Proceedings of the 14th IASTED International Conference
PARALLEL AND DISTRIBUTED COMPUTING AND SYSTEMS
November 4-6, 2002, Cambridge, USA

A-61

Complexity of k-Pairwise Disjoint Shortest Paths in the Undirected Hypercubic Network and Related Problems

Teofilo F. Gonzalez and David Serena
Department of Computer Science
University of California
Santa Barbara, CA, USA
email: {teo, dserena}@cs.ucsb.edu

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 results for the undirected n-cube topology. It is determined that the k-pairwise node disjoint shortest paths decision problem is NP-complete even for a restricted form of a partial permutation routing request – a subset of the 1-1 routing request. We also discuss our results for the node disjoint paths and node disjoint near-shortest paths problems.

KEY WORDS Fault-tolerance, hypercube, undirected n-cube, node disjoint paths and shortest paths, node disjoint near-shortest paths, k-pairwise routing, NP-completeness.

1 Introduction

The n-cube is a fundamental 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 computational complexity of routing problems which are common to many applications. The topology chosen for analysis is the **undirected** n-cube, hereafter referred to as the n-cube.

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 $X_i = (s_i, t_i)$, 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 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. The distance $d(a, b)$ is none other than the "Hamming Distance" between the vertices a and b in an n-cube. 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. Since each vertex contains at most one endpoint or blocking node, the problem is said to be a partial permutation routing request. When every vertex contains at most one source and one target then it is said to be a 1-1 routing request. The *edge disjoint shortest paths problem* is given X (without faulty nodes or $q = 0$) 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. Note that in the context of undirected graphs the order of the source to target in the routing request is not important. Therefore, we use the undirected pairs $X_i = \{s_i, t_i\}$ in lieu of the directed pair (s_i, t_i). Now 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 those 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 [10] 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 [18] presented a polynomial time algorithm to construct node disjoint paths in a graph for two pairs of vertices and Watkin [20] showed that $(2k-1)$-connectedness is a necessary condition for a graph to admit disjoint paths for a set of k pairs of vertices.

The related problem where one seeks to find p-pairwise node disjoint paths (*arbitrary distance pairs*) from vertices in set $\{s_1, s_2, \ldots, s_p\}$ to vertices in set $\{t_1, t_2, \ldots, t_p\}$ is called the set-to-set node disjoint paths problem. In this problem one needs to find p node disjoint paths from one set to the other such that the paths are from s_i to $t_{\phi(i)}$ where $\phi : \{1, 2, \ldots p\} \rightarrow \{1, 2, \ldots p\}$ and ϕ is any 1-1 function. Whereas in the k-pairwise problem ϕ is the identity function, namely $\phi(i) = i$. The undirected vertex version of Menger's theorem is applicable to the former problem but not the latter [14]. It is not applicable to the set-to-set node disjoint *shortest* paths problem as Menger's Theorem may imply *arbitrary* length paths.

Madhavapeddy and Sudborough [12] developed an $O(n^3 \log n)$ time algorithm to find disjoint paths for k pairs in an n-cube when $\lceil n/2 \rceil \geq k > 2$ and $n \geq 4$. Each of the paths is of length at most $2n$. Then Gu and Peng [7] 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 [17] proposed an algorithmic strategy which addresses fault-tolerance, security and load balancing. 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 [8] address the problem of constructing set-to-set node disjoint paths. They also presents algorithms for the node-to-set node disjoint paths problem. Gao, Novick and Qiu [2] present an algorithm for finding node-to-set node disjoint *shortest* paths. 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, [2, 11]. Finding k disjoint paths between two nodes in the hypercube has also been studied [7], but for brevity we do not discuss in detail these results. There are other interesting results and conjectures regarding the doubled n-cube [19].

Gonzalez and Serena [5] present polynomial time algorithms for a restricted subproblem called the *extreme* version. The extreme version of the *node disjoint shortest paths problem* has the additional constraint $d(X_i) = n$ for $1 \leq i \leq p$.

Madhavapeddy and Sudborough [13] show that the k-pairwise edge disjoint paths problem in the hypercube is NP-complete. Unfortunately they did not include a proof that the problem is in NP and we are unable to validate that proposition. Their reduction utilizes a vertex occupancy of the routing request that is well above the 1-1 routing request [13]. They conjecture that the k-pairwise node disjoint paths problem is also NP-complete. In Section 3 we prove that this problem in NP-hard.

In this paper we show that the node disjoint shortest path problem for the n-cube is NP-complete even when $d(X_i) = 3$ for $1 \leq i \leq p$. We also establish that the node disjoint paths problem (proving the conjecture in [13]), and the node disjoint near-shortest paths problems are NP-hard. 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.

2 n-Cube: Node Disjoint Shortest Paths

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 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.

We show that the node disjoint shortest paths problem when $d(X_i) \leq 2$ is solvable in polynomial time by reducing it to 2-SAT, which we know can be solved in linear time [1, 15] via *resolution*. This approach is identical to the one by Raghavan, Cohoon and Sahni [16] for a similar problem defined on a grid (mesh). 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. Then we introduce clauses that disallow choices that are infeasible. Using this approach it is simple to construct an algorithm that generates 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.

However, 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 [9].

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 1 *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 [9] 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 [3].

A simple polynomial reduction from 3-SAT to L3-SAT exists; however, it is omitted for brevity. It is a well known result that the L3-SAT decision problem is an NP-complete problem.

In Theorem 2 we show that our problem is NP-complete. We establish this result by reducing the L3-SAT problem to it. Before we present the proof of Theorem 2, 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 $000 \leftrightarrow 001 \leftrightarrow 011 \leftrightarrow 111$.

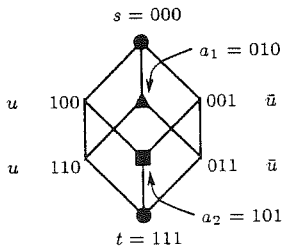

Figure 1. Duplication of consistent variable state: "setting and fan-out" component.

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.

The convey apparatus is used to transport "almost" consistent variable state information from the setting and

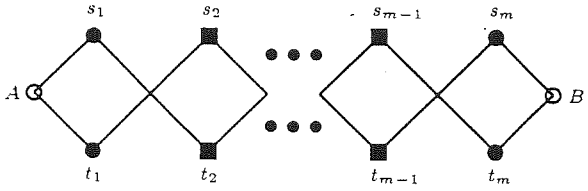

Figure 2. Convey Apparatus

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 conjec-

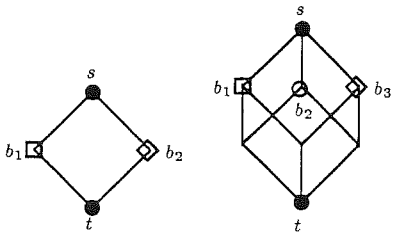

Figure 3. Same parity clause-checkers for 2 and 3 literals.

ture 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 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.

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:

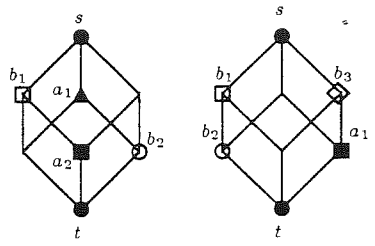

Figure 4. Left: Two literal clause-checker with different parity terminals. Right: Three literal clause-checker with two same parity terminals (b_1 and b_3) and one differing one (b_2).

$D_0D_1 \ldots D_{\lceil \log_2(v+1) \rceil}$ $E_0E_1 \ldots E_{\lceil \log_2(w+1) \rceil}$ 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 2 *The k-pair node disjoint shortest paths problem for the n-cube is NP-complete.*

Proof: Showing the problem is in NP is straightforward [6]. 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 proof that the instance of 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. []

We should point out that in the above reduction every pair has the property that $d(X_i) = 3$. Thus the problem is NP-complete even under this condition. We have also shown that the problem is NP-hard even if there are no singletons [6].

3 n-Cube: Related Problems

In this section we show that finding k-pairwise node disjoint *arbitrary* length paths in the n-cube is an NP-hard problem. The idea is to take the construction in Section 2 and just wrap blocking nodes around it so that every path for each pair does not go outside the corresponding shortest path for the pair. We state our result in the following theorem without a proof.

Theorem 3 *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.*

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.

How about if the paths connecting the pairs are allowed to have a length constrained by an **approximation** of the shortest path? Such as $d(X_i) + c$ or $c\ d(X_i)$ where $c > 1$ is a constant. In this section we demonstrate that NP-hardness persists even under this relaxation. Now let us define the *node disjoint near-shortest paths problem* in the n-cube by allowing paths of length at most $c\ d(X_i)+2b$, where b and c are constants such that $c = 1$ and $b > 0$, or $c > 1$ and $b \geq 0$. The reason for the constant time is that any detour from the shortest path in the n-cube must return on that dimension to reach a target node. A reduction from the node disjoint problem cannot be applied directly as a "yes" instance in that context may map to a "no" instance of the near-shortest paths problem. Namely an actual path length may in fact exceed the upper bound $c\ d(X_i) + 2b$.

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

Proof: Take the construction which is derivative of L3-SAT as shown in Section 2 with the constraint on pair

distance of 3. All of the constructions listed in Section 2 function for any length pairs even when wrapped. Now by definition $d(X_i) \leq c\ d(X_i) + 2b$ so a "yes" instance of the construction has a set of node disjoint shortest paths and therefore maps to a "yes" instance of the k-pairwise near-shortest paths problem. A "no" instance of the construction does not have node disjoint paths and therefore maps to a "no" instance of the k-pairwise near-shortest paths problem. Further the construction takes polynomial time for the same reason stated in the proof of Theorem 3. []

4 Conclusion

We have established that the k-pairwise node disjoint paths, and the node disjoint shortest paths problems in the n-cube are NP-hard. The result holds even when $d(X_i) \leq 3$. However the problem is polynomially solvable when $d(X_i) \leq 2$ even when defined over general graphs. Given current assumptions about the equivalence of classes P and NP this indicates that the likelihood of finding polynomial algorithms for these problems on these topologies is negligible. What is even more problematic is that the k-pairwise node disjoint near-shortest paths problem is NP-hard. This problem may be viewed as approximating the original problem. A similar result has been established by Gonzalez and Serena [6] for the edge disjoint shortest paths problems. However, Gonzalez and Serena [5] have developed polynomial time algorithms for a restricted subproblem called the *extreme* version.

References

[1] DOWLING, W. F., AND GALLIER, J. H. Linear-Time Algorithms for Testing the Satisfiability of Propositional Horn Formulae. *J. Logic 3* (1984), 267–284.

[2] GAO, S., NOVICK, B., AND QUI, K. From Hall's Matching Theorem to Optimal Routing on Hypercubes. *Journal of Combinatorial Theory, Series B 74*, 2 (November 1998), 291–301.

[3] GAREY, M. R., AND JOHNSON, D. S. *Computers and Intractability A Guide to the Theory of NP-Completeness*. Freeman, 1979.

[4] GONZALEZ, T. F., AND SERENA, F. D. n-Cube Network: Node Disjoint Shortest Paths for Pairs of Vertices. Tech. Rep. TRCS-2001-12, University of California at Santa Barbara, July 2001.

[5] GONZALEZ, T. F., AND SERENA, F. D. 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)* (2001), IASTED, pp. 278–282.

[6] GONZALEZ, T. F., AND SERENA, F. D. Complexity of k-Pairwise Disjoint Shortest Paths in the Hypercube and Grid Networks. Tech. Rep. TRCS-2002-14, University of California at Santa Barbara, May 2002.

[7] GU, Q.-P., AND PENG, S. k-Pairwise Cluster Fault Tolerant Routing in Hypercubes. *IEEE Transactions on Computers 46*, 9 (September 1997).

[8] GU, Q.-P., AND PENG, S. Node-to-Set and Set-to-Set Cluster Fault Tolerant Routing in Hypercubes. *Parallel Computing 24* (1998), 1245–1261.

[9] HOPCROFT, J., AND KARP, R. M. An $n^{2.5}$ Algorithm for Maximum Matchings in Bipartite Graphs. *SIAM J. Computing* (1973), 225–231.

[10] KARP, R. On the Computational Complexity of Combinatorial Problems. *Networks 5* (1975), 45–68.

[11] LATIFI, S., KO, H., AND SRIMANI, P. K. Node-to-Set Vertex Disjoint Paths in Hypercube Networks. *Computer Science Technical Report, Colorado State University CS-98-107* (1998).

[12] MADHAVAPEDDY, S., AND SUDBOROUGH, I. H. A Topological Property of Hypercubes: Node Disjoint Paths. *Proc. Second IEEE Symp. Parallel and Distributed Processing* (1990), 532–539.

[13] MADHAVAPEDDY, S., AND SUDBOROUGH, I. H. Disjoint Paths in the Hypercube. In *Graph-Theoretic Concepts in Computer Science, 15th International Workshop, WG '89* (June 1990), M. Nagl, Ed., vol. 411 of *Lecture Notes in Computer Science*, pp. 3–18.

[14] MENGER, K. Zur Allgemeinene Kurventheorie. *Fund. Math 10* (1927), 95–115.

[15] PRETOLANI, D. A Linear Time Algorithm for Unique Horn Satisfiability. *Information Processing Letters 48* (1993), 61–66.

[16] R. RAGHAVAN, J. C., AND SAHNI, S. Single Bend Wiring. *Journal of Algorithms 7* (1984), 232–257.

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

[18] SHILOACH, Y. Two Paths Problem is Polynomial. Tech. Rep. TR-CS-78-654, Stanford University, 1978.

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

[20] WATKIN, M. Graph is $(2k-1)$-Connected is a Necessary Condition to Admit k Paths. *Duke Math. Journal* (1968).