Complexity of k-Pairwise Disjoint Shortest Paths in the Hypercube and Grid Networks

Teofilo Gonzalez and David Serena
{teo, dserena}@cs.ucsb.edu
University of California at Santa Barbara
Department of Computer Science

May 30, 2002

Abstract

In parallel and distributed systems many communications take place concurrently, so the routing algorithm as well as the underlying interconnection network plays a vital role in delivering all the messages efficiently. Fault tolerance and performance are often obtained by delivering the messages through node disjoint shortest paths. In this paper we present two basic graph topologies, the grid and the n-cube wherein it is determined that the k-pairwise node or edge disjoint shortest paths decision problem is NP-complete or NP-hard for a partial half permutation routing request – a subset of a 1-1 routing request. Further k-pairwise edge disjoint path decision problem in the doubled n-cube for a 1-1 routing request found to be NP-complete.

Keywords: fault-tolerance, hypercube, n-cube, doubled n-cube, grids, node disjoint shortest paths, edge disjoint shortest paths, k-pairwise routing, NP-completeness, NP-hardness.

1 Introduction

The n-cube is a vital structure for parallel computing. Several systems with this communication architecture have been built. The SGI Origin 2000 is a computing platform whose interconnection network is a variation of the n-cube. There are many algorithms for several different routing problems that arise while executing code on an n-cube connected machine. In this paper we study the computional complexity of routing problems which are common to many applications.

The *node disjoint shortest paths problem* for the n-cube is given p pairs of nodes and q blocking nodes denoted by

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

where $X_i = (s_i, t_i)$, for $1 \leq i \leq p$, and $X_i = (a_i)$ for $p+1 \leq i \leq p+q$, find node disjoint shortest paths in the n-cube for all the pairs X_i that do not include blocking nodes and such that 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. Every node in the n-cube is represented by an n-bit string of bits and there is an edge between two nodes if their bit representation disagrees in exactly one bit. The distance between the source and target nodes of pair X_i in the n-cube is denoted by $d(X_i) = d(s_i, t_i)$ and it is the number of bits that differ in the bit representation of s_i and t_i. 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 $s_{i'}$ and $t_{i'}$ for $i' \neq i$. The nodes a_i (or blocking nodes or faulty processors) may also be included as part of the input. The *edge disjoint shortest paths problem* is given X in the n-cube find shortest paths connecting each s_i to t_i such that no two paths have an edge in common.

Both the decision problems and the search problems are important for analysis. In the undirected n-cube there are "yes" and "no" instances of the k-pairwise node disjoint shortest paths decision problem. In the 2-cube $X = \{\{00, 11\}, \{01\}, \{10\}\}$ has no solution because any path between 00 and 11 must go through 01 and 10. On the other hand, $Y = \{\{00, 11\}, \{01\}\}$ does have such a solution with pair $\{00, 11\}$ yielding route $00 \leftrightarrow 10 \leftrightarrow 11$. The algorithms presented herein are search algorithms in the sense that they construct for yes instances of the decision problem a set of node disjoint paths. For problem instance Y given above a successful routing is $00 \leftrightarrow 10 \leftrightarrow 11$. Note that while problem instance $Z = \{\{000, 011\}, \{001\}, \{010\}\}$ has no shortest path solution, it does have a routing with any length paths: $000 \leftrightarrow 100 \leftrightarrow 101 \leftrightarrow 111 \leftrightarrow 011$. To clarify the problem the possible shortest path routings are $000 \leftrightarrow 010 \leftrightarrow 011$ and $000 \leftrightarrow 001 \leftrightarrow 011$. Though these paths are shortest paths by our definition, neither of these paths may be traversed due to the blocking nodes in the input: $\{001\}$ and $\{010\}$. For edge disjoint shortest paths both of thos paths are possible. However the problem instance $Z = \{\{00, 11\}, \{10, 01\}\}$ does not have edge disjoint shortest paths.

Node disjoint paths have been studied extensively. Karp [20] showed that determining whether or not there exist node disjoint paths for a set of pairs of vertices in a graph is an NP-complete problem. Shiloach [29] presented a polynomial time algorithm to construct node disjoint paths in a graph for two pairs of vertices and Watkin [33] showed that $(2k-1)$-connectedness is a necessary condition for a graph to admit disjoint paths for a set of k pairs of vertices.

Madhavapeddy and Sudborough [26] developed an $O(n^3 \log n)$ time algorithm to find disjoint paths for k pairs in an n-cube when $k > 2$ and $n \geq 4$. Each of the paths is of length at most $2n$. Then Gu and Peng [14] presented an algorithm that takes $O(kn \log n)$ time to find node disjoint paths for k pairs. Even if there are $n - 2k + 1$ faulty clusters of diameter 1 and $k \leq \lceil n/2 \rceil$.

The main difference between the work mentioned above and ours is that we are interested in finding node or edge disjoint **shortest** paths rather than just node disjoint paths (i.e. for each pair X_i the length of the path must be $d(X_i)$). Rabin proposed an algorithmic strategy which addresses fault-tolerance, security and load balancing [28]. While the paper is an interesting starting point it is clear that in general, three arbitrary parameters of optimization may sometimes be contradictory or at least mutually interdependent.

Gu and Peng [15] address an interesting and very similar problem; the problem of constructing set-to-set node disjoint paths. The main difference between their problem and ours is that they just need to find a path from each vertex in one set to a (different) vertex in the other set; where as, in our problem one constructs **shortest** paths for a **given set of pairs** of vertices. Our problem has applications when network traffic endpoints are defined by empirically observed flows; or are specifically initiated by the individual nodes in the network to designated destinations. Gu and Peng's paper [15] also presents algorithms for the node-to-set node disjoint paths problem. Gao, Novick and Qiu [7] presented an algorithm for finding node-to-set node disjoint *shortest* paths. Gao, Novick and Qiu's paper [7] exploits the fact that local n-cube constraints, the existence of a first step in a path, dominate in determining the existence of node-to-set node disjoint paths. Namely, as shown in [7], the existence of a System of Distinct Representatives (SDR) is a necessary and sufficient condition for the existence of node disjoint shortest paths. We have observed to some extent, similar characteristics in restricted versions of our problem. Node-to-set approaches in the n-cube are highly pragmatic in the sense that it has applications in the context of fault-tolerant distributed networks. There are several important papers which also address this issue for the n-cube, [7, 23]. The node-to-set problem reduces to our problem after finding the first link for each path and of course iterating over destination nodes. Finding k disjoint paths between two nodes in the hypercube has also been studied [14], but for brevity we do not discuss in detail these results.

Gonzalez and Serena [12] present polynomial time algorithms for a restricted subproblem called the *extreme* version. The extreme version of the *node disjoint shortest paths problem* for the n-cube is given p pairs of nodes and q blocking nodes denoted by

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

where $X_i = \{s_i, t_i\}$, $d(X_i) = n$, for $1 \leq i \leq p$, and $X_i = \{a_i\}$ for $p + 1 \leq i \leq p + q$, find node disjoint shortest paths in the n-cube for all the pairs X_i that do not include blocking nodes and such that no two such paths have a node in common. Remember that the path for each pair X_i must have length equal to $d(X_i)$. For this version of the problem the endpoints of every pair X_i are complements ($\oplus(2^n - 1)$) of each other because they are in the n-cube and $d(X_i) = n$, i.e., $\bar{s}_i = s_i \oplus (2^n - 1) = t_i$. They show that under certain conditions a set of shortest paths exists for the set of p pairs in X even in the presence of q blocking nodes, single vertices which can be considered as faulty or simply blocking nodes.

Gonzalez and Serena's first algorithm in [12], for the case when $p \leq \lceil n/2 \rceil$ and $2p+q \leq n+1$, constructs the set of node disjoint shortest paths by finding for each pair X_i one link incident to an endpoint of X_i which will be part of the final path and subsequently reduce the problem of finding p shortest paths in the presence of q blocking nodes in an n-cube into two independent problems in $(n-1)$-cubes with a total number of p pairs and $p+q$ blocking nodes. Their approach is to provide constructive proofs for their claims and then present an algorithm based on their constructive proofs. Their algorithm, has time complexity is $O((p^2+q)n^2)$. Latifi, Ko and Srimani [23] use a variation of this approach, albeit for a different routing problem, for which they generate node disjoint paths that are not necessarily shortest paths. The second algorithm in [12] is for the case when $p+q \leq n-1$. This algorithm finds a link for each pair and reduces the problem, using a balanced transformation, to finding a set of node disjoint paths in an $(n-1)$-cube. Since the two resulting subproblems do not fall within the induction hypothesis, another transformation is applied using the special bit position shown to exist which further reduces the cube size to an $(n-2)$-cube. These two transformations conducted multiple times yield an algorithm which generates paths in a time efficient manner. The algorithm has time complexity $O((p^2+pq)n^3)$.

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 given vertex in the graph [16]. This routing request problem has also been called the multimessage cd unicasting message routing problem. 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 [16]. With regards to the concept of h-k-rearrangeability or ℓ-rearrangeability the edge disjoint paths need not be restricted to the shortest path. 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 if vertex occupancy is constrained so vertices are occupied by a source or target, but not both. Define this request as a *partial half permutation routing request*. Furthermore define the routing request in Equation 1 as a *partial half permutation routing request with singletons*.

Many of these problems are know to be computationally difficult for general graphs even if one allows any path, rather than just shortest ones. In [20], Karp analyzes the k-pairwise disjoint paths problem in a general graph and shows NP-completeness. In this paper we concentrate on the corresponding problem but in specific topologies. The doubled n-cube, by our definition, which differs from that in Lubiw [25], takes the undirected form of the n-cube and replaces each undirected edge with two directed edges: for a graph, $G=(V,E)$, undirected edge $\{a,b\} \in E$ is replaced by (a,b) and (b,a), $a,b \in V$.

Szymanski's Weak Conjecture [31] is that the doubled n-cube is 1-1-rearrangeable with any length path. This conjecture holds for $n \in \{2,3,4\}$. *Szymanski's Strong Conjecture* [31] is that the doubled n-cube is 1-1-rearrangeable with shortest paths. The work of Lubiw [25] has demonstrated that the Szymanski strong conjecture is false.

From a practical standpoint near-shortest paths often suffice for efficient transmission in both circuit-switched and packet-switched networks. This paper demonstrates that even under many common topologies and extremely restrictive assumptions, namely shortest paths, the problem remains or is NP-complete or NP-hard. The network topologies under consideration are both the grid and the n-cube. Remember that shortest paths are defined by the underlying graph topology not the mutual interaction of paths. Hence detailed analysis in this context, namely the *k-pairwise disjoint shortest paths problem* is warranted. Herein we attempt to resolve complexity issues related to these problems. Hereafter when we talk about our routing problem we refer to the decision version of the problem as opposed to the search problem.

With regards to the n-cube we establish the k-pairwise node (Section 2) and edge (Section 4) disjoint shortest paths for a half partial routing request are NP-complete even when $d(X_i) \leq 3$. We also show that these problems are polynomially solvable when $d(X_i) \leq 2$. In Section 3 we show that even if we relax the condition of shortest paths and allow arbitrary paths the problem remains computationally intractable. For the doubled n-cube the 1-1 routing request problem is shown to be NP-complete (Section 5).

In VLSI routing the problem of finding disjoint paths in 1 and 2 dimensions has been extensively studied in [1, 21, 24]. In [21] the classical grid routing model is used and explores the grid (i.e. an $n \times n$ array of square cells). The capacity between adjacent vertices is determined by an arbitrary "channel width." The number of pins p describes the occupancy of the vertices in the grid. The case when $p = 1$ corresponds to the half partial permutation routing request without singleton nodes in the grid. The paper analyzes constant bend routing methodologies in the edge disjoint case and some provably good heuristics are developed. Further it is shown that optimal one turn routing is NP-complete when $p = 2$.

In this paper our edge disjoint shortest paths work deals only with a constant "channel width" of one for the connection of vertices in the grid. We show that the k-pairwise (edge and node) disjoint shortest paths problem for any number of turns is NP-hard in the grid. Further in Section 6.1 we show that for any constant number bends greater than 3 the edge disjoint version of the problem is NP-hard.

Kramer and Ven Leeuwen [22] show that the k pairwise node disjoint any length paths problem in the grid is shown to be NP-hard. This problem has also been stidied in the multilayer grid [1]. They discuss the node disjoint any length path with constant bend routing. They prove an upper bound trade off between the number of layers and area in a general multilayer grid model. The model discussed devolves to ours when the number of layers is one. In Section 6.2 we analyze the node disjoint shortest paths case for the problem and determine that the problem for a half partial permutation routing request is in fact NP-hard.

2 n-Cube: Node Disjoint Shortest Paths Problem

In this section we show that when $d(X_i) \leq 2$ finding node disjoint shortest paths for pairs of vertices is polynomially soluble not only in the n-cube but also in general graphs. We establish this result by performing the reduction from our problem to 2-SAT which we know can be solved in linear time [4, 27] via *resolution*. We also show that for the case when $d(X_i) \leq 3$ the problem becomes NP-complete. We prove this result by reducing from a restricted version of

3-SAT, which we call L3-SAT, to the k-pairwise node disjoint shortest path problem.

The 2-SAT problem is defined as follows:

INPUT: Given a set U of boolean variables $\{u_1, u_2, \ldots, u_v\}$, and a collection of clauses $C = (c_1, c_2, \ldots, c_w)$ over U such that each clause has at most two literals.

QUESTION: Is there a satisfying truth assignment for C?

The naïve implementation of the algorithm known as *resolution* is polynomial in the product of v and w, $O(vw)$ [3, 6, 8]; however there are known improved resolution algorithms which are of order $O(v + w)$ [4, 27]. We now outline a simple polynomial time reduction to the k-pairwise node disjoint shortest path problem from 2-SAT.

Given an instance (X, k, n) of the k-pairwise node disjoint shortest path problem defined in the n-cube, we construct an instance (U, C) of 2-SAT as follows. The idea is that for each pair $X_i \in X$ we introduce a boolean variable u_i. Since $d(X_i) \leq 2$, there are two possible shortest paths from s_i to t_i. Let us denote such paths by $P_{i,0}$ and $P_{i,1}$. If the boolean variable u_i ends up with the value of true then the pair X_i is connected by path $P_{i,1}$, otherwise it is connected by the path $P_{i,0}$. If either of these paths includes a vertex in X, then we add a clause of length one that forbids such a path (either u_i or $\bar{u}_i$). Now for every two paths $P_{i,a}$ and $P_{j,b}$, $i \neq j$, with at least one vertex in common, we add the clause $\{\bar{u}_i, \bar{u}_j\}$ if $a = b = 1$; $\{\bar{u}_i, u_j\}$ if $a = 1$ and $b = 0$; $\{u_i, \bar{u}_j\}$ if $a = 0$ and $b = 1$; and $\{u_i, u_j\}$ if $a = b = 0$.

Lemma 1 *Given an instance (X, k, n) of the k-pairwise node disjoint shortest path problem, and the corresponding instance (U, C), then (X, k, n) has a set of node disjoint shortest paths for the k-pairs if and only if (U, C) is satisfiable.*

Proof: The proof is simple because of the direct correspondence of all the possible paths and boolean variables. Qualitatively the clauses directly enumerate possible paths. Similarly the demand that all paths are node disjoint shortest paths is enforced by "and" operation between the clauses. []

Using the above lemma we construct the algorithm 2-NDSP for the k-pairwise node disjoint shortest path problem as follows: first construct the corresponding instance of 2-SAT as described above, then solve it by the well known linear time algorithm for 2-SAT. The algorithm generates node disjoint shortest paths, if they exist, from the values of the boolean variables in the solution to the 2-SAT problem instance.

Theorem 1 *Algorithm 2-NDSP constructs a set of paths for any instance of the k-pairwise node disjoint shortest paths problem in the n-cube, where the distance between every pair is at most two in $O(k^2 n)$ time, whenever such paths exist.*

Proof: Correctness follows from the previous lemma. The instance of 2-SAT constructed from the k-pairwise node disjoint shortest path problem instance has k variables and at most $O(k^2)$ clauses. The time required to construct the instance is $O(k^2 n)$ time because for every two pairs one needs to check if the shortest paths have a node in common which takes $O(n)$

time since the vertices are represented by strings n bits long. The solution of the 2-SAT problem instance takes linear time with respect to the number of clauses and literals, which in this case is $O(k^2)$. The construction of the solution, if it exists, takes $O(kn)$ time, since there are $O(k)$ pairs and each one has an intermediate vertex (n-bits to represent it). []

For arbitrary graphs we cannot use the same technique to solve our problem. The reason is that the number of shortest paths between a pair of vertices at a distance two from each other may be more than two. It does not seem possible with an instance of 2-SAT to select one of these possible paths. However, the problem remains polynomially solvable by reducing it to finding a maximum matching in a bipartite graph which we know is polynomially solvable [17].

Note that the time complexity for our new algorithm is very different from that for the n-cube case simply because the input to the algorithm is a set of vertices, V, with an arbitrary set of edges, E, followed by k pairs of vertices. For the n-cube the graph structure is implicit and the whole structure is simply characterized by the integer n. Thus, the input size is just n followed by the k pairs of vertices ($O(kn)$).

Theorem 2 *Given any instance of the k-pairwise node disjoint shortest path problem defined over graph $G = (V, E)$ with $v = |V|$ vertices and $e = |E|$ edges, where the distance between each pair of vertices is at most 2, algorithm 2G-NDSP solves the problem in $O(kv\sqrt{k+v})$ time.*

Proof: We present a constructive proof. First we construct a bipartite graph and then we find a maximum matching in it. If the cardinality (number of edges) of the matching is k, then there are k-pairwise node disjoint shortest paths for the pairs and they can be constructed from the matching. Otherwise there is no solution to the instance. For every pair $X_i = \{s_i, t_i\}$ and let $I_i = \{v_{i,1}, v_{i,2}, \ldots, v_{i,\ell_i}\} \subset V$ be the middle (or intermediate) nodes in the shortest paths from s_i to t_i that are not part of the other vertices of the k-pairs. Note that each of the shortest paths must have a length equal to two. We construct the bipartite graph $G' = (V', E')$ as follows:

$$\begin{aligned} L &= \{x_1, x_2, \ldots, x_k\} \\ R &= \bigcup_{j=1}^{k} \bigcup_{u \in I_j} u \\ V' &= L \cup R \\ E' &= \{(a, b) \mid a \in L,\ b \in R\} \end{aligned}$$

Clearly, the vertex sets L and R are distinct, $L \cap R = \emptyset$. We claim that G' has a matching of cardinality k if, and only if, there exist node disjoint shortest paths for the k-pairs in G. The reason is that a matching of cardinality k chooses for each pair a path which by construction of G' are all node disjoint, and vice-versa. The construction of the graph takes $O(kv)$ time and the resulting graph has $k + v$ vertices and $O(kv)$ edges. Hopcroft and Karp's [17] algorithm takes $O(kv\sqrt{k+v})$ time. The node disjoint shortest paths can be constructed in $O(k)$ time from the maximum matching. Thus the overall time complexity is $O(kv\sqrt{k+v})$. []

We now show that the k-pair node disjoint shortest paths problem in the n-cube is NP-complete when $d(x_i) \leq 3$. We prove this by reducing the L3-SAT problem to the k-pair node disjoint problem. The L3-SAT problem is defined as follows.

INPUT: Given a set U of boolean variables $\{u_1, u_2, \ldots, u_v\}$, and a collection of clauses $C = (c_1, c_2, \ldots, c_w)$ over U so that each clause has two or three variables. As an additional criteria, all clauses contain no more than two instances of the variable u and the same holds for $\bar{u}$.

QUESTION: Is (U, C) satisfiable?

The 3-SAT problem is L3-SAT without the last condition and removal of size one clauses. 3-SAT was shown to be NP-complete in [8]. For completeness, we briefly outline a simple reduction from 3-SAT to L3-SAT.

Assume without loss of generality that we start with an instance of 3-SAT such that when every variable $u \in U$, there is at least one clause that contains u and at least a clause that contains $\bar{u}$. When either u or $\bar{u}$ is not present in the clauses one may eliminate the variable and a subset of the clauses and end up with a problem instance that is satisfiable if and only if the original problem instance is satisfiable. To transform the instance of 3-SAT to an instance of L3-SAT we need to change all the variables that occur in at least four clauses. Let $u \in U$ be any variable that occurs in $\ell \geq 4$ clauses. Now introduce ℓ additional variables denoted by $a_1, a_2, \ldots, a_\ell$ and the set of clauses

$$\{a_1, \bar{a}_2\}, \{a_2, \bar{a}_3\}, \ldots, \{a_{\ell-2}, \bar{a}_{\ell-1}\}, \{a_{\ell-1}, \bar{a}_\ell\}, \{a_\ell, \bar{a}_1\}.$$

This cyclic set of clauses guarantees that all these ℓ variables have the identical value when the set of clauses is satisfiable. Furthermore, each variable and its complement appears exactly once. Now the first occurrence of u or $\bar{u}$ is replaced by a_1 or $\bar{a}_1$, the second occurrence of u or $\bar{u}$ in a clause is replaced by a_2 or $\bar{a}_2$, and so on. Hence variable u and its complement are eliminated and the variables such as a_1 occur twice for each usage. The negation of a_1, $\bar{a}_1$ similarly occurs at most twice. After applying this transformation to each variable that occurs in more than three clauses, we end up with an instance of the L3-SAT problem which is satisfiable if and only if the original 3-SAT problem instance is satisfiable. The transformation takes polynomial time and it is simple to show that L3-SAT is in NP. It then follows that the L3-SAT decision problem is an NP-complete problem. As we said before, this is not a new result, but it is included for clarity of exposition.

In Theorem 3 we show that when $d(X_i) \leq 3$ the problem is NP-complete. We establish this result by reducing the L3-SAT problem to it. Before we present the proof of Theorem 3, we start by discussing the transformation details. There are three types of components: *setting and fan-out*, *convey* and *clause-checker*. The setting and fan-out component assigns to each variable a value, making two copies of the variable and its negation. The convey apparatus transports the value of a boolean variable or its negation from one vertex in the n-cube to another. The clause checker makes sure that a clause is satisfied if at least one of its literals has the value true.

Figure 1 presents the setting and fan-out component which generates two copies of a boolean variable and its complement. The construction consists of a length three pair $(s,t) = (000, 111)$ and two blocking nodes $a_1 = 010$ and $a_2 = 101$: $\{(000, 111), (010), (101)\}$. Note that there are only two possible node disjoint shortest paths for the pair: $000 \leftrightarrow 100 \leftrightarrow 110 \leftrightarrow 111$ and $000 \leftrightarrow 001 \leftrightarrow 011 \leftrightarrow 111$.

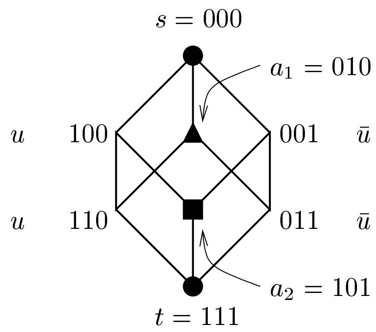


Figure 1: Duplication of consistent variable state: "setting and fan-out" component.

There are two consistent copies of the value of a variable and its negation. In particular when the nodes 100 and 110 are in the path for pair $\{000, 111\}$ the boolean variable u has the value of **false**, and when the nodes 001 and 011 are in the path for the pair then the variable u has the value of **true**. The opposite holds for $\bar{u}$, the negation of u. The reason one needs only two copies of each value of a variable and its negation is that we are reducing from the L3-SAT problem which has the property that no variable or its negation is in more than two clauses.

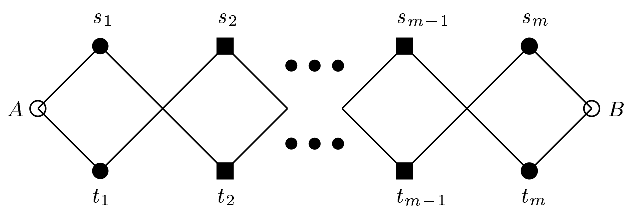


Figure 2: Convey Apparatus

The convey apparatus is used to transport "almost" consistent variable state information from the setting and fan-out components to the clause-checkers. A convey apparatus is shown in Figure 2 and consists of the pairs $\{(s_1, t_1), (s_2, t_2), \ldots, (s_{p+q}, t_{p+q})\}$. Each pair has vertices at a distance two from each other. All these pairs are called *conveyor pairs*. Note that in Figure 2 if vertex A is covered by a path from a non-conveyor pair then B will be covered by the paths joining the pair $\{s_m, t_m\}$. However if A is not covered by a path from a non-conveyor pair the vertex B may or may not be covered. Note that in Figure 2 if A is covered by a path from a non-conveyor pair then B will be covered by the path joining the pair (s_m, t_m). However if A is not covered by a path from a non-conveyor pair then B may or may not be covered by the path joining the pair (s_m, t_m). Ideally we would want a perfect transmission of information which in this case means that B is covered by the path joining the pair (s_m, t_m) if and only if A is covered by a path for a non-conveyor pair. Convey apparatus which transmit vertex coverage precisely exist, but they have a complex structure and are based on pairs whose nodes are at a distance four from each other with overlapping 4-cubes. Our construction has a possible faulty transmission, but the component is simple and the fault does not cause a problem. The reason for this is that the components are used as follows: a false value will be transmitted to B as a covered vertex, whereas a true value may be received at B as either a covered or non-covered vertex. Note however, it is always possible to receive it as a true value; namely as an uncovered node at vertex B in the convey apparatus. A false value for u corresponds with A being covered by a path from a non-conveyor pair in Figure 1. A true value for u corresponds with A not being covered by a path for a non-conveyor pair. A similar argument holds for the negation of u. If u is in a clause, the vertex A of the convey apparatus will be coincident with vertex 100 or 110 in Figure 1 and vertex B will be a vertex in the clause-checker. If u has a true value the pair $(000, 111)$, as shown in Figure 1, is connected by the path $000 \leftrightarrow 001 \leftrightarrow 011 \leftrightarrow 111$ and vertex $A = 100$ is not covered. Thus vertex B may or may not be covered by the path in pair (s_m, t_m). On the other hand if u is false it results in vertex 100 and 110 being covered, hence $A = 100$ is covered and the path for pair (s_m, t_m) must cover node B. A similar argument holds when A is coincident with nodes 110, 001 or 011. The latter two nodes are used when $\bar{u}$ is in the clause.

Any clause of size one may be eliminated since the variable or its negation must be true when the problem instance is satisfiable. Therefore all clauses contain two or three literals. The clause-checkers for this case are given in Figure 3. The convey apparatus terminates at a clause-checker vertex b_1, b_2 and b_3 for clauses with three literals and at vertex b_1 and b_2 for clauses with two literals. The convey apparatus transmits the value of a variable or its negation from vertex A to vertex B and vertex B will be coincident with a vertex labeled b_1, b_2 or b_3 in a clause-checker. Remember that if in the convey apparatus, a literal has the value of false, vertex A is covered and thus vertex B will be covered as part of the path for pair (s_m, t_m). So if all the literals in a clause have the value of false there will be no way of connecting (s, t), in Figure 3, by a shortest path since all of the neighbors surrounding the shortest path from s to t are covered. On the other hand if at least one of these values is true, it is possible that vertex B will not be covered and hence there is at least one shortest path available for the pair (s, t).

At this point one may conjecture that it is just matter of embedding all of these components in an n-cube, but this alone does not suffice. There are some subtle details which need to be addressed. Let us define the *parity* of a vertex as **even** if the number of ones in its bit representation is even, and **odd** otherwise. The convey apparatus has the property that the parity of A must be equal to the parity of B. Since the parity of the vertices in the setting and fan-out components vary, we need to construct clause-checkers whose terminals have all

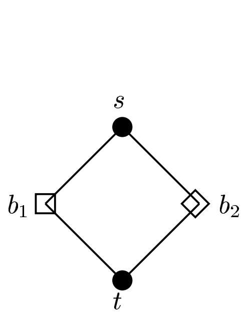


Figure 3: Same parity clause-checkers for 2 and 3 literals.

possible combinations of parities. Figure 3 shows components whose terminals have identical parities. Figure 4 shows the case when the terminals have differing parities. For a clause with two literals, the left hand side construction in Figure 4 can be used by assigning b_1 and b_2 to the appropriate vertices; so that both have even parity, odd parity, or differing parity. For clauses with three literals one may use the right hand component in Figure 3 for even parity or odd parity by placing the terminals in appropriate vertices, and for mixed parity one can use the component in the right hand side of Figure 4 by placing the terminals in appropriate vertices. The actual placement of the terminals so that the desired parity for each vertex is correct is straightforward. Clearly, Figures 3 and 4 demonstrate that one may construct clause-checkers where the terminals have all possible combinations of parities.

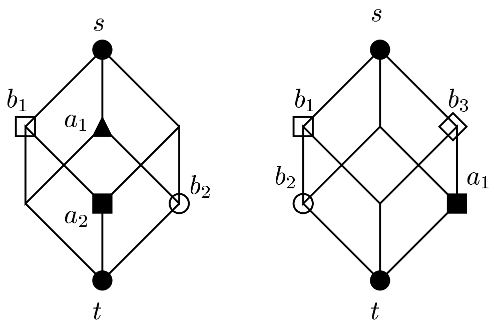


Figure 4: Left: Two literal clause-checker with different parity terminals. Right: Three literal clause-checker with two same parity terminals (b_1 and b_3) and one differing one (b_2).

Given an instance of the L3-SAT problem with w clauses and v variables we construct an instance of the k-pairwise node disjoint shortest path problem as follows. The nodes of the n-cube will be a sequence of bits of the following form:

$$D_0D_1 \dots D_{\lceil \log_2(v+1) \rceil} \;\; E_0E_1 \dots E_{\lceil \log_2(w+1) \rceil} \;\; Y \;\; FGH \;\; JKL$$

The 3-cubes for the setting and fan-out components use the bits FGH and clause-checkers use bits JKL. The index of a clause number is represented in binary by the bits $E_0E_1 \ldots E_{\lceil \log_2(w+1) \rceil}$ and we use the bits $D_0D_1 \ldots D_{\lceil \log_2(v+1) \rceil}$ to represent the index of variables. The Y bit is used to make the parity of an intermediate vertex equal to the parity of the vertices A and B in a convey apparatus.

The 3-cube for the setting and fan-out component representing the i^{th} variable is defined using the bits FGH, the bits $D_0D_1 \ldots D_{\lceil \log_2(v+1) \rceil}$ represent the value of i and all the remaining bits are zero. Now we match the terminals of the setting and fan-out component to the appropriate clauses to determine the type of clause-checker needed. The j^{th} clause-checker will be set so that bits JKL are the 3-cube as indicated. $E_0E_1 \ldots E_{\lceil \log_2(w+1) \rceil}$ represents the integer j in binary. Again all the other bits set to zero. The terminals in the setting and fan-out components matched to the terminals in the clause-checkers effect a polynomial reduction to the instance of L3-SAT and insure that they have the same parity. This is possible because the clause-checkers can be constructed with any combination of parities. The third part is to specify a convey apparatus for every pair of matched terminals. Let us now define the path of the convey apparatus that joins a terminal in the i^{th} variable with a terminal in the j^{th} clause-checker. The convey apparatus will start at node $bitrep(i)$ $bitrep(0)$ 0 FGH 000 where $bitrep(\ell)$ is the bit representation of integer ℓ and FGH are the bits for the setting and fan-out terminal in the matching being implementing. By changing two bits at a time the conveyor pairs end at $bitrep(0)$ $bitrep(j)$ 0 000 JKL. We require that the conveyor visits the intermediate vertex $bitrep(i)$ $bitrep(j)$ Y FGH JKL. Y is such that the parity of this vertex is exactly the parity of vertex A and B in the convey apparatus. There is no overlap with other pairs and their respective paths because $(\ldots\ bitrep(j)\ \ldots\ FGH\ \ldots)$ is unique in the first part of the path and $(bitrep(i)\ \ldots\ JKL)$ is unique in the second part of the path.

Theorem 3 *Given a partial half permutation routing request with singleton nodes the k-pairwise node disjoint shortest paths problem for the n-cube is NP-complete.*

Proof: The problem is in NP. Let p and q be as defined in the basic problem statement, then the input length is $O(n(p+q))$. The nondeterministic algorithm guesses each path in $O(n)$ and inserts the nodes in a binary trie, checking for duplicates. If the guess is correct and there are no duplicates the problem has a solution. Since each path has at most $n-1$ intermediate vertices then the whole process takes polynomial time with respect to the input length. Hence it is in NP.

We now show that L3-SAT polynomially transforms to the k-pairwise node disjoint shortest paths problem for the n-cube. Given any instance of L3-SAT we construct an instance of the k-pairwise node disjoint shortest paths problem as shown just before this theorem. The problem is defined in the n-cube, where $n = (\lceil \log_2(w+1) \rceil + \lceil log_2(v+1) \rceil + 7)$. w is the number of variables and v is the number of clauses in the L3-SAT problem instance.

Since each pair has n bits and the nodes in each pair are at a distance at most 3; the instance can be constructed in polynomial time with respect to $O(wv)$, the input length of the L3-SAT problem instance. Further the input length of the k-pairwise disjoint paths problem is $O(nwv)$, which is clearly polynomial in the input length. The proof that the instance of

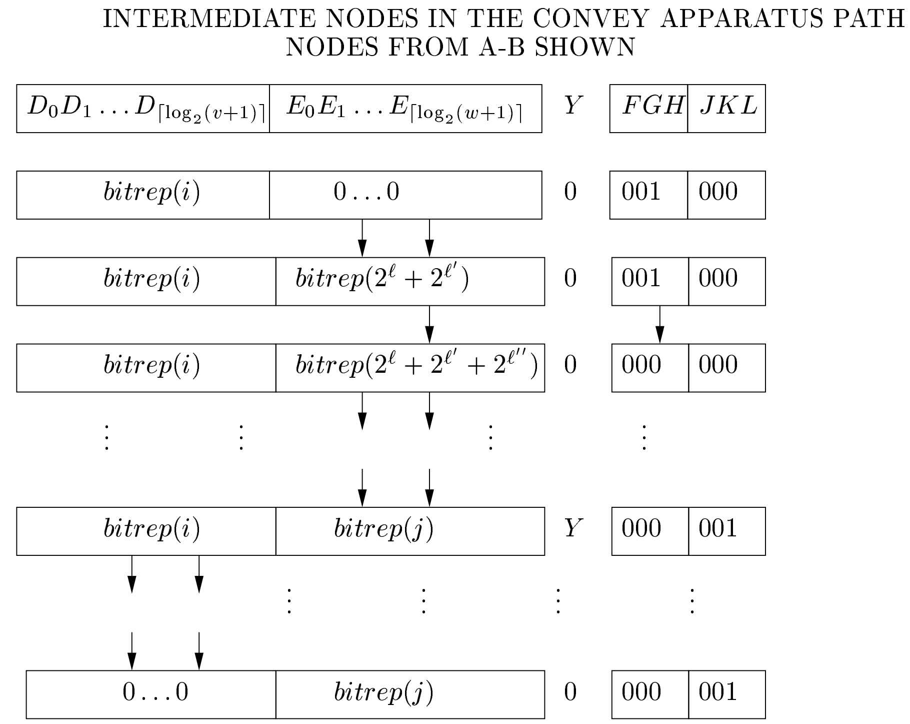


Figure 5: n-Cube Node Disjoint Problem Example Convey Apparatus

L3-SAT is satisfiable if and only if the problem instance constructed from the k-pairwise node disjoint shortest path problem has k node disjoint paths; is straightforward and follows the arguments in the discussion just before the theorem. Thus the problem is NP-complete. []

Corollary 1 *Given X a partial half permutation routing request with singleton nodes k-pairwise node disjoint shortest paths problem for the n-cube is NP-complete when $d(X_i) \leq 3$.*

Proof: By the proof of Theorem 3. []

Clearly both the Theorem 3 and the Corollary 1 hold in the undirected n-cube for any routing request which properly contains the partial half permutation request without singleton nodes, for example a 1-1 routing request or a partial half permutation routing request with singleton nodes. To improve the result the following corollary follows directly from a trivial modification of the *setting and fan-out* component.

Corollary 2 *Given X a partial half permutation routing request without singleton nodes k-pairwise node disjoint shortest paths problem for the n-cube is NP-complete when $d(X_i) \leq 3$.*

Proof: Follows directly by replacing blocking nodes in the 3-cube constructions for the clause-checker and the setting and fan-out component with pairs of length one. For example the pairs in the revised construction are precisely as follows $Z \subset X$. (A and B are the bit sequences corresponding to the specific fan-out component 3-cube to be modified.)

$$Z = \{\{0000, 0111\}, \{0010, 1010\}, \{0101, 1101\}\}$$

The revised construction thus remains polynomial and NP-complete even for the partial half permutation routing request without singleton nodes. []

3 n-Cube: Node Disjoint Paths Problem

How about if the paths connecting the pairs are allowed to have any arbitrary length? Namely paths of the form $011 \leftrightarrow 010 \leftrightarrow 110 \leftrightarrow 100$ are allowed as a route for input pair $\{000, 011\}$. In this section we show that finding k-pairwise node disjoint paths in the n-cube remains difficult in the sense that it is an NP-hard problem.

The idea is to take the construction in the previous section and just wrap around it blocking nodes so that every path for each pair does not go outside the corresponding shortest path for the pair. The construction in Section 2 was shown to be polynomial in the input length. The three type of components *setting and fan-out*, *convey* and *clause-checker* can be easily surrounded by individual blocking nodes given X as a partial half permutation routing request with singleton nodes.

Theorem 4 *Given X a partial half permutation routing request with singleton nodes the k-pairwise node disjoint paths problem for the n-cube is NP-hard.*

Proof: The input size is $O((p+q)n)$. We know that the total size of the construction outlined in Theorem 3 is polynomial in the input length. Each component as a function of the number of input pairs only introduces an $O(n)$ number of singleton blocking nodes needed to surround each of the components. The construction is clearly polynomial and hence the problem is NP-hard. []

The difficulty of showing that the problem is in NP is due to the fact that some paths may be of exponential length in the size of the input. This factor makes it difficult to prove that a "yes" instance has a nondeterministic polynomial solution. This is due to the fact that there are an exponential number of nodes in the n-cube: 2^n. Therefore if a problem instance only has a solution with a "long" (exponential in n) path the problem may not be nondeterministically polynomial.

Corollary 3 *Given X a partial half permutation routing request the k-pairwise node disjoint paths problem for the n-cube is NP-hard even for $d(X_i) \leq 3$.*

Proof: The proof follows by replacing individual singleton blocking inputs with length one pairs as in Corollary 2 and noting that all pairs are at a length of at most three. Thus $d(X_i) \leq 3$. The problem remains polynomial in the input size. []

4 n-Cube: Edge Disjoint Shortest Path

Consider that the routing request, X is a partial half permutation routing request (without singleton nodes). As in the case of the node disjoint shortest path problem this problem is polynomially solvable for $d(X_i) \leq 2$, and it becomes NP-hard when $d(X_i) \leq 3$. Our techniques to prove that these claims are slightly different than the ones in the previous section.

Given an instance (X, k, n) of the k-pairwise edge disjoint shortest path problem defined in the n-cube, we construct an instance (U, C) of 2-SAT as follows. For each pair $X_i \in X$ we introduce a boolean variable u_i. Since $d(X_i) \leq 2$, there are two possible shortest paths from s_i to t_i. Let us denote such paths by $P_{i,0}$ and $P_{i,1}$. If the boolean variable u_i ends up with the value of true then the pair X_i is connected by path $P_{i,1}$, otherwise it is connected by the path $P_{i,0}$.

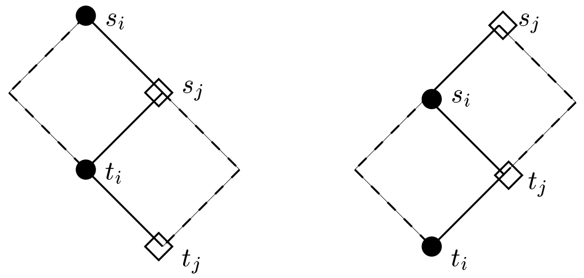


Figure 6: The two **yes** instances, $|X| = 2$, with overlapping Edge Configurations

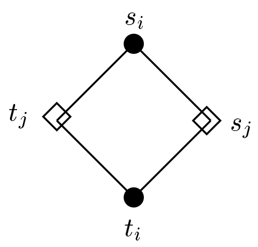


Figure 7: The **no** instance with $|X| = 2$ and $d(X_i) \leq 2$.

Unlike the vertex disjoint case, there are no clauses of length one because a pair may not block a possible path for another pair. However for every two paths $P_{i,a}$ and $P_{j,b}$, $i \neq j$, with at least one edge in common, we add the clause $\{\bar{u}_i, \bar{u}_j\}$ if $a = b = 1$; $\{\bar{u}_i, u_j\}$ if $a = 1$ and $b = 0$; $\{u_i, \bar{u}_j\}$ if $a = 0$ and $b = 1$; and $\{u_i, u_j\}$ if $a = b = 0$.

Lemma 2 *Given an instance (X, k, n) of the k-pairwise edge disjoint shortest path problem, and the corresponding instance (U, C), then (X, k, n) has a set of edge disjoint shortest paths for the k-pairs if and only if (U, C) is satisfiable.*

Proof: The proof is simple because of the direct correspondence of all the possible paths and boolean variables. The clauses directly enforce the property that all paths must be edge disjoint. Similarly the demand that all paths are edge disjoint shortest ones enforces the "and" operation between the clauses. The "or" operation within the clause is used to eliminate paths that are not pairwise edge disjoint for different pairs. []

Using the above lemma we define algorithm 2-EDSP for the k-pairwise edge disjoint shortest path problem as follows: first construct the corresponding instance of 2-SAT as described above, then solve it by the well known linear time algorithm for 2-SAT. The algorithm generates edge disjoint shortest paths, if they exist, from the values of the boolean variables in the solution to the 2-SAT problem instance.

Theorem 5 *Algorithm 2-EDSP constructs for any instance of the k-pairwise edge disjoint shortest path problem in the n-cube, where the distance between every pair is at most two in $O(k^2n)$ time, whenever such paths exist.*

Our algorithm for node disjoint shortest paths for arbitrary graphs cannot be adapted to the edge disjoint shortest paths problem. The reason is that node disjointness is guaranteed by characterizing the path by the intermediate node, but that is not the case for edge disjointness. In this case one with two edges, a bipartite graph is insufficient to model this situation. In Theorem 6 we show that when $d(X_i) \leq 3$ the edge disjoint shortest path problem is NP-complete. We establish this result by reducing the L3-SAT problem to it. Before we present the proof of Theorem 6, we start by discussing the transformation details, as in the node disjoint case, there are three type of components: *setting and fan-out*, *convey* and *clause-checker*. The setting and fan-out component assigns to each variable a value, making two copies of the variable and its negation. The convey apparatus transports the value of a boolean variable or its negation from one edge in the n-cube to another. The clause checker makes sure that a clause is satisfied if at least one of its literals has the value true.

Figure 11 presents the setting and fan-out component to represent two copies of a boolean variable and its complement. The construction consists of a length three pair $\{s', t'\}$ and two blocking edges $(t'', s') = (010, 000)$ and $(s'', t') = (101, 111)$. Note that with the blocking edges there are only two possible edge disjoint shortest paths for the pair: $000 \leftrightarrow 100 \leftrightarrow 110 \leftrightarrow 111$ and $000 \leftrightarrow 001 \leftrightarrow 011 \leftrightarrow 111$. We explain in subsection 4.0.1 how we can introduce blocking edges. This is not a trivial matter because node 000 is already occupied by a pair. The construction provides two consistent copies of the value of the variable and its negation. In particular when the edges $(000, 100)$ and $(110, 111)$ are in the path for pair $\{000, 111\}$ the boolean variable u has the value of **false**, and when the edges $(000, 001)$ and $(011, 111)$ are in the path for the pair then the variable u has the value of **true**. The reason one needs only two copies of each value of a variable and its negation is that we are reducing from the L3-SAT problem which has the property that no variable or its negation is in more than two clauses.

4.0.1 The Edge Blocking Mechanism

To complete the construction of the setting and fan-out component we need a component to block an edge without using both of its vertices to do it.

Extrinsic node s or t

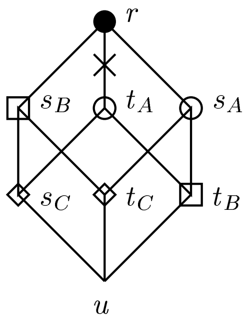


Figure 8: Edge Blocking Mechanism

Consider the edge blocking mechanism given in Figure 8. The edge to be blocked is the one from node r to t_A. When the route $s_A \leftrightarrow r \leftrightarrow t_A$ is selected for the pair $\{s_A, t_A\}$ then edge disjoint shortest paths exist for all the pairs in the 3-cube and are given below.

$$\begin{array}{ccc} s_A \leftrightarrow & r & \leftrightarrow t_A \\ s_B \leftrightarrow & t_C \leftrightarrow s_A & \leftrightarrow t_B \\ s_C \leftrightarrow & u & \leftrightarrow t_C \end{array}$$

On the other hand, if the path $s_A \leftrightarrow t_B \leftrightarrow t_A$ is selected, then the path for $\{s_B, t_B\}$ must be either

$$t_B \leftrightarrow \mathbf{u} \leftrightarrow \mathbf{t_C} \leftrightarrow s_B$$

or

$$t_B \leftrightarrow u \leftrightarrow \mathbf{s_C} \leftrightarrow \mathbf{s_B}.$$

Since the path for the pair $\{s_C, t_C\}$ is either $s_C \leftrightarrow \mathbf{u} \leftrightarrow \mathbf{t_C}$ or $\mathbf{s_C} \leftrightarrow \mathbf{s_B} \leftrightarrow t_C$. It then follows that there are no edge disjoint paths for the edge blocking mechanism, because each path for the pair $\{s_B, t_B\}$ has an edge in common with both of the remaining feasible paths for $\{s_C, t_C\}$. These shared edges are shown in bold, namely $\mathbf{u} \leftrightarrow \mathbf{t_C}$ or $\mathbf{s_C} \leftrightarrow \mathbf{s_B}$.

4.0.2 Connecting the Edge Blocking Mechanism with the Setting and Fan-out Mechanism

Now we need to incorporate the edge blocking mechanism into the setting and fan-out mechanism. The setting and fan-out component represents the i^{th} variable and is defined using the bits $F_0G_0H_0$. The bits $F_0G_0H_0$ are the 3-cube in Figure 11. The appropriate edges are blocked by overlapping the blocking component with bits F_1G_1 as the extrinsic dimensions in the 3-cube.

Each edge labeled in Figure 11, with an $\times$ is coincident with the edge labeled $\times$ in the blocking mechanism in Figure 8. There are two extrinsic dimensions (bits F_1G_1 in the following) required for a distinct 3-cube overlapping the 3-cube on one edge. To be precise the 3-cube in Figure 11 has $s' = 000\ 00$, $s'_1 = 001\ 00$, $s'_2 = 100\ 00$, $t'_1 = 110\ 00$, $t'_2 = 011\ 00$ and $t' = 111\ 00$. The latter two dimensions are used for the blocking 3-cube one notes that the two 3-cubes which are the blocking mechanisms are defined by pairs (s_A, t_A, s_B, t_B, s_C, t_C, u) = (000 01, 010 00, 000 10, 010 01, 010 10, 000 11, 010 11) and (s'_A, t'_A, s'_B, t'_B, s'_C, t'_C, u') = (111 01, 101 00, 111 10, 101 01, 101 10, 111 11, 101 11).

In both, note that the second two bits shown form a distinct signature that insures that all pairs and the intermediate paths between them are distinct. In the preceding the first three bits are $F_0G_0H_0$ and the remaining two bits are F_1 and G_1.

4.0.3 The Clause-Checker

The clause-checking component is a single pair in a 3-cube with edges (b_1, s), (b_2, s) and (b_3, s) that are conditionally covered, on a 2-cube with edges (b_1, s) and (b_2, s). Clearly, there is no shortest path from s to t if all the outgoing edges are covered.

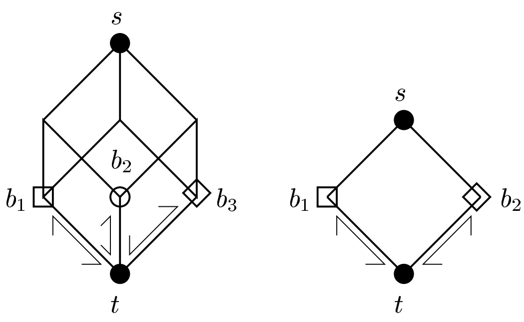


Figure 9: Left: Construction for Clause with Three Literals. Right: Size two clause. Edges which are conditionally occupied are labeled with a double arrow.

At most three independent conveyor apparatus will be joined to a setting and fan-out component. This is because the reduction is from L3-SAT and clause sizes are either two or three variables.

4.0.4 The Convey Apparatus

The convey apparatus is used for connecting the setting and fan-out components to the appropriate clause-checker. First we outline the basic technique and then indicate why the connections do not interfere with each other. Figure 10 gives the basic construction we will be using to connect variable state and fan-out to a clause-checker. The conveyor consists of the pairs $\{s_1, t_1\}$, $\{s_2, t_2\}$, ... $\{s_{p+q}, t_{p+q}\}$. As in the case for the node disjoint shortest paths problem, our conveyors do not always transmit information precisely. The simpler technique for conveyance of the edges suffices. Unlike the node disjoint shortest path convey apparatus in Figure 2 we have not yet determined whether a "perfect" construction exists for the edge disjoint case. However, we conjecture that in fact it does exist, but it is more complex than required for Theorem 6. That is why we use the simple convey apparatus shown in Figure 10. The conveyor operates as follows. If the edge A is covered by a path for a non-conveyor pair, then it will always be the case that the edge B will be covered by the paths in the conveyor pairs. However if edge A is not covered by a path for a non-conveyor pair then the edge B may or may not be covered by a path for the conveyor pair $\{s_{p+q}, t_{p+q}\}$.

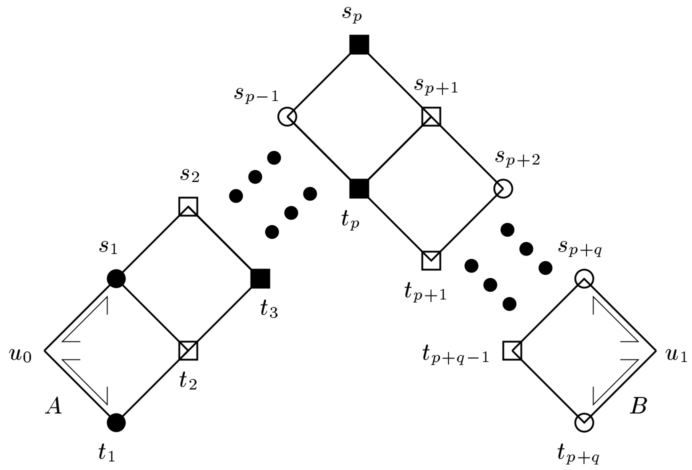


Figure 10: Convey Apparatus

The joining of the conveyor apparatus to a setting and fan-out, and a clause checker will in general be as follows. The edge A in the convey apparatus (Figure 10) will be the same as one of the edges labeled x and $\bar{x}$ in the setting and fan-out component. To do this the vertices s'_1, s'_2, t'_1 and t'_2 in Figure 11 are coincident with t_1 in the convey apparatus (Figure 10). Of course we also need two independent instances of the edge blocking mechanism in Figure 11 to be attached at the appropriate places in the setting and fan-out component. Now, the edges from t to b_1, b_2, and b_3 in the clause checkers will be the same as the edge labeled B in the conveyor.

Consider a node labeling similar to that in Section 2.

$$D_0D_1\ldots D_{\lceil\log_2(v+1)\rceil}\;\; E_0E_1\ldots E_{\lceil\log_2(w+1)\rceil}\;\; F_0G_0H_0\; F_1G_1\; \alpha_0\alpha_1\;\; JKL\; \beta_0\beta_1\beta_2$$

As before the bits $D_0D_1\ldots D_{\lceil\log_2(v+1)\rceil}$ represent the value of i, the index of the i^{th} variable u_i. The setting and fan-out component plus the corresponding 3-cube blocking components use the bits $F_0G_0H_0\ F_1G_1$. The index of the j^{th} clause-checker will be set, using the bits, $E_0E_1\ldots E_{\lceil\log_2(w+1)\rceil}$. The clause checker will use the bits JKL (3-cube or 2-cube as indicated in Figure 9).

Depending on the instance of L3-SAT that we start with, we map, via the convey apparatus, the edges $\{y, y_{i'}\}$, where $y \in \{s', t'\}$ and $i' \in \{1, 2\}$, is in the setting and fan-out component

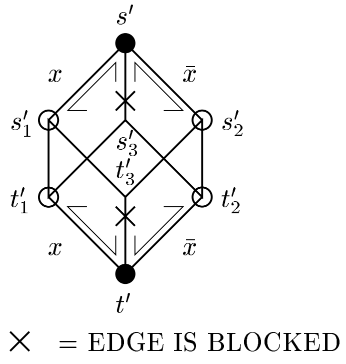


Figure 11: Setting and Fan-Out Component

to the edges $\{t, b_{j'}\}$, where $j' \in \{1,2,3\}$ in the clause checker. Once this mapping has been defined we introduce a convey mechanism for every such pair. In the case when the edge $\{y, y_{i'}\}$ in the i^{th} setting and fan-out component is mapped to the edge $\{t, b_{j'}\}$ in the j^{th} clause checker is given below.

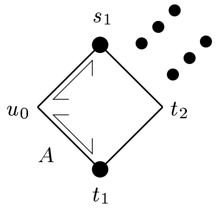


Figure 12: Beginning of the Convey Apparatus. 2-Cube with nodes u_0, t_1, s_1 and t_2.

Note that the partitions of the vertices not identified below consist of all zeros. Note that $bitone(\alpha_0)$ indicates that bit α_0 in the node label is a one. $bitrep(i)$ is the binary representation of the integer i where the length is implicitly $\lceil \log_2(v+1) \rceil$. Similarly the length of $bitrep(y)$ and $bitrep(y_{i'})$ is implicitly five bits "$F_0G_0H_0\ F_1G_1$."

$$
\begin{aligned}
u_0 &= bitrep(i) \ldots bitrep(y) \ldots \\
t_1 &= bitrep(i) \ldots bitrep(y) \ldots bitone(\alpha_{i'}) \ldots \\
s_1 &= bitrep(i) \ldots bitrep(y_{i'}) \ldots \\
t_2 &= bitrep(i) \ldots bitrep(y_{i'}) \ldots bitone(\alpha_{i'}) \ldots
\end{aligned}
$$

Clearly, the difference between nodes u_0 and t_1 is that t_1 has bit $\alpha_{i'}$ with a one. The same holds for the two nodes (s_1, t_2), (s_2, t_3), …, (s_{p-1}, t_p) and (s_p, s_{p+1}). Then the transition from s_1 to s_2, s_2 to s_3 and so on, is by changing a 0-bit to a 1-bit in such a way that we end up in

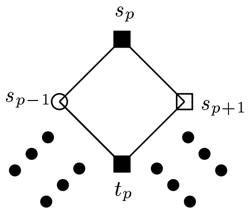


Figure 13: Middle of the Convey Apparatus. 2-Cube with nodes s_{p-1}, t_p, s_p and s_{p+1}.

$$\begin{aligned}
s_{p-1} &= bitrep(i)bitrep(j)bitrep(y_{i'})bitrep(b_{j'})\dots onebit(\beta_{j'})\dots \\
t_p &= s_{p-1} \oplus \dots onebit(\alpha_{i'})\dots \\
s_p &= bitrep(i)bitrep(j)bitrep(y_{i'})bitrep(t)\dots onebit(\beta_{j'})\dots \\
s_{p+1} &= s_p \oplus \dots onebit(\alpha_{i'})\dots
\end{aligned}$$

The next pair, t_{p+1} and s_{p+2} differs from t_p and s_{p+1} in that one 1-bit in $bitrep(y_i')$ is zero. Note that if $y_{i'}$ is s_1' or s_2' then all the bits in the partition for $bitrep(y_{i'})$ will be zero. On the other hand if $y_{i'}$ is t_1' or t_2' then the pairs after t_{p+2} and s_{p+3} will have all the bits in the portion for $bitrep(y_{i'})$ equal to zero.

Note that the difference between the two nodes in (s_{p-1}, s_p), (t_p, s_{p+1}), (t_{p+1}, s_{p+2}), $\dots$, (t_{p+q-1}, s_{p+q}) and (t_{p+q}, u_1) is that the first element of the pair has $bitrep(b_{j'})$ and the second one has $bitrep(t)$.

From here on the subsequent pairs will have one fewer bit one in the portion $bitrep(i)\dots onebit(\alpha_{i'})\dots$. This process ends at t_{p+q-1}, s_{p+q}, u_1 and t_{p+q}.

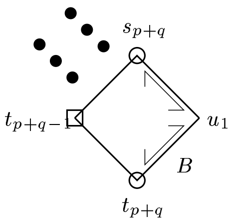


Figure 14: End of the Convey Apparatus. 2-Cube with nodes t_{p+q-1}, s_{p+q}, t_{p+q} and u_1.

$$\begin{aligned}
t_{p+q-1} &= \dots bitrep(j)\dots bitrep(b_{j'})\dots bitone(\beta_{j'})\dots \\
s_{p+q} &= \dots bitrep(j)\dots bitrep(b_{j'})\dots bitone(\beta_{j'})\dots \\
t_{p+q} &= \dots bitrep(j)\dots bitrep(b_{j'})\dots \\
u_1 &= \dots bitrep(j)\dots bitrep(t')\dots
\end{aligned}$$

Vertices t_{p+q} and u_1 correspond to the edge $(b_{j'}, t)$ in the j^{th} clause-checker component.

Lemma 3 *The convey apparatus which connects setting and fan-out component i and clause-checker j does not overlap with any other convey apparatus and nodes u_1, u_2, s_1, s_2, ..., s_{p+q}, t_1, t_2, ..., t_{p+q} of this convey apparatus are internally all unique.*

Proof: Besides the nodes incident to the edges A and B in the conveyors, no other nodes in the conveyors belong to more than one conveyor nor is it present more than once in the same conveyor. The reason for this is that in the first part of the conveyor except for one node, every node has the signature $bitrep(i) \ldots bitrep(y_{i'}) \ldots$ This is unique for every such node in a pair. All of these nodes are unique because each one has one more bit set to one in another portion of the node. The only other node that does not have the above signature is $t_1 = bitrep(i) \ldots bitrep(y') \ldots onebit(\alpha_{i'}) \ldots$ which is different from other nodes in different conveyors because of the bit(s) $bitrep(i)$, $bitrep(y_{i'})$, or $onebit(\alpha_{i'})$.

In the second part of the conveyor all the nodes have the signature $\ldots bitrep(j) \ldots bitrep(b_{i'}) \ldots onebit(\beta_{i'}) \ldots$ or $\ldots bitrep(j) \ldots bitrep(t) \ldots onebit(\beta_i) \ldots$. All of these nodes are unique because each one has a different number of ones. The only exception is (s_{p-1}, s_p), (t_p, s_{p+1}) and (t_{p+1}, s_{p+2}) because the first pair has the same number of ones as the third pair. Note however that these two pairs of vertices are different because the transition from (s_{p-1}, s_p) to (t_p, s_{p+1}) is one $\alpha_{i'}$ and the one from (t_p, s_{p+1}) to (t_{p+1}, s_{p+2}) is on a bit in $bitrep(y_{i'})$.

We now show that a node in the first part of the conveyor is not identical to one in the second part of another conveyor and vice-versus. This follows trivially from the uniqueness of the signature. Convey apparatus which share a common source or destination are disambiguated by either the α or β bits respectively. Thus no two convey apparatus overlap. []

Theorem 6 *Given a partial half permutation routing request the k-pairwise edge disjoint shortest paths problem for the n-cube (k-PAIRWISE-NCUBE-EDSP) is NP-complete.*

Proof: The problem is in NP. For a given pair $X_i = \{s_i, t_i\}$, shortest paths are a permutation of dimensions in the set

$$\{\ell \mid (s_i \oplus t_i)\,\&2^\ell = 2^\ell\}.^1$$

Since there is a maximum number of n edges for any given pair, one may nondeterministically obtain paths for all these elements in $O(kn)$. The maximum length of the construction is a constant number of edges. Therefore determining with a binary trie insert operation that they are all distinct is an $O(kn^2)$ operation. Clearly polynomial in the input length.

By the previous discussion we note that only a polynomial number of pairs are required to reduce from L3-SAT to the present problem. Therefore the problem is NP-complete. []

We have conjectured that the any length edge disjoint path problem possesses no easily constructed blocking mechanism. Therefore the decision problem for any length edge disjoint paths in the n-cube is an open problem.

[1] "&" is the binary "and" operation.

5 Doubled n-Cube, 1-1 Routing Request: Edge Disjoint Shortest Paths

Even with highly restricted pair lengths, less than 4, the problem is NP-complete. The strong conjecture of Szymanski was disproved by Lubiw [25]. Her work informs the following proof of NP-completeness. The routing request herein is less restrictive than the previous problem. In the following a 1-1 routing request is allowed as input.

Consider a doubled n-cube where each edge is replaced with two directed edges: $\{a, b\} \in E$ is replaced by (a, b) and (b, a), $a, b \in V$. The input is as before

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

where $X_i = (s_i, t_i)$, for $1 \leq i \leq p$, and $X_i = (a_i)$ for $p + 1 \leq i \leq p + q$. The problem is to find edge disjoint directed shortest paths in the n-cube for all the pairs X_i that do not include blocking nodes and such that no two such paths have an edge in common. Since there are two edges traversal of both directions is permitted.

In Theorem 7 we show that when $d(X_i) \leq 3$ the edge disjoint directed shortest path problem in the doubled n-cube is NP-complete. We establish this result by reducing the L3-SAT problem to it. Before we present the proof of Theorem 7, we start by discussing the transformation details. As in the previous section, there are three type of components: *setting and fan-out*, *convey* and *clause-checker*. The setting and fan-out component assigns to each variable a value, making two copies of the variable and its negation. The convey apparatus transports the value of a boolean variable or its negation from one edge in the n-cube to another. The clause checker makes sure that a clause is satisfied if at least one of its literals has the value true.

Figure 11 presents the setting and fan-out component to represent two copies of a boolean variable and its complement. The construction consists of a length three pair $\{s', t'\}$ and two blocking edges $(t'', s') = (010, 000)$ and $(s'', t') = (101, 111)$. Note that with the blocking edges there are only two possible edge disjoint shortest paths for the pair: $000 \leftrightarrow 100 \leftrightarrow 110 \leftrightarrow 111$ and $000 \leftrightarrow 001 \leftrightarrow 011 \leftrightarrow 111$. We explain in the next paragraph how we can introduce blocking edges. This is not a trivial matter because node 000 is already occupied by a pair. The construction provides two consistent copies of the value of the variable and its negation. In particular when the edges $(000, 100)$ and $(110, 111)$ are in the path for pair $\{000, 111\}$ the boolean variable u has the value of **false**, and when the edges $(000, 001)$ and $(011, 111)$ are in the path for the pair then the variable u has the value of **true**. The reason one needs only two copies of each value of a variable and its negation is that we are reducing from the L3-SAT problem which has the property that no variable or its negation is in more than two clauses.

Part of the Lubiw [25] construction in Figure 16 suffices as an edge blocking mechanism in the context of the n-cube with doubled edges. Once given the blocking mechanism the remainder of the other constructions parallel those given in Section 4. In Lubiw's [25] construction the pairs are $\{(00000, 11000), (10000, 11001), (10001, 11101), (10011, 10101), (10111, 10001), (10110, 10011), (00110, 10010), (00010, 10000)\}$.

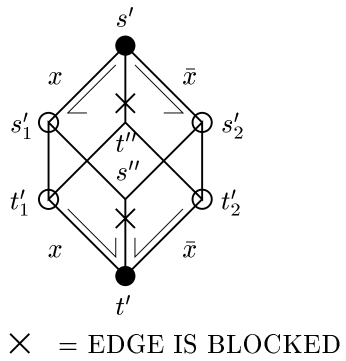


Figure 15: Basic "setting and fan-out" or "variable state" mechanism for the doubled n-cube

Edges labeled $\times$ have an occupied source node or target node in order to block an edge. We claim that the path for pair (s'', t'') must be $s'' = 00000 \to 01000 \to 11000 = t''$. Suppose not. Suppose that the path for this pair is $00000 \to 10000 \to 11000$. This forces the paths $00010 \to 10010 \to 10000$ and $10000 \to 10001 \to 11001$ for the pairs $\{00010, 10000\}$ and $\{10000, 11001\}$. These paths force the paths $00110 \to 10110 \to 10010$ and $10001 \to 10101 \to 11101$. Which implies paths $10011 \to \mathbf{10111} \to \mathbf{10101}$ and $10110 \to \mathbf{10111} \to \mathbf{10011}$. But then the only two possible paths for the the remaining pair $(10111, 10001)$ is $\mathbf{10111} \to \mathbf{10101} \to 10001$ or $\mathbf{10111} \to \mathbf{10011} \to 10001$ are not feasible because at least one edge (in **bold**) is already occupied. A contradiction, therefore the only possible path for pair (s'', t'') must be $s'' = 00000 \to 01000 \to 11000 = t''$. We now show that such a path allows paths for all of the pairs.

$$
\begin{gathered}
00010 \to 10010 \to 10000 \\
00110 \to 10110 \to 10010 \\
10110 \to 10111 \to 10011 \\
10111 \to 10101 \to 10001 \\
10011 \to 10001 \to 10101 \\
10001 \to 11001 \to 11101 \\
10000 \to 11000 \to 11001\ .
\end{gathered}
$$

In Figures 15 and 20, s'' and t'' correspond to the blocking mechanism described above.

Now the convey apparatus for this problem is precisely as shown in Figure 10, albeit the source and target pairs occasionally exchanged for some configurations. Figure 19 illustrates the alternate configuration of the source and target pairs. There are two possible starting points for the convey apparatus contingent upon whether s_1 or t_1 in the convey apparatus overlaps s'_1, s'_2, t'_1 or t'_2. Both the starting configurations are shown in Figure 20.

Now to simplify the discussion we introduce a construction so that the terminal of the revised convey apparatus is always edge B as shown in Figure 10, albeit with directional edges. The directional edges may necessarily be reversed so that edge B may be (u_1, s_{p+q})

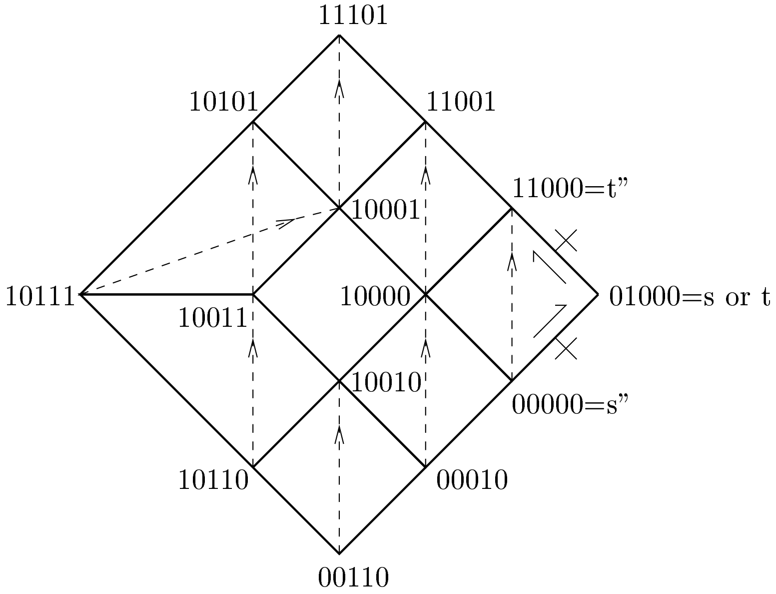


Figure 16: Blocking Mechanism for the Directional edges. $\times$ indicates a blocked edge.

or (u_1, t_{p+q}). Figure 21 indicates how one may mix the parity of the clause-checker. Edge B may be coincident with (b_1, t), (b_2, t) or (b_3, t) in Figure 17. As shown in Figure 21 the edge B in the convey apparatus may connect to edges (s, b_1), (s, b_2) or (b_3, t). This construction allows for the vacant node u_1 to be coincident with s or t in the clause-checker. Note that one may change the construction so that the destination edges are (b_1, t), (b_2, t) or (b_3, t) instead. Nodes are labeled as before with the bit representation

$$D_0 D_1 \ldots D_{\lceil \log_2(v+1) \rceil} \;\; E_0 E_1 \ldots E_{\lceil \log_2(w+1) \rceil} \;\; F_0 G_0 H_0 \; F_1 G_1 H_1 I_1 \; \alpha_0 \alpha_1 \;\; J_0 K_0 L_0 \; J_1 K_1 L_1 M_1 \beta_0 \beta_1 \beta_2$$

As before the bits $D_0 D_1 \ldots D_{\lceil \log_2(v+1) \rceil}$ represent the value of i, the index of the i^{th} variable u_i. The setting and fan-out component plus the corresponding 5-cube blocking components use the bits $F_0 G_0 H_0$ $F_1 G_1 H_1 I_1$. The two blocking components for the setting and fan out component use the bits $F_1 G_1 H_1 I_1$. The bits $J_1 K_1 L_1 M_1$ are used only in the clause checkers depicted in Figure 21 as blocking mechanism. The index of the j^{th} clause-checker will be set, using the bits, $E_0 E_1 \ldots E_{\lceil \log_2(w+1) \rceil}$. The clause checker will use the bits $J_0 K_0 L_0$ (3-cube or 2-cube as indicated in Figures 17, 18 or 21). The bits $\alpha_0 \alpha_1$ and $\beta_0 \beta_1 \beta_2$ are used precisely as in the Section 4 to insure that the convey apparatus remains distinct from the other incident convey apparatus. The bits $\alpha_0 \alpha_1$ and $\beta_0 \beta_1 \beta_2$ are for the setting and fan out and clause checking apparatus respectively.

Depending on the instance of L3-SAT that we start with, we map, via the convey apparatus, the edges $(s', y_{i'})$ or $(y_{i'}, t')$, where $y \in \{s', t'\}$ and $i' \in \{1, 2\}$, is in the setting and fan-out component to the edges $(b_{j'}, t)$ or $(s, b_{j'})$, where $j' \in \{1, 2, 3\}$ in the clause checker. Once this mapping has been defined we introduce a convey mechanism for every such pair. The case when the edge $(s, y_{i'})$ in the i^{th} setting and fan-out component is mapped to the edge $(b_{j'}, t)$

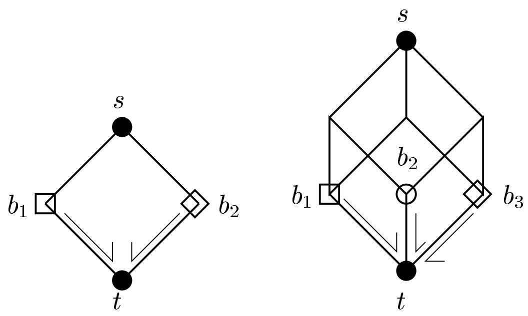


Figure 17: Empty node u_1 coincident with target in clause-checkers for 2 and 3 literals.

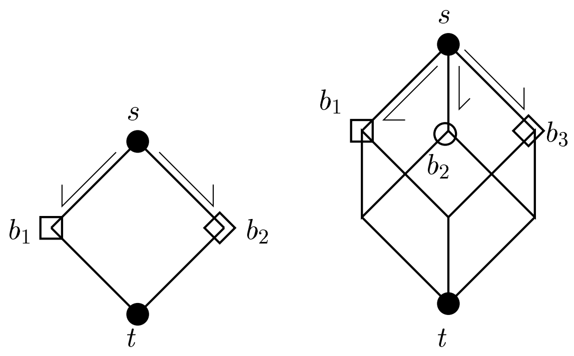


Figure 18: Empty node u_1 coincident with source in clause-checkers for 2 and 3 literals.

or $(s, b_{j'})$ in the j^{th} clause checker is illustrative of the technique. If in a clause checker you need to use edges (s, b_1), (s, b_2) and (s, b_3) or (b_1, t), (b_2, t) and (b_3, t) then it is going to be like in Figure 17. If it is instead required to have (s, b_1), (s, b_2) and (b_3, t) or (b_1, t), (b_2, t) and (s, b_3) then it will be like in Figure 21. In this case one need to add an edge blocking mechanism using the bits $J_1K_1L_1M_1$. The blocked edge in Figure 21 is labeled with the "$\times$."

The procedure for connecting a given edge A to an edge B via the Directional Convey Apparatus given in Figure 19 is identical to that given in Section 4. As in this section auxiliary bits α and β insure that incident convey apparatus do not overlap.

Theorem 7 *Given a 1-1 routing request as input the k-pairwise edge disjoint shortest paths problem for the doubled n-cube (k-PAIRWISE-DNCUBE-EDSP) is NP-complete.*

Proof: The problem is in NP. The argument is precisely as in Theorem 6, albeit with doubled edges and more pairs. Therefore it remains in non-deterministically polynomial time.

Other than the extra dimensions required for the blocking mechanism, the mixed parity construction, and the possible interchange of source and targets in the convey apparatus the proof is similar to Theorem 6.

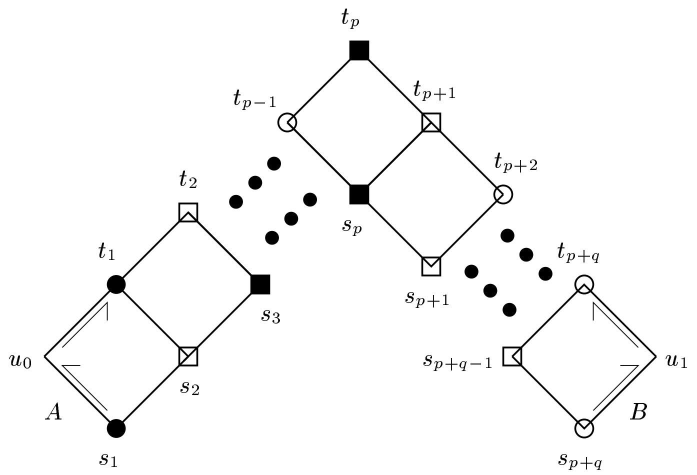


Figure 19: Directional Edge Convey Apparatus

As the number of pairs in the edge blocking mechanism is proportional to the number of variables and clauses, the transformation described above is polynomial in the input size of the L3-SAT problem. Hence the problem is NP-complete. []

6 Grid: Disjoint Shortest Paths

The grid or mesh is an important planar graph structure. In this section we show that the edge disjoint shortest paths and the node disjoint shortest path problems are NP-hard, but in restricted form solvable in polynomial time. We begin with the reduction for the edge disjoint shortest path because it is simpler than the node disjoint one.

6.1 Grid: Edge Disjoint Shortest Paths

The n_0 by n_1 grid, $G = (V, E)$, is a graph structure where vertices are labeled by pairs of integers $(e_0, e_1) \in V$ such that $0 \leq e_i < n_i$ for $i \in \{0, 1\}$. Further, edge $\{(e_0, e_1), (e_2, e_3)\} \in E$, if both $e_0 + \delta = e_2$ and $e_1 + (1 - \delta) = e_3$ hold for $\delta \in \{0, 1\}$. In all of our figures of the grid the first coordinate increases from left to right and the second coordinate increases from top to bottom.

In Theorem 8 we show that the edge disjoint shortest path in the grid graph is NP-hard. We establish this result by reducing the SAT problem to it. Before we present the proof of Theorem 8, we start by discussing the transformation details which are similar to the node disjoint case given in Section 3. There are two types of components: *setting* and *clause-checker*. The setting component assigns to each variable a value. For each variable u_i there is a pair indicated by the x_i's in Figure 22. As it will turn out all such pairs will be connected by a path with one turn or bend. The top-down vertical to horizontal path corresponds to the

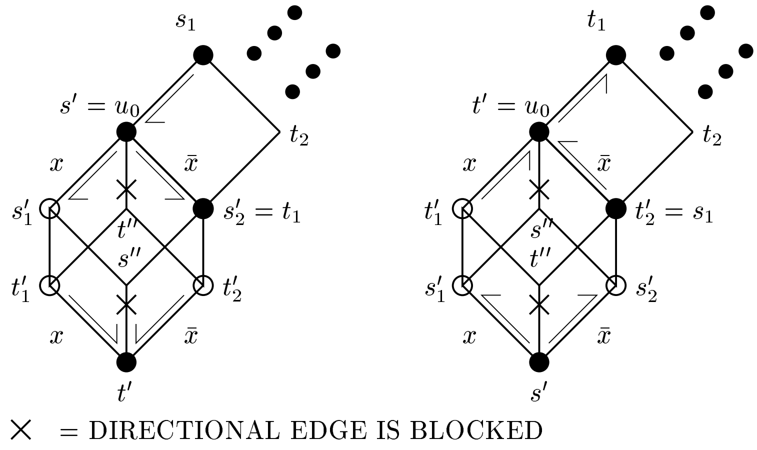


Figure 20: Two possible connections to a "setting and fan-out" component from the start of the convey apparatus.

false value for x_i and the top-down horizontal to vertical path corresponds to the true value for x_i. There is a clause checker for each clause C_j. Each checker consists of two pairs, c_j and d_j (see Figure 22). Further, shortest paths for pairs c_j and d_j have 2 bends or 4 bends and, as we show later on, exist only if the clause they represent is satisfied, i.e. at least one of its literals has the value true. Each pair c_j and d_j must have a path that crosses in at least 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.

The pair x_i is located at (a, b) and $(a + 2, d)$ for some value of b and d to be determined later. The pair c_j is located at (b, a) and $(d, a + 2)$ and the ones for the d_j pairs are located at $(b, a + 2)$ and (d, a) for some other values of b and d. For any pair of vertices we define their *shortest path region* as all the vertices which may lie in a shortest path for the pair. The additional pairs located at the intersection of the shortest path region for the i^{th} variable pair x_i and the shortest path region for the j^{th} 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.

Figure 22 (a) depicts the intersection region of the i^{th} variable with the j^{th} clause pair when the variable appears uncomplemented (Figure 22 (a)) and complemented (Figure 22 (c)) in a clause. Vertical ellipses indicate a continuation of the i^{th} variable and the location for other clauses. Horizontal ellipses indicate the intersection region of the clause with other variable pairs. The middle portion of Figure 22 (**b**) indicates the configuration when the variable is **not** in the clause. Possible paths are indicated by thin solid or dashed lines. The thick horizontal lines in Figure 22 indicates a blocking pair.

In order for shortest paths for pairs c_j and d_j to exist it must be that at the intersection region for the clause pairs and a variable pair, the paths for c_j and d_j cross, i.e., the path for d_j will move to the upper row (dashed line) and the one for c_j will move to the lower row

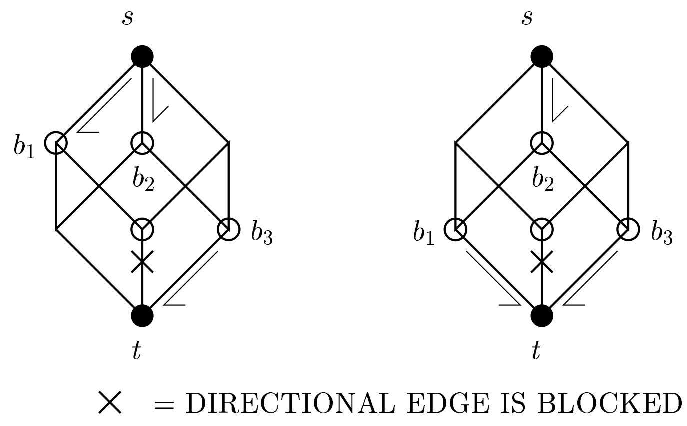


Figure 21: Empty node u_1 connects to source and/or target. Left: Two sources as destination and one target. Right: Two targets and one source as a destination.

(dashed line). As we shall establish, our construction is such that this will only be possible if the value of the variable satisfies the clause.

The intersection region for the i^{th} variable and the j^{th} clause pair is shown in Figure 22 (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". Note that consistency of variable values are enforced by the fact that in all intersection regions there are always two rows for the paths for the pairs, c_j and d_j, going from the left to the right. Given that invariant, there are only two possible paths for the variable pair. If the path for the variable pair for x_i traverses any path other than that shown it would not yield a solution to the k-pairwise problem, because one of the d_j or c_j pairs would not have a shortest path. Figure 23 (a) depicts the crossing of the paths for c_j and d_j when the variable has the value of "true". Figure 23 (b) illustrates that the crossing is not possible when the variable has the value of "false". The intersection region when the negation of a variable is in a clause is given if Figure 22 (c). Using arguments similar to the ones given above, 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. Figure 23 (b) depicts the crossing of the paths for c_j and d_j when the variable has the value of "false". Figure 23 (d) illustrates that the crossing is not possible when the variable has the value of "true".

When a variable is not present in a clause the intersection region is such that the crossing of the path for pairs c_j and d_j is not possible. (Figure 22 (b)). Note that in addition the one unit vertical link shown in Figure 22 (b) may be shared with an adjacent variable; however, this does not cause a problem. Figure 23 (e) and (f) illustrate that fact that it is not possible for the paths for pairs c_j and d_j to cross no matter what the value the variable is assigned. The pairs c_j and d_j need only cross at one variable pair in the proper state. This directly parallels the fact that only one variable in a clause need to have the correct value for the clause to be satisfied.

Each pair c_j and d_j must have a path that crosses in at least one position. This is due

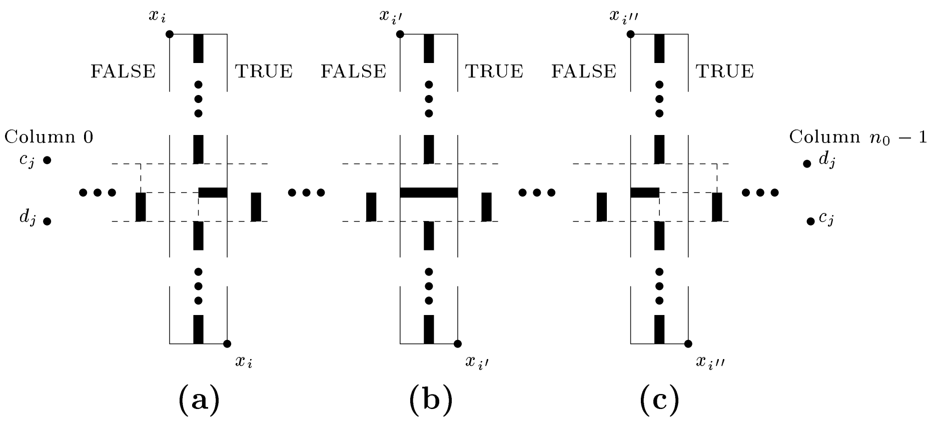


Figure 22: (a) : Variable u_i is (uncomplemented) in clause C_j. (b): Variable (uncomplemented) $u_{i'}$ or (complemented) variable $\bar{u}_{i'}$ are **not** in clause C_j. **(c)**: Negation of variable $u_{i''}$ is in the clause C_j.

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. The pairs are separated vertically by two units. The next figure demonstrates the only possible manner in which the pairs can cross at a given variable in the clause. The variable pair path for x_i must be in the correct state in order for the crossing to occur. Figure 23 (a) and (b) depicts the two possible paths crossing configurations for the clause pairs c_j and d_j as shown in Figure 22 (a) and (c) when the value of the variable satisfies the clause. Figures 23 (c) and (d) do not allow any crossing of the clause pairs c_j and d_j. This is because the value of the variable does not satisfy the clause and for each one unit step from left to right the path for c_j remains above d_j. Figures 23 (e) and (f) demonstrate that no crossing of both pairs can occur when the variable is not in the given clause.

Note that two or more possible crossings may be available for c_j and d_j (see c_1 and d_1 in Figure 24).

When at least one variable or its negation is in the clause a crossing may occur and when this occurs 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 that case there are no edge disjoint paths for both the pairs c_j and d_j.

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 Figure 24 with a satisfying assignment $u_2 =$false and $u_1 = u_3 =$true.

The instance $\{\bar{u}_1, u_2\}$, $\{u_1, \bar{u}_2\}$, $\{u_1, u_2\}$ and $\{\bar{u}_1, \bar{u}_2\}$ is not satisfiable and the problem instance generated from it is depicted in Figure 25 with 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 j^{th} 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 n_1 is a function of the

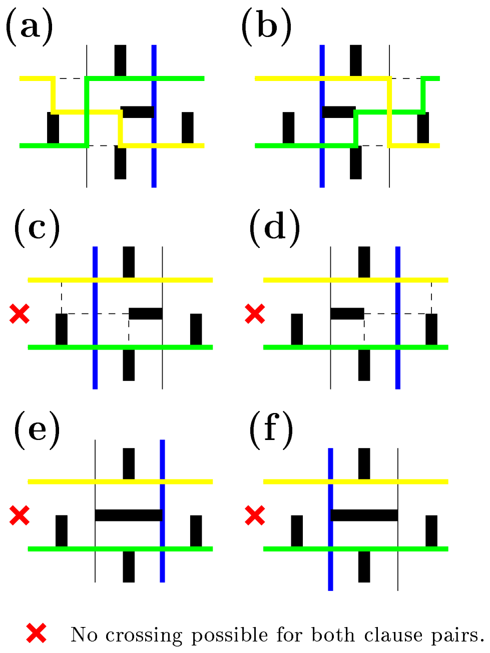


Figure 23: (a): Crossing of clause pairs is possible when path x_i traverses the rightmost vertical path and the variable (not its negation) is the clause. (b): Crossing of clause pairs is possible when path x_i traverses the leftmost vertical path and the negation of the variable is in the clause. (c): A crossing is not possible when path x_i traverses the leftmost vertical path and the variable is the clause. (d): A crossing is not possible when path x_i traverses the rightmost vertical path and the variable's negation is the clause. (e): A crossing is not possible when the variable is not in the clause and the rightmost vertical path is taken for pair x_i. (f): A crossing is not possible when the variable is not in the clause and the leftmost path is taken for pair x_i.

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

When the additional pairs are introduced, the net effect is to preclude paths not shown in the Figure 22. They are defined as follows:

The vertical pairs between the variable pairs x_i and x_{i+1} are the pairs $\{(4i, 3a+2), (4i, 3a+3)\}$, for $0 \leq b < w$. The vertical pairs inside each variable x_i are the pairs $\{(4i-2, 3b), (4i-2, 3b+1)\}$, for $0 \leq b \leq w$.

Let $h_1 < h_2 < \ldots < h_r$ be the variables that clause j contains. Further let $d_1, d_2, \ldots, d_r \in \{0, 1\}$ be defined as follows: If the variable u_i, $i = h_k$ (necessarily the variable u_i is in the clause j) is the negation then $d_k = 0$ otherwise $d_k = 1$. Now we write down the pairs for clause j as follows.

$$\{(4h_1 + d_1 - 3, 3j - 1), (4h_1 + d_1 - 2, 3j - 1)\},$$

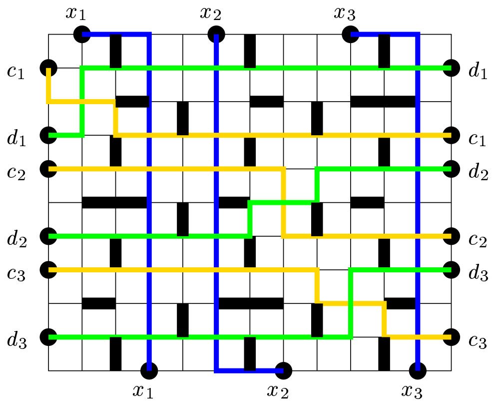


Figure 24: Problem instance generated from $\{u_1, u_2\}$, $\{\bar{u}_2, \bar{u}_3\}$, $\{u_3, \bar{u}_1\}$. The solution is shown for $u_2 =$False and $u_1 = u_3 =$True.

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

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

It is not clear whether the k-pairwise edge disjoint shortest paths decision problem is in NP. The difficulty in establishing this result 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 is $p \log(n_0 + n_1)$, where p is the number of pairs. Therefore the amount of information to represent the path may not be polynomial in the input length.

Theorem 8 *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 SAT is satisfiable. Given any assignment that satisfies all the clause, we just construct the corresponding path for each pair x_i and the paths for all the c_j and d_j pairs exist because each clause is satisfied. Similarly, if edge disjoint shortest paths exist, then each pair x_i is connected by a path that corresponds to the value of "true" or "false" for each value and all these values satisfy all the clauses.

Further the reduction is polynomial as the number of pairs introduced is $O(vw)$. The representation of pairs in the grid is no more than $O(vw \log(v + w))$ as well and thus the reduction is polynomial and therefore the problem is NP-hard. []

When $d(X_i) \leq 2$ the same algorithm that we introduced for the n-cube in Section 4 works for this problem because there are only two possible paths for each pair. We have proved NP-completeness for edge disjoint shortest paths in the grid without a restriction on the number

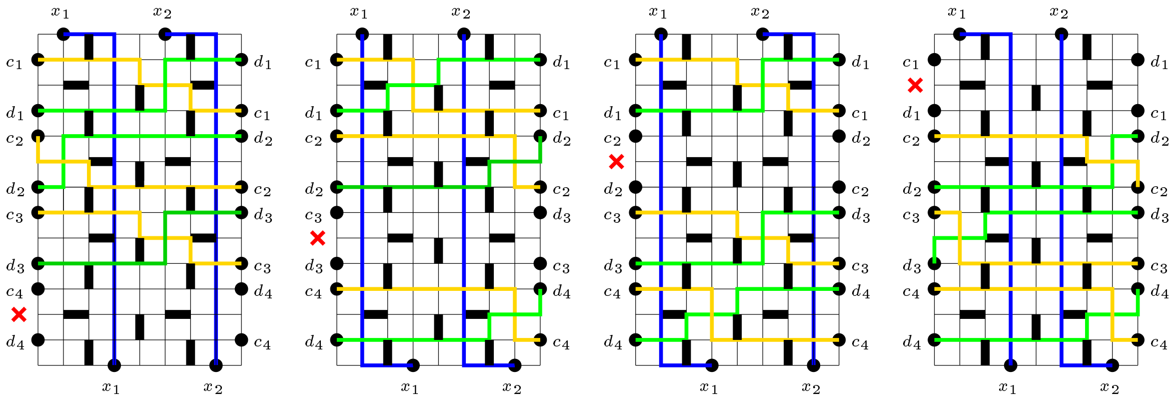


Figure 25: Problem instance generated from $\{\bar{u}_1, u_2\}$,$\{u_1, \bar{u}_2\}$, $\{u_1, u_2\}$ and $\{\bar{u}_1, \bar{u}_2\}$ has **no** solution. All of the configurations of true and false paths for pairs x_1 and x_2 are shown. A "$\times$" symbol indicates that both pairs do not have shortest paths.

of turns. Note that much of the previous literature concerned with a constant number of turns is subsumed by this construction. In fact, the algorithm also works 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.

We have established the NP-completeness for edge disjoint shortest paths in the grid without a restriction on the number of bends. A careful examination of the reduction for Theorem 8 reveals that where there is a solution every pair is connected by a path with at most three bends. Therefore we have the following corollary.

Corollary 4 *With at most three bends the k-pairwise edge disjoint shortest paths problem for the grid (k-PAIRWISE-GRID-EDSP-3TURNS) is NP-complete.*

Proof: To show that the problem is in NP is straight forward. One non-deterministically determines how many bends there are and where the bends are for each pair. Pairwise evaluation of each segment for overlap with another segment can be performed in polynomial time with respect to the number of pairs. Therefore this restricted problem is in NP. The result now follows from the comments that precede the corollary. []

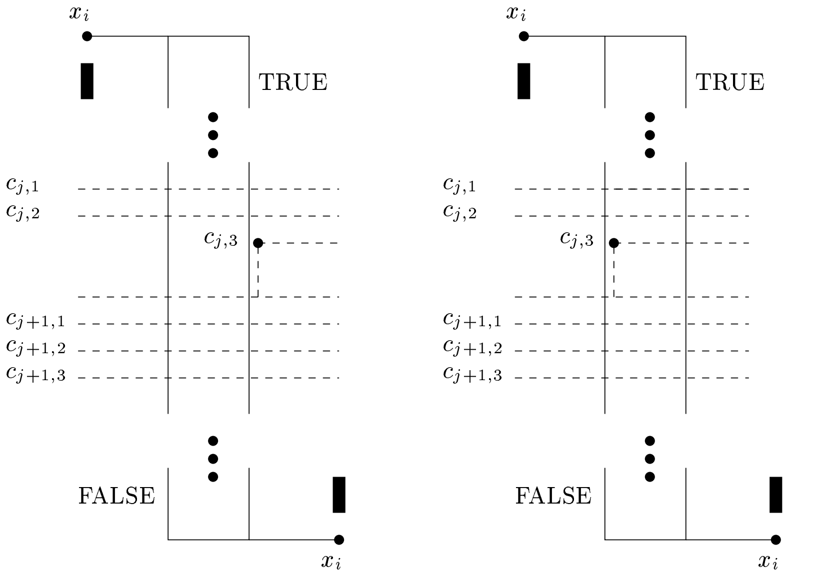
When we restrict further the possible paths to allow for only one bend per path, the problem becomes polynomially solvable. The reason for this is that every pair has two possible paths and it is simple to check for interference between them. Therefore the same algorithm as the one in Section 4 works for this case. We summarize this result in Theorem 9.

Theorem 9 *The k-pairwise edge disjoint shortest paths problem with only 1 bend allowed per path is polynomially solvable for a partial half permutation routing request in the grid.*

Proof: Note that a one bend configuration is either overlapping or not and hence one may apply the same logic as applied in algorithm 2-EDSP and the proof is similar to the one for Theorem 5. []

While the construction outlined in Theorem 8 does not work in the 1-turn case propounded in [21] it does function with simple changes for four or more constant turns per path with one source or destination at a given vertex. None of the previous theorems cover the case when one limits each path to have at most two turns. For this case the number of different paths for a pair of vertices is not bounded by a constant. We now show that even this problem is an NP-complete problem. To prove this result we polynomially reduce L3-SAT to the edge disjoint shortest paths problem in the grid when every path is allowed at most two bends.

The following construction suffices for two constant turns. In Figure 26 the dashed lines designate the only possible paths. Basically there are two possibilities for each construction. Either a path comes in above the level B in I and level B' in I' in Figures 27 or 28 respectively or below it. If it comes in below one of the variables is in the proper state, otherwise, the variable may or may not have been in the proper state.



CONSTRUCTION IF $\bar{u}_i$ IS THE 3rd VARIABLE IN CLAUSE C_j.

CONSTRUCTION IF u_i IS THE 3rd VARIABLE IN CLAUSE C_j.

Figure 26: Placement of $c_{j,3}$ leftmost pair, dependent upon whether the variable u_i or its negation is in clause C_j. Paths for extrinsic pairs $c_{j,1}$ and $c_{j,2}$ are shown here. Both $\{(4i,1),(4i,2)\}$ and $\{(4i+3,n_1-2),(4i+3,n_1-3)\}$ are the vertical blocking pairs shown in bold.

Similar to the previous construction, each variable x_i pair is placed as shown in Figure 26 vertically. Now consider the placement of the clause pairs in conjunction with the variable pairs x_j. Due to the fact that each clause may be constrained by L3-SAT reduction so that only two or three pairs are required: $c_{j,1}$, $c_{j,2}$, and conditionally $c_{j,3}$. So two constructions are required. Figure 27 for size three clauses and Figure 28 for size two clauses. Note that always at least one path must be incident from the left and no more than two or one respectively incident from above. This requires that at least one variable in the clause must be in the proper state. There are other cases wherein all paths are incident from the left; however, we need only demonstrated that a solution exists and we need not enumerate all paths incident from the left in the proof.

At the rightmost boundary of the grid the clauses are arranged as shown in Figure 27. This configuration admits at most one of the pairs $c_{j,1}$, $c_{j,2}$, and $c_{j,3}$ from the left and at most two from above. By enumeration of the possible paths one notes that the structure functions for two constant turns. In addition if the clause path $c_{j,k}$ goes down in a variable to the right of its starting vertex it will create one turn in excess of the constant allowed. Therefore such paths are precluded. The placement of the horizontal access to the structure in Figure 27 insures that one path is incident from the left, A and two from the left above B and B'.

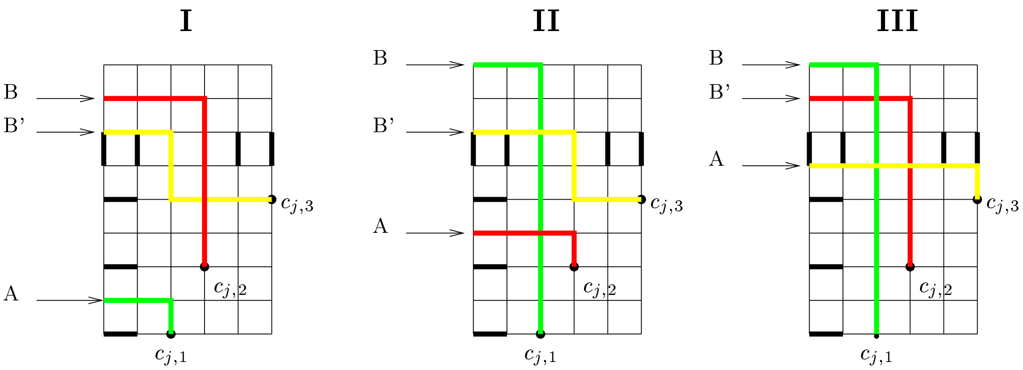


Figure 27: Size three clause: The three valid rightmost configurations in the grid of $c_{j,1}$, $c_{j,2}$, and $c_{j,3}$.

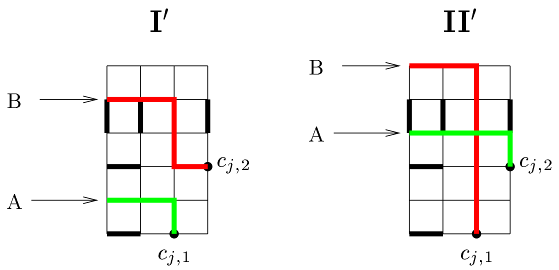


Figure 28: Size two clause: The two valid rightmost configurations in the grid of $c_{j,1}$, $c_{j,2}$.

A trivial modification admits any constant number of turns greater than two. Therefore for shortest paths, the problem is NP-complete for a constant number of turns greater than one. A trivial modification of this structure yields a constant number of turns for any length path. However, Leeuwen has already demonstrated this, namely node disjoint paths for any length path in the grid the problem is NP-complete [22]. To show Theorem 10 one merely adds a constant number ($t-2$ turns) of turns before the leftmost nodes $c_{j,1}$, $c_{j,2}$, and $c_{j,3}$.

This results in a slightly larger spacing between the variable pairs x_j. In the two turn case the following construction suffices. The variable pairs introduced when $0 \leq i < w$ are $\{(4i, 0), (4i+3, n_1-1)\}$. To insure variable consistency, introduce the following two pairs for the i^{th} variable $\{(4i, 1), (4i, 2)\}$ and $\{(4i+3, n_1-2), (4i+3, n_1-3)\}$ are the vertical blocking pairs shown in bold in Figure 26.

If variable u_{h_k} is the k^{th} 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 at most three variables in the clause C_j are represented by pairs.

$$c_{j,k} \quad \text{pair is} \quad \{(4h_k + 1 + \delta_k, 8j + 2 + k), (n_0 - (3-k) - 1, 8(j+1) + 2 - (k-1))\}$$

As indicated in the formal specification of the placement of the pairs the pair $c_{j,3}$ is the rightmost of the pairs $c_{j,1}$, $c_{j,2}$.

The reduction is to L3-SAT. Consider the above described construction. Starting at the leftmost node of pairs $c_{j,1}$, $c_{j,2}$, or $c_{j,3}$, note that if the path goes downward then it is clear that it introduces at least one additional turn. In order for such a path to move downward in a column other than the starting column at least two additional turns will be introduced.

With respect to Figure 27 there are precisely three cases enumerated. While there are other solutions none admit more than two paths entering from above. The cases shown are conditioned upon which ending vertex path A meets. In all cases path A has precisely 2 turns. In case I, path A connects to $c_{j,1}$. The figure also illustrates how paths incident from B and B' may connect to $c_{j,2}$ or $c_{j,3}$ with one or two additional turns respectively. There are two possible leftmost entry points for the path A, but all result in a path for A $(c_{j,1})$ with precisely two turns. In case II, again path B' is chosen so that it connects to $c_{j,3}$ and path B connects to $c_{j,1}$. These paths have two and one turns respectively. For case III the path B connects to $c_{j,1}$ and path B' connects to $c_{j,2}$. Both paths make precisely one turn.

In the case of a size two clause the case I' and II' in Figure 28 similarly requires two turns from edges incident from A and one or two turns from edges incident from B.

Since the reduction outlined above follows similar patterns as the previous ones, we state our result omitting some simple details.

Theorem 10 *The edge disjoint shortest paths problem for k pairs of vertices is NP-complete for a number of turns less than or equal to $t = 2$.*

Proof: The problem is in NP. Since there are at most three segments per path, we just nondeterministically guess the 0,1, or 2 possible turns. Determining pairwise overlap of the segments is polynomial in the number of input pairs. Therefore, the problem is in NP.

The previous discussion shows that L3-SAT polynomially reduces to to the edge disjoint shortest paths problem for k pairs when each path has at most 2-turns. The number of pairs required is $O(v + w)$ where v and w are the number of variables and clauses respectively. Therefore the reduction is clearly polynomial in the size of the original problem instance and hence the problem is NP-complete. []

A slight modification, namely the introduction of one turn at the beginning of the leftmost $c_{j,k}$ pairs allows at most three turns per path. Although this is not necessary it demonstrates that either the construction of Theorem 8 or 10 can be used to obtain the result of Corollary 4. A similar modification but introducing a $t-2$ turns at the beginning of the leftmost $c_{j,k}$ pairs allows at most t turns per path. So the problem remains NP-complete when one limits the total turns allowed per path to any constant number t.

This structure yields a constant number of turns for any length path. However, Leeuwen has already demonstrated this, namely node disjoint paths for any length path in the grid the problem is NP-complete [22]. To establish this result we use the previous reduction and merely add pairs to force a constant number $(t-2)$ of turns before the leftmost nodes $c_{j,1}$, $c_{j,2}$, $c_{j,3}$ and x_i. One also needs to change the spacing of the pairs in this case. For brevity we do not discuss this further.

6.2 Grid: Node Disjoint Shortest Paths

We show that the node disjoint shortest problem in the grid is an NP-complete problem. We reduce SAT to this problem. Our reduction is similar in nature to the one in Section 6.1, but it is slightly more complex. The main difference is that crossing of the shortest paths is not allowed because they are not node disjoint. 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 constructions in which the direction of a path in one part of the grid force a certain types of path in another remote part of the grid. This communication of state throughout the graph is qualitatively indicative of NP-hardness or NP-completeness.

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 certain pattern. In Figure 29 we depict the general architecture of our construction. The chains x_1, x_2, x_3, x_4... are used to represent the value of the variables. The paths for x_1, x_2 and x_4 indicate a "true" value of the variable, and the one for x_3 indicates a "false" value. As we show later on, these values are consistently communicated diagonally across the grid. In 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 the clause-checker. Then in the next region below that we introduce another pair for the second clause which has similar properties. This is the second clause checker, and so on. It is spaced sufficiently so that each clause checking pair does not overlap with any other. This insures that the clause checking pairs interact only with horizontal and vertical pairs shown and the variable state information (i.e. x_1, x_2, x_3 and x_4).

Once that the direction of the paths for the variables have been decided, shortest paths for the clause-checkers may or may not exist. If they exist then they will be of the form given in Figure 29 for the pair c_1. The form is that as long as a variable does not satisfy a clause (either because the variable is not in the clause or the variable has not been assigned a value that satisfies the clause) then the path for c_1 will advance to the other side of the chain representing the variable but its vertical distance will only drop by a value proportional to the number of variables. On the other hand when a variable has a value that satisfies the clause then it may escape vertically for a distance proportional to the number of variables and then it will be able to connect to the other c_1 point. The scenario when the clause path connects is shown with the dashed and dotted line in Figure 29. If none of the values for the variables satisfy the clause the path for c_1 will end up at a region where it is be impossible to reach the other endpoint of c_1. This clause path which cannot connect is marked by "×" in Figure 29.

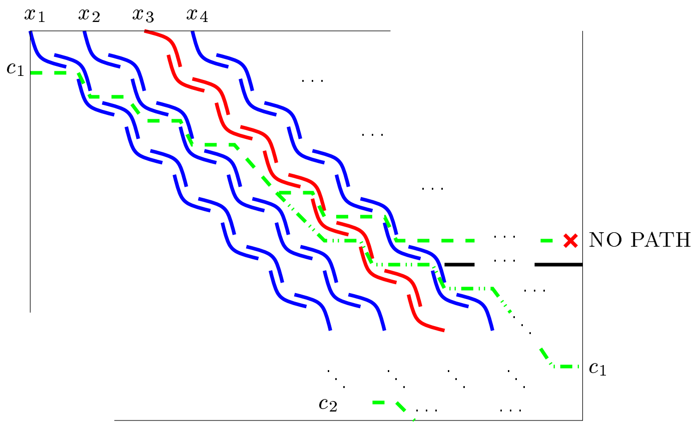


Figure 29: Overall Architecture: Diagonal pairs are variable states. Dashed lines show possible paths. The upper path cannot connect (a "×" indicates no connection is possible). The lower path can connect.

Specifically the diagonal pairs are spaced in a manner that preserves the state of the variable pairs. The i^{th} variable has the following pairs for all non-negative integers a such that both end vertices are within the bounds of the grid:

$$
\begin{aligned}
P_1(i,a) &= \{(24(i-1)+12a, 12a), (24(i-1)+12(a+1)-2, 12(a+1)-2)\} \\
P_2(i,a) &= \{(24(i-1)+12a+1, 12a+1), (24(i-1)+12(a+1)-3, 12(a+1)-3)\} \\
P_3(i,a) &= \{(24(i-1)+12a+6, 12a+6), (24(i-1)+12(a+1)+4, 12(a+1)+4)\} \\
P_4(i,a) &= \{(24(i-1)+12a+7, 12a+7), (24(i-1)+12(a+1)+3, 12(a+1)+3)\}
\end{aligned}
$$

Now the consistent state can be defined by the following two scenarios: (1.) both paths for pairs $P_1(0,0)$ and $P_2(0,0)$ contain nodes which are in column 6 and below node $(24(i-1)+12a+6, 12a+6)$ or (2.) both paths for pairs $P_1(0,0)$ and $P_2(0,0)$ contain nodes which are in row 6 and to the right of node $(24(i-1)+12a+6, 12a+6)$. Inductively one can show that only one of two possibilities exists for the variable state. For each clause j the clause pair c_j is positioned as follows.

$$\{(0, 24v(j-1)+6), (n_0, 24vj+5)\}$$

Note that no clause path may possibly overlap another. Even though this simplifies analysis one may overlap the clause paths; however, it does not effect the polynomial nature of the reduction from SAT. Therefore placement of the clause pairs is such that they do not overlap at all.

Now for a given clause j assume that the variables $h_1 < h_2 < \ldots < h_r$ or their negations are in the clause. In addition let $\ell_1 < \ell_2 < \ldots < \ell_{v-r}$ be the other variables not in the clause j. For the appropriate b and variable i the the leftmost point of the rightmost horizontal pair is always given by $(24(i-1)+12b+10, 12b+6)$. When the variable is not in the clause the rightmost point of the leftmost horizontal pair is given by $(24(\ell_k-1)+12(b+2)-7, 12(b+2)-2)$. Further when the negation of the variable h_s is not in the clause the vertex is $(24(h_s-1)+12(b+2)-8, 12(b+2)-2)$ and when the variable h_s is in the clause the vertex is $(24(h_s-1)+12(b+2)-13, 12(b+2)-8)$. The remaining pairs are simply given in a manner that prevents the clause pair from moving below the designated level as shown in Figure 38.

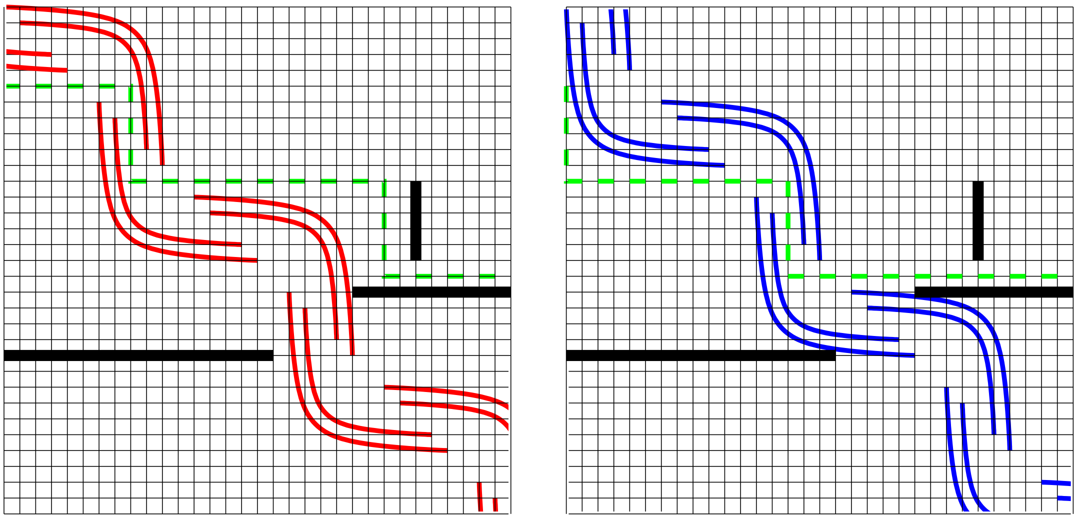
From left to right in the grid, as the clause path crosses a variable construction that is not in the clause it moves downward in the vertical direction by at most twelve units per each variable. The same holds when a variable or its complement does not satisfy the clause. However, if the value of the variable or its negation satisfies the clause, then either the vertical movement may be more than twelve units or the path may only drop the usual twelve vertical units. The Figures 30, 33, 34 and 35 represent the intersection of the clause path with the variable path.

Figure 30 shows the case when a variable or its complement is not in the clause. Figure 33 depicts the case when the variable's negation is in the clause. The case when the uncomplemented variable is in the clause is given in Figures 34 and 35. Additional blocking pairs are indicated with solid bold lines. The dashed lines only indicate one or more of the possible paths. Figures 30, 33, 34 and 35 actually admit more paths than those that are shown with the dashed lines; however the exit location is invariant. To facilitate later proof both the true and false variable states for variable x_i are shown by the curved lines. These also are indicative of a collection of possible paths.

In Figures 30, 33, 34 and 35 the region between two horizontal lines (for the blocking pairs) is called the *passage to the lower region*. It is a step which exceeds a twelve vertical unit step per variable. In Figures 30 (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 paths for the four pairs with endpoints on the opposite sides of the passage to the lower region do not leave enough space for the clause pair. This is illustrated in Figure 31 (a).

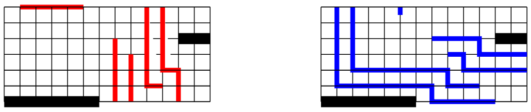
Figure 33 shows our construction when the negation of a variable is in clause C_j. In Figure 33 (a) shows two possible configurations for the clause path. One of these paths goes through the passage to the lower region. This is possible because the value of the variable satisfies the clause. The passage region is shown in more detail in Figure 36 (a). The configuration in Figure 33 (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.

The reason for this is simple. In the true state for the variable the paths associated with



(a) **(b)**

Figure 30: Variable u_i or its complement is **not** in clause C_j. (a): Variable u_i has a true value. (b): Variable u_i has a false value.



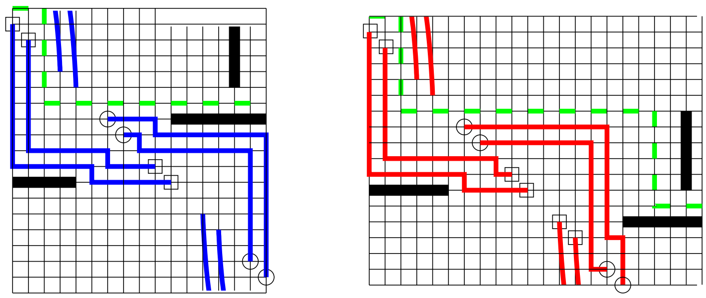
(a) **(b)**

Figure 31: Specific paths when variable or its complement u_i is **not** in clause C_j. (a): Variable u_i has the value true. (b): Variable u_i has the value false.

the pairs labeled with "□" must go below paths for pairs labeled with "∘." There are at most two nodes from the left portion of the right horizontal pair to the shorter pair labeled with "∘." And since both of the pairs labeled "□" must traverse these nodes thus no path, including the clause path and the longer of the pairs labeled with "∘" will move downward. Thus the pairs labeled with "∘" block any exit by the clause path from above. This is illustrated in Figure 32 (a).

In Figure 34 the clause path must exit to the right at the level indicated. The reason for this is that the node disjoint shortest path for the pairs labeled with a "∘" must be to the right of all the pairs labeled with "□." The passage region is depicted in more detail in Figure 32 (b) and shows that clause path must exit in the upper region.

Figure 35 shows two possible paths when the variable u_i is in clause C_j. In Figure 35 (a) the path exists through the upper region and in Figure 35 (b) it exits through the bottom region. Figure 36 (b) shows the passage to the lower region in more detail.



(a) **(b)**

Figure 32: (a): Specific path when negation of variable u_i is in C_j and variable has the value of true. (b): Specific path when varible u_i is in C_j and the variable has the value false.

Figure 38 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. It shows how after a clause path moves to the lower region it may take twelve vertical steps per variable traversed. In this example the clause path move down as u_1 is false and then connects to its appropriate destination. Figure **??** shows the same clause $C_1 = \{\bar{u}_1, u_2\}$ when the values of the variables are not set correctly ($u_3 =$false and $u_1 = u_2 =$true). Note there is no method for the clause path c_j to pass to the lower region and thus there is no solution for this variable state configuration.

Theorem 11 *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: Note that the grid in the construction described is of dimension of $O(vw) \times O(vw)$. Clearly the construction is polynomial in the input length. The reduction is from SAT.

The only way that a path may escape from the twelve 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 only connects if one of its boolean variable paths allows it to do so. If a variable in the SAT clause is in the clause and of the the appropriate value, the Figures 35 (b) and 33 (a) and their attendant discussions 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 twelve vertical units for each variable state construction crossed horizontally. The path for the clause shown in Figure 30 indicates that no more than 12 units is required for each individual variable

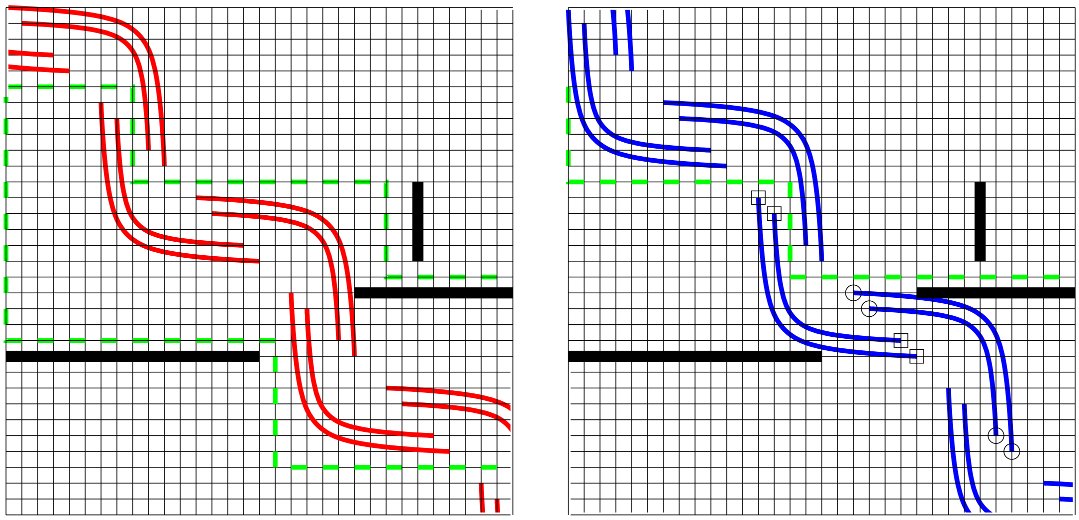


Figure 33: Negation of Variable u_i is in Clause C_j. (a): Variable u_i has the value false. (b): Variable u_i has the value true.

crossed. Therefore if any one construction allows the clause path to move into the lower region it may connect to the terminus of the clause path at the far right of the grid.

Further 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. []

7 Conclusion

The paper has described the k-pairwise problem for both the node and edge-disjoint shortest paths. While algorithms are not presented, polynomial transformations are given from known NP-complete problems thus proving NP-completeness or NP-hardness. Given current assumptions about the equivalence classes P and NP this indicates that the likelihood of finding polynomial algorithms for these topologies is negligible for these problems.

References

[1] A. Aggarwal, J. Kleinberg, and D. P. Williamson. Node-Disjoint Paths on the Mesh and a New Trade-Off in VLSI Layout. *Siam J. Comput.*, 29(4):1321–1333, 2000.

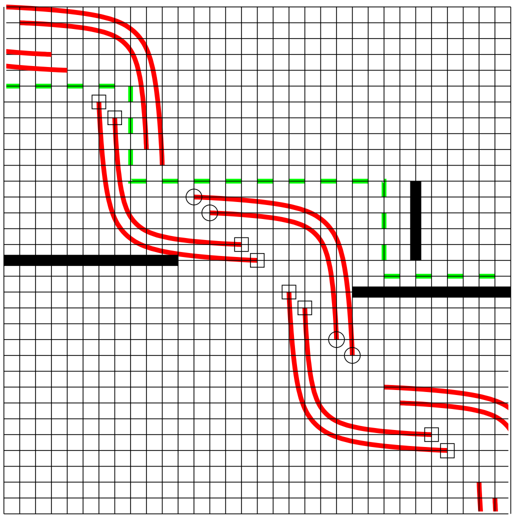


Figure 34: Variable u_i has the value false and it is in the clause C_j.

[2] S. B. Akers and B. Krishnamurthy. A Group-Theoretic Model for Symmetric Interconnection Networks. *IEEE Transactions on Computers*, 4(38):555–566, April 1989.

[3] S. A. Cook. The Complexity of Theorem-Proving Procedures. *Proc. 3rd Ann. ACM Symp. on Theory of Computing*, pages 151–158, 1971.

[4] W. F. Dowling and J. H. Gallier. Linear-Time Algorithms for Testing the Satisfiability of Propositional Horn Formulae. *J. Logic*, 3:267–284, 1984.

[5] A.-H. Esfahanian. Generalized Measures of Fault Tolerance with Application to N-Cube Networks. *IEEE Transactions on Computers*, 38(11):1586–1591, November 1989.

[6] S. Even, A. Itai, and A. Shamir. On the Complexity of Timetable and Multicommodity Flow Problems. *SIAM Journal on Computing*, 5(4):691–703, December 1976.

[7] S. Gao, B. Novick, and K. Qui. From Hall's matching theorem to optimal routing on hypercubes. *Journal of Combinatorial Theory, Series B*, 74(2):291–301, November 1998.

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

[9] W. H. Gates and C. H. Papadimitriou. Bounds for Sorting by Prefix Reversal. *Discrete Math*, 27(1):47–57, 1979.

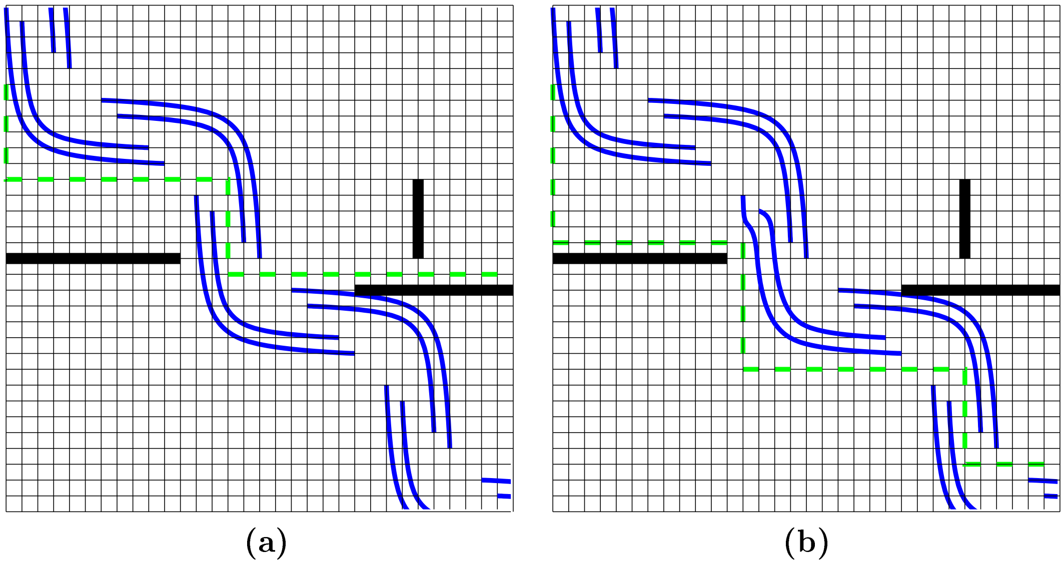


Figure 35: Variable u_i is in a Clause C_j. u_i has the value true and the path for clause C_j exits through the (a): upper portion. (b): lower portion.

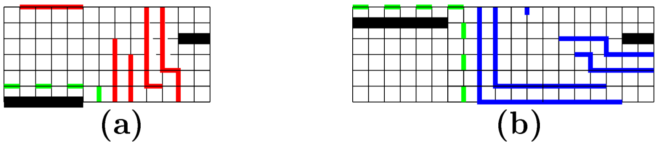


Figure 36: Clause path (dashed line) passage to the lower region. (a): Variable $\bar{u}_i$ is in clause C_j and has a false value. (b): Variable u_i is in clause C_j and has the true value.

[10] T. F. Gonzalez and F. D. Serena. n-Cube Network: Node Disjoint Shortest Paths for Pairs of Vertices. Technical Report TRCS-2001-12, University of California at Santa Barbara, July 2001.

[11] T. F. Gonzalez and F. D. Serena. Node Disjoint Shortest Paths for Pairs of Vertices in an n-Cube Network. In *Proceedings of the International Conference on Parallel and Distributed Computing and Systems (PDCS2001)*, pages 278–282. IASTED, 2001.

[12] T. F. Gonzalez and F. D. Serena. n-Cube Network: Node Disjoint Shortest Paths for Maximal Distance Pairs of Vertices. Technical Report TRCS-2002-15, University of California at Santa Barbara, May 2002.

[13] Q.-P. Gu and S. Peng. An Efficient Algorithm for Node-to-node Routing in Hypercubes with Faulty Clusters. *The Computer Journal*, 39(1):14–19, 1996.

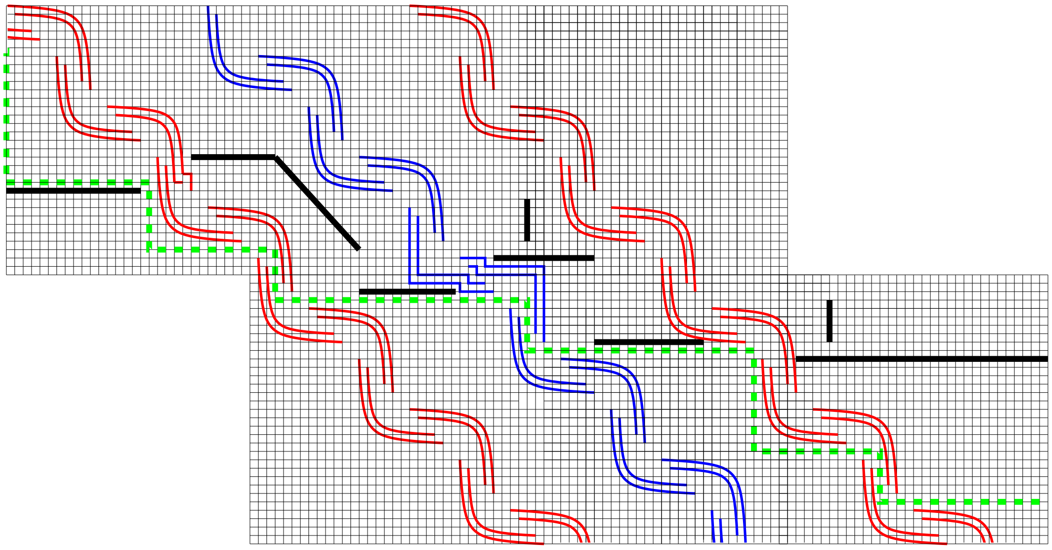


Figure 37: Segment of clause $C_1 = \{\bar{u}_1, u_2\}$, The solution is shown for $u_1 = u_3$ =false and u_2 =true.

[14] Q.-P. Gu and S. Peng. k-Pairwise Cluster Fault Tolerant Routing in Hypercubes. *IEEE Transactions on Computers*, 46(9), September 1997.

[15] Q.-P. Gu and S. Peng. Node-to-Set and Set-to-Set Cluster Fault Tolerant Routing in Hypercubes. *Parallel Computing*, 24:1245–1261, 1998.

[16] Q.-P. Gu and H. Tamaki. Routing a Permutation in the Hypercube by Two Sets of Edge-Disjoint Paths. *Journal of Parallel and Distributed Computing*, 44:147–152, 1997.

[17] J. Hopcroft and R. M. Karp. An $n^{2.5}$ Algorithm for Maximum Matchings in Bipartite Graphs. *SIAM J. Computing*, pages 225–231, 1973.

[18] D. F. Hsu. On Container Width and Length in Graphs, Groups and Networks. *IEICE Trans. Fundamentals*, E77-A(4):668–680, 1994.

[19] S. L. Johnsson and C.-T. Ho. Optimum Broadcasting and Personalized Communication in Hypercubes. *IEEE Transactions on Computers*, 38(9):1249–1268, September 1989.

[20] R. Karp. On the Computational Complexity of Combinatorial Problems. *Networks*, 5:45–68, 1975.

[21] R. Karp, F. T. Leighton, R. L. Rivest, C. D. Thompson, U. Vazirani, and V. Vazirani. Global Wire Routing in Two-Dimensional Arrays. *IEEE Comput. Soc. Press*, pages 453–459, 1983. 24th Annual Symposium on Foundations of Computer Science.

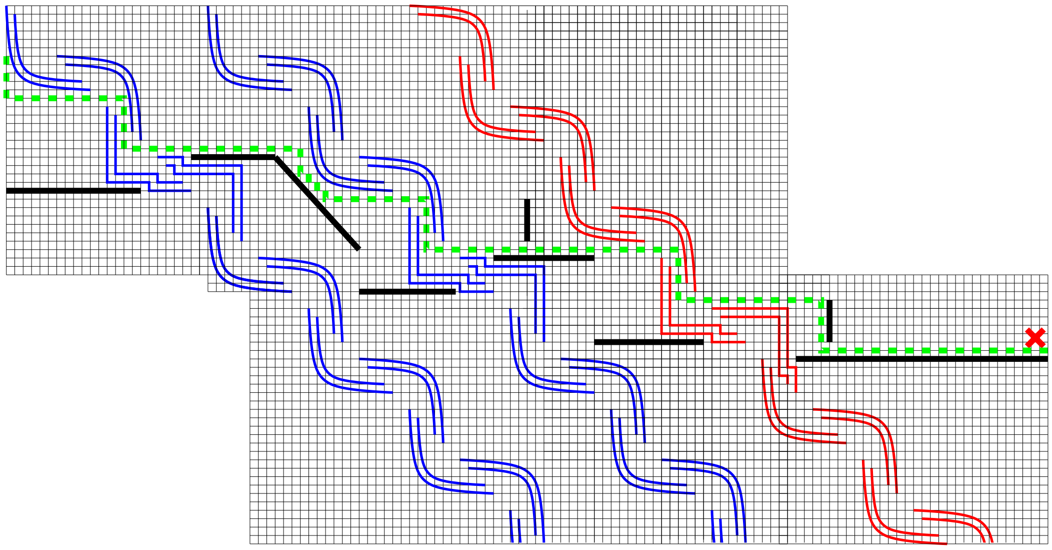


Figure 38: Segment of clause $C_1 = \{\bar{u}_1, u_2\}$, There is no solution for u_3 =false and $u_1 = u_2$ =true. "$\times$" indicates that no path exists below this point.

[22] M. R. Kramer and J. van Leeuwen. The Complexity of Wirerouting and Finding Minimum Area Layouts for Arbitrary VLSI Circuits. *Advances in Computing Research*, 2:129–146, 1984.

[23] S. Latifi, H. Ko, and P. K. Srimani. Node-to-Set Vertex Disjoint Paths in Hypercube Networks. *Computer Science Technical Report, Colorado State University*, CS-98-107, 1998.

[24] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout.* John Wiley & Sons, 1990.

[25] A. Lubiw. Counterexample to a Conjecture of Szymanski on Hypercube Routing. *Information Processing Letters*, 35:57–61, June 1990.

[26] S. Madhavapeddy and I. H. Sudborough. A Topological Property of Hypercubes: Node Disjoint Paths. *Proc. Second IEEE Symp. Parallel and Distributed Processing*, pages 532–539, 1990.

[27] D. Pretolani. A linear time algorithm for unique Horn satisfiability. *Information Processing Letters*, 48:61–66, 1993.

[28] M. O. Rabin. Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance. *Journal of the Association of Computing Machinery*, 36(2):335–348, April 1989.

[29] Y. Shiloach. Two Paths Problem is Polynomial. Technical Report TR-CS-78-654, Stanford University, 1978.

[30] I. H. Sudborough and T. Zhang. Circuit-Switched Routings for Automorphisms of a Hypercube. In *Proceedings of the Fourth International Symposium on Parallel Architectures, Algorithms and Networks: ISPAN 1999*. IEEE, June 1999.

[31] T. Szymanski. On the Permutation Routing of a Circuit-Switched Hypercube. *Proc. International Conference on Parallel Processing (ICPP)*, 1:103–110, 1989.

[32] S.-B. Tien and C. S. Raghavendra. Algorithms and Bounds for Shortest Paths and Diameter in Faulty Hypercubes. *IEEE Transactions on Parallel and Distributed Systems*, 4(6):713–718, June 1993.

[33] M. Watkin. Graph is (2k-1)-connected is a necessary condition to admit k paths. *Duke Math. Journal*, 1968.