

Disjoint and Shortest Paths Routing in Hypercubes

Eddie Cheng

Department of Mathematics and Statistics
Oakland University
Rochester, MI 48309, USA

Ke Qiu

Department of Computer Science
Brock University
St. Catharines, Ontario, L2S 3A1
Canada

Zhizhang Shen

Department of Computer Science
Plymouth State University
Plymouth, NH 03264, USA

Abstract

One of the routing paradigms on interconnection networks is as follows: given a source node and m destination nodes, find disjoint paths from the source to the destination nodes. If we impose the condition that these paths be the shortest ones, this problem becomes harder and more interesting because such shortest and disjoint paths do not always exist. This problem has been studied previously for Q_n , the hypercube of dimension n , when $m = n$ and a necessary and sufficient condition has been found for these paths to exist. In this paper, we review the previous work for the hypercube and then consider the problem in the more general case for arbitrary m , where $1 \leq m \leq 2^n - 1$ in an n -cube by designing routing algorithms that always finds disjoint and shortest paths for a maximum subset of the destination nodes for which such paths exist. The size of such a set is no more than n , the degree of the Q_n .

1 Introduction

One of several well-known disjoint path paradigms [1, 3, 16] for any interconnection network is as follows: given a source node and m destination (or target) nodes, find m disjoint paths from the source to each target node (node-to-set). Other paradigms include node-to-node and set-to-set. There are large amount of work done on many well-known interconnection networks for various paradigms that are too numerous to list, among them, vast number of publications in the literature are devoted to various routing algorithms on the hypercube alone. This specific node-to-set routing problem has also been studied extensively for the hypercube in, for example, [7, 10, 11], to list just a few. Algorithms for this routing are shown to be useful in designing efficient and fault tolerant routings for the corresponding network [3].

The hypercube is perhaps one of the best known interconnection networks [12, 17]. An n -dimensional hypercube Q_n has 2^n nodes $0, 1, 2, \dots, 2^n - 1$ where (u, v) is an edge if u 's and v 's binary representations differ in exactly one position, i.e., $u = u_{n-1}u_{n-2} \dots u_{i+1}u_i u_{i-1} \dots u_1 u_0$ and $v = u_{n-1}u_{n-2} \dots u_{i+1}\bar{u}_i u_{i-1} \dots u_1 u_0$, $0 \leq i \leq n - 1$. Fig. 1 shows a 3-cube Q_3 .

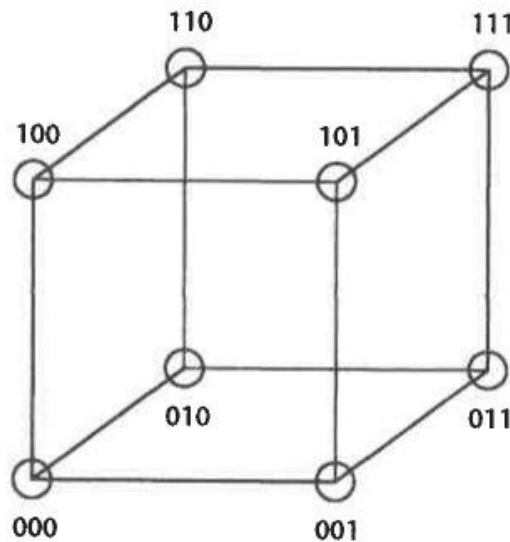


Figure 1: A 3-Cube.

For any two nodes u and v in a hypercube, the shortest path between u and v has a length of $H(u, v)$ where $H(u, v)$ is the Hamming distance between u and v , namely, the number of bits in which u and v differ. Please note that we also use $H(u)$ to represent the Hamming weight of u , the number of 1's in u 's binary representation.

In this paper, we study a special version of this disjoint path problem for

the hypercube: given a fixed node (source) and m other nodes (destination or target nodes) in a hypercube of dimension n , find *shortest* and *disjoint* paths for a maximum subset of the target nodes from the source node where two paths are disjoint if they do not share any node except the source node, and a path between nodes u and v is the shortest if and only if the length of the path is the Hamming distance between u and v . Of course, none of these paths should intersect any of the remaining target nodes. Because of the symmetry, without loss of generality, from now on, we assume that the source is the node $0 = 0^n$. Therefore, the shortest distance between a node u to the source node is simply the Hamming weight of u .

To the best of our knowledge, [14] and [6] are two of the earliest works on this subject for the case when $m = n$. Most of the routing algorithms for the hypercube find disjoint paths from 0 to n target nodes in an n -cube with $N = 2^n$ nodes whose lengths are bounded but are not necessarily the shortest. For example, it is shown in [15] that there exist n disjoint paths in an n -cube such that all paths have lengths no longer than $n + 1$ while in [14], it is shown that n disjoint paths exist such that at most one path has length $n + 1$ and all other paths have length no more than n . Please refer to [4] for the general problem of finding the *Rabin Number* of interconnection networks. Also note that some routing algorithms may find the disjoint paths that are shortest possible for the set of target nodes but not the shortest.

For the traditional routing problem between a source and a set of target nodes, on any interconnection network, when we impose the condition that these disjoint paths be the shortest, it is clear that such paths do not always exist for all target nodes. For example, on an n -cube, if $m > n$, then there can exist disjoint and shortest paths from the source node to at most n of the target nodes. because the degree of any node in an n -cube is n . Even when $m = n$, such paths do not always exist. See Fig. 2. Fig. 3 gives an example where such paths do exist. A necessary and sufficient condition was presented for such paths (disjoint and shortest) to exist for a set of n target nodes in [14]. This condition was later generalized in [6]. In addition, an $O(n^4) = O(\log^4 N)$ routing algorithm was given to find n disjoint shortest paths in cases they do exist. An $O(n^3)$ algorithm was given later in [2].

In the next section, we review the work done for the case when $m = n$ that includes a necessary and sufficient condition for these paths to exist and the $O(n^3)$ routing algorithm that finds such paths should they exist. We then study the problem of the general case for arbitrary m target nodes in Section 3 where we first find the cardinality of a maximum subset of target nodes to which there exist disjoint and shortest paths from the source node, we then derive approaches that will find shortest and disjoint paths from the source node to this set of target nodes.

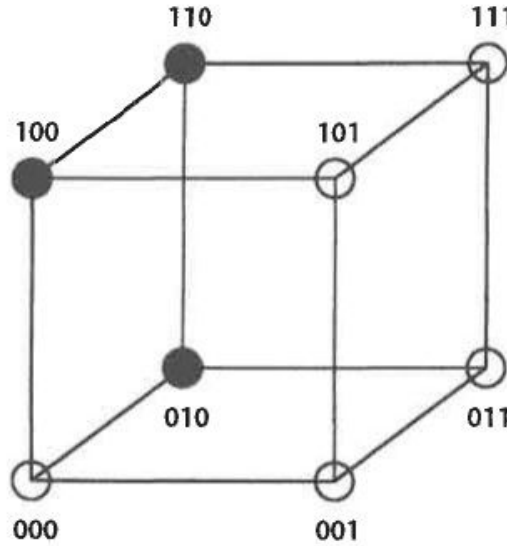


Figure 2: No Disjoint and Shortest Paths.

2 Node-to-Set Disjoint Shortest Paths Routing When $m = n$, A Review

In this section, we assume that $m = n$. Before we present our routing algorithm, we briefly describe the following result given in [14, 6]:

Given n non-zero distinct target nodes u_1, u_2, \dots, u_n in a hypercube of dimension n with $N = 2^n$ nodes, there exist n disjoint shortest paths from node 0 to them if and only if for any $1 \leq k \leq n - 1$, at most k nodes in the set of target nodes all have 0 at the same $n - k$ coordinates.

Although the proof was somewhat involved, simply stated, this condition says that such paths exist if and only if no k -subcube containing the node 0 contains more than k target nodes. For example, in Fig. 2, a 2-cube containing the node 0 also contains three target nodes, thus there do exist three disjoint and shortest paths for all three target nodes, while no such a k -cube, $1 \leq k \leq 2$, exists in Fig. 3.

We can state this condition yet another way that is crucial to developing an efficient routing algorithm as follows.

Assume that the n target nodes' binary representations are

$$\begin{aligned}
 &u_{11}u_{12}\dots u_{1n}, \\
 &u_{21}u_{22}\dots u_{2n}, \\
 &\vdots \\
 &u_{n1}u_{n2}\dots u_{nn}.
 \end{aligned}$$

The binary representations of the n nodes can be viewed as the adjacency matrix of a bipartite graph and the condition that for any $1 \leq k \leq n - 1$,

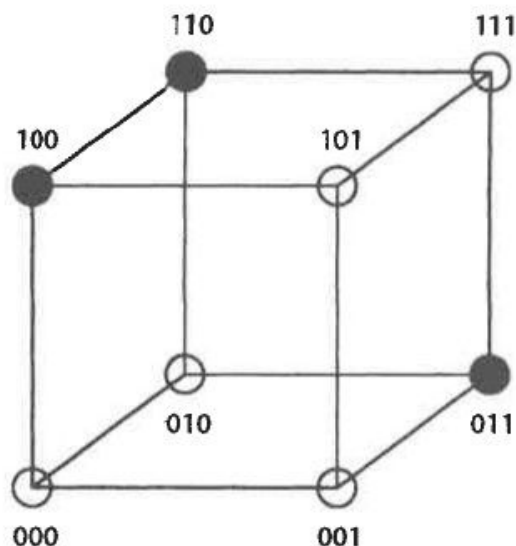


Figure 3: Disjoint and Shortest Paths.

at most k nodes in the set of target nodes all have 0 at the same $n - k$ coordinates means that Hall's condition [8] is satisfied, i.e., n disjoint shortest paths exist if and only if there exists a permutation $i_1 i_2 \dots i_n$ of symbols from $\{1, 2, \dots, n\}$ such that $u_{1, i_1} = u_{2, i_2} = \dots = u_{n, i_n} = 1$, namely, there is a 1 on each row and each column in the adjacency matrix. This is a direct result of Corollary 1 [6]. In other words, there exists a perfect matching for the bipartite graph or, equivalently, there is an optimal solution with cost n to the corresponding classic assignment problem [13] represented by the cost matrix which is the set of binary representations (note that to view the problem as an assignment problem, all 0 entries should be changed to a fixed value greater than 1). This description of the necessary and sufficient condition plays a critical role in our routing algorithm.

For example, for the following four nodes represented as a 4×4 cost matrix

$$\begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

it represents a bipartite graph $G = (\{A, B, C, D, a, b, c, d\}, \{(A, b), (A, c), (B, a), (C, b), (C, c), (C, d), (D, a), (D, d)\})$. One possible assignment is

$$\begin{pmatrix} 0 & 1 & \textcircled{1} & 0 \\ \textcircled{1} & 0 & 0 & 0 \\ 0 & \textcircled{1} & 1 & 1 \\ 1 & 0 & 0 & \textcircled{1} \end{pmatrix}$$

where the assignment solution is circled. This particular assignment is equivalent to a perfect matching of $\{(A, c), (B, a), (C, b), (D, d)\}$. Note that the solution to the assignment problem (and thus the perfect matching problem) is not unique.

Finding a maximum (perfect) matching for a bipartite graph with $2n$ vertices can be done in $O(n^{5/2})$ [5, 9]. Once a maximum matching of size n is found, we can proceed to develop an algorithm that finds these n disjoint and shortest paths. An $O(n^4)$ is first given in [6]. It is then improved to $O(n^3)$ in [2].

3 Node-to-Set Disjoint Shortest Paths Routing for Arbitrary m Nodes

Given m target nodes in Q_n , the bipartite graph mentioned in Section 2 can be generalized and constructed as follows. For each node $u_i = u_{i1}u_{i2} \cdots u_{in}$, it corresponds to a set U_i . If $u_{ij} = 1$, then $j \in U_i$. For example, for $u = 11010 \in Q_5$, $U = \{1, 2, 4\}$ because u has 1's in dimensions 1, 2, and 4. Now we have a bipartite graph $G = (V_1, V_2, E)$ where $V_1 = \{U_1, U_2, \dots, U_m\}$, $V_2 = \{1, 2, \dots, n\}$, and $(U_i, j) \in E$ if $j \in U_i$. For the target nodes 100, 010, 110, and 111 in Q_3 , the corresponding bipartite graph is given in Fig. 4.

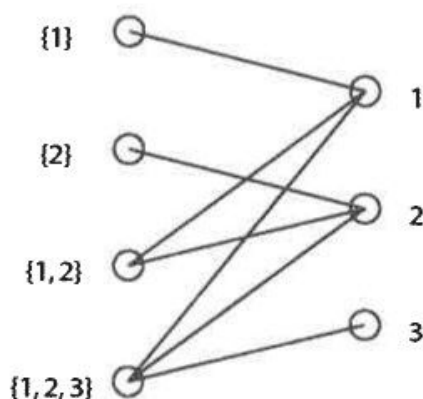


Figure 4: A Bipartite Graph for Four Target Nodes in Q_3 .

We claim that given m target nodes in an n -cube, the number of nodes that have disjoint and shortest paths to the source node without intersecting any other target nodes is the size of a maximum matching of the bipartite graph as constructed above from the target nodes. The proof is given by our routing algorithms below which actually find these paths.

Clearly, even though m is an arbitrary integer less than 2^n , the maximum possible such paths is n , the degree of the n -cube. If the number of such paths is k and $k \leq n$, there are k nodes in V_1 that are part of this maximum matching. If we ignore all other nodes, then clearly, we can find k disjoint

and shortest paths from these k nodes to 0^n using the routing algorithm from [2]. However, when other $m - k$ nodes are present, these paths may intersect some of these remaining $m - k$ nodes. So all we need to do is to devise a way to find these k paths without intersecting any of the target nodes. We consider two approaches

Our first approach is obtained by modifying the algorithm given in [2]. We will first describe that algorithm here. We then show how to use this algorithm to obtain disjoint and shortest paths for a maximum subset of the m target nodes.

We will first describe a key step in our routing algorithm. This step is based on the following observation:

Given n non-zero nodes satisfying the necessary and sufficient condition for n disjoint shortest paths to exist,

$$\begin{aligned}
 u_1 &= u_{11}u_{12}\dots u_{1n} \\
 u_2 &= u_{21}u_{22}\dots u_{2n} \\
 &\vdots \\
 u_n &= u_{n1}u_{n2}\dots u_{nn},
 \end{aligned}$$

then for any node u_i , we can find u'_i such that $H(u'_i \oplus u_i) = 1$, that is, $H(u'_i) = H(u_i) - 1$ and (u_i, u'_i) is an edge in the hypercube. Furthermore, if $H(u_i) > 1$, i.e., $u'_i \neq 0$, then nodes $u_1, u_2, \dots, u_{i-1}, u'_i, u_{i+1}, \dots, u_n$ still satisfy the necessary and sufficient condition.

We call the process to find such an u'_i from u_i a *reduction*. We now show how a reduction can be performed.

We assume that an aforementioned assignment has been found.

For node u_i , for each 1 other than the 1 in the assignment solution, remove it and see whether the resulting u'_i is not equal to any of the other $n - 1$ nodes. If such a 1 exists, remove it and we have found our u'_i . Otherwise (removing any such a 1 results in a node equal to u_j for some $j \in \{1, 2, \dots, n\} - \{i\}$), the situation can be illustrated below when removing the bold 1 will result in u_i becoming u_j :

$$\begin{array}{l}
 u_1 \\
 u_2 \\
 \vdots \\
 u_j \quad u_{j1}u_{j2}\dots 0 \dots \textcircled{1} \dots 1 \dots u_{jn} \\
 \vdots \\
 u_i \quad u_{i1}u_{i2}\dots 1 \dots 1 \dots \textcircled{1} \dots u_{in} \\
 \vdots
 \end{array}$$

u_n

Note that because the n target nodes satisfy the necessary and sufficient condition, there have to be two indices l and s ($l \neq s$) such that

$$u_{jl} = u_{js} = u_{il} = u_{is} = 1,$$

and u_{jl} and u_{is} are part of the assignment. We can get a new assignment by switching the two ①'s while keeping the remaining assignment intact as follows:

$$\begin{array}{r}
 u_1 \\
 u_2 \\
 \vdots \\
 u_j \quad u_{j1}u_{j2} \cdots 0 \cdots 1 \cdots \textcircled{1} \cdots u_{jn} \\
 \vdots \\
 u_i \quad u_{i1}u_{i2} \cdots 1 \cdots \textcircled{1} \cdots 1 \cdots u_{in} \\
 \vdots \\
 u_n
 \end{array}$$

Also note that because the nodes do satisfy the necessary and sufficient condition for disjoint shortest paths to exist and crossing any 1 in u_i other than 1 results in another target node, we know for sure that removing the 1 will generate a node that is different from any other target node. More precisely, let $v = v_1v_2 \cdots v_n$ be the node such that v and u_i only differ at position r , namely, $v_k = u_{ik}$ for $k \neq r$, $u_{ir} = 1$, and $v_r = 0$. Let $P = \{p \in \{1, 2, \dots, n\} - \{r\} | u_{ip} = 1\}$, i.e., the collection of positions except at the position r where u_i takes a 1. By assumption, for every $p \in P$, there is a target node u such that $u = u_i$, except that u has a 0 at position p . Hence, the assignment as guaranteed by the sufficient and necessary condition must include a position $p' \in P - \{p\}$, where u has a 1 at position p' . Since every such a position p corresponds to such a u , every position in P is included in the aforementioned assignment for $|P|$ such nodes.

Now consider v . For all positions $q \notin P$, including position r , v has a 0 at position q . Therefore, no such position q may be included in the assignment for v . Furthermore, by the definition of such an assignment, no position $p \in P$, at which position v has a 1, can be included in the assignment for v , either. Thus, none of the 1's in v can be "circled", a contradiction to the assumption that "an assignment has been found." Hence, v is indeed different from any other target node.

Now we can remove the 1 (in bold) formerly as part of the assignment in u_i to get u'_i and clearly, the new set of nodes still satisfy the necessary

and sufficient condition as an assignment still exists, $H(u_i \oplus u'_i) = 1$, and subsequently, $H(u'_i) = H(u_i) - 1$.

We now discuss how to perform the above reduction and its time complexity.

Before the reduction, the following pre-processing is done:

1. Convert the n binary representations into integers, also compute the Hamming weights of all target nodes. Both sequences are then sorted.
2. Find an assignment (perfect matching).

Step 1 takes $O(n^2 + n \log n) = O(n^2)$ time while Step 2 takes $O(n^{5/2})$ time. Note that both steps are done only once at the very beginning of the entire routing algorithm.

Note that in order to reduce a node u_i , if we delete 1's that are not in the assignment solution, in the order from the most significant position to the least significant position, we will get a sorted sequence of numbers. This observation leads to an $O(n)$ reduction:

If deleting a 1 results in a number v that is not equal to any of the remaining $n - 1$ target nodes ($O(1)$ time), delete this 1, the new value v is inserted into the list of sorted targets ($O(n)$ time). Similarly, the Hamming weight of the new node is also computed in constant time since it is $H(u_i) - 1$ and inserted to a sorted list of Hamming weights.

For example, for the cost matrix

$$\begin{array}{rcccc} u_1 : & 1 & 0 & 1 & \textcircled{1} \\ u_2 : & 0 & 0 & \textcircled{1} & 1 \\ u_3 : & 1 & \textcircled{1} & 0 & 1 \\ u_4 : & \textcircled{1} & 0 & 1 & 0 \end{array}$$

and we want to reduce u_1 . The sorted sequence of target nodes is 3, 10, (11,) 13. Crossing out the first 1 results in 3; Crossing out the second 1 results in 9. So we delete the second 1 and insert 9 into the sorted sequence of target nodes.

On the other hand, if no matter which 1 is deleted, the resulting node becomes one of the remaining target nodes, as illustrated in the two cases below where node u_1 is to be reduced, then we can simply find any one of the remaining target nodes and perform the reduction (by switching the 1's in the assignment first) and the updating mentioned above. The time for this operation is also $O(n)$.

$$\begin{array}{rcccc} u_1 : & 1 & 1 & \textcircled{1} \\ u_2 : & 0 & \textcircled{1} & 1 \\ u_3 : & \textcircled{1} & 0 & 1 \end{array}$$

$$\begin{array}{rcccc}
u_1 : & 1 & 0 & \textcircled{1} & 1 \\
u_2 : & 0 & 0 & 1 & \textcircled{1} \\
u_3 : & 1 & \textcircled{1} & 0 & 1 \\
u_4 : & \textcircled{1} & 0 & 1 & 0
\end{array}$$

Therefore, a node reduction can be done in $O(n)$ time.

We can now state the routing algorithm as follows:

1. while there exist nodes whose Hamming weights are all greater than 1, select one node w such that $H(w) = \max_u \{H(u) > 1\}$; (The Hamming weights of nodes can be computed at the beginning of the algorithm and updated after each reduction so that this step takes $O(n)$ time)
2. perform a reduction as described above.

The routing algorithm is to simply apply the reduction $O(n^2)$ times, as there are $O(n^2)$ 1's in the n target nodes, until they are reduced to an assignment solution, namely, the n neighbours of the node 0.

As for the total time $t(n)$ of the routing algorithm for an n -cube, we need $O(n^2)$ time to convert the n nodes to integer values, $O(n \log n)$ time to sort them, $O(n^{5/2})$ to find a perfect matching to get an assignment (these two steps need to be done only once), and perform the $O(n)$ -time reduction $O(n^2)$ times, resulting in a total time of $O(n^3) = O(\log^3 N)$.

Note that from the algorithm we always pick a node with the largest Hamming weight to reduce at any time. This is important in order to avoid routing conflict as described below.

$$\begin{array}{ccccccc}
& & & & & & \vdots \\
& & & & & & \vdots \\
& \dots & \bar{1} & \dots & 1 & \dots & 1 & \dots & : & u \\
& & & & & & \vdots \\
& & & & & & \vdots \\
& \dots & 1 & \dots & 1 & \dots & 1 & \dots & : & v \\
& & & & & & \vdots \\
& & & & & & \vdots
\end{array}$$

where $H(u) = 3$ and $H(v) = 4$ and $\bar{1}$ indicates that the 1 was previously deleted from u . If u is reduced first, we would have such a situation as described above. Then when v is reduced, the leftmost 1 is tried first and it is found that the resulting node is different from any of the other node, so that 1 is incorrectly removed, while in fact, the resulting node is actually u .

The proof that the paths so obtained are disjoint is thus clear: if after a reduction, the resulting node becomes a node that is reduced earlier, then

it means that a node with smaller Hamming weight is reduced earlier, a contradiction.

Similarly, the paths found are the shortest since for each node, a 1 is removed each time.

We end this section by an example. Given $n = 5$ nodes as follows:

$$\begin{array}{l} u_1 : \textcircled{1} \quad 1 \quad 0 \quad 0 \quad 0 \\ u_2 : 0 \quad 1 \quad \textcircled{1} \quad 0 \quad 0 \\ u_3 : 1 \quad \textcircled{1} \quad 1 \quad 0 \quad 0 \\ u_4 : 0 \quad 0 \quad 0 \quad \textcircled{1} \quad 1 \\ u_5 : 0 \quad 0 \quad 1 \quad 1 \quad \textcircled{1} \end{array}$$

with assignment solution circled. Since $H(u_3) = 3$, we do the reduction on u_3 . We first try the first 1 (from the left) and the resulting node is u_2 . We then try the third 1 and the resulting node is u_1 . We then switch the two 1's in the assignment solution in u_1 and u_3 and remove the second 1 in u_3 to get:

$$\begin{array}{l} u_1 : 1 \quad \textcircled{1} \quad 0 \quad 0 \quad 0 \\ u_2 : 0 \quad 1 \quad \textcircled{1} \quad 0 \quad 0 \\ u_3 : \textcircled{1} \quad 0 \quad 1 \quad 0 \quad 0 \\ u_4 : 0 \quad 0 \quad 0 \quad \textcircled{1} \quad 1 \\ u_5 : 0 \quad 0 \quad 1 \quad 1 \quad \textcircled{1} \end{array}$$

The next node to reduce is u_5 . When the third 1 is removed, the resulting node becomes u_4 . So we remove the fourth 1 to obtain

$$\begin{array}{l} u_1 : 1 \quad \textcircled{1} \quad 0 \quad 0 \quad 0 \\ u_2 : 0 \quad 1 \quad \textcircled{1} \quad 0 \quad 0 \\ u_3 : \textcircled{1} \quad 0 \quad 1 \quad 0 \quad 0 \\ u_4 : 0 \quad 0 \quad 0 \quad \textcircled{1} \quad 1 \\ u_5 : 0 \quad 0 \quad 1 \quad 0 \quad \textcircled{1} \end{array}$$

The next five reductions are straightforward and will end up with:

$$\begin{array}{l} u_1 : 0 \quad \textcircled{1} \quad 0 \quad 0 \quad 0 \\ u_2 : 0 \quad 0 \quad \textcircled{1} \quad 0 \quad 0 \\ u_3 : \textcircled{1} \quad 0 \quad 0 \quad 0 \quad 0 \\ u_4 : 0 \quad 0 \quad 0 \quad \textcircled{1} \quad 0 \\ u_5 : 0 \quad 0 \quad 0 \quad 0 \quad \textcircled{1} \end{array}$$

The five disjoint shortest paths are:

$$\begin{array}{l} 11000 \rightarrow 01000 \rightarrow 00000 \\ 01100 \rightarrow 00100 \rightarrow 00000 \\ 11100 \rightarrow 10100 \rightarrow 10000 \rightarrow 00000 \\ 00011 \rightarrow 00010 \rightarrow 00000 \\ 00111 \rightarrow 00101 \rightarrow 00001 \rightarrow 00000 \end{array}$$

When we have m target nodes, we can use the routing algorithm described above to find disjoint and shortest paths to the source node for a maximum subset of the target nodes. Let G be the bipartite graph constructed from the target nodes and k the size of a maximum matching. Then we know that the maximum number of target nodes that can have disjoint and shortest paths to the source node is at most k , where $k \leq n$, as each edge in the matching corresponds to a neighbor of the source node 0^n . As mentioned earlier, if we only consider the target nodes in the matching alone while ignoring all other target nodes, the routing algorithm just described can be used directly to find these k paths. Since we have to consider all target nodes and no path should intersect any target node, we need to modify the routing algorithm: we proceed as normal until removing a 1 from a target node u in the matching results in another target node v that is not in the matching. In this case, all we need to do is to replace u with v and put v in the set of target nodes in the matching. The number of disjoint and shortest paths remains unchanged. It is trivial to see that paths found will not intersect any target nodes.

It is now apparent that in order to avoid intersections, thus making sure the paths are disjoint, another approach is to simply find a minimum weight matching in the bipartite graph, where weights are assigned as follows: all the edges from a target node u have a weight of $H(u)$. The weighted bipartite graph of Fig. 4 is shown in Fig. 5. Once a matching is found, we can throw away the nodes and edges not in the matching and proceed with the standard routing algorithm from [2].

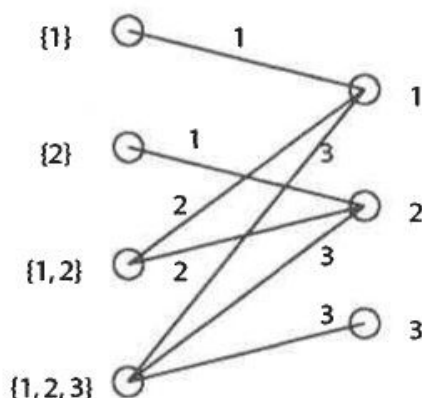


Figure 5: A Weighted Bipartite Graph.

For the first approach, the running time depends on m . If $m = O(n)$, then the time will be $O(n^3)$, the same as in the case considered in [2]. If $m \gg n$, then instead of searching the entire list of target nodes, thus requiring $O(m)$ for a row reduction, and a total time of $O(n^2m)$, we can do a binary search, resulting in a running time of $O(n \log m)$ for the row reduction, and a total time of $O(n^3 \log m)$. For the second approach, finding

a minimum weight matching takes $O(|E|\sqrt{|V|}) = O(mn\sqrt{m})$ time, assuming $m = \Omega(n)$, and $O(n^{5/2})$ otherwise, as there are $O(m+n)$ nodes and $O(mn)$ edges in the bipartite graph. But once a minimum weight matching is found, the size of the matching is no more than n , the routing can be completed in $O(n^3)$ time.

4 Conclusion

We studied the problem of, given a set of m target nodes and one source node, finding a maximum subset of target nodes to which disjoint and shortest paths from a source node in a hypercube of dimension n . This problem has been studied previously for the special case where $m = n$ and a necessary and sufficient condition was found for n disjoint and shortest paths to exist. Subsequently, a routing algorithm was presented to find these paths should they exist. Our problem is more general in that m is arbitrary. As a result, we are able to find the maximum possible number of target nodes for which disjoint and shortest paths exist between them and the source node and together with an algorithm that actually finds the paths for these target nodes. All the algorithms have a running time of polynomials in m and n . We note that if $m = \Omega(n)$, then a trivial lower bound to our routing problem of finding disjoint shortest paths in an n -cube is $\Omega(n^2)$ as there are n paths each of length $O(n)$ (Indeed, if we drop the condition that the disjoint paths be the shortest, then n -disjoint paths can be found in the optimal $O(n^2)$ time [7]).

References

- [1] Chen, C.C., Chen, J.: Nearly Optimal One-to-Many Parallel Routing in Star Networks. *IEEE trans. on Parallel and Distributed Systems*, Vol. 8, No. 12, 1196-1202 (1997)
- [2] Cheng, E., Gao, S., Qiu, K. Shen, Z.: On Disjoint Shortest Paths Routing on the Hypercube. *Proc. of the 3rd Annual International Conference on Combinatorial Optimization and Applications*, Yellow Mountains, China, LNCS 5573, 375-383 (2009)
- [3] Dietzfelbinger, M., Madhavapeddy, S., Sudborough, I.H.: Three Disjoint Path Paradigms in Star Networks. *Proc. of the 3rd IEEE Symposium on Parallel and Distributed Processing*, Dallas, Texas, 400-406 (1991)
- [4] Duh, D.R., Chen, G.H.: On the Rabin Number Problem. *Networks*, Vol. 30, No. 3, 219-230 (1998)

- [5] Even, S. and Kariv, O.: An $O(n^{2.5})$ Algorithm for Maximum Matching in General Graphs. IEEE 16th Annual Symposium on Foundations of Computer Science, 100-112 (1975)
- [6] Gao, S., Novick, B., Qiu, K.: From Hall's Matching Theorem to Optimal Routing on Hypercubes. Journal of Combinatorial Theory (B), Vol. 74, No. 2, 291-301 (1998)
- [7] Gu, Q.P., Peng, S.T.: Node-to-Set and Set-to-Set Cluster Fault Tolerant Routing in Hypercubes. Parallel Computing, 24, 1245-1261 (1998)
- [8] Hall, P.: On Representatives of Subsets. J. London Math. Soc., 10, 26-30 (1935)
- [9] Hopcroft, J.E., Karp, R.M.: An $n^{5/2}$ Algorithm for Maximum Matching in Bipartite Graphs. J. SIAM COMP., 2, 225-231 (1973)
- [10] Lai, C.N.: One-to-Many Disjoint Paths in the Hypercube and Folded Hypercube, Ph.D. Thesis, National Taiwan University (2001).
- [11] Latifi, S., Ko, H., Srimani, P.K.: Node-to-Set Vertex Disjoint Paths in Hypercube Networks. Technical Report CS-98-107, Colorado State University (1998)
- [12] Leighton, F.T.: Introduction to Parallel Algorithms and Architecture: Arrays-Trees-Hypercubes. Morgan Kaufmann Publishers, San Mateo, California (1992)
- [13] Papadimitriou, C.H., Steiglitz, K.: Combinatorial Optimization: Algorithms and Complexity. Dover, Mineola, New York (1998)
- [14] Qiu, K., Novick, B.: Disjoint Paths in Hypercubes. Congressus Numerantium, Vol. 119, 105-112 (1996)
- [15] Rabin, M.O.: Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance. Journal of ACM, Vol. 36. No. 2, 335-348 (1989)
- [16] Stewart, I.A., Xiang, Y.H.: One-to-Many Node-Disjoint Paths in (n, k) -Star Graphs. Discrete Applied Mathematics, 158, 62-70 (2010)
- [17] Saad, Y., Schultz, M.H.: Topological Properties of Hypercubes. IEEE Transaction on Computers, Vol. 37, No. 7, 867-872 (1988)