International Journal of Networking and Computing – www.ijnc.org
ISSN 2185-2839 (print) ISSN 2185-2847 (online)
Volume 3, Number 2, pages 182–191, July 2013

A Faster Algorithm for Finding Disjoint Ordering of Sets

E. Cheng
Mathematics and Statistics
Oakland University
Rochester, MI 48309
echeng@oakland.edu

K. Qiu
Computer Science
Brock University
St. Catharines, Ontario
kqiu@brocku.ca

Z. Shen
Computer Science and Tech.
Plymouth State University
Plymouth, NH 03264
zshen@plymouth.edu

Received: February 7, 2013
Revised: May 17, 2013
Accepted: June 20, 2013
Communicated by Sayaka Kamei

Abstract

Consider the problem of routing from a single source node to multiple target nodes with the additional condition that these disjoint paths be the shortest. This problem is harder than the standard one-to-many routing in that such paths do not always exist. Various sufficient and necessary conditions have been found to determine when such paths exist for some interconnection networks. And when these conditions do hold, the problem of finding such paths can be reduced to the problem of finding a disjoint ordering of sets. In addition to the applications in finding disjoint shortest paths in interconnection networks, the problem of finding disjoint ordering of sets is an interesting combinatorial problem in its own right. We study the problem of finding a disjoint ordering of sets A_1, A_2, ..., A_m where $A_i \subseteq A = \{a_1, a_2, \cdots, a_n\}$ and $m \leq n$. We present an $O(n^3)$ algorithm for doing so, under certain conditions, thus improving the previously known $O(n^4)$ algorithm, and consequently, improving the corresponding one-to-many routing algorithms for finding disjoint and shortest paths.

1 Introduction

In an interconnection network, there exist several well-known disjoint path paradigms [3]: (1) disjoint paths between two nodes (one-to-one); (2) disjoint paths from one fixed node to a set of nodes

(one-to-many); and (3) disjoint paths from a set of nodes to another set of nodes of the same cardinality (many-to-many). There are large amount of work done on many well-known interconnection networks for various paradigms, especially for the first two, that are too numerous to list. As for the third paradigm where given k pairs of nodes (s_1, t_1), (s_2, t_2), ..., (s_k, t_k) in a graph G, the problem of checking for the existence of k disjoint paths in arbitrary graph G for $k \geq 3$ is NP-complete [10]. For the n-star S_n, k pairwise paths of length no more than $D(S_n) + 5$, where $D(S_n)$ is the diameter of the n-star, can be found in $O(n^2)$ time [6]. For the hypercube, the problem is studied in [5, 7].

We can also impose certain conditions for each of these paradigms, creating different versions of the disjoint path problems. For example, we can ask that disjoint paths be the shortest, or that the sum of all distances be as small as possible.

In this paper, we consider the routing paradigm (1) with the additional condition that all disjoint paths be the shortest paths as well. This problem of finding disjoint shortest paths between a source node and multiple target nodes in a graph is a very interesting problem considering that such paths do not always exist. For some trivial graphs, e.g., linear arrays and 2-d meshes, it is easy to derive a necessary and sufficient condition for such paths to exist. For example, for a linear array, two such paths exist iff the two target nodes are on different sides of the source node. Similarly, for a 2-d mesh, 4 such paths exist iff no quadrant contains more than 2 target nodes and no two nodes in the same quadrant are on the same row or column as the source node (i, j) (here, for any node in general position (i, j), we divide the mesh into four quadrants, for example, one quadrant consists of points (x, y), where $x > i$ and $y \geq j$, etc.). For non-trivial graphs, it is in general not easy to find such conditions.

The hypercube is one of the most popular interconnection networks. A hypercube of dimension n, or an n-cube, consists of 2^n nodes labeled as 0, 1, 2, ..., $2^n - 1$. Two nodes u and v are connected if and only if their binary representations differ in exactly one bit. For example, nodes $u = 1$ and $v = 3$ are linked in a 3-cube since 001 and 011 differ in one bit. For any node v in an n-cube, its binary representation is $v = v_1 v_2 \cdots v_n$, where $v_i \in \{0, 1\}$ for $1 \leq i \leq n$.

For the hypercube, because of the symmetry, without loss of generality, we assume that the source is the identity node $0 = 00 \cdots 0$. Most of the routing algorithms for the hypercube find disjoint paths from 0 to n target nodes in an n-cube whose lengths are bounded but are not necessarily the shortest. For example, it is shown in [14] that there exist n disjoint paths in an n-cube such that all paths have lengths no longer than $n + 1$ while in [13], 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. Note that a path from 0 to node u is shortest if $d(0, u) = H(u)$, the Hamming weight of u defined to be the number of 1's in u. 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.

When we impose the condition that these disjoint paths be the shortest, it is clear that such paths exist only for certain sets of n target nodes in an n-cube. A necessary and sufficient condition was presented for such paths (disjoint and shortest) to exist for a set of n target nodes in [13]. This condition was later generalized in [4]. In addition, an $O(n^4)$ routing algorithm was given to find the paths if they do exist. This routing algorithm finds the disjoint shortest paths by finding a disjoint ordering of sets with a system of distinct representatives (SDR). The existence of an SDR, and subsequently a disjoint ordering, are important to the disjoint shortest paths routing in the hypercube. They can also be used to characterize the sufficient and necessary condition for such paths to exist in the star graph, a popular member of the Cayley graphs to which the hypercube also belong, and possibly other Cayley graphs based on the symmetric group [1]. Once it has been determined that disjoint and shortest paths do exist in an interconnection network using the iff condition involving SDR's, a disjoint ordering will generate such paths. In addition to the application in finding disjoint shortest paths in interconnection networks, the problem of finding disjoint ordering of sets is an interesting combinatorial problem in its own right. In this paper, we present an algorithm that finds disjoint ordering of n sets in $O(n^3)$ time for reasonably small n's and in $O(n^3)$ expected time for arbitrary n's, improving the previous $O(n^4)$ algorithm. This algorithm immediately implies an $O(n^3)$ algorithm for finding n disjoint shortest paths in an n-cube. In addition, it immediately gives an routing algorithm for finding disjoint shortest paths in any interconnection network when such paths can be characterized by SDR's and disjoint ordering

of sets. We give necessary background in the next section and present our algorithm for finding disjoint ordering of sets in Section 3. Further discussions on the relation between the problems of routing and finding disjoint ordering of sets and that of matching as well as future work are given in Section 4.

2 System of Distinct Representatives and Disjoint Ordering of Sets

Definition 1 *([4]) Let $(A_1, A_2, \cdots, A_m)$ be a collection of subsets of a set $A = \{a_1, a_2, \cdots, a_n\}$, $m \leq n$. An ordered set of distinct elements $[a_{i_1}, a_{i_2}, \cdots, a_{i_m}]$ is called a system of distinct representatives (SDR) if $a_{i_j} \in A_j$, for $1 \leq j \leq m$.*

For example, if $A = \{1, 2, 3, 4\}$ and $A_1 = \{1, 2\}$, $A_2 = \{3\}$, and $A_3 = \{2, 3\}$, then $[1, 3, 2]$ is an SDR for A_1, A_2, and A_3. On the other hand, if $A = \{1, 2, 3, 4\}$ and $A_1 = \{1\}$, $A_2 = \{1, 4\}$, and $A_3 = \{4\}$, then there does not exist an SDR for A_1, A_2, and A_3. Clearly, an SDR of $(A_1, A_2, \cdots, A_m)$ corresponds to a matching of size m of the bipartite graph where the partite sets are A and $\{A_1, A_2, \cdots, A_m\}$ such that there is an edge from $a \in A$ to A_i iff $a \in A_i$. The well-known Hall's Theorem [8] gives the iff condition for the existence of an SDR for a collection of finite sets.

For n target nodes in an n-cube with 2^n nodes:

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

they correspond to n sets U_1, U_2, ..., U_n which are defined as follows. If $u_{ij} = 1$, then we include j in set U_i. It was shown in [4] that an iff condition for n disjoint shortest paths to exist in the n-cube is that there exists an SDR for U_1, U_2, ..., and U_n. For example, for nodes

$$\begin{aligned} u_1 &= 0110 \\ u_2 &= 1000 \\ u_3 &= 0111 \\ u_4 &= 1001, \end{aligned}$$

we have

$$\begin{aligned} U_1 &= \{2, 3\} \\ U_2 &= \{1\} \\ U_3 &= \{2, 3, 4\} \\ U_4 &= \{1, 4\}. \end{aligned}$$

One SDR for the four sets is $[2, 1, 3, 4]$.

The binary representations of the n nodes can be viewed as the adjacency matrix of a bipartite graph and the condition that an SDR exists means that Hall's condition [8] is satisfied, i.e., n disjoint shortest paths exist if and only if there exists a permutation $i_1 i_2 \cdots i_n$ of symbols from $\{1, 2, \cdots, n\}$ such that $u_{1,i_1} = u_{2,i_2} = \cdots = u_{n,i_n} = 1$, namely, there is a 1 on each row and each column in the adjacency matrix. 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 [12] 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).

For the above example, 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 $O(n^{5/2})$ [9].

Once an SDR is found, an actual routing is then found by finding a disjoint ordering of the sets U_i's. The concepts of ordering and disjoint ordering of elements of a finite set are defined below.

Definition 2 *([4]) A permutation of the elements of a finite set is called an ordering. Let X and Y be two sets ordered as $O_X = (x_1, x_2, \cdots, x_a)$ and $O_Y = (y_1, y_2, \cdots, y_b)$, we say that O_X and O_Y are disjoint if*

$$\{x_1, x_2, \cdots, x_i\} \neq \{y_1, y_2, \cdots, y_i\},$$

for $1 \leq i \leq \min(|X|, |Y|)$ unless $i = |X| = |Y|$.

Note that by the definition, there exists a disjoint ordering even if $X = Y$. A simple example is when $X = Y = \{1, 2\}$, then (1, 2) and (2, 1) are disjoint.

A collection of finite sets have a disjoint ordering if each set has an ordering and all the orderings are pairwise disjoint. Specifically, if all singletons in the collection are distinct, then the first elements of a disjoint ordering of the collection form an SDR. For example, for the following collection of sets and their disjoint ordering, 1, 3, 2, 4 form an SDR $[1, 3, 2, 4]$:

$$\begin{array}{lcll} A & = & \{1, 2, 4\} & O_A = (1, 2, 4) \\ B & = & \{1, 3, 4\} & O_B = (3, 4, 1) \\ C & = & \{1, 2, 3\} & O_C = (2, 3, 1) \\ D & = & \{1, 2, 4\} & O_D = (4, 1, 2). \end{array}$$

Once again, for our previous example with SDR $= [2, 1, 3, 4]$, a disjoint ordering is

$$\begin{array}{l} (2, 3) \\ (1) \\ (3, 4, 2) \\ (4, 1), \end{array}$$

implying a routing:

$$\begin{array}{l} 0000 \rightarrow 0100 \rightarrow 0110 \\ 0000 \rightarrow 1000 \\ 0000 \rightarrow 0010 \rightarrow 0011 \rightarrow 0111 \\ 0000 \rightarrow 0001 \rightarrow 1001 \end{array}$$

Note that neither the SDR nor the disjoint ordering for a collection of sets is necessarily unique, if they do exist. An $O(n^4)$ algorithm is given in [4] to find a disjoint ordering for any collection of sets with an SDR. This immediately implies an $O(n^4)$ routing for disjoint shortest paths on an n-cube.

It has been shown that the existence of an SDR is an iff condition for disjoint shortest paths to exist in not only the hypercube, but also star graphs, and quite possibly many other interconnection networks [1] and disjoint ordering gives actual routing for these paths. In addition, the problem of finding disjoint ordering for a given SDR is an interesting combinatorial problem in its own right. In the next section, we present an $O(n^3)$ algorithm for this problem. Without loss of generality, we assume that $A = \{1, 2, \cdots, n\}$, $m = n$ so we have A_1, A_2, ..., A_n where $A_i \subseteq A$. We also assume that an SDR is given. Clearly, $m \leq n$ because if $m > n$, it is impossible to have an SDR. If $m < n$, we can always add sets $A_j = \{1, 2, \cdots, n\}$, $j = m+1, m+2, \cdots, n$.

3 Finding a Disjoint Ordering from a Given SDR

An important step in our algorithm is to represent the n sets as an $n \times n$ matrix $(a_{ij})_{n\times n}$ where

$$a_{ij} = \begin{cases} 0 & if\ j \notin A_i \\ 1 & if\ j \in A_i. \end{cases}$$

With this representation, we then perform a sequence of steps that we call row reductions to be described next until the representation matrix is reduced to an SDR. Note that this final SDR does not have to be the originally given SDR. The actual disjoint ordering for the n sets is then constructed from the reductions.

3.1 Row Reduction

The key step in our algorithm is based on the following observation:

Theorem 1 *Given n sets with an SDR in their matrix representation*

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}$$

then for any row A_i, we can find $A_i^{'}$ such that $H(A_i^{'} \oplus A_i) = 1$ and , $H(A_i^{'}) = H(A_i) - 1$, where $H(A_i)$ is the Hamming weight of vector representing A_i. Furthermore, if $H(A_i) > 1$, i.e., $A_i^{'} \neq 0$, then sets A_1, A_2, ..., A_{i-1}, $A_i^{'}$, A_{i+1}, ..., A_n still have an SDR.

In essence, A_i has been reduced to A_i' by eliminating one specific 1 from A_i. We call the process to find such an $A_i^{'}$ from A_i a *reduction*. The theorem is proved by the reduction algorithm given next. Note that the elements of the SDR are circled.

For set A_i, for each 1 other than the 1 in the SDR, remove it and see whether the resulting $A_i^{'}$ is not equal to any of the other $n-1$ sets. If such a 1 exists, remove it and we have found our $A_i^{'}$. Otherwise (removing any such a 1 results in a set equal to A_j for some $j \in \{1, 2, \cdots, n\} - \{i\}$), the situation can be illustrated below when removing the bold **1** will result in A_i becoming A_j:

$$\begin{array}{ll} A_1 & \\ A_2 & \\ \vdots & \\ A_j & a_{j1}a_{j2}\cdots 0 \cdots \textcircled{1} \cdots 1 \cdots a_{jn} \\ \vdots & \\ A_i & a_{j1}a_{j2}\cdots \mathbf{1} \cdots 1 \cdots \textcircled{1} \cdots a_{jn} \\ \vdots & \\ A_n & \end{array}$$

Note that because the n sets have an SDR, there have to be two indices l and m ($l \neq m$) such that

$$a_{jl} = a_{jm} = a_{il} = a_{im} = 1,$$

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

$$\begin{array}{ll} A_1 & \\ A_2 & \\ \vdots & \\ A_j & a_{j1}a_{j2}\cdots 0\cdots 1\cdots ①\cdots a_{jn} \\ \vdots & \\ A_i & a_{j1}a_{j2}\cdots 1\cdots ①\cdots \mathbf{1}\cdots a_{jn} \\ \vdots & \\ A_n & \end{array}$$

Note that because the sets do have an SDR and crossing any 1 in A_i other than **1** results in another set, we know for sure that removing the **1** will generate a set that is different from any other set.

Now we can remove the 1 (in bold) formerly as part of the SDR in A_i to get $A_i^{'}$ and clearly, the new set of sets still have an SDR, $H(A_i \oplus A_i^{'}) = 1$, and subsequently, $H(A_i^{'}) = H(A_i)$ - 1. Also, if we remove a 1 at position j for A_i, it implies that we should insert the element j in the left end of the disjoint ordering for A_i. This step is repeated each time a reduction is performed, and at the end, we should have our disjoint ordering.

3.2 Implementation and Analysis

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

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

1. Create the binary representation matrix for the given sets A_1, A_2, ..., A_n;
2. Convert the n rows into integers, also compute the Hamming weights of all sets. Both sequences are then sorted;
3. Find an SDR.

Step 1 takes $O(n^2)$. Step 2 takes $O(n^2 + n \log n) = O(n^2)$ time while Step 3 takes $O(n^{5/2})$ time. Note that these steps are done only once at the very beginning.

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

If deleting a 1 (at position k) from A_i that is not in the SDR results in a number v that is not equal to any of the remaining $n-1$ values corresponding to the $n-1$ other rows of the matrix (we can compute v in constant time because $v = A_i$'s value - 2^{n-k}), delete this 1, the new value v is inserted into the list of sorted targets ($O(n)$ time, if done sequentially, or $O(\log n)$ time by a binary search). Similarly, the Hamming weight of the new row is also computed in constant time since it is $H(A_i)$ - 1 and inserted to a sorted list of Hamming weights. To check whether v is equal to any of the $n-1$ values, we can simply perform a linear search (even though these $n-1$ values are sorted) and let's say that it takes m steps, $m \leq n$. If this is not the case, we are done with reducing A_i. If v is equal to one of the $n-1$ values, then we try to reduce the next 1 on A_i. But when we do the checking again, we can simply start from the place where the last search ended. This is because the new value to be searched for is larger than v. As we can see, even though each search could take $O(n)$ time, the total search time remains $O(n)$. Therefore, the time for reducing A_i to A_i' is $O(n)$.

If there are t 1's ($t \leq n$) in A_i, the amortized search time for each 1 is $O(n/t)$. If $t = \theta(n)$, this time becomes constant.

For example, for the representation matrix

$$\begin{array}{lcccc} A_1: & 1 & 0 & 1 & \textcircled{1} \\ A_2: & 0 & 0 & \textcircled{1} & 1 \\ A_3: & 1 & \textcircled{1} & 0 & 1 \\ A_4: & \textcircled{1} & 0 & 1 & 0 \end{array}$$

and we want to reduce A_1. The sorted sequence of the sets 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 sets.

On the other hand, if no matter which 1 is deleted, the resulting row becomes one of the remaining rows, as illustrated in the two cases below where row A_1 is to be reduced, then we can simply find any one of the remaining sets and perform the reduction (by switching the 1's in the SDR first) and the updating mentioned above. The time for this operation is also $O(n)$.

$$\begin{array}{lccc} A_1: & 1 & 1 & \textcircled{1} \\ A_2: & 0 & \textcircled{1} & 1 \\ A_3: & \textcircled{1} & 0 & 1 \end{array}$$

$$\begin{array}{lcccc} A_1: & 1 & 0 & \textcircled{1} & 1 \\ A_2: & 0 & 0 & 1 & \textcircled{1} \\ A_3: & 1 & \textcircled{1} & 0 & 1 \\ A_4: & \textcircled{1} & 0 & 1 & 0 \end{array}$$

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

3.3 Generating a Disjoint Ordering

We can now state the algorithm as follows:

1. while there exist sets whose Hamming weights are greater than 1, select one set A_i such that $H(A_i) = \max\limits_{1 \leq j \leq n} \{H(A_j) > 1\}$; (The Hamming weights of sets 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 disjoint ordering generating algorithm is to simply apply the reduction $O(n^2)$ times, as there are $O(n^2)$ 1's in the n sets, until they are reduced to an SDR.

As for the total time $t(n)$ of the algorithm for n sets, we need $O(n^2)$ time to convert the n binary set representations to integer values, $O(n \log n)$ time to sort them, $O(n^{5/2})$ to find a perfect matching to get an SDR (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 $O(n^3)$.

For this running time, the following discussion is in order. When we have an n-bit binary number, the range of the corresponding integer is $\{1, 2, \cdots 2^n - 1\}$. It is certainly not reasonable to assume that two numbers of the magnitude $O(2^n)$ can be compared in constant time. Therefore, our running time stands only when n is relatively small. For arbitrary n, one remedy is to compute all integers modulo q for some suitable prime number q, for example, a q such that $10 \times q$ just fits into a computer word [2, page 912]. In this sense, our situation is similar to the one faced by the Rabin-Karp string matching algorithm [11]. Accordingly, our algorithm has an $O(n^3)$ expected running time. The justification is similar to that given in [2, page 915].

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

$$\begin{array}{c} \vdots \\ \cdots \quad \bar{1} \quad \cdots 1 \cdots 1 \cdots : A \\ \vdots \\ \cdots 1 \cdots \quad 1 \quad \cdots 1 \cdots 1 \cdots : B \\ \vdots \end{array}$$

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

The proof that the ordering so obtained are disjoint is thus clear: if after a reduction, the resulting set becomes a set that is reduced earlier, then it means that a set with smaller Hamming weight is reduced earlier, a contradiction.

We end this section by an example. Given five sets A_i's, $i = 1, 2, 3, 4, 5$ whose matrix representation is as follows:

$$\begin{array}{lccccc} A_1: & \textcircled{1} & 1 & 0 & 0 & 0 \\ A_2: & 0 & 1 & \textcircled{1} & 0 & 0 \\ A_3: & 1 & \textcircled{1} & 1 & 0 & 0 \\ A_4: & 0 & 0 & 0 & \textcircled{1} & 1 \\ A_5: & 0 & 0 & 1 & 1 & \textcircled{1} \end{array}$$

with the SDR circled. Since $H(A_3) = 3$, we do the reduction on A_3. We first try the first 1 (from the left) and the resulting set is A_2. We then try the third 1 and the resulting set is A_1. We then switch the two 1's in the SDR in A_1 and A_3 and remove the second 1 in A_3 to get:

$$\begin{array}{lccccc} A_1: & 1 & \textcircled{1} & 0 & 0 & 0 \\ A_2: & 0 & 1 & \textcircled{1} & 0 & 0 \\ A_3: & \textcircled{1} & 0 & 1 & 0 & 0 \\ A_4: & 0 & 0 & 0 & \textcircled{1} & 1 \\ A_5: & 0 & 0 & 1 & 1 & \textcircled{1} \end{array}$$

We now have a disjoint ordering as follows:

$$\begin{aligned} O_{A_1} &= (\ ) \\ O_{A_2} &= (\ ) \\ O_{A_3} &= (2) \\ O_{A_4} &= (\ ) \\ O_{A_5} &= (\ ) \end{aligned}$$

The next set to reduce is A_5. When the third 1 is removed, the resulting row becomes A_4. So we remove the fourth 1 to obtain

$$\begin{array}{lccccc} A_1: & 1 & \textcircled{1} & 0 & 0 & 0 \\ A_2: & 0 & 1 & \textcircled{1} & 0 & 0 \\ A_3: & \textcircled{1} & 0 & 1 & 0 & 0 \\ A_4: & 0 & 0 & 0 & \textcircled{1} & 1 \\ A_5: & 0 & 0 & 1 & 0 & \textcircled{1} \end{array}$$

The disjoint ordering now becomes

$$O_{A_1} = (\ )$$

$$\begin{aligned} O_{A_2} &= (\,) \\ O_{A_3} &= (2) \\ O_{A_4} &= (\,) \\ O_{A_5} &= (4) \end{aligned}$$

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

$$\begin{array}{lccccc} A_1: & 0 & \textcircled{1} & 0 & 0 & 0 \\ A_2: & 0 & 0 & \textcircled{1} & 0 & 0 \\ A_3: & \textcircled{1} & 0 & 0 & 0 & 0 \\ A_4: & 0 & 0 & 0 & \textcircled{1} & 0 \\ A_5: & 0 & 0 & 0 & 0 & \textcircled{1} \end{array}$$

The disjoint ordering generated is:

$$\begin{aligned} O_{A_1} &= (2,1) \\ O_{A_2} &= (3,2) \\ O_{A_3} &= (1,3,2) \\ O_{A_4} &= (4,5) \\ O_{A_5} &= (5,3,4) \end{aligned}$$

4 Conclusion

We have developed an $O(n^3)$ algorithm that generates a disjoint ordering for n sets which have an SDR and whose elements are from a set A of size n. improving the previously known $O(n^4)$ algorithm. A trivial lower bound to this particular problem of finding a disjoint ordering of n sets is clearly $\Omega(n^2)$. It is certainly our intention to examine the possibility of further improving the $O(n^3)$ time algorithm we have just developed and for arbitrary n or finding a bigger lower bound.

As shown in [4], a disjoint ordering implies a set of shortest disjoint paths from the source node to the target nodes for the hypercube. Thus we immediately have an $O(n^3)$ routing algorithm for finding n disjoint shortest paths on an n-cube, provided that they do exist. In terms of finding disjoint shortest paths for one-to-many paradigm in interconnection networks, it is reasonable to think that this disjoint ordering finding problem/algorithm could be applicable to networks where nodes of the graph have this coordinate type of representations such as nodes in the hypercube and the star graph. And it is also possible that the problem of finding disjoint and shortest paths in other paradigms such as the one-to-one paradigm between two nodes (one-to-one) can be characterized by SDR/disjoint ordering. These issues are further discussed in [1]. We plan to find more applications for disjoint ordering of sets in the general area of routing in interconnection networks.

So far, we know that the existence of an SDR is the necessary and sufficient condition for the existence of disjoint and shortest paths from a single node to a set of nodes in some networks (e.g. the hypercube and star graph). In these cases, the disjoint ordering of sets correspond to the actual paths. In general, for any given sets A_1, A_2, ..., A_m, where $A_i \subseteq \{a_1, a_2, \cdots, a_n\}$, $m \leq n$, an SDR of A_1, A_2, ..., A_m implies a disjoint ordering of A_1, A_2, ..., A_m and vice versa. Also, finding an SDR is equivalent to finding a maximum matching in the bipartite graph with A_1, A_2, ..., A_m as one independent set and $\{a_1, a_2, , \cdots, a_n\}$ as the other where there is an edge between A_i and a_j iff $a_j \in A_i$. The necessary and sufficient condition from [4] is equivalent to the existence of a perfect matching for a balanced bipartite graph. To date, the best known algorithm by Hopcroft and Karp for finding such a matching takes $O(n^{5/2})$ time. In fact, both the $O(n^4)$ algorithm in [4] and our $O(n^3)$ algorithm for finding disjoint ordering of sets use this matching algorithm as a subroutine. Our goal is to improve the $O(n^3)$ algorithm. To this end, there are two possible

approaches. The first one is to keep using the matching algorithm as a subroutine. This approach immediately implies an $\Omega(n^{5/2})$ lower bound for the disjoint ordering of sets. The second one is to find an algorithm independent of the matching algorithm. This approach has an $\Omega(n^2)$ lower bound. But more importantly, this gives us another angle to the problem of matching. For example, for a balanced bipartite graph, its adjacency matrix can be viewed as consisting of rows, each of which corresponds to a set X_i. Clearly, a disjoint ordering of these sets implies a perfect matching. If a simpler and/or more efficient disjoint ordering of these sets can be found without using the matching algorithm, it would immediately imply a simpler and/or more efficient algorithm for a perfect matching for a balanced bipartite graph. However, whether it is possible to find a disjoint ordering of sets without first finding an SDR remains to be seen.

References

[1] E. Cheng, K. Qiu, and Z.Z. Shen. On Disjoint Shortest Paths Routing in Interconnection Networks: A Case Study in the Star Graph. *Manuscript*, 2012.

[2] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms,* 2nd ed. The MIT Press, 2001.

[3] M. Dietzfelbinger, S. Madhavapeddy, and I.H. Sudborough. Three Disjoint Path Paradigms in Star Networks. In *Proc. of the 3rd IEEE Symposium on Parallel and Distributed Processing*, Dallas, Texas, IEEE Computer Society Press, pages 400-406, 1991.

[4] S. Gao, B. Novick, and K. Qiu. From Hall's Matching Theorem to Optimal Routing on Hypercubes. *Journal of Combinatorial Theory* (B), 74(2): 291-301, 1998.

[5] T.F. Gonzalez and D. Serena. n-Cube Network: Node Disjoint Shortest Paths for Maximal Distance Pairs of Vertices. *Parallel Computing*, 30:973-998, 2004.

[6] Q.P. Gu and S.T. Peng. An Efficient Algorithm for k-Pairwise Disjoint Paths in Star Graphs. *Information Processing Letters*, 67:283-287, 1998.

[7] Q.P. Gu and S.T. Peng. An Efficient Algorithm for the k-Pairwise Disjoint Paths Problem in Hypercubes. *Journal of Parallel and Distributed Computing*, 60:764-774, 2000.

[8] P. Hall. On Representatives of Subsets. *J. London Math. Soc.*, 10:26-30, 1935.

[9] J.E. Hopcroft and R.M. Karp. An $n^{5/2}$ Algorithm for Maximum Matching in Bipartite Graphs. *J. SIAM COMP.*, 2:225-231, 1973.

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

[11] R.M. Karp and M.O. Rabin. Efficient Randomized Pattern-Matching Algorithms. Technical Report TR-31-81, Aiken Computation Laboratory, Harvard University, 1981.

[12] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity* Dover, Mineola, New York, 1998.

[13] K. Qiu and B. Novick. Disjoint Paths in Hypercubes. *Congressus Numerantium*, 119:105-112, 1996.

[14] M.O. Rabin. Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance. *Journal of ACM*, 36(2):335-348, 1989.