n-Cube Search Algorithm for Finding $(n-1)$-Pairwise Node Disjoint Shortest Paths

Teofilo Gonzalez and David Serena
University of California at Santa Barbara
Department of Computer Science

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 an efficient algorithm to construct, under certain conditions, pairwise node disjoint shortest paths for pairs of vertices in an n-cube in the presence of faulty nodes. The vertices in each pair are assumed to be at a distance n. The time complexity of our algorithm is $O(m^3)$, where m is the input length.*

Keywords: routing, fault-tolerance, n-cube, k-pairwise node disjoint shortest paths.

1 Introduction

The n-cube is a vital structure for parallel computing. Several systems with this communication architecture have been built in the past. The SGI Origin 2000 is such a computing system with an n-cube interconnection network. There are many algorithms utilized for finding disjoint shortest paths given a routing request on an n-cube connected network. In this paper we present an efficient algorithm for a restricted routing request problem prevalent in many applications.

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

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

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. The nodes a_i (or blocking nodes or faulty processors) may also be included as part of the input. 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. In this paper we show that when $p+q \leq n-1$ a solution always exists and we present a $O(m^3)$ time algorithm, where m is the input length ($\Theta((p+q)n)$), to construct a set of such paths.

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 short-

est 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 $\{010\}$.

Node disjoint paths have been studied extensively. Karp [6] showed that determining whether or not there exist node disjoint paths for a set of pairs of points in a graph is an NP-complete problem. Madhavapeddy and Sudborough [8] 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 [4] 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 disjoint **shortest** paths rather than just node disjoint paths. Rabin [9] proposed algorithms in which an attempt is made to simultaneously analyze fault-tolerance, security and load balancing.

Gu and Peng [5] 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 every 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 [5] also presents algorithms for the node-to-set node disjoint paths problem. Gao, Novick and Qiu [1] present an algorithm for finding node-to-set node disjoint *shortest* paths. Gao, Novick and Qiu's paper [1] 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. 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, [1, 7]. 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 [4], but for brevity we do not discuss in detail these results.

Gonzalez and Serena [2] have shown that finding k node disjoint shortest paths in an n-cube is NP-hard even when the length of each of the paths is at most three, but polynomially solvable, even on general graphs, when the length of each path is at most two. The problem remains NP-hard even when arbitrary length node disjoint paths are allowed. Since the problem of finding node disjoint shortest paths is NP-hard, we propose an algorithm for a restricted version of this problem.

2 Previous Algorithms

An n-cube, or equivalently an n dimensional hypercube, is an undirected graph $G = (V, E)$ with vertex set $V = \{0, 1, \ldots, 2^n - 1\}$ represented by a string of bits ($|V| = 2^n$ and

$|E| = n2^{n-1}$).

$$\underbrace{00\ldots0}_{n}{}_2 = 0 \leq v \leq 2^n - 1 = \underbrace{11\ldots1}_{n}{}_2 .^1$$

The undirected edges E are given by the set $\{\{a,b\} \mid d(a,b) = 1\}$, where $d(a,b)$ is the number of bits with value one in the binary representation of $a \oplus b$ and $\oplus$ is the bitwise "exclusive or" operation.

Hereafter we assume that the vertices in each pair X_i are at a distance n. Gonzalez and Serena [3] proved that when $p \leq \lceil n/2 \rceil$ and $2p+q \leq n+1$ node disjoint shortest paths exist. These constructive proofs can be easily implemented to take $O(m^2)$, where $m = O(n^2)$ is the input length [3]. Gonzalez and Serena's [3] approach constructs a set of node disjoint shortest paths as follows. First find an integer k which represents the position of one of the bits in the binary representation of the vertices and satisfies the property that for each endpoint $x \in X_i$ in every pair X_i, $1 \leq i \leq p$, and its neighbor $g(x) = x \oplus 2^k$ (x and $g(x)$ differ only on bit k) in the n-cube are such that all the $g(x)$ nodes are distinct and every $g(x) \notin e(X)$, where $e(X)$ is the set of all end points and blocking nodes in X.

The selection of this bit k is very important because it is used to reduce the original problem of finding node disjoint paths in an n-cube for p pairs of vertices in the presence of q blocking nodes to two independent problems. One subproblem, which is rather simple, is in the sub-cube where bit k is zero and the other one in the sub-cube where bit k is one. The first subproblem consists of finding a path for one pair in the presence of a number of blocking nodes. One isolates this subproblem by transitioning on an endpoint of one pair using bit k. It is relatively simple to establish that a solution exists for the first subproblem. The other subproblem has $p-1$ pairs of vertices with another set of blocking nodes, which is solved inductively. Gonzalez and Serena [3] showed that this result is tight in the sense that for $p \leq \lceil n/2 \rceil + 1$ there exist problem instances which do not have a set of k node disjoint shortest paths for the p pairs. In the next section we show that when $p+q \leq n-1$ a solution always exists and we present a $O(m^3)$ time algorithm, where m is the input length, to construct a set of such paths. The input length m is of $\Theta((p+q)n)$.

Let us illustrate this approach with the following example. The value of n is 3, $X_1 = \{000, 111\}$, and $X_2 = \{100, 011\}$. Transitioning on the bit $k = 0$ is not allowed because the neighbor of 000 along bit 0 is $100 \in e(X)$, but bit 1 and bit 2 are possible choices for k. Using bit $k = 1$ our approach is to select from X_1 the endpoint $e_1 = 000$ and from X_2 the endpoint $e_2 = 011$. Their corresponding neighbors are $g(e_1) = 010$ and $g(e_2) = 001$. Now the problem is to find a shortest path from 010 to 111 and one from 001 to 100. Since both paths have to go through nodes in which bit one is never changed (the bit is always one for the first subproblem and zero for the second one), it follows that the resulting problems are independent of each other. Therefore, we may refer to the resulting problems as finishing up the path for X_1 in the sub-cube $K(010, 111)$ [2] with blocking node 011 and finishing up the path for X_2 in the sub-cube $K(001, 100)$ with blocking node 000. By deleting bit k the resulting problems reduce to finding a path in $K(00, 11)$ with blocking node 01 and finding a path in $K(01, 10)$ with blocking node 00. Once we find these paths we add the bit just deleted as well as the transition introduced on bit k and we get the node disjoint shortest paths in the 3-cube.

We use (n, p, q) to represent all problem instances in an n-cube with p paths and q blocking nodes. Note that these equivalence classes do not correlate with the underlying decision problem of whether or not node disjoint short-

[1] Hereafter binary numbers will appear without the subscript "$_2$."

[2] The sub-cube defined by the source and destination nodes s and t is defined by

$$K(s,t) = \{u \mid (d(s,u) + d(u,t) == d(s,t))\}.$$

In other words, the sub-cube includes all the nodes in any shortest path from s to t.

est paths exist between the endpoints. For example note that some instances of $(4,2,3)$ do not have node disjoint paths where as others do. For example $\{\{0000,1111\}, \{0011,1100\}, \{0101\}, \{0110\}, \{1001\}\}$ has no solution, but $\{\{0000,1111\}, \{0011,1100\}, \{0001\}, \{0110\}, \{1001\}\}$ does.

For the case when $n = 4$ it is impossible to strengthen our results because for $p = \lceil n/2 \rceil + 1 = 3$ all the problem instances do not have a solution, i.e., node disjoint shortest paths do not exist. One such instance is given in Example 1 and the proof of our claim appears in [3].

Example 1 *The instance of* $(n,p,q) = (4,3,0)$ *with*

$$X = \{\{0000,1111\},\{1100,0011\},\{1010,0101\}\}$$

has no solution in the 4-cube. i.e., node disjoint shortest paths do not exist for this problem instance.

However note that there is a bit $k = 3$ (i.e. $0000 \oplus 2^3 = 1000$) as noted in Lemma 1 of [3]. Applying the 1-bit step we end up with the problems $\{X_1 = \{000,111\}, X_2 = \{100\}, X_3 = \{010\}\}$, and $\{X_1 = \{100,011\}, X_2 = \{010,101\}, X_3 = \{000\}\}$. The former problem can be solved, but not the latter one simply because there are not enough vertices in a 3-cube for two paths of length 3 and a blocking node.

As will be shown, the conditions $p \leq \lceil n/2 \rceil$ and $2p + q \leq n + 1$ can be improved when the value of n is larger. Example 2 gives a problem instance with $n = 7$ and $p = 6$ for which node disjoint shortest paths exist while the algorithm and proof technique in [3] does not cover this scenario.

Example 2 *Instance (n,p,q)=(7,6,0) with the pairs defined by*

$$\begin{aligned} X = \{ & \{0000000,1111111\},\{0000001,1111110\}, \\ & \{0000010,1111101\},\{0000100,1111011\}, \\ & \{0001000,1110111\},\{0010000,1101111\}\} \end{aligned}$$

Even though there exist two bits for transition, after applying the divide and conquer step twice, we end up with the problem instance in Example 1 which we know does not have node disjoint paths. Our approach in the next section covers the problem in Example 2. The main reason is that the partition into two subproblems is different.

3 The $p + q \leq n - 1$ Problem

By adding dummy pairs transform any problem for which $p+q \leq n-1$ to one with $p = n-1$ and $q = 0$. We refer to this problem as the $(n,p,0)$ problem. To construct node disjoint shortest paths for any instance of this problem we use the *1-bit balanced transformation.* This transformation begins by selecting a bit k, which we can prove it always exists, since $p < n$. The reduction uses $\lceil p/2 \rceil$ transitions of an endpoint of each of these p pairs from 1 to 0 along the k^{th} bit and $\lfloor p/2 \rfloor$ transitions from 0 to 1 for an endpoint of the remaining pairs. The subproblem in which all the k^{th} bit positions are 1 will have the $\lfloor p/2 \rfloor$ pairs that make transitions from 0 to 1, plus one endpoint of the $\lceil p/2 \rceil$ pairs that make the transition from 1 to 0. Therefore the resulting problem is an instance of $(n-1, \lfloor p/2 \rfloor, \lceil p/2 \rceil)$. The other subproblem in which all the k^{th} bit positions are 0 will have the $\lceil p/2 \rceil$ pairs that make transitions from 1 to 0, plus one endpoint of the $\lfloor p/2 \rfloor$ pairs that make the transition from 0 to 1. This resulting problem is an instance of $(n-1, \lceil p/2 \rceil, \lfloor p/2 \rfloor)$. The resulting subproblems will not fall into the induction hypothesis. So another transformation needs to be applied, which we call the *q-dependent transformation.*

In order to use the q dependent transformation we need to find a bit k in which to transition. The bit k may have at most one conflict which we can avoid. The existence of this bit is established in Lemma 1.

Lemma 1 *Let* $X = \{X_1, X_2, \ldots, X_p\}$ *be pairs of vertices inside the an n-cube with* $d(X_i) = n$ *for* $1 \leq i \leq p$, *and* $\{X_{p+1}, X_{p+2}, \ldots, X_{p+q}\}$ *be a set of blocking nodes such that* $1 \leq p \leq$

$p+q \leq n$, $p < n$ and $n > 3$. There exist at least $n-p-q+1$ bit positions k such that the cardinality of the set

$$e(X) \bigcup \{s_i \oplus 2^k \mid 1 \leq i \leq p\} \bigcup \{t_i \oplus 2^k \mid 1 \leq i \leq p\}$$

is $4p+q-1$ (conflict or blocked case) or $4p+q$ (conflict-free or unblocked case). This means that for each of these bit k positions there is at most one conflicting transition. Further the conflict is incident on a blocking node: for fixed k it occurs due to one unique $\ell \in \{1,\ldots,p\}$ and either $s_\ell \oplus 2^k$ or $t_\ell \oplus 2^k$, but not both, is a conflicting blocking node.

Proof: Omitted for brevity. □

For the q-dependent transformation we start by selecting the bit k for the transformation which can be shown to exist because $p+q \leq n$ and $p < n$. Let i be the number of q blocking nodes with a one in the k^{th} bit position. Without loss of generality assume that $i \leq \lfloor q/2 \rfloor$. Because if this is not the case all the vertices may be complemented with 2^n-1 and the new value for i will be at most $\lfloor q/2 \rfloor$.

First lets discuss the case where there is a bit k that is conflict-free, i.e., none of the endpoints has another endpoint of X as its neighbor along bit k. As it turns out we may assume that $p \geq q$ whenever we apply the q-dependent transformation. The transformation makes i transitions for a endpoint of the p pairs from 1 to 0 along the k^{th} bit and $p-i$ transitions from 0 to 1 for all of an endpoint of the remaining pairs. Therefore, the subproblem in which all the k^{th} bit positions are 1 will have the $p-i$ pairs that make initial transitions from 0 to 1, plus the i blocking nodes with a one on the kth bit position plus one endpoint of the i pairs that make the transition from 1 to 0. Therefore the resulting problem is an instance of $(n, p-i, 2i)$. The other subproblem in which all the k^{th} bit positions are 0 will have the i pairs that make transitions from 1 to 0, plus the $q-i$ blocking nodes with a zero on the kth bit position plus one endpoint of the $p-i$ pairs that make the transition from 0 to 1. This resulting problem is an instance of $(n, i, p+q-2i)$.

Now let us consider the case when each possible bit k has a conflict with a blocking node. By definition when $i = 0$ all the q blocking nodes have a one in the k-bit position. Therefore a transition from a 1 to a 0 along bit k in a pair will not have a conflict with a blocking node. Since all pairs make this type of transition, it then follows that our transformation rules are valid. When $i > 0$ one may always select pairs for a 0 to 1 transition on bit k that do not have a conflict since at most $p/2$ pairs make a 0 to 1 transition. Therefore the above q-dependent transformation is valid on all cases. Using the above transformations we can prove the following theorem.

Theorem 1 *Given X consisting of p pairs of nodes and q blocking nodes in an n-cube. If for all i and j such that $1 \leq i \leq p < j \leq p+q$, $d(X_i) = n > 6$, $d(X_j) = 0$, $p+q \leq n-1$ then node disjoint shortest paths exist for X.*

Proof: Omitted for brevity. □

An algorithm based on a straightforward implementation of the construction proof of Theorem 1 takes exponential time. Let us explain why this is the case with an example. Consider an instance of $(64, 63, 0)$. The algorithm generates an instance of $(63, 32, 31)$ and one of $(63, 31, 32)$. The instance of $(63, 31, 32)$ reduces to $(63, 32, 31)$ by transforming a blocking node into a pair. Further reduction of the instance of $(63, 32, 31)$ using the conflict-free part of Lemma 1 and applying the q-dependent transformation yields an instance of $(62, 32-i, 2i)$ and one of instance $(62, i, 63-2i)$. When $i = 2$ the resulting instances are $(62, 30, 4)$ and $(62, 2, 59)$. Now applying the induction hypothesis we transform the instance of $(62, 2, 59)$ into one of $(62, 61, 0)$. Therefore to solve the instance $(64, 63, 0)$ we need to solve two instances of $(62, 61, 0)$ plus we need to do some extra work. So the time complexity of the algorithm appears to be exponential, because we are con-

verting blocking nodes into pairs per the simplification of the proof of Theorem 1. However, the algorithm does not really need to produce paths for the dummy pairs being introduced.

To reduce the time complexity the algorithm mimics the conversion of blocking nodes into dummy pairs. If there are no (original) pairs, then the algorithm is done; if there is only one pair it is trivial to find a path; and if there is more than one pair, it applies the constructive proof of Theorem 1 to construct the paths. When the dimension is 7 or 8 then it just applies the constructive proof for the base case of Theorem 1 to find the paths. The algorithm uses the proof of Theorem 1 in a two stage process to partition the problem. Our algorithm is initially invoked with Find_Paths_0(n, X).

Theorem 1 may be used in a two stage process to partition the problem. Our algorithm is initially invoked with Find_Paths_0(n, X). The routine Find_Paths_1 implements the q-dependent transformation.

```
Find_Paths_0(n, X = {X_1, X_2, ..., X_p,
                     X_{p+1}, ..., X_{p+q}})
{
assume that d(X_1) = d(X_2) = ... = d(X_p) = n,
 d(X_{p+1}) = d(X_{p+2}) = ... = d(X_{p+q}) = 0,
      all the X_i pairs are in the same n dimensional
    sub-cube, and p + q ≤ n - 1
if p = 0 return;
if p = 1 construct path and return;
if n ≤ 8 return the paths constructed by the
          base case in Theorem 3;
  //find an unobstructed bit k for transition
    k ← Find_Bit_0(X);
    X' ← OneToZeroBalancedTransition(k, X);
    X'' ← ZeroToOneBalancedTransition(k, X);
    Output_Paths(k, X);
    Find_Paths_1(n - 1, X');
    Find_Paths_1(n - 1, X'');
}
```

The ZeroToOneBalancedTransition(k, X) and OneToZeroBalancedTransition(k, X) return the problem partitioned via the balanced transformation. The bit k is conflict-free due to the proof given in [3]. The ZeroToOneBalancedTransition(k, X) takes all pairs and returns $X' = \{X'_1, X'_2, \ldots, X'_{\lceil p/2 \rceil}, X'_{\lceil p/2 \rceil + 1} \ldots, X'_{p+q_0}\}$, $\lceil p/2 \rceil$ pairs reduced in size by transition from a 0 to 1 on bit k. Endpoints of the remaining $\lfloor p/2 \rfloor$ with a one in the k^{th} bit position are returned as blocking nodes. Further q_0 is the number of singleton nodes with a one in the k^{th} bit position. X'' = OneToZeroBalancedTransition(k, X) returns the complement albeit returning $\lfloor p/2 \rfloor$ paths with $q - q_0 + \lceil p/2 \rceil$ singleton nodes.

```
Find_Paths_1(n, X = {X_1, X_2, ..., X_p,
                     X_{p+1}, ..., X_{p+q}})
 assume that d(X_1) = d(X_2) = ... = d(X_p) = n,
   d(X_{p+1}) = d(X_{p+2}) = ... = d(X_{p+q}) = 0,
      all the X_i pairs are in the same n dimensional
     sub-cube, and p + q ≤ n
 if p = 0 return;
 if p = 1 construct path and return;
 if n ≤ 8 return the paths constructed by the
     base case in Theorem 3;
 if p + q < n then Find_Paths_0(X);
 else { // p + q = n
     //Find an appropriate bit for the transition.
    k ← Find_Bit(X);
    // If a conflict-free bit exists select it.
    // Define CountBlockingNodesWithOne(k, X)
    // as the number of blocking nodes with a
    //      1 bit in the k^th position.
    if (CountBlockingNodesWithOne(k, X) > ⌊q/2⌋)
     then X = ComplementProblem(n, X);
    i ← CountBlockingNodesWithOne(k, X);
    X' ← OneToZero_qTransition(i, k, X);
    X'' ← ZeroToOne_qTransition(i, k, X);
    Output_Paths(k, X);
    Find_Paths_0(n - 1, X');
    Find_Paths_0(n - 1, X'');
 }
```

Both OneToZero_qTransition(i, k, X) and ZeroToOne_qTransition(i, k, X) are based on the q-dependent transformation. Function CountBlockingNodesWithOne(k, X) returns the number of blocking nodes with a one in the k^{th} bit position. ComplementProblem(n, X) complements all nodes with $2^n - 1$ while maintaining the pair and blocking node structure. Note that under complementation the procedure Output_Paths must complement its output. To simplify the presentation of the algorithm we omit the details.

Due to the aforementioned complementation

function OneToZero_qTransition(i, k, X) does not have a conflict. This procedure is given p pairs and it selects $p - i$ of them for transition from 1 to 0. The residual blocking nodes from pairs is $2i$. However in the event that $i = 0$ and there is a conflict with a blocking node then an additional path is added to its output and blocking nodes with a zero in the k^{th} bit position are selected.

The function ZeroToOne_qTransition(i, k, X) returns i pairs which are capable of transitioning from 0 to 1 on bit k. If there is a conflict, then when $i = 0$ one removes that pair from the output. If however $i > 0$ one simply selects i other paths for transition from 0 to 1. The number of residual blocking nodes with a one in the k^{th} bit position are selected.

For brevity we omit the proof from the following theorem. Given that the input length m the overall time complexity is $O(m^3)$. The input length m is of $\Theta((p+q)n)$.

Theorem 2 *Given X consisting of p pairs of nodes and q blocking nodes in an n-cube. If for all i and j such that $1 \leq i \leq p < j \leq p+q$, $d(X_i) = n > 6$, $d(X_j) = 0$, $p + q \leq n - 1$ then algorithm* Finds_Paths_0 *constructs node disjoint shortest paths for X and can be implemented to take $O((p^2+q)n^3)$ time.*

4 Conclusion

We have presented a polynomial time algorithm for the case when $p+q \leq n-1$. Note that there are instances of (n, p, q) with $p + q = n$ that do not have node disjoint shortest paths, so one cannot solve the more general search problem. The problem of determining whether or not node disjoint shortest paths exist in an n-cube is an NP-hard problem even when $d(X_i) \leq 3$, but it is polynomially solvable when $d(X_i) \leq 2$ [2]. The problem remains NP-hard even for node disjoint paths. The extreme version of the problem is not known to to be NP-complete.

References

[1] 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.

[2] T. F. Gonzalez and F. D. Serena. n-cube network: Complexity of node disjoint paths for k-pairs of vertices. Technical Report TRCS-2001-16, UCSB, Sep. 2001.

[3] 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.

[4] Q.-P. Gu and S. Peng. k-pairwise cluster fault tolerant routing in hypercubes. *IEEE Transactions on Computers*, 46(9), September 1997.

[5] 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.

[6] R. Karp. On the computational complexity of combinatorial problems. *Networks*, 5:45–68, 1975.

[7] 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.

[8] 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.

[9] 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.