ and ϕ is any bijective function. Whereas in the k-pairwise problem ϕ is the identity function, namely $\phi(i) = i$. The undirected vertex version of Menger's theorem is applicable to the former problem, but not to the latter [16]. It is not applicable to the set-to-set node disjoint *shortest* paths problem as Menger's Theorem may imply *arbitrary* length paths.

The node disjoint paths problem for the n-cube has been studied even in the presence of node faults [15,7,8]. Gonzalez and Serena [5,6] studied the node and the edge disjoint *shortest* paths problem for the n-cube. Similarly, algorithms for the node-to-set node disjoint *shortest* paths problem for the n-cube or hypercube have been developed [3,13,18].

Typically *routing requests*, like X above, are classified by the occupancy of source and destination nodes at the vertices. An *h-k routing request* constrains each source node to appear at most h times, and each target node to appear at most k times at a vertex in the graph [9]. This routing request problem has also been called the multimessage unicasting message routing problem [4]. If edge disjoint paths exist for all h-k routing requests in a graph or digraph, the graph is said to be h-k-rearrangeable. A routing request is ℓ-rearrangeable if any 1-1 routing request can be partitioned into ℓ routing requests each of which has an edge disjoint routing [9]. Concerning the concept of h-k-rearrangeability and ℓ-rearrangeability the edge disjoint paths are not restricted to shortest paths. A *permutation routing request* is a 1-1 routing request wherein all vertices are occupied by a source and a target. A *partial permutation routing request* relaxes the constraint that all vertices be occupied. An example of a partial permutation routing request is when vertex occupancy is constrained to a source or target, but not both. We call this request the *partial half permutation routing request*. Furthermore, the routing request for the problem defined at the beginning of this section is called the *partial half permutation routing request with singletons*.

The problem of finding disjoint paths in two dimensions has been extensively studied in [1,11,14] because of its applications to VLSI routing. In [11] the classical grid routing model is used and explores the grid routing problems. Formally, the $n_0 \times n_1$ grid, $G = (V, E)$, is a graph structure where vertices are labeled by pairs of integers $(a_0, a_1) \in V$ such that $0 \leqslant a_i < n_i$ for $i \in \{0, 1\}$. Furthermore, edge $\{(a_0, a_1), (a_2, a_3)\} \in E$, if both $a_0 + \delta = a_2$ and $a_1 + (1 - \delta) = a_3$ hold for $\delta \in \{0, 1\}$. The capacity, c, is the maximum number of paths that may use any given edge. The number of pins p describes the occupancy of the vertices in the grid. The case when $p = 1$ corresponds to the partial half permutation routing request without singleton nodes in the grid. Karp et al. [10] analyzes constant bend routing methodologies for the k-pairwise edge disjoint case and establishes provably good heuristics. Furthermore, they have shown that the optimal one turn routing problem is NP-complete when $c = 2$. In other words, they allow two paths to travel along each edge, but every path has at most one bend. In this paper, we show that when $c = 1$ (one path per edge) the one turn routing problem is polynomially solvable, but the two turn problem is NP-complete. All of our node and edge disjoint shortest paths problems are defined only for $c = 1$. In the literature this corresponds to c-realizability for edges. Edge (node) c-realizability allows at most c paths to traverse any edge (node) [19].

We show in Sections 2.1 that the k-pairwise edge disjoint shortest paths problem is NP-hard in the grid (when $c = 1$). *Bends* in the grid path correspond directly to a change in the path direction from vertical to horizontal, or from horizontal to vertical. In Section 2.2

we show that the edge disjoint shortest path problem remains NP-complete even when we restrict all the paths to have no more than some fixed constant number of bends (greater than or equal to two). However, when every path is restricted to have one bend we show that the problem is solvable in polynomial time. In addition, we show that the *k*-pairwise edge disjoint shortest paths problem remains NP-complete even when every pair distance is bounded by some fixed constant (Section 2.3).

Kramer and vanLeeuwen [12] established that the *k*-pairwise node disjoint (arbitrary length) paths problem in the grid is NP-hard. This problem has also been studied for multi-layer grids [1]. Agrawal et al. [1] also discuss the node disjoint arbitrary length path with constant number of bends, and establish an upper bound for the trade off between the number of layers and area in a general multilayer grid model. In this paper the number of layers is one.

In Section 3.1, we discuss the node disjoint *shortest* paths problem and establish that for a partial half permutation routing request the problem is NP-hard. In addition in Section 3.2 we show that for a sufficiently large constant bound on the pair distance the *k*-pairwise node disjoint shortest paths problem remains NP-complete. The grid structure is generalizable to higher dimensional topologies. Since the grid is embeddable in those topologies, NP-completeness and NP-hardness results for those structures directly follow from our grid results.

2. Complexity of grid edge disjoint shortest path problems

We show that the edge disjoint shortest path problem in the (undirected) grid (Section 2.1) is NP-hard. Then we establish that the problem remains NP-hard even when every path is limited to have at most a constant number of bends (Section 2.2), and when every the pair distance is bounded by a constant (Section 2.3). When there is a sufficient constraint on the number of bends the problem is shown to be polynomial time solvable (Section 2.2, Theorem 2). It is interesting to note that the result in Section 2.3 does in fact imply a constant bound on the number of bends; however, it is a far larger constant than the three bends required by the construction in Section 2.2.

2.1. Edge disjoint shortest paths

We establish that the *k*-pairwise edge disjoint shortest path problem in the grid graph is NP-hard by reducing the SAT problem to it. Our reduction has some architectural similarities to the reductions for the corresponding problems defined over the *n*-cube [5,6]. From an instance I of the SAT problem, we construct an instance $f(I)$ of the *k*-pairwise edge disjoint shortest path problem. First we give a general overview of the reduction, specify the pairs to be introduced for the variables and clauses as well as for additional pairs, called blocking pairs, that force certain paths for all the variable and clause pairs. We give a couple of examples to illustrate our informal reduction and then we formally define our reduction.

There are two types of components: *setting* and *clause-checker*. The *setting* component assigns to each variable a value (true or false). For each variable u_i there is a pair indicated by the x_i's in Fig. 1. We will also introduce blocking pairs (which are represented by thick

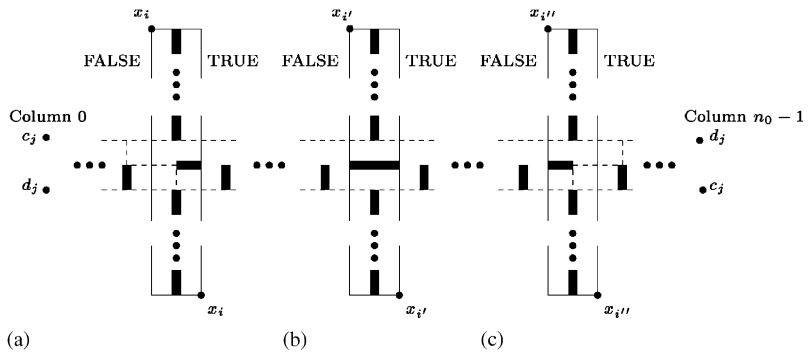

Fig. 1. Basic architecture of the construction: (a) Variable u_i is (uncomplemented) in clause C_j. (b) Variable (uncomplemented) $u_{i'}$ or (complemented) variable $\bar{u}_{i'}$ are not in clause C_j. (c) Negation of variable $u_{i''}$ is in the clause C_j. The thick lines represent the only possible shortest path for blocking pairs.

lines in Fig. 1 joining the pair by the only possible shortest path for the pair) that will guarantee that all the x_i pairs can only be connected by a path with one turn or bend in a shortest path solution. The top–down vertical to horizontal path corresponds to the value false assigned to x_i and the top–down horizontal to vertical path corresponds to the value true assigned to x_i. There is a *clause checker* for each clause C_j. Each checker consists of two pairs, c_j and d_j (see Fig. 1). Furthermore, shortest paths for pairs c_j and d_j have 2 bends or 4 bends. We will introduce *blocking pairs* (which are represented by thick lines in Fig. 1 joining the pair by the only possible shortest path for the pair) in such a way that shortest paths for pairs c_j and d_j exist if and only if the clause they represent is satisfied, i.e. at least one of its literals has the value true. The pairs c_j and d_j must have paths that cross in one position. This is due to the fact that in column 0, c_j is above d_j and in the rightmost column $n_0 - 1$, d_j is above c_j. In Fig. 1 we use vertical ellipses to indicate a continuation of the ith variable and the location for other clauses. Horizontal ellipses in Fig. 1 indicate the intersection regions of the clause with other variable pairs. Possible paths are indicated by thin solid or dashed lines.

Let us now discuss in more detail the pairs for the variables and clauses. The pair x_i is located at $(a, 0)$ and $(a + 2, n_1 - 1)$, for some positive integers a and n_1 to be determined later. Remember that in all of our figures the first coordinate increases from left to right and the second one increases from top to bottom. The pair c_j is located at $(0, c)$ and $(n_0-1, c+2)$ and the ones for the d_j pairs are located at $(0, c+2)$ and (n_0-1, c), for some positive integer c which will be determined later on. For every pair of vertices we define their *shortest path region* as the set of all the vertices which lie in any shortest path for the pair. We introduce additional pairs, called *blocking* pairs, located at the intersection of the shortest path region for the ith variable pair and the jth clause pair as well as between adjacent intersection regions. As mentioned above, these pairs are represented by thick lines along the only possible shortest path for the blocking pair. These pairs limit the possible paths for the variable (x_i) pairs and/or the clause (c_i and d_i) pairs. The pairs added in the shortest path region for the ith variable pair and the jth clause pair are different depending upon whether or not the variable or its complement is present in the clause. In what follows we refer

to this intersection as the *intersection region for the variable and clause pair* or simply the *intersection region* if the variable and clause pairs are understood. The *neighborhood* of an intersection region of a variable and a clause includes the intersection region plus the previous and next columns inside the shortest paths region for the clause. As we shall see later on, the neighborhood for a clause and two adjacent variables have a column in common.

In what follows we specify the location of the blocking pairs for the intersection regions. The blocking pairs introduced at the intersection regions for the three different cases (variable is not in the clause, variable is in the clause and appears uncomplemented, or complemented) are given in Fig. 1. We justify our constructions below.

The neighborhood of the intersection for the ith variable and the jth clause pair when the variable is in the clause is shown in Fig. 1(a). The region allows a crossing of the path for the pairs c_j and d_j when the variable has been assigned the value of true, i.e., it is connected by a path that uses the vertical right-hand side of the shortest path region for the variable. If the path for the variable pair x_i traverses the vertical left-hand side of the shortest path region for the variable, i.e. the variable has the value of false, then it is not possible for the paths for the pairs d_j and c_j to cross in this neighborhood. Fig. 2(a) depicts the crossing of the paths for c_j and d_j when the variable has the value of true. Fig. 2(c) can be used to show that the crossing is not possible when the variable has the value of false.

The neighborhood region when the negation of a variable is in a clause is given in Fig. 1(c). The region allows a crossing of the path for the pairs c_j and d_j when the variable has been assigned the value of false. Fig. 2(b) depicts the crossing of the paths for c_j and d_j when the variable has the value of false. Fig. 2(d) can be used to show that the crossing is not possible in the neighborhood when the variable has the value of true.

When a variable is not present in a clause the neighborhood region is such that the crossing of the path for pairs c_j and d_j is not possible. (Fig. 1(b)). Fig. 2(e) and (f) can be used to show that it is not possible for the paths for pairs c_j and d_j to cross in the neighborhood no matter what value the variable is assigned.

The pairs c_j and d_j need only cross at one variable pair in the proper state. Note that two or more possible crossings may be available for c_j and d_j (see c_1 and d_1 in Fig. 3). When a crossing occurs we know the clause is logically satisfied. If the clause cannot be satisfied then no crossings occur throughout the structure. There is no solution as the paths have not crossed. In this case there are no edge disjoint paths for both the pairs c_j and d_j.

Before we formally define our reduction, we give a couple of examples to illustrate our informal reduction. The instance of SAT, $\{\{u_1, u_2\}, \{\bar{u}_2, \bar{u}_3\}, \{u_3, \bar{u}_1\}\}$ is satisfiable. We illustrate our construction for this instance in Fig. 3 with a satisfying assignment $u_2 =$ False and $u_1 = u_3 =$ True.

The instance $\{\{\bar{u}_1, u_2\}, \{u_1, \bar{u}_2\}, \{u_1, u_2\}, \{\bar{u}_1, \bar{u}_2\}\}$ is not satisfiable and the problem instance generated from it is depicted in Fig. 4 where we show all configurations of "true" and "false" values for u_1 and u_2. As we traverse the figures from left to right, none have edge disjoint shortest paths for clauses C_4, C_3, C_2 and C_1, respectively.

Formally, the construction places for each variable x_i a pair at the vertices $\{(4i - 3, 0), (4i - 1, n_1 - 1)\}$. For the jth clause there are two pairs $\{(0, 3j - 2), (n_0 - 1, 3j)\}$ and $\{(0, 3j), (n_0 - 1, 3j - 2)\}$ representing the "c_j"and "d_j" pairs, respectively. Of course

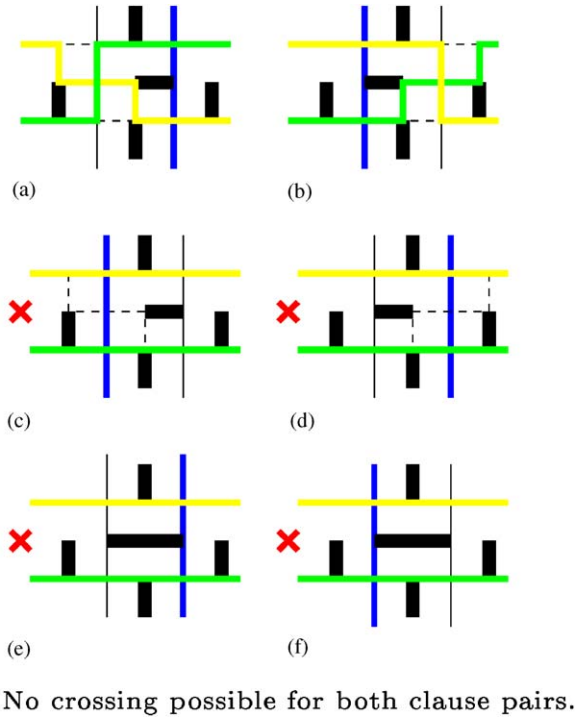

Fig. 2. (a) and (b) Crossing of clause pairs are possible because the value assigned to the variable satisfies the clause. (c) and (d) Crossing of clause pairs is impossible because the value assigned to the variable does not satisfy the clause. (e) and (f) Crossing of clause pairs is impossible because the variable is not in the clause.

n_1 is a function of w, the number of clauses, $n_1 = 3w + 1$ and n_0 is a function of v, the number of variables v, $n_0 = 4v$.

The additional blocking pairs are defined as follows: The vertical pairs between the variable pairs x_i and x_{i+1} are located at $\{(4i, 3j + 2), (4i, 3j + 3)\}$, for $0 \leqslant j < w$. The vertical pairs inside the shortest path region for each variable x_i are located at $\{(4i - 2, 3j), (4i - 2, 3j + 1)\}$, for $0 \leqslant j \leqslant w$.

We assume without loss of generality that none of the clauses contain a variable and its complement, as otherwise we could just delete the clause without changing the satisfiability of the clauses. Let $h_1 < h_2 < \cdots < h_r$ be the variables that clause j contains. We define $d_1, d_2, \ldots, d_r \in \{0, 1\}$ as follows: If the variable u_{h_k}, appears negated in the clause then $d_k = 0$, otherwise $d_k = 1$. Now we define the blocking pairs for clause j as follows:

$$\{(4h_1 + d_1 - 3, 3j - 1), (4h_1 + d_1 - 2, 3j - 1)\},$$
$$\{(4h_2 + d_2 - 3, 3j - 1), (4h_2 + d_2 - 2, 3j - 1)\}, \ldots,$$
$$\{(4h_r + d_r - 3, 3j - 1), (4h_r + d_r - 2, 3j - 1)\}.$$

All other variables, u_ℓ, that are not contained in the clause C_j have a blocking pair of the form $\{(4\ell - 3, 3j - 1), (4\ell - 1, 3j - 1)\}$.

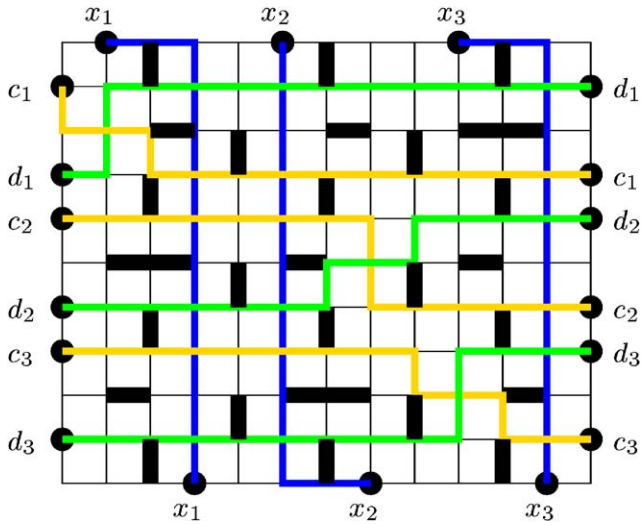

Fig. 3. Problem instance generated from $\{\{u_1, u_2\}, \{\bar{u}_2, \bar{u}_3\}, \{u_3, \bar{u}_1\}\}$. The solution shown corresponds to the truth assignment $u_2 =$ False, and $u_1 = u_3 =$ True.

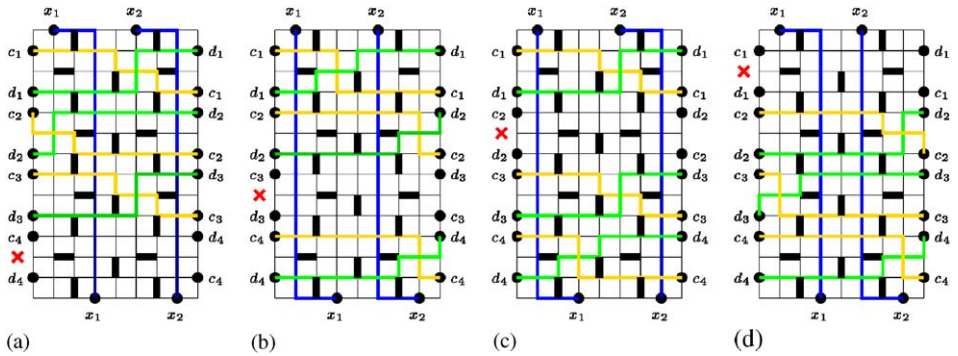

Fig. 4. Problem instance generated from $\{\{\bar{u}_1, u_2\}, \{u_1, \bar{u}_2\}, \{u_1, u_2\}, \{\bar{u}_1, \bar{u}_2\}\}$ has no solution. A "×" symbol indicates that the clause pairs do not have shortest paths. (a) $u_1 =$ True and $u_2 =$ True. (b) $u_1 =$ False and $u_2 =$ False. (c) $u_1 =$ False and $u_2 =$ True. (d) $u_1 =$ True and $u_2 =$ False.

In Theorem 1 we show that the k-pairwise edge disjoint shortest paths problem for the grid (k-PAIRWISE-GRID-EDSP) is NP-hard. We cannot establish that the problem is NP-complete because it is not clear whether the k-pairwise edge disjoint shortest paths decision problem is in NP. The difficulty in proving the problem is in NP is that it may be that for a grid of size n_0 by n_1 the number of bends for a path may be as large as $\min(n_0, n_1)$. But the input length is $k \log(n_0 + n_1)$, where k is the number of pairs. Therefore, the amount of information needed to represent the path might not be polynomial with respect to the input length.

Theorem 1. *Given the partial half routing request, the k-pairwise edge disjoint shortest paths problem for the grid* (*k-PAIRWISE-GRID-EDSP*) *is NP-hard.*

Proof. We use the reduction from SAT given above. We now show that the instance constructed from SAT has edge disjoint shortest paths for all pairs if and only if the instance of SAT is satisfiable.

First we connect all blocking pairs by their only shortest path. Now, given any assignment that satisfies all the clauses, we just connect each pair x_i by the path that corresponds to the truth assignment for the corresponding variable. Since the satisfying assignment is such that every clause is satisfied, then we know that edge disjoint shortest paths exist for each of the two pairs for each clause. The paths will cross as indicated in Fig. 2. Therefore, edge disjoint shortest paths exist for all pairs.

On the other hand, if edge disjoint shortest paths exist for the instance of the *k*-pairwise edge disjoint shortest paths problem that we defined, we need to show that the instance of SAT that we started from is satisfiable. Consider first the path for the ith variable x_i. Since the path is a shortest path it must be inside the shortest path region for the variable. If it starts by moving to the right, then it cannot move down on the next column because there is a blocking pair that uses that edge. So it must move to the right to the next column and then down. This corresponds to assigning the value of true to the corresponding variable. If the path starts by moving down first, then it must continue moving down until row $n_1 - 1$ and then continues to the right. This corresponds to the variable having the value of false. The reason why the path must continue moving down is that in the shortest path region for pair x_i between the intersection regions for adjacent clauses the path for x_i must be on the left side or the right side of the shortest path region for pair x_i because of the vertical blocking pairs inside the region. So if the path is not continually moving down, then inside an intersection region with a clause it must move to the right side. This is not possible because two rows must be used by the paths for the c_j and d_j pairs and one row for the path for the x pair. There are three rows, but at least part of the middle row is blocked. Therefore, the path for x_i must continue moving down until row $n_1 - 1$. Now lets consider the paths for any pair c_i and d_i. Since these are connected by edge disjoint paths then they must cross at the neighborhood of the intersection of a variable and the ith clause. By construction, it is easy to show that the crossing is not possible in regions that represent variables that are not in the clause. In addition, by construction one can easily show that such crossing is only possible at the neighborhood region where a variable that has the value true appears in the clause, or the variable has the value of false and it appears complemented in the clause. Therefore, the existence of edge disjoint shortest paths implies the satisfiability of the instance of SAT we started with.

Furthermore, the reduction is polynomial as the number of pairs introduced is $O(vw)$. The representation of pairs for the grid requires about $vw \log(v + w)$ bits, and thus the reduction is polynomial. This completes the proof of the theorem. □

When $d(X_i) \leqslant 2$ the algorithm by Gonzalez and Serena [5,6] for the corresponding problem in the *n*-cube works also for this problem simply because there are only two possible paths for each pair. This problem is solved by reducing it to the 2SAT problem. It is well known that 2SAT is polynomially solvable [2,17]. We have also developed a polynomial time algorithm for the case when $d(X_i) \leqslant 3$. This algorithm is tedious, so for brevity we do

not include it. In the next section we discuss the *k*-pairwise edge disjoint problem when we restrict paths to have a constant number of bends.

2.2. Restricting the number of path bends

The algorithm in the previous section can be easily adapted for arbitrary values of $d(X_i)$ as long as the only valid paths have just one bend, i.e., the path consists of a vertical line followed by a horizontal one or vice versa. The reduction to 2SAT is quite simple. A variable is used to indicate one of the possible two paths and two-literal clauses are added to avoid possible pairs of paths that overlap. We state our result in Theorem 2 without discussing further details.

Theorem 2. *Given the partial half routing request, the k-pairwise edge disjoint shortest paths problem with at most one bend allowed per path is polynomially solvable for a partial half permutation routing request in the grid.*

Proof. The problem is reduced to 2SAT (see the comment just before this theorem) which in turn can be solved in polynomial time with respect to the number of pairs. □

In the previous subsection we have established NP-hardness for the *k* pair edge disjoint shortest paths problem in the grid without a restriction on the number of bends or turns. A careful examination of the reduction for Theorem 1 reveals that when there is a solution every pair is connected by a path with at most four bends. Given any set of paths, each with at most four bends, once can easily verify in polynomial time whether or not the paths are shortest, and are edge disjoint. Therefore, this restricted version of the problem is in NP. The following corollary is a consequence of the previous theorem and the above comments.

Corollary 1. *Given the partial half routing request, the k-pairwise edge disjoint shortest paths problem for the grid* (*k-PAIRWISE-GRID-EDSP-3TURNS*) *when every path is restricted to have at most four bends is an NP-complete problem.*

None of the previous theorems cover the case when one limits each path to have at most two turns. We now show that even this problem is an NP-complete problem. To prove this result we polynomially reduce 3SAT to the edge disjoint shortest paths problem in the grid when every path is allowed at most two bends. We start by discussing the transformation details which are similar in nature to the ones in the previous section. The setting components are similar. The main difference is that in the reduction we use 3SAT rather than SAT, and the clause checkers are different. We use 3SAT in order to simplify our clause checker structure. The clause checker pairs need to be changed because half of the clause pairs in the previous reduction required paths with four bends. From an instance I of the 3SAT problem, we construct an instance $f(I)$ of the *k*-pairwise edge disjoint shortest path problem as follows. First we give a general idea about our reduction and give an example. Then we formally define the reduction and prove our result.

As in the construction given in the previous section, there are two types of components: *setting* and *clause-checker*. The *setting* component assigns to each variable a boolean value. For each variable u_i there is a pair indicated by the x_i's in Fig. 5. We will also introduce

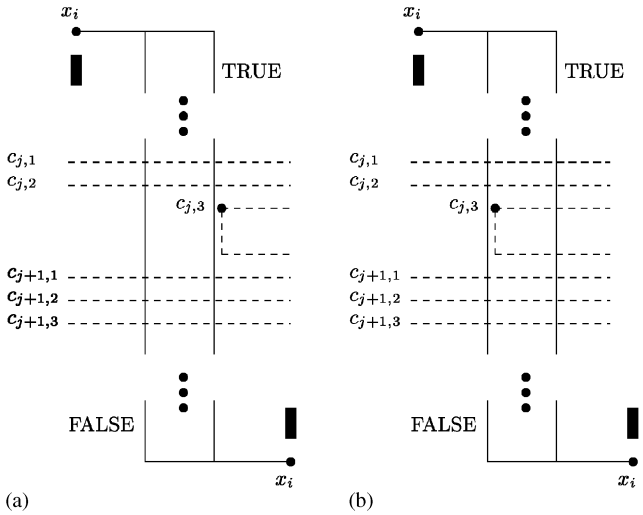

Fig. 5. Placement of the leftmost point in the $c_{j,3}$ pair. (a) When $\bar{u}_i$ is the third variable in clause C_j. (b) When u_i is the third variable in clause C_j. The thick lines represent the only possible path for blocking pairs.

some *blocking* pairs (which are represented by thick lines in Fig. 5 joining the pair by the only possible shortest path for the pair) that will guarantee that all the x_i's pairs can only be connected by paths that move to the right, then down, and then again to the right. We say that the path whose vertical segment is on the left side represents the value of false and the other one represents the value of true (Fig. 5). For each clause we introduce three pairs, one for each literal. The pairs for the jth clause are denoted by: $c_{j,1}$, $c_{j,2}$, and $c_{j,3}$. Notice the difference from the previous reduction. In the previous reduction we had two pairs per clause, where as now we have one pair per literal in a clause. Each of these pairs will have two different possible rows for the path. If a variable is the third literal of the jth clause, then the pair originates as indicated in Fig. 5. This implies that if the value assigned to the variable does not satisfy the clause, then the path for the corresponding clause pair must be on the upper row for the pair, where as if the value assigned to the variable satisfies the clause then the path may be on the upper or lower row for the pair. The idea is that if a clause is satisfied by the values assigned to the variables, then at least one of the paths for the clause pairs will be on its lower row and the remaining paths will use their upper rows. On the other hand if the values do not satisfy the clauses, then the paths for the three pairs for the clauses must be on their upper rows, but our construction will not allow one of these paths to connect to its destination, as we shall see later on. Let us now discuss the other endpoints for the clause pairs.

Consider the jth clause. Fig. 6 shows the rightmost endpoints for the three pairs. The one unit thick black lines represent the only possible shortest paths for the blocking pairs that are used to prevent a connection without a bend near the neighborhood of a clause pair, or prevent connections from three pairs using the upper rows of the clause pairs. If one allows the former case, then a clause that is not satisfied can have its three pairs connected by shortest paths by just introducing a bend along one of the unused columns of

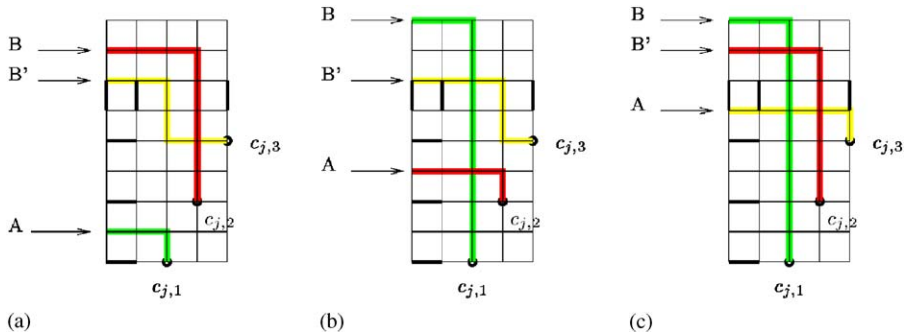

Fig. 6. Size three clause: the three valid rightmost configurations in the grid of $c_{j,1}$, $c_{j,2}$, and $c_{j,3}$.

other variable pairs. The latter case prevents connections when all the three paths use their upper rows which occurs when a clause is not satisfied. The leftmost diagram in Fig. 6 shows the connections when the clause is satisfied by the first variable, the middle diagram is for the case when the second literal satisfies the clause, and the rightmost is when the third literal satisfies the clause. Note that two or more literals may satisfy a clause. However, it is sufficient to show that connections are possible if only one pair uses its lower row.

Before we formally define our reduction it is important that we apply the reduction to the following instance of 3SAT: $\{\{u_1, u_2, u_3\}, \{\bar{u}_1, \bar{u}_2, u_4\}, \{\bar{u}_2, \bar{u}_3, \bar{u}_4\}, \{u_1, u_2, u_4\}\}$. The solution shown in Fig. 7 for the case when u_1 and u_4 have the value of true and the other variables the value of false.

Let us now formally define our reduction. The pair introduced for the ith variable, for $0 < i \leqslant w$, has its end points located at $\{(4i-4, 0), (4i-1, n_1-1)\}$. To insure variable consistency introduce two vertical blocking pairs (thick black lines in Fig. 5) for the ith variable $\{(4i-4, 1), (4i-4, 2)\}$ and $\{(4i-1, n_1-2), (4i-1, n_1-3)\}$.

If variable u_{h_k} is the kth variable in clause C_j then let the variable δ_k be 1; otherwise if the negation of the variable u_{h_k} is in the clause let the variable δ_k be 0. Each of the most three variables in the clause C_j are represented by pairs.

$$c_{j,k} \quad \text{pair is } \{(4h_k - 3 + \delta_k, 9j - 6 + k), (n_0 - 4 + k, 9j + 5 - 2k)\}$$

The additional blocking pairs are: $\{(n_0-5, 9j-3), (n_0-5, 9j-2)\}$, $\{(n_0-4, 9j-3), (n_0-4, 9j-2)\}$, $\{(n_0-1, 9j-3), (n_0-1, 9j-2)\}$, $\{(n_0-5, 9j-1), (n_0-4, 9j-1)\}$, $\{(n_0-5, 9j+1), (n_0-4, 9j+1)\}$, and $\{(n_0-5, 9j+3), (n_0-4, 9j+3)\}$.

We now establish the main result for this subsection.

Theorem 3. *The edge disjoint shortest paths problem for k pairs of vertices is NP-complete when every path is restricted to have no more than two turns or bends.*

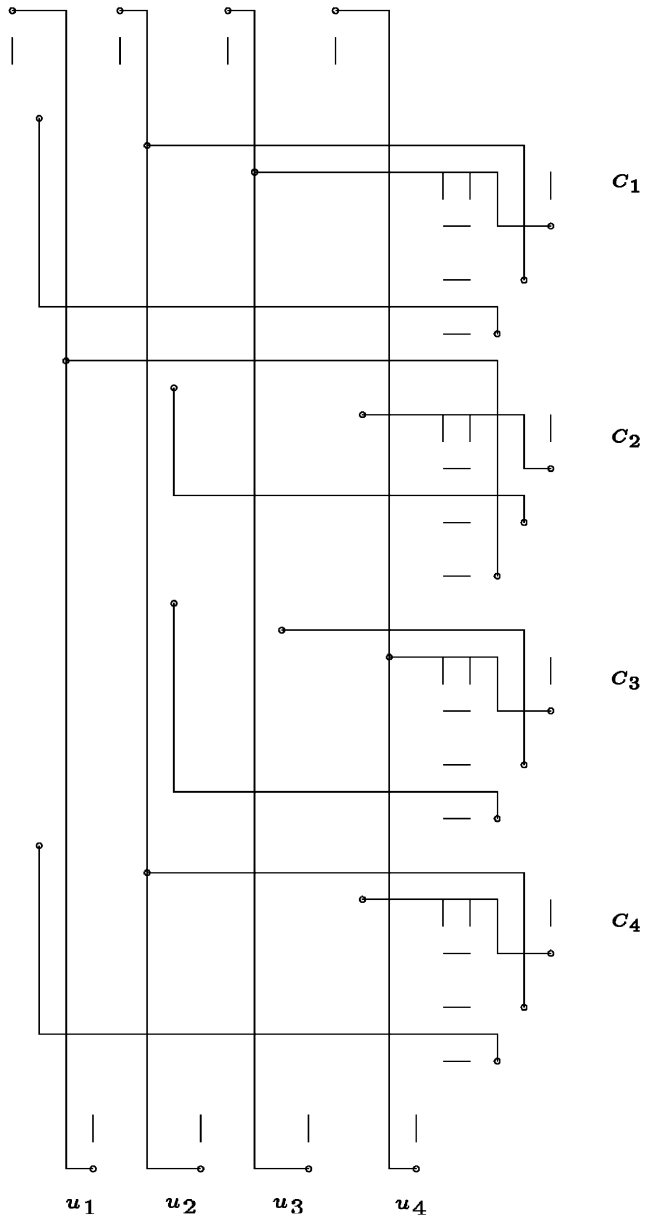

Fig. 7. Problem instance constructed from $\{\{u_1, u_2, u_3\}, \{\bar{u}_1, \bar{u}_2, u_4\}, \{\bar{u}_2, \bar{u}_3, \bar{u}_4\}, \{u_1, u_2, u_4\}\}$. The solution given is u_1 and u_4 with value of true and the remaining variables have the value of false.

Proof. Since there are at most three straight line segments per path, we can verify in polynomial time that any set of paths, each with at most two turns, whether they are pairwise disjoint shortest paths. Therefore, the problem is in NP.

We reduce 3SAT to our problem as shown above. We now show that the instance constructed from 3SAT has edge disjoint shortest paths for all pairs if and only if the instance of 3SAT that we started from is satisfiable.

First we connect all blocking pairs since they have exactly one shortest path. Now, given any assignment that satisfies all the clauses, we just connect each pair x_i by the path that corresponds to the truth assignment for the corresponding variable. Since the satisfying assignment is such that every clause is satisfied, then at least one of the three pairs for each clause can be connected through its lower row. Therefore, edge disjoint shortest paths exist for the three pairs for each clause as shown in Fig. 6. Hence, edge disjoint paths exist for all pairs.

On the other hand, if edge disjoint shortest paths exist for the instance of the *k*-pairwise edge disjoint shortest paths problem that we defined, we need to show that the instance of 3SAT that we started from is satisfiable. Consider first the path for the ith variable x_i. Since the path is a shortest path with at most two turns, it must be of one of the two forms shown in Fig. 7. Each of these two possible paths represents the value of true or false as indicated in Fig. 5. Now, the three pairs for each clause must be such that at least one of them is connected by a path on its lower track, as otherwise the pairs cannot be connected by shortest paths with at most two bends, because if the three pairs are joined by paths using their upper tracks then the component given in Fig. 6 will not allow for their connection using shortest paths with at most two bends. This implies that at least one of the pairs for each clause is connected by a path that starts moving down (when viewing the path from left to right) which means that the corresponding variable satisfies the clause. Therefore, the existence of edge disjoint shortest paths implies the satisfiability of the instance of 3SAT we started with.

The reduction can be carried out in polynomial time as the number of pairs introduced is $O(vw)$. The representation of pairs in the grid requires about $vw \log(v+w)$ bits. Therefore, the reduction takes polynomial time and our problem is NP-complete. □

A slight modification to the above reduction, namely the introduction of blocking edges that force one additional turn on the left side of the path of each pair can be used to show that the problem, when we limit the number of turns or bends to at most three, remains NP-compete. A similar modification, but forcing the introduction of $t-2$ turns on the leftmost side of every path shows that the problem remains NP-complete when one limits the total turns allowed per path to any constant number *t*. This structure also shows that the problem of finding *arbitrary* length paths with a (given) constant number of bends is also an NP-complete problem. We should point out that Kramer and vanLeeuwen [12] have already established this result, namely that the node disjoint paths for arbitrary length paths in the grid is an NP-complete problem, they limit neither the number of bends nor the path length to the shortest one.

2.3. Restricting the pair distance

In this subsection we consider the restricted version of the problem when the distance between the endpoints of every pair is bounded by a constant, i.e., for all i, $d(X_i)$ is less or equal than some fixed constant. We show that even under this condition the edge disjoint shortest paths problem for *k*-pairs in the grid remains NP-complete. To establish this result

we make non-trivial modifications to the reduction given in Section 2.1. A closer inspection of that reduction reveals that the only pairs whose endpoints are not at a distance bounded by some fixed constant are the ones for the variables (x_i in Fig. 1), and the clause checkers (c_i and d_i in Fig. 3). Our approach is to replace each of these pairs by a set of pairs whose shortest paths have properties similar to the ones for the previous pairs, and the endpoints of each of the new pairs are at a distance bounded by some fixed constant.

Consider first a replacement for each x_i pair. As you recall from Section 2.1 (Fig. 1) each variable in SAT is represented by a pair and the connecting path specifies whether the variable has the value of true or false. The distance between the two endpoints is proportional to the number of clauses and therefore is not bounded by a constant. The idea is to introduce a number of pairs equal to the number of clauses. The paths used to connect all of these pairs need to be consistent and each pair will cover the area for one clause.

Figs. 8(a) and (b) give an example of our reduction for an instance of SAT with three clauses. The interpretation of the paths for the odd numbered clause regions is exactly as in Section 2.1. The interpretation for the even numbered clause regions is the opposite of the one for the odd ones. I.e., from top to bottom, a connecting path consisting of a vertical line segment and then a horizontal line segment represents the value of false, and a connecting path consisting of a horizontal line segment followed by a vertical one represents the value of true.

All the pairs introduced (Fig. 8) to replace one of the variable pairs in Fig. 1 have their endpoints at a distance bounded by a constant and all of them have consistent values. For example, if a variable pair is connected by a top to bottom path consisting of a vertical line segment followed by a horizontal one, then the next pair below it is connected by a path consisting of a vertical line segment followed by a horizontal line segment, and the same pattern keeps on repeating periodically until we reach the bottom of the grid. In order to see that this holds one must take into account the pairs which supplant the c_i and d_i clause pairs in the construction in Section 2.1. The pairs replacing the clause pairs in this construction are such that the upper and lower row for the clause will be used by the clause pairs. The middle line (row) for the clause will be used by some blocking pairs to be introduced later on. As we shall see later on, the pairs replacing the clause pairs in this construction enforce variable consistency by precluding paths other than the two configurations shown in Figs. 8(a) and (b).

When one defines the neighborhood region for the intersection of a variable and a clause one needs to introduce mirror images about a vertical axis for Figs. 1(a)–(c) for the even numbered clauses. I.e., for even numbered clauses Figs. 1(a) and (c) are swapped. In Fig. 9(a) we present the instance generated for the set of clauses $\{\{u_1, u_2\}, \{\bar{u}_2, \bar{u}_3\}, \{u_3, \bar{u}_1\}\}$. At this point ignore the clause pairs in the figure. We will discuss them later on. The intersection regions have the same property for flow in the grid as those in the Section 2.1. With the exception that the crossing information is propagated with constant distance clause pairs from left to right in the grid. The intersection given in Fig. 1(a) is used at the intersection region of clause 1 with the variable pairs associated with u_1 and u_2, and clause 3 with variable u_3. Now the negations of u_2 and u_3 in clause 2 appear as the mirror image of Fig. 1(c). In other words, while in Section 2.1 the segments appear as Fig. 1(c), here they appear in clause 2 as Fig. 1(a). Fig. 1(c) is used at the intersection region of clause 3 and variable u_3. Fig. 1(b) is used at the intersection region of clause 1 and variable u_3, clause 2

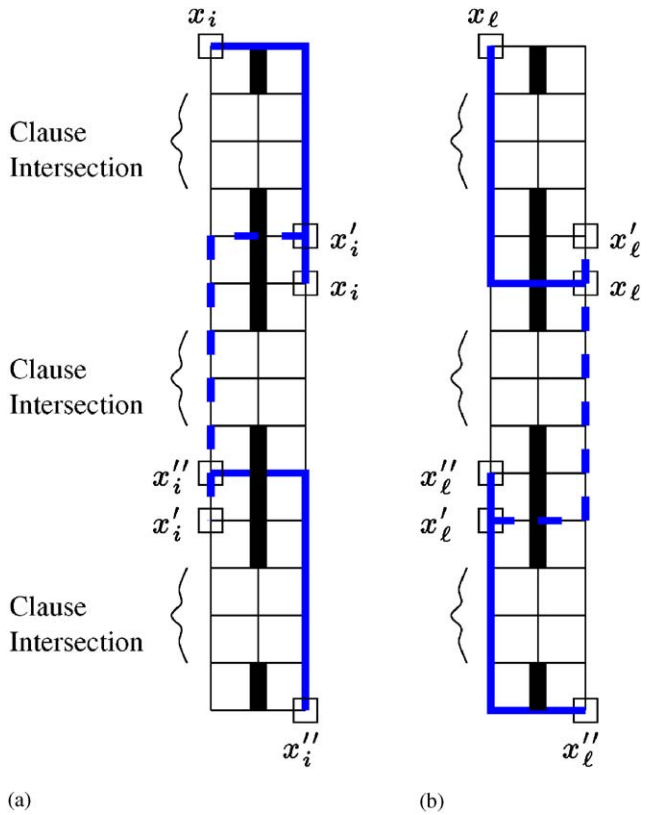

Fig. 8. Consistent variable state is communicated vertically throughout the grid with finite distance pairs. (a) True state of variable i. (b) False state of variable ℓ.

and variable u_1, and clause 3 and variable u_2. In Fig. 9(b) we give actual paths that correspond to the assignments $u_2 =$ False and $u_1 = u_3 =$ True. The paths correspond to the paths given in Fig. 2 for odd numbered clauses, and the mirror images of the same figure for the even numbered ones.

As you recall from Section 2.1 (see Fig. 1) the jth clause in SAT is represented by the pairs c_j and d_j. The distance between the endpoints in these two pairs is proportional to the number of variables. As in the case for the variables, the idea is to introduce a number of pairs equal to twice the number of variables. I.e., one c_j and d_j pair for each variable. The paths used to connect all of these pairs need to be consistent. Furthermore, each pair will cover the area for a variable. Figs. 10(a) and (b) give an example of how this works without any pair for the variables shown. The idea is that if at some point the value of a variable satisfies the clause then the crossing of the c_i and d_i paths will be made at the intersection region with the pairs for the variable. Fig. 11 shows the zoomed section for the clause 1, $\{u_1, u_2\}$, for the the example given in Fig. 9. From left to right if the value of a variable and all the previous variables do not satisfy a clause, then the crossing of the pair occurs

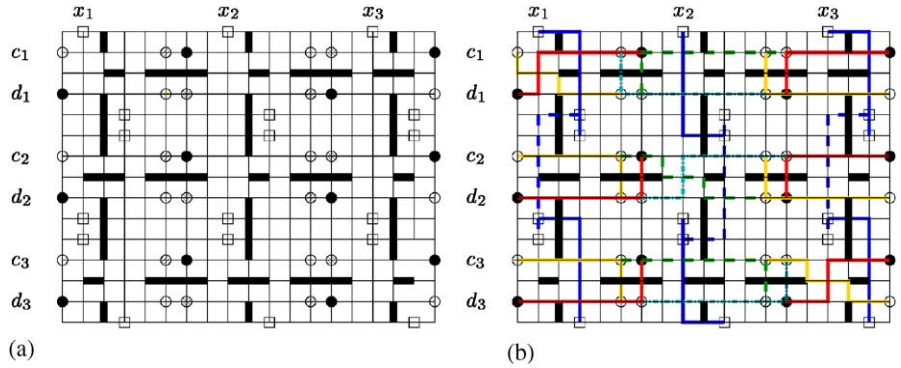

Fig. 9. Problem instance generated from $\{\{u_1, u_2\}, \{\bar{u}_2, \bar{u}_3\}, \{u_3, \bar{u}_1\}\}$. (a) Solid lines indicate the shortest path for pairs with a unique path (vertical or horizontal line), and the other symbols are the endpoints of clause and variable pairs. (b) Shows the solution for $u_2 =$ False and $u_1 = u_3 =$ True.

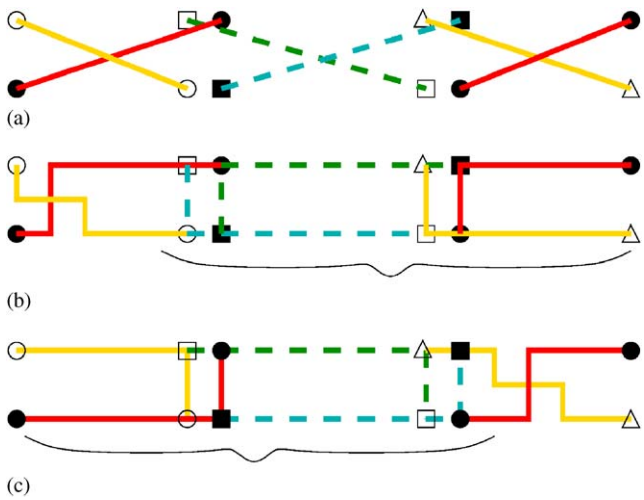

Fig. 10. (a) Shows clause pairs independent from (b) and (c). (b) Illustrates post-crossing region. (c) Illustrates pre-crossing region.

just after the intersection region, and the next two pairs start precisely as the first one did. Eventually, if none of the values of the variables satisfies a clause then there is no possible path for the last pair.

The instance $\{\{\bar{u}_1, u_2\}, \{u_1, \bar{u}_2\}, \{u_1, u_2\}, \{\bar{u}_1, \bar{u}_2\}\}$ is not satisfiable and the problem instance generated from it is depicted in Fig. 12 where we show all configurations of "true" and "false" values for u_1 and u_2. As we traverse the figures from left to right, none have edge disjoint shortest paths for clauses C_4, C_3, C_2 and C_1, respectively.

For brevity we will not formally specify the location of all the pairs. Interested readers may find the details in [6]. The pairs for the variables are denoted by $x_i^{(k)}$ and the

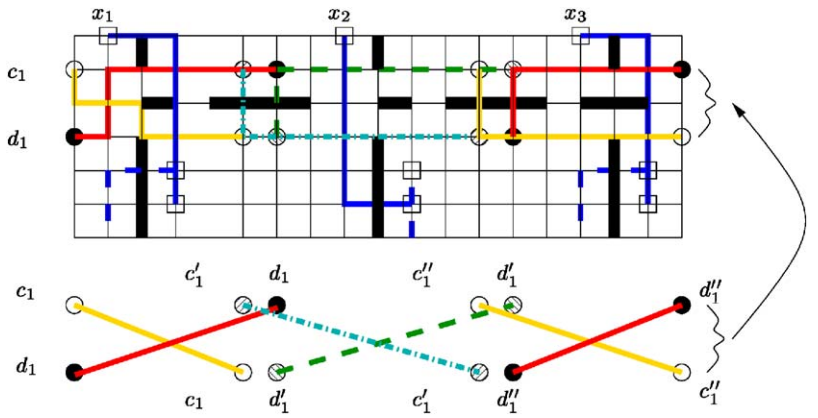

Fig. 11. Constant constraint on pair distance. Shows only clause $\{u_1, u_2\}$ when $u_2 =$ False and $u_1 =$ True. Shows labeling of the constant distance pairs below the grid.

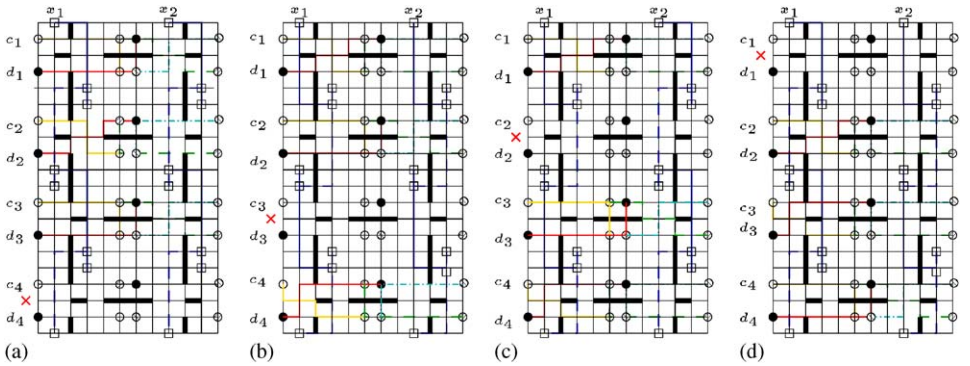

Fig. 12. Problem instance generated from $\{\{\bar{u}_1, u_2\}, \{u_1, \bar{u}_2\}, \{u_1, u_2\}, \{\bar{u}_1, \bar{u}_2\}\}$ has no solution. A " $\times$" symbol indicates that the clause pairs cannot be connected via shortest paths. (a) $u_1 =$ True and $u_2 =$ True. (b) $u_1 =$ False and $u_2 =$ False. (c) $u_1 =$ False and $u_2 =$ True. (d) $u_1 =$ True and $u_2 =$ False.

ones for the clause pairs by $c_j^{(k)}$ and $d_j^{(k)}$. Let us now state the main result in this subsection.

Theorem 4. *Given a partial half routing request on the grid, when pair distances are constrained by a constant, the k-pairwise edge disjoint shortest paths problem* (*k-PAIRWISE-GRID-CEDSP*) *is NP-complete.*

Proof. As before v and w are the number of variables and clauses, respectively. Given a set of paths for all the pairs, one can easily check to see if each path is shortest path (number of edges must be equal to the pair distance) and check that all of the edges in the paths are different. Since all pairs have at most of distance c, then each path has no more than

c edges and all the above checks can be performed in polynomial time with respect to the input length ($O(vw \log_2(vw))$). Therefore, the problem is in NP.

Now to show NP-completeness we use the reduction from SAT given above. We now show that the instance constructed from SAT has edge disjoint shortest paths for all pairs if and only if the instance of SAT is satisfiable.

Given any assignment that satisfies all the clauses, we just construct the corresponding paths for pairs $x_i^{(k)}$ and the paths for all the $c_j^{(k)}$ and $d_j^{(k)}$ paths exist because each clause is satisfied. If a variable in a clause is in the proper state a crossing of the c_j and d_j occurs at an intersection region, then that is propagated by pairs $c_j^{(\ell)}$ and $d_j^{(\ell)}$, $\ell > k$ to the rightmost extent of the grid (Fig. 10). As in the previous proof, if the instance we constructed from an instance of SAT has a solution, then the instance of SAT we started from is satisfiable. The proof for this is based on the result in the previous section and the properties of the construction given above.

Furthermore, the reduction is polynomial as the number of pairs introduced is $O(vw)$. The representation of pairs in the grid requires about $vw \log(v+w)$ bits. Thus the reduction takes polynomial time, and the problem is NP-complete. □

3. Complexity of grid node disjoint shortest path problems

We show that the k-pairwise node disjoint shortest paths problem in the grid is an NP-hard problem by reducing SAT to this problem. Our reduction is similar in nature to the one in Section 2.1, but it is slightly more complex. The main difference is that the crossing of the shortest paths is not allowed because they must be node disjoint. Edge disjoint paths may cross through a given node; however, node disjoint paths may not. One might conjecture that this topological constraint implies that paths for pairs should be decided solely on local information. However, this is not the case because there are problem instances in which the direction of a path in one part of the grid forces changes in certain types of path in another remote part of the grid. This communication of routing decisions throughout the graph many times is qualitatively indicative of NP-hardness. In Section 3.2 we show that the problem remains computationally intractable (NP-complete) even when every pair distance is bounded by a (small) constant.

3.1. Node disjoint shortest paths

Instead of representing a variable by a pair and the value of the variable by the direction of the path connecting the pair, we use a sequence of pairs. These pairs are designed in such a way that all of them are connected by paths that follow a certain pattern (like the ones in Section 2.3, see Fig. 8). In Fig. 13 we depict the general architecture of our construction. The chains $x_1, x_2, x_3, x_4, \ldots$ are used to represent the value of the variables. The paths for x_1, x_2 and x_4 indicate the value of true for the variable, and the one for x_3 indicates the value of false. As we show later on, these values are propagated consistently across the grid. On the top portion we introduce one pair for the first clause such that it has a shortest path if and only if the first clause is satisfied. This is called a *clause-checker* pair. Then in the next region below we introduce another pair for the second clause which has similar properties

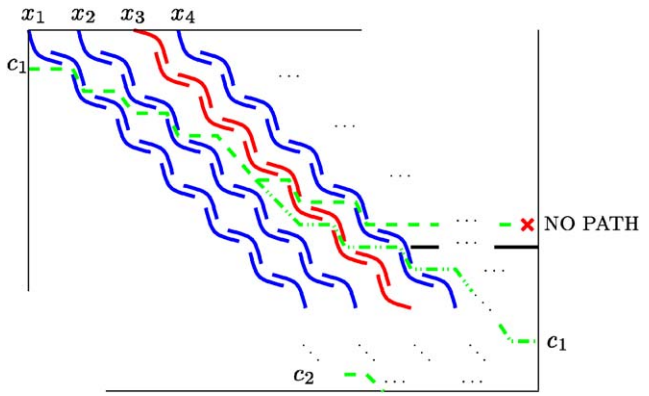

Fig. 13. Overall architecture: diagonal pairs are variable states. Dashed lines show possible paths. The upper path cannot connect (a "×" indicates no connection is possible). The lower path can connect.

(second clause checker), and so on. These pairs are spaced sufficiently apart so that clause checking pairs do not overlap with each other. This guarantees that the clause checking pairs interact only with the horizontal and vertical blocking pairs that we introduce later on and with the variable state information (i.e. the chains for x_1, x_2, x_3 and x_4).

Once that the direction of the paths for the variables has been decided (which means that the variables have been assigned boolean values), shortest paths for the clause-checker pairs may or may not exist. if they exist then they will be of the form given in Fig. 13 for the clause checker pair c_1. The form of the path is such that if the variables do not satisfy a clause, every possible path for c_1 will advance to the other side of the chains representing the variables, but its vertical distance will only drop by a value equal to 12 times the number of variables. On the other hand, when at least one variable has a value that satisfies the clause, then the path will escape vertically to the other side of the chains for a distance equal to 12 times the number of variables plus a small constant, and then it will be able to connect to the other c_1 point. The scenario when there is a path for the clause pair is shown with the dashed and dotted line in Fig. 13. If none of the values for the variables satisfy the clause, then every path for pair c_1 ends up at a region where it is impossible to reach the other endpoint of c_1, because there is a blocking pair whose only path will not allow it to move lower. This clause path which cannot connect to the other endpoint is marked by "×" and is represented by dashed lines in Fig. 13. For brevity we will not specify the actual positions of all the pairs in the reduction. Interested readers are referred to [6] for additional details. However, we will discuss additional details of the reduction in what follows.

From left to right in the grid, as the clause path crosses a chain for a variable that is not in the clause, the path will move downward in the vertical direction by at most 12 units per each variable. The same holds when a variable or its complement does not satisfy the clause. However, if the value of the variable or its negation satisfies the clause, then either the vertical movement may be more than 12 units or the path may only drop the usual twelve vertical units. Let us now explain the general implementation framework and then give additional details.

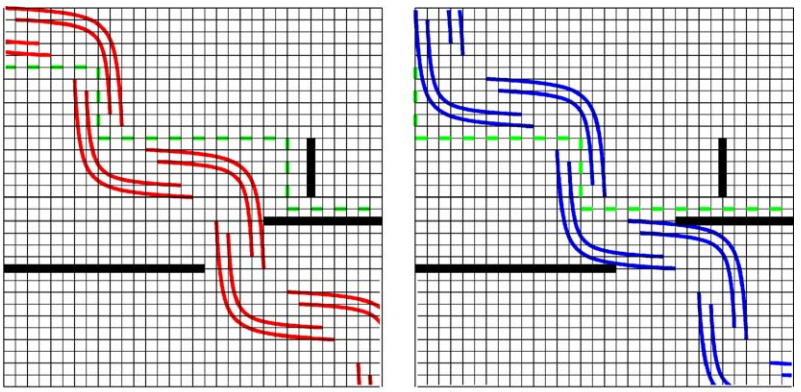

Fig. 14. Variable u_i or its complement is not in clause C_j. (a) Variable u_i has the value of false. (b) Variable u_i has the value true.

Fig. 14 shows the case when a variable or its complement is not in the clause. The only possible path for the additional blocking pairs is indicated with thick black lines. The dashed lines indicate one or more of the possible clause paths. Fig. 16 depicts the case when the variable's negation is in the clause. Notice that this figure is different from Fig. 14. Fig. 16 has five columns between the two endpoints of the horizontal blocking pairs, but Fig. 16 has six. The only possible path for the additional blocking pairs are indicated with solid bold lines. The dashed lines indicate one or more of the possible paths. The case when the uncomplemented variable is in the clause is given in Figs. 17 and 18. Figs. 14, 16–18 actually admit more paths than those that are shown with the dashed lines; however the exit location is invariant. To facilitate the exposition both the "true" and "false" variable states for variable x_i are shown by curved lines. These also are indicative of a collection of possible paths.

In Figs. 14, 16–18 the region between the two horizontal thick lines (the only paths for the blocking pairs) is called the *passage to the lower region*. If a path begins on the left side of the region above the left horizontal thick line and ends below the right horizontal line on the right side of the region it will exceed the 12 vertical unit step for this variable. This is only allowed if the variable pairs are in the appropriate state. In Figs. 14(a) and (b) the clause path cannot go through the passage to the lower region. The main reason for this is that the paths, for the four pairs with endpoints on opposite sides of the passage to the lower region, do not leave enough space for a clause pair to move to the lower region. This is illustrated in more detail in Figs. 15(a) and (b). In Figs. 14, 16–18 a clause path will enter five units below the top-left corner or below the left horizontal thick line. The path will leave on the right side one unit above the right horizontal thick line or below it. When either the current variable value or one of the previous ones satisfies the clause, then the path may leave above or below the right horizontal thick line. On the other hand if the values assigned to these variables do not satisfy the clause, then path can only leave above the right horizontal thick line.

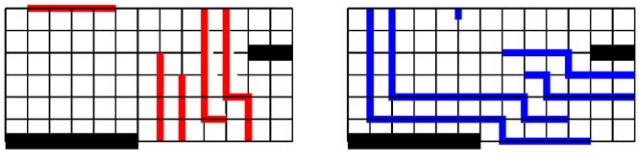

Fig. 15. Specific paths when variable or its complement u_i is not in clause C_j. (a) Variable u_i has the value false. (b) Variable u_i has the value true.

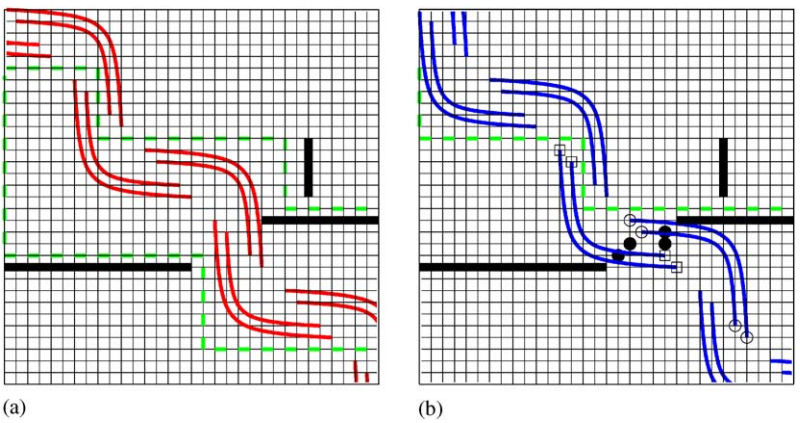

Fig. 16. Negation of variable u_i is in clause C_j. (a) Variable u_i has the value false. (b) Variable u_i has the value true.

Fig. 16 shows our construction when the negation of a variable is in clause C_j (notice that the number of columns in the passage to the lower region is more than the ones in Fig. 14). In Fig. 16(a) shows two possible configurations for the clause path. One of these paths goes through the passage to the lower region. This is possible because the value of the variable satisfies the clause. The passage region is shown in more detail in Fig. 19(a). The configuration in Fig. 16(b) does not allow the clause path to go through the passage to the lower region because the value of the variable does not satisfy the clause. This blockage is illustrated in Fig. 20(a).

The reason for this blockage is simple (Fig. 16(b)). In the "true" state for the variable the paths associated with the pairs labeled with a square must go below paths for pairs labeled with "$\circ$." It is simple to see that every path that goes through the passage to the lower region must use one of the four solid "$\bullet$" in Fig. 16(b). Since the paths for the two pairs labeled with "$\circ$." and the two paths labeled with a square must go through the passage, it then follows that no other path may use the passage.

Fig. 17 gives the construction when the variable u_i is in the clause and it has the value false. In this case the clause path must exit to the right at the level indicated in the figure. The passage to the lower region is depicted in more detail in Fig. 20(b) where one can see

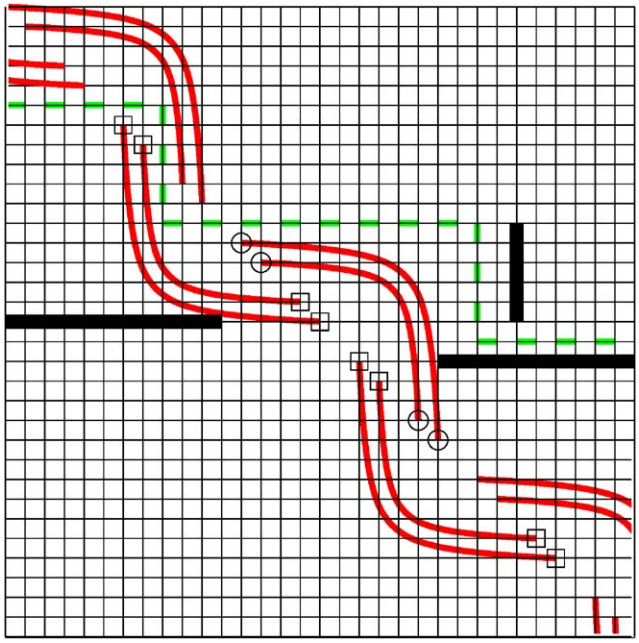

Fig. 17. Variable u_i has the value false and it is in the clause C_j.

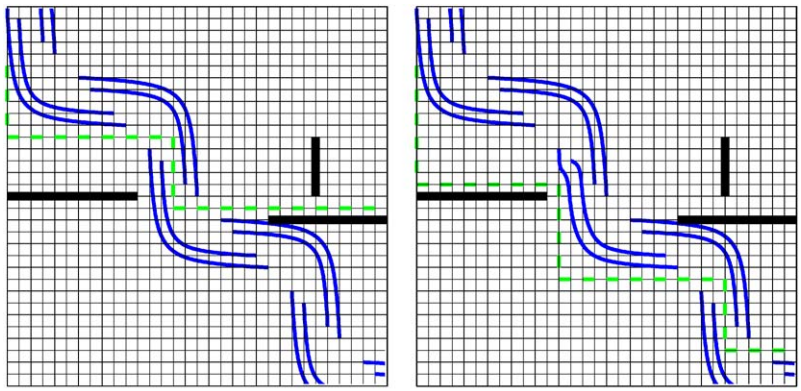

Fig. 18. Variable u_i is in a clause C_j. Variable u_i has the value true and the path for clause C_j exits through the (a) upper portion or (b) lower portion.

that the passage to the lower region is blocked by the paths for the variable pairs. In this case one can prove that clause path must exit in the upper region.

Fig. 18 shows two possible paths when the variable u_i is in clause C_j and has the value true. In Fig. 18(a) the path exists through the upper region and in Fig. 18(b) it exits through

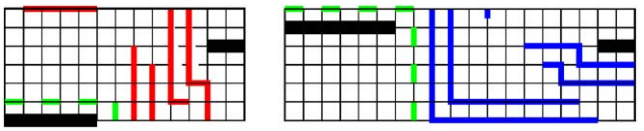

Fig. 19. Clause path (dashed line) passage to the lower region. (a) Variable $\bar{u}_i$ is in clause C_j and has the value false. (b) Variable u_i is in clause C_j and has the value true.

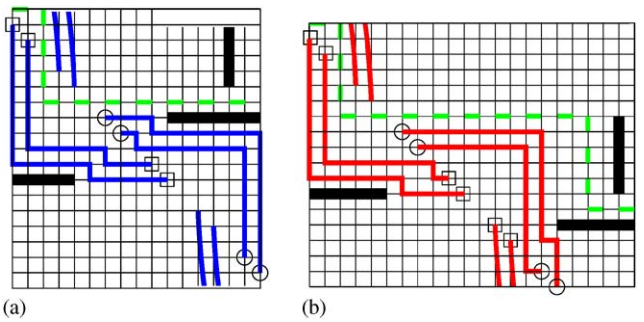

Fig. 20. (a) Specific path when negation of variable u_i is in C_j and variable has the value true. (b) Specific path when variable u_i is in C_j and the variable has the value false.

the bottom region. Fig. 19(b) shows the passage to the lower region in more detail. As in the previous cases one can prove that the clause path may advance to the lower region via the passage only if a variable has the value that satisfies the clause. For brevity we will not prove our result. However, in the following two examples we give additional details needed to assemble all the components in our reduction.

Fig. 21 shows the segment of clause $C_1 = \{\bar{u}_1, u_2\}$. The solution is shown for $u_1 = u_3 =$ False and $u_2 =$ True. The three larger squares, which we now call *l-squares*, correspond to the intersection of a variable with the clause. The top-left corner of the ith l-square is 16 units to the right and 12 units below the top-right corner of the $i-1$st l-square. Notice the dashed–dotted line joining the horizontal blocking pairs of adjacent l-squares. These paths are introduced to guarantee that if a path leaves the ith l-square above the right horizontal blocking pair, then it will enter the $i-1$st l-square above its left horizontal blocking pair. This also guarantees that if a path leaves below the right horizontal blocking pair, then it will enter below the left horizontal blocking pair of the next l-square. These paths can be realized by either introducing a new pair or by combining the blocking pairs of adjacent l-squares into one pair. Now, the first l-square corresponds to variable $\bar{u}_1$ belonging to the clause. The blocking pairs are the ones given in Figs. 16(a) and (b), the clause path is given by Fig. 16(a), and the actual passage to the lower region is given by Fig. 19(a). The second l-square corresponds to the variable u_2 being a literal of the clause. Figs. 17 and 18(a) and (b) show the blocking pairs for this l-square, the clause path is given by Figs. 18(a) and (b), and the actual passage to the lower region is given by Fig. 19(b), except if the path entered from below the left blocking pair for the l-square. The third l-square corresponds to the third

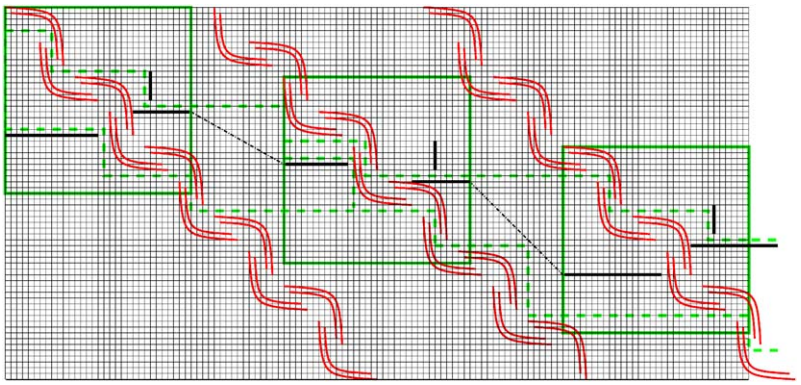

Fig. 21. Segment of clause $C_1 = \{\bar{u}_1, u_2\}$. The solution is shown for $u_1 = u_3 =$ False and $u_2 =$ True.

variable not being part of the clause. The blocking pairs are the ones given in Figs. 14(a) and (b), the clause path is given by Fig. 14(a), and the actual passage to the lower region is given by Fig. 15(a).

Fig. 22 shows the same clause C_1 when the values of the variables are not set correctly ($u_2 = u_3 =$ False and $u_1 =$ True). Note there is no way for the clause path to pass to the lower region and thus there is no solution for this variable state configuration. The first l-square corresponds to variable $\bar{u}_1$ belonging to the clause. The blocking pairs are the ones given in Figs. 16(a) and (b), the clause path is given by Fig. 16(b), and the actual passage to the lower region is given by Fig. 20(a). The second l-square corresponds to the variable u_2 being a literal of the clause. Figs. 17 and 18(a) and (b) show the blocking pairs for this l-square, the clause path is given by Fig. 17, and the actual passage to the lower region is given by Fig. 20(b). The third l-square corresponds to the third variable not being part of the clause. The blocking pairs are the ones given in Figs. 14(a) and (b), the clause path is given by Fig. 14(a), and the actual passage to the lower region is given by Fig. 15(a).

The main reason why we have not been able to show that this problem is in NP is that the input size is $k \log(v + w)$, but there may be a pair with distance $v + w$ that has $v + w$ bends. The amount of space needed to represent the occupied vertices may be exponential with respect to the input length. So it seems that polynomial time verification may not be possible.

Theorem 5. *The partial half permutation routing request for the k-pairwise node disjoint shortest paths problem for the grid* (*k-PAIRWISE-GRID-NDSP*) *is NP-hard.*

Proof. To prove NP-hardness, we use the above reduction from SAT, where v is the number of variables and w is the number of clauses. The only way that a path may escape from the 12 vertical unit step function is if the variable or its negation is in the clause and the variable is in the proper state. The previous discussion demonstrates that the path for a given clause exists only if one of the paths for its variables allows it to do so. If a variable in the SAT clause has the appropriate value, then Figs. 18(b) and 16(a) and their attendant discussions

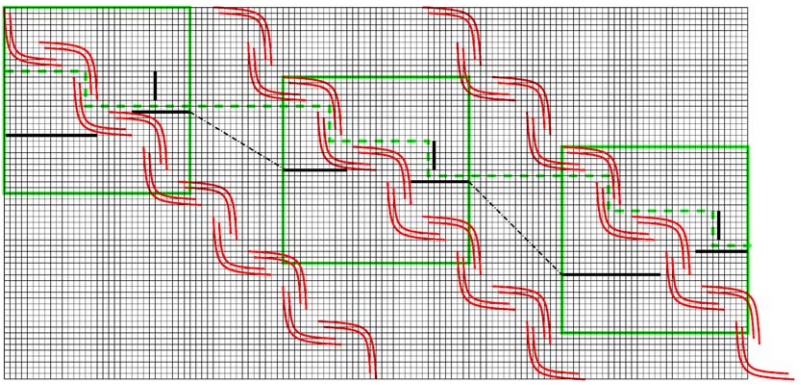

Fig. 22. Segment of clause $C_1 = \{\bar{u}_1, u_2\}$. There is no solution for $u_2 = u_3 =$ False and u_1 True. "×" indicates that no path exists below this point.

demonstrate that a clause path may pass to the lower region. However if no variable in the clause is the appropriate value then the clause path may not move below the level $12v$ below its starting point. This is observed merely because the path for traversal is blocked by the individual variable states and constructions described above.

Once the clause path has moved to the lower region, it still must move downward 12 vertical units for each variable state construction crossed horizontally. The path for the clause shown in Fig. 14 indicates that no more than 12 units are required for each individual variable crossed. Therefore, if any one construction allows the clause path to move into the lower region it may connect to the terminal point of the clause path at the far right of the grid.

Furthermore, the transformation is of polynomial time complexity as the number of pairs generated from an instance of SAT is $O(vw)$. Each pair is labeled by a node label of maximum length $O(vw)$. The length of the input to the *k*-pairwise shortest path problem in the grid is therefore polynomial in the input length. So the problem is NP-hard. □

3.2. Constant constraint on the pair distance

We show that when the problem outlined in Section 3.1 is such that pair distances are constrained by a sufficiently large constant the problem remains NP-hard. Since the pair distance is constrained we can easily prove that the problem is in NP. Hence the problem is NP-complete. Our result (Theorem 6) implies that the constant number of bends problem is also NP-complete. The reduction we use in this section is similar to the one in Section 3.1. The only parts we need to change are the clause checker pairs since they have a pair distance that is not bounded by a constant.

The clause checker in the previous Section 3.1 consists of a pair whose vertices are at a distance that is linearly dependent on the number of variables. We need to modify this so that the clause checker pair distance is bounded above by a small constant. Our modified architecture involves multiple pairs for each pair C_i. Note that C_1 and C_2 shown in Fig. 23

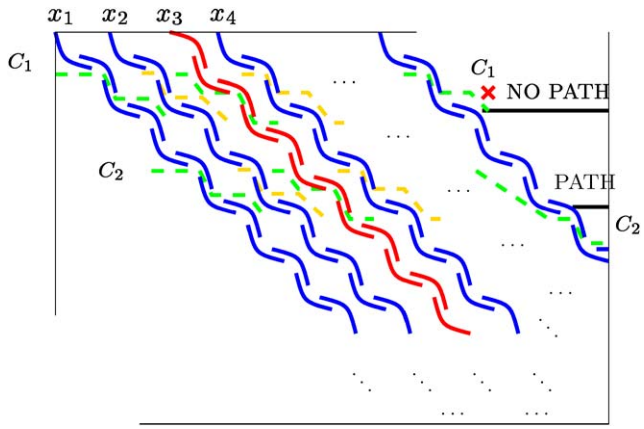

Fig. 23. Overall architecture: diagonal pairs are variable states. Dashed lines show possible paths. The upper path for c_1 pairs cannot connect ("×" indicates no connection is possible). The lower path composed of the c_2 pairs which can connect.

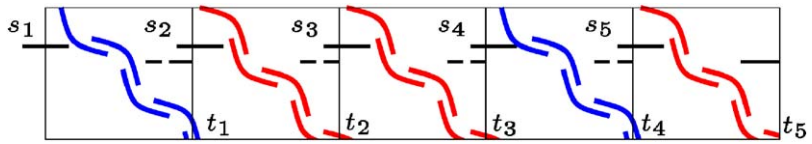

Fig. 24. Overall architecture: adjacent connecting cells for the multiple clause pairs.

are now multiple pairs whereas in Section 3.1 only one pair crossed the entire horizontal expanse of the grid. Fig. 24 shows an example of the placement of multiple pairs instead of just c_1 as in Section 3.1.

The way the pair (s_i, t_i) is routed affects the way pair (s_{i+1}, t_{i+1}) is routed. The area where each pair is routed is like the ones in the previous section in the sense that if the corresponding variable satisfies the given clause, then the pair is routed through a path that ends below the "passage to the lower region" as in the left-hand side of Fig. 25(a). On the other hand, if the corresponding variable does not satisfy the clause then the pair can only be routed through a path that ends above the passage to the lower region and it needs to use the gap to the right side in order to connect to its destination (see left-hand side of Fig. 25(b)). Henceforth we will call the gap on the right side of each component the "gap to the right". In the former case the next pair may use the gap to the right as passage to the lower region and start in the bottom of the component and then connect to its destination (see right-hand side of the Fig. 25(a)). In the latter case the next pair must be routed by a path going through the upper portion of the area. The gap to the right is occupied in this case. However, it may use the passage to the lower region in the next area (see right-hand side of Fig. 25(b)). The lower path shown here represents a variable or its negation in the proper state. The upper path represents a variable not in its proper state or a variable not in the clause. The upper path uses

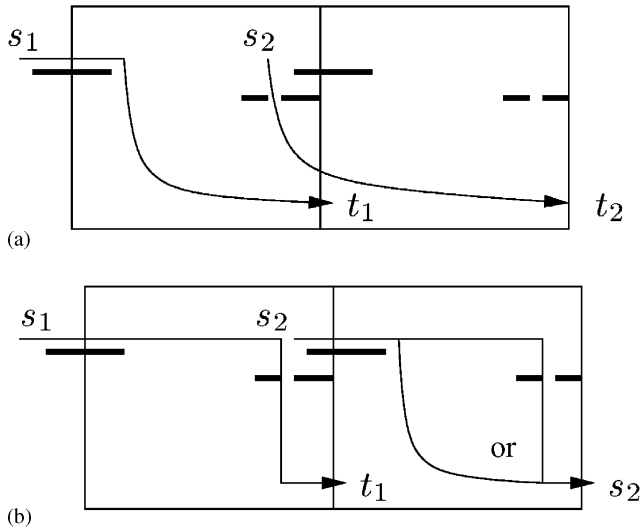

Fig. 25. (a) Clause pair (s_1, t_1) is routed by a path through the lower region and thus the clause path for (s_2, t_2) may move downward first. (b) No clause pair to the left has been routed to the lower region therefore the next clause path for (s_2, t_2) moves to the right first.

the gap to the right. The idea is that if a variable satisfies the clause, then the corresponding pair will be routed as in the left part of Fig. 25(a) and the same routing that appears on the right side of Fig. 25(a) will be possible for the path for the remaining pairs. When none of the variables satisfies the clause then the rightmost pair will not have a connecting path because there is no gap to the right in the rightmost pair ((s_5, t_5) in Fig. 24).

Fig. 26(a) illustrates the case where a variable u_i is not in the clause j and the variable u_i is in the "true" state. Similarly Fig. 26(b) indicates the case when a variable u_i is not in the clause and the variable is in the "false" state. In both cases if the gap is not used at the starting point of the path for the pair there will be no passage to the lower region as in the proof in Section 3.1 in particular Figs. 14, 18 and 16(b). Figs. 27(a) and (b) show the variable paths in the "false" and "true" states, respectively. The negation of the variable u_i is in the clause and if the variable is in the "false" state there will be a passage to the lower region (as indicated in Fig. 27(a)). However, as Fig. 27(b) indicates there will not be such a passage for the clause path when the variable paths are in the "true" state. Figs. 28(a) and (b) show the variable paths in the "false" and "true" states, respectively. The variable u_i is in the clause and if the variable is in the "true" state there will be a passage to the lower region (as indicated in Fig. 28(b)). However, as Fig. 28(a) indicates there will not be such a passage for the clause pair when the variable paths are in the "false" state.

Theorem 6. *The partial half permutation routing request with a constant constraint on pair distance for the k-pairwise node disjoint shortest paths problem for the grid* (*k-PAIRWISE-GRID-CNDSP*) *is NP-complete.*

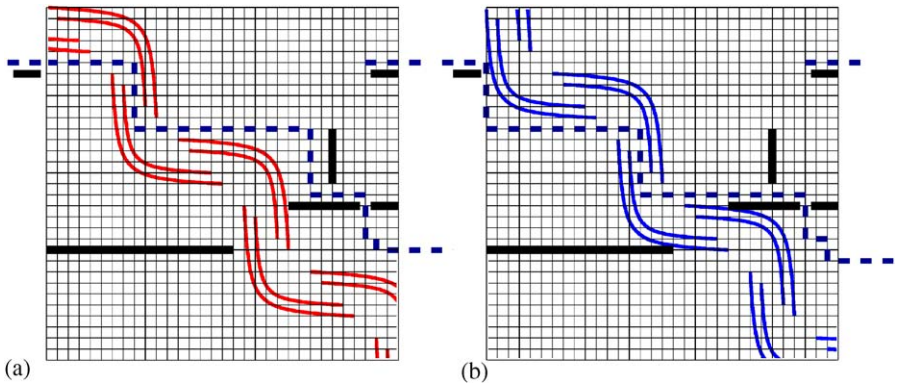

Fig. 26. (a) Variable u_i or its complement is not in clause C_j. Variable u_i has the value true. (b) Variable u_i or its complement is not in clause C_j. Variable u_i has the value false.

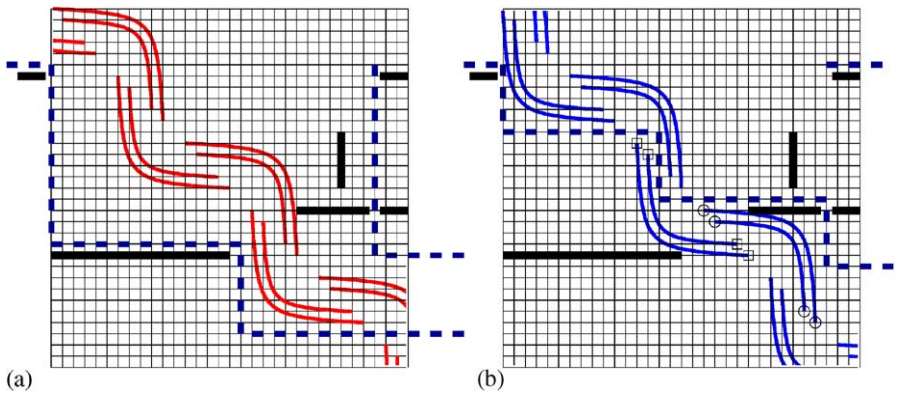

Fig. 27. Negation of variable u_i is in clause C_j. (a) Variable u_i has the value false. (b) Variable u_i has the value true.

Proof. As the pair distance is constrained by a constant, each path can be characterized by constant amount of information. So the given a set of paths checking pairwise disjointness can be performed in polynomial time. Therefore, the problem is in NP. The remaining part of the proof follows from the one in Theorem 5 after taking care of the modification discussed above. □

The constant constraint on the pair distance result (Theorem 6) also implies the following result.

Corollary 2. *The partial half permutation routing request with a constant on the number of bends for the k-pairwise node disjoint shortest paths problem for the grid* (*k-PAIRWISE-GRID-CBENDS-NDSP*) *is NP-complete.*

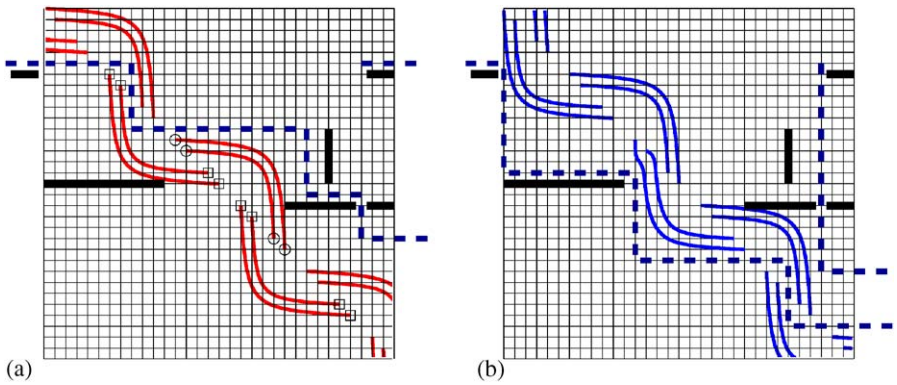

Fig. 28. Variable u_i is in a clause C_j. (a) Variable u_i has the value false. (b) u_i has the value true and the path for clause C_j and exists through the lower region.

4. Discussion

While not employed in as many high performance systems as the n-cube, the network architecture of the grid, and its higher dimensional generalizations, are important network topologies. The results stated in this paper for the grid are applicable to the higher dimensional generalizations of the grid, because we are considering shortest paths and the grid is embeddable in its higher dimensional generalizations. In Section 2.1 we show that the edge disjoint shortest paths in the grid is an intractable problem. Even with a constant constraint on the number of bends the problem remains NP-hard (Section 2.2). When there is a limit of one on the number of bends for each route the problem can be solved efficiently (Section 2.2, Theorem 2). When a constant constraint on pair distance is applied, the problem remains intractable (Section 2.3). While counter-intuitive, when doubling the grid topology and maintaining the partial half permutation routing request we find that the problem remains intractable [6]. When node disjoint shortest paths are considered, the basic topology of the grid continues to imply intractability for the partial half permutation routing request (Section 3.1) and further constant constraints or constraints on number of bends do little to eliminate the characteristics of intractability (Section 3.2). Similar results apply for the doubled grid; however, for brevity we do not include all of this work in this paper.

References

[1] A. Aggarwal, J. Kleinberg, D.P. Williamson, Node-disjoint paths on the mesh and a new trade-off in VLSI layout, SIAM J. Comput. 29 (4) (2000) 1321–1333.
[2] W.F. Dowling, J.H. Gallier, Linear-time algorithms for testing the satisfiability of propositional horn formulae, J. Logic 3 (1984) 267–284.
[3] S. Gao, B. Novick, K. Qui, From Hall's matching theorem to optimal routing on hypercubes, J. Combin. Theory Ser. B 74 (2) (1998) 291–301.
[4] T.F. Gonzalez, Complexity and approximations for multimessage multicasting, J. Parallel Distributed Comput. 55 (1998) 215–235.

[5] T.F. Gonzalez, F.D. Serena, Node disjoint shortest paths for pairs of vertices in an n-cube network, in: Proc. Internat. Conf. Parallel and Distributed Computing and Sytems (PDCS2001), 2001, IASTED, pp. 278–282 (full version appears as UCSB TRCS-2001-16, September 2001).
[6] T.F. Gonzalez, F.D. Serena, Complexity of k-pairwise disjoint shortest paths in the undirected hypercubic network and related problems, in: Proc. Internat. Conf. Parallel and Distributed Computing and Systems (PDCS 2002), 2002, IASTED, pp. 61–66 (full version appears as UCSB TRCS-2001-14, May 2002).
[7] Q.-P. Gu, S. Peng, k-Pairwise cluster fault tolerant routing in hypercubes, IEEE Trans. Comput. 46 (9) (1997).
[8] Q.-P. Gu, S. Peng, Node-to-set and set-to-set cluster fault tolerant routing in hypercubles, Parallel Comput. 24 (1998) 1245–1261.
[9] Q.-P. Gu, H. Tamaki, Routing a permutation in the hypercube by two sets of edge-disjoint paths, J. Parallel Distributed Comput. 44 (1997) 147–152.
[10] R. Karp, On the computational complexity of combinatorial problems, Networks 5 (1975) 45–68.
[11] R. Karp, F.T. Leighton, R.L. Rivest, C.D. Thompson, U. Vazirani, V. Vazirani, Golbal wire routing in two-dimensional arrays, in: 24th Ann. Symp. on Foundations of Computer Science,IEEE Computer Society Press, 1983, pp. 453–459.
[12] M.R. Kramer, J. van Leeuwen, The complexity of wirerouting and finding minimum area layouts for arbitrary VLSI circuits, Adv. Comput. Res. 2 (1984) 129–146.
[13] S. Latifi, H. Ko, P.K. Srimani, Note-to-set vertex disjoint paths in hypercube networks, Computer Science Tech. Rep. CS-98-107, Colorado State University, 1998.
[14] T. Lengauer, Combinatorial Algorithms for Integrated Circuit Layout, Wiley, New York, 1990.
[15] S. Madhavareddy, I.H. Sudborough, A topological property of hypercubes: node disjoint paths, in: Proc. Second IEEE Symp. Parallel and Distributed Processing,1990, pp. 532–539.
[16] K. Menger, Zur allgemeinene kurventheorie, Fund. Math. 10 (1927) 95–115.
[17] D. Pretolani, A linear time algorithm for unique horn satisfiability, Inform. Process. Lett. 48 (1993) 61–66.
[18] M.O. Rabin, Efficient dispersal of information for security, load balancing, and fault tolerance, J. Assoc. Comput. Machinery 36 (2) (1989) 335–348.
[19] X. Shen, Q. Hu, W. Liang, Realization of an arbitrary permutation on a hypercube, Inform. Process. Lett. 51 (51) (1994) 237–243.
[20] Y. Shiloach, To paths problem is polynomial, Tech. Rep. TR-CS-78-654, Stanford University, 1978.
[21] M. Watkin, Graph is $(2k-1)$-connected is a necessary condition to admit k paths, Duke Math. J. (1968).