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

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. We study the problem of finding a disjoint ordering of sets \( X_1, X_2, \ldots, X_s \) where \( X_i \subseteq \{1, 2, \ldots, n\} \) and \( s \leq n \). We present an \( O(n^3) \) algorithm for doing so, under certain conditions, thus improving the previously known \( O(n^3) \) 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 \((s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)\) 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 as short as possible, 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 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.

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 with \( N = 2^n \) nodes 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 \( u \). 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 $O(n^3)$ 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 a 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.

## 2 System of Distinct Representatives and Disjoint Ordering of Sets

**Definition 1** Let $A = (A_1, A_2, \ldots, A_m)$ be a collection of subsets of a set $Y$. An ordered set of distinct elements $[a_1, a_2, \ldots, a_m]$ is called a system of distinct representatives (SDR) if $a_i \in A_i$, for $1 \leq i \leq m$.

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

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

\[
\begin{align*}
    u_1 &= u_{11}u_{12}\ldots u_{1n}, \\
    u_2 &= u_{21}u_{22}\ldots u_{2n}, \\
    \vdots \\
    u_n &= u_{n1}u_{n2}\ldots u_{nn}.
\end{align*}
\]

if $u_{ij} = 1$, then we include $j$ in set $U_i$. It was shown that an iff condition for $n$ disjoint paths to exist in this $n$-cube is that there exists a SDR for $U_1, U_2, \ldots, U_n$. For example, for nodes

\[
\begin{pmatrix}
    0 & 1 & 1 & 0 \\
    1 & 0 & 0 & 0 \\
    0 & 1 & 1 & 1 \\
    1 & 0 & 0 & 1 \\
\end{pmatrix}
\]

we have

\[
\begin{align*}
    U_1 &= \{2, 3\} \\
    U_2 &= \{1\} \\
    U_3 &= \{2, 3, 4\} \\
    U_4 &= \{1, 4\}.
\end{align*}
\]

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 a 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 $i_1i_2\cdots i_n$ of symbols from $\{1, 2, \ldots, n\}$ 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

\[
\begin{pmatrix}
    0 & 1 & 1 & 0 \\
    1 & 0 & 0 & 0 \\
    0 & 1 & 1 & 1 \\
    1 & 0 & 0 & 1 \\
\end{pmatrix}
\]

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 a 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** 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 a SDR. For example, for the following collection of sets and their disjoint ordering, 1, 3, 2, 4 form a SDR [1, 3, 2, 4]:
\[
\begin{align*}
A &= \{1, 2, 4\} & O_A &= (1, 2, 4) \\
B &= \{1, 3, 4\} & O_B &= (3, 4, 1) \\
C &= \{1, 2, 3\} & O_C &= (2, 3, 1) \\
D &= \{1, 2, 4\} & O_D &= (4, 1, 2).
\end{align*}
\]

Once again, for our previous example with SDR = [2, 1, 3, 4], a disjoint ordering is
\[
\begin{align*}
(2, 3) \\
(1) \\
(3, 4, 2) \\
(4, 1),
\end{align*}
\]

implying a routing:
\[
\begin{align*}
0000 &\rightarrow 0100 \rightarrow 0110 \\
0000 &\rightarrow 1000 \\
0000 &\rightarrow 0010 \rightarrow 0011 \rightarrow 0111 \\
0000 &\rightarrow 0001 \rightarrow 1001
\end{align*}
\]

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 a 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 a 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 \(Y = \{1, 2, \cdots, n\}\), \(m = n\) so we have \(A_1, A_2, \cdots, A_n\) where \(A_i \subseteq Y\). We also assume that a SDR is given. Clearly, \(m \leq n\) because if \(m > n\), it is impossible to have a 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
\[
a_{ij} = \begin{cases} 
0 & \text{if } j \notin A_i \\
1 & \text{if } j \in A_i.
\end{cases}
\]

With this representation, we then perform a sequence of row reductions to be described next until the representation matrix is reduced to a 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 a SDR in their matrix representation
\[
\begin{pmatrix}
        a_{11} & a_{12} & \cdots & a_{1n} \\
        a_{21} & a_{22} & \cdots & a_{2n} \\
        \vdots & \vdots & \ddots & \vdots \\
        a_{n1} & a_{n2} & \cdots & a_{nn}
\end{pmatrix}
\]

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 \(A_1, A_2, \cdots, A_{i-1}, A'_i, A_{i+1}, \cdots, A_n\) still have a SDR.

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 \(i\) 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 $u_i$. Otherwise (removing any such a 1 results in a set equal to $A_j$ for some $j \in \{1, 2, \ldots, n\} - \{i\}$), the situation can be illustrated below when removing the bold 1 will result in $A_i$ becoming $A_j$:

$$
\begin{align*}
A_1 & \\
A_2 & \\
\vdots & \\
A_j & a_{j1}a_{j2} \cdots 0 \cdots 1 \cdots a_{jn} \\
\vdots & \\
A_i & a_{i1}a_{i2} \cdots 1 \cdots 1 \cdots a_{jn} \\
\vdots & \\
A_n &
\end{align*}
$$

Note that because the $n$ sets have a 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:

$$
\begin{align*}
A_1 & \\
A_2 & \\
\vdots & \\
A_j & a_{j1}a_{j2} \cdots 0 \cdots 1 \cdots a_{jn} \\
\vdots & \\
A_i & a_{i1}a_{i2} \cdots 1 \cdots 1 \cdots a_{jn} \\
\vdots & \\
A_n &
\end{align*}
$$

Note that because the sets do have a 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 a 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 $A_1, A_2, \ldots, A_n$;

2. Convert the $n$ rows into integers, also compute the Hamming weights of all sets. Both sequences are then sorted;

3. Find a 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 $O(n)$.

For example, for the representation matrix

$$
\begin{align*}
A_1 & : \ 1 \ 0 \ 1 \ \now 1 \\
A_2 & : \ 0 \ 0 \ \now 1 \\
A_3 & : \ 1 \ \now 0 \ 0 \\
A_4 & : \ \now 1 \ 0 \ 0 
\end{align*}
$$

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 $A_4$ 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 $O(n)$.

\[
\begin{align*}
A_1 &: 1 1 \boxed{1} 1 \\
A_2 &: 0 \boxed{1} 1 \\
A_3 &: \boxed{1} 0 0 1 \\
A_4 &: \boxed{1} 0 0 1 \\
\end{align*}
\]

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 $H(A_i) = \max_{1 \leq j \leq n} \{H(A_j) > 1\}$; (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 a 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 a 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.

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:

\[
\begin{align*}
A_1 &: \boxed{1} 1 0 0 0 \\
A_2 &: 0 1 \boxed{0} 0 0 \\
A_3 &: 1 \boxed{0} 0 0 0 \\
A_4 &: 0 0 0 \boxed{1} 1 \\
A_5 &: 0 0 1 1 \boxed{1} \\
\end{align*}
\]

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:

\[
\begin{align*}
A_1 &: 1 \boxed{1} 0 0 0 \\
A_2 &: 0 1 \boxed{0} 0 0 \\
A_3 &: \boxed{1} 0 1 0 0 \\
A_4 &: 0 0 0 \boxed{1} 1 \\
A_5 &: 0 0 1 1 \boxed{1} \\
\end{align*}
\]

We now have a disjoint ordering as follows:

\[
\begin{align*}
O_{A_1} &= ( ) \\
O_{A_2} &= ( ) \\
O_{A_3} &= (2) \\
O_{A_4} &= ( ) \\
O_{A_5} &= ( ) \\
\end{align*}
\]

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
\[
\begin{align*}
  A_1 & : 1 \; \mathbf{0} \; 0 \; 0 \; 0 \\
  A_2 & : 0 \; 1 \; \mathbf{0} \; 0 \; 0 \\
  A_3 & : \mathbf{0} \; 0 \; 1 \; 0 \; 0 \\
  A_4 & : 0 \; 0 \; 0 \; \mathbf{1} \; 1 \\
  A_5 & : 0 \; 0 \; 1 \; 0 \; \mathbf{1}
\end{align*}
\]

The disjoint ordering now becomes
\[
\begin{align*}
  O_{A_1} & = ( ) \\
  O_{A_2} & = ( ) \\
  O_{A_3} & = (2) \\
  O_{A_4} & = ( ) \\
  O_{A_5} & = (4)
\end{align*}
\]

The next five reductions are straightforward and we will end up with:
\[
\begin{align*}
  A_1 & : 0 \; \mathbf{1} \; 0 \; 0 \; 0 \\
  A_2 & : 0 \; 0 \; \mathbf{1} \; 0 \; 0 \\
  A_3 & : \mathbf{0} \; 0 \; 0 \; 0 \; 0 \\
  A_4 & : 0 \; 0 \; 0 \; \mathbf{1} \; 0 \\
  A_5 & : 0 \; 0 \; 0 \; 0 \; \mathbf{1}
\end{align*}
\]

The disjoint ordering generated is:
\[
\begin{align*}
  O_{A_1} & = (2, 1) \\
  O_{A_2} & = (3, 2) \\
  O_{A_3} & = (1, 3, 2) \\
  O_{A_4} & = (4, 5) \\
  O_{A_5} & = (5, 3, 4)
\end{align*}
\]

## 4 Conclusion

We have developed an \(O(n^3)\) algorithm that generates a disjoint ordering for \(n\) sets who have a SDR and whose elements are from the set \(\{1, 2, \ldots, n\}\), 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.

## References


