# The Maximum Partition Matching Problem with Applications

Chi-Chang Chen* Jianer Chen†

**Abstract**

Let $\mathcal{S} = \{C_1, C_2, \ldots, C_k\}$ be a collection of pairwise disjoint subsets of $U = \{1, 2, \ldots, n\}$ such that $\bigcup_{i=1}^{k} C_i = U$. A *partition matching* of $\mathcal{S}$ consists of two subsets $\{a_1, a_2, \ldots, a_m\}$ and $\{b_1, b_2, \ldots, b_m\}$ of $U$ together with a sequence of distinct partitions of $\mathcal{S}$: $(\mathcal{A}_1, \mathcal{B}_1)$, $(\mathcal{A}_2, \mathcal{B}_2)$, ..., $(\mathcal{A}_m, \mathcal{B}_m)$ such that $a_i$ is contained in a subset in the collection $\mathcal{A}_i$ and $b_i$ is contained in a subset in the collection $\mathcal{B}_i$ for all $i = 1, \ldots, m$. An efficient algorithm is developed that constructs a maximum partition matching for a given collection $\mathcal{S}$. The algorithm can be used to construct optimal parallel routing between two nodes in interconnection networks.

## 1 Introduction

Matching is one of the most extensively studied areas in computer science, since it is interesting from a combinatorial point of view and has wide applications as well. Examples of matching are maximum graph matching problem [14], maximum graph adjacency matching problem [6], stable marriage problem [9], and 3-dimensional matching problem [8].

In this paper, we introduce a new maximum matching problem, study its computational complexity, and demonstrate its applications in interconnection networks. Let $\mathcal{S} = \{C_1, C_2, \ldots, C_k\}$ be a collection of subsets of the universal set $U = \{1, 2, \ldots, n\}$ such that $\bigcup_{i=1}^{k} C_i = U$, and $C_i \cap C_j = \phi$ for all $i \neq j$. A *partition matching* (of order $m$) of $\mathcal{S}$ consists of two ordered subsets $L = \{a_1, a_2, \ldots, a_m\}$ and $R = \{b_1, b_2, \ldots, b_m\}$ of $m$ elements of $U$ (the subsets $L$ and $R$ may not be disjoint), together with a sequence of $m$ distinct partitions of $\mathcal{S}$: $(\mathcal{A}_1, \mathcal{B}_1)$, $(\mathcal{A}_2, \mathcal{B}_2)$, ..., $(\mathcal{A}_m, \mathcal{B}_m)$ such that for all $i = 1, \ldots, m$, $a_i$ is contained in a subset in the collection $\mathcal{A}_i$ and $b_i$ is contained in a subset in the collection $\mathcal{B}_i$. The *Maximum Partition Matching* problem is to construct a partition matching of order $m$ for a given collection $\mathcal{S}$ with $m$ maximized.

The Maximum Partition Matching problem can be formulated in terms of 3-Dimensional Matching problem as follows: given an instance $\mathcal{S} = \{C_1, C_2, \ldots, C_k\}$ of the Maximum Partition Matching problem, we construct an instance $M$ for 3-Dimensional Matching problem such that a triple $(a, b, P)$ is contained in $M$ if and only if $a$ and $b$ are elements in $U$ and $P = (\mathcal{A}, \mathcal{B})$ is a partition of $\mathcal{S}$ such that $a$ is contained in a set in $\mathcal{A}$ and $b$ is contained in a set in $\mathcal{B}$. Unfortunately, the number of partitions of the collection $\mathcal{S}$ can be as large as $2^n$ thus the above reduction is not polynomial-time bounded. Moreover, 3-Dimensional Matching problem is NP-hard [8].

We will present an algorithm of running time $O(n^2 \log n)$ that solves the Maximum Partition Matching problem. We first show that when the number of subsets in the collection $\mathcal{S}$ is sufficiently large, a maximum partition matching can be constructed from a simpler "pre-matching" on the elements in $U$. For the case that the number of subsets in the collection $\mathcal{S}$ is small, we develop a greedy algorithm that uses a "chain justification" technique and finds a maximum partition matching. A sophisticated combinatorial analysis is given to prove the correctness of the algorithm.

*Department of Computer Science and Engineering, Tatung Institute of Technology, Taipei 10451, Taiwan, R.O.C.. Supported in part by Engineering Excellence Award from Texas A&M University. Email: ccchen@cse.ttit.edu.tw.

†Corresponding author. Department of Computer Science, Texas A&M University, College Station, TX 77843-3112. Supported in part by the National Science Foundation under Grant CCR-9110824. Email: chen@cs.tamu.edu.

The Maximum Partition Matching problem arises in connection with the parallel routing problem in interconnection networks. We show how the above algorithm can be applied to construct an optimal parallel routing in star networks, which have received considerable attention recently and have been shown to be an attractive alternative to the widely used hypercube networks [1]. In particular, we present an efficient algorithm that constructs between two arbitrary nodes in the $n$-dimensional star network a maximum number of node-disjoint *shortest* paths, which can be further used to construct between the two nodes $n-1$ node-disjoint paths of minimum bulk length (the *bulk length* of $n-1$ node-disjoint paths between two nodes is defined to be the length of the longest path among the $n-1$ paths). Note that these two problems on general graphs are NP-hard [7, 8]. These results significantly improve the previous parallel routing results in star networks [5, 10, 12, 13].

We introduce necessary terminologies that are used in the rest of our discussion.

Let $M = \langle L, R, (\mathcal{A}_1, \mathcal{B}_1), \ldots, (\mathcal{A}_m, \mathcal{B}_m)\rangle$ be a partition matching of the collection $\mathcal{S}$, where $L = \{a_1, \ldots, a_m\}$ and $R = \{b_1, \ldots, b_m\}$. We will say that the partition $(\mathcal{A}_i, \mathcal{B}_i)$ *left-pairs* the element $a_i$ and *right-pairs* the element $b_i$. An element $a$ is said to be *left-paired* if it is in the set $L$. Otherwise, the element $a$ is *left-unpaired*. Similarly we define *right-paired* and *right-unpaired* elements. The collections $\mathcal{A}_i$ and $\mathcal{B}_i$ are called the *left-collection* and *right-collection* of the partition $(\mathcal{A}_i, \mathcal{B}_i)$. The partition matching $M$ may also be written as $M[(a_1, b_1), \ldots, (a_m, b_m)]$ if the partitions $(\mathcal{A}_1, \mathcal{B}_1)$, …, $(\mathcal{A}_m, \mathcal{B}_m)$ are implied. The partition matching $M[(a_1, b_1), \ldots, (a_m, b_m)]$ is *maximum* if $m$ is the largest among all partition matchings of $\mathcal{S}$.

A permutation $u = a_1 a_2 \cdots a_n$ of the elements in the set $U = \{1, 2, \ldots, n\}$ can be given by a product of disjoint cycles [2], which will be called the *cycle structure* of the permutation. A $\pi[1, i]$ *transposition* on $u$ is to exchange the positions of $a_1$ and $a_i$ in $u$. More specifically, $\pi[1, i](u) = a_i a_2 a_3 \cdots a_{i-1} a_1 a_{i+1} \cdots a_n$. It is sometimes more convenient to write the transposition $\pi[1, i](u)$ as $\pi[a_i](u)$ to indicate that the transposition exchanges the position of the first symbol and the symbol $a_i$ in $u$. Let us consider how a transposition changes the cycle structure of a permutation. Write $u$ in its cycle structure

$$u = (a_1^{(1)} \cdots a_{n_1}^{(1)}\ 1)(a_1^{(2)} \cdots a_{n_2}^{(2)}) \cdots (a_1^{(k)} \cdots a_{n_k}^{(k)})$$

Now suppose we apply the transposition $\pi[1, i]$ on $u$. There are two cases.

If $a_i$ is not in the cycle containing the symbol 1, then $\pi[1, i]$ "merges" the cycle containing 1 with the cycle containing $a_i$. More precisely, suppose that $a_i = a_1^{(2)}$ (note that each cycle can be cyclically permuted and the order of the cycles is not important), then the permutation $\pi[1, i](u)$ will have the following cycle structure:

$$\pi[1, i](u) = (a_1^{(2)} \cdots a_{n_2}^{(2)} a_1^{(1)} \cdots a_{n_1}^{(1)}\ 1)(a_1^{(3)} \cdots a_{n_3}^{(3)}) \cdots (a_1^{(k)} \cdots a_{n_k}^{(k)})$$

If $a_i$ is in the cycle containing the symbol 1, then $\pi[1, i]$ "splits" the cycle. More precisely, suppose that $a_i = a_j^{(1)}$, $j > 1$ (note that $a_1^{(1)} = a_1$ and we assume $i > 1$), then $\pi[1, i](u)$ will have the following cycle structure:

$$\pi[1, i](u) = (a_1^{(1)} \cdots a_{j-1}^{(1)})(a_j^{(1)} \cdots a_{n_1}^{(1)} 1)(a_1^{(2)} \cdots a_{n_2}^{(2)}) \cdots (a_1^{(k)} \cdots a_{n_k}^{(k)})$$

Note that if a symbol $a_i$ is in a single symbol cycle in a cycle structure of a permutation $u = a_1 a_1 \cdots a_n$, then the symbol is in its "correct" position, i.e., $a_i = i$, and that if a symbol is in a cycle containing more than one symbol, then the symbol is not in its correct position. Denote by $\varepsilon$ the identity permutation $\varepsilon = 12 \cdots n$.

**Algorithm.** PRE-MATCHING

INPUT: The collection $\mathcal{S} = \{C_1, \ldots, C_k\}$ of subsets of $U$, with $2^k \geq 2n$.

OUTPUT: A partition matching $M$ in $\mathcal{S}$

1. $\mathcal{T} = \mathcal{S}$;
2. **while** $\mathcal{T}$ contains more than one set
   but does not consists of exactly three singular sets **do**
   
   Pick two sets $C$ and $C'$ of largest cardinality in $\mathcal{T}$;
   
   Pick an element $a$ in $C$ and an element $a'$ in $C'$;
   
   Pick two unused partitions of $\mathcal{S}$ of the forms $(\mathcal{A} \cup \{C\}, \mathcal{B} \cup \{C'\})$ and $(\mathcal{B} \cup \{C'\}, \mathcal{A} \cup \{C\})$, where $\mathcal{A} \cup \mathcal{B} = \mathcal{S} - \{C, C'\}$;
   
   Ues the partition $(\mathcal{A} \cup \{C\}, \mathcal{B} \cup \{C'\})$ to left-pair $a$ and right-pair $a'$;
   
   Use the partition $(\mathcal{B} \cup \{C'\}, \mathcal{A} \cup \{C\})$ to left-pair $a'$ and right-pair $a$;
   
   $C = C - \{a\}$; $\quad C' = C' - \{a'\}$;
   
   If any of the sets $C$ and $C'$ is an empty set now, then delete it from the collection $\mathcal{T}$;
3. **if** $\mathcal{T}$ consists of exactly three singular sets $C_1 = \{a_1\}$, $C_2 = \{a_2\}$, and $C_3 = \{a_3\}$
   **then** Pick three unused partitions of $\mathcal{S}$ of the forms $(\mathcal{A}_1 \cup \{C_1\}, \mathcal{B}_1 \cup \{C_2\})$, $(\mathcal{A}_2 \cup \{C_2\}, \mathcal{B}_2 \cup \{C_3\})$, and $(\mathcal{A}_3 \cup \{C_3\}, \mathcal{B}_3 \cup \{C_1\})$, to left-pair $a_1$ and right-pair $a_2$, left-pair $a_2$ and right-pair $a_3$, and left-pair $a_3$ and right-pair $a_1$, respectively.

Figure 1: The algorithm PRE-MATCHING

## 2 Partition matching via element pre-matching

For the rest of this paper, we assume that $U = \{1, 2, \ldots, n\}$ and that $\mathcal{S} = \{C_1, C_2, \ldots, C_k\}$ is a collection of pairwise disjoint subsets of $U$ such that $\bigcup_{i=1}^{k} C_i = U$.

A necessary condition for two subsets $\{a_1, a_2, \ldots, a_m\}$ and $\{b_1, b_2, \ldots, b_m\}$ of $U$ to form a partition matching for the collection $\mathcal{S}$ is that $a_i$ and $b_i$ belong to different subsets in the collection $\mathcal{S}$, for all $i = 1, 2, \ldots, m$. This motivates the following definition.

**Definition** Two subsets $\{a_1, a_2, \ldots, a_m\}$ and $\{b_1, b_2, \ldots, b_m\}$ of $U$ form an *element pre-matching* $P = \{(a_i, b_i) \mid 1 \leq i \leq m\}$ if $a_i$ and $b_i$ do not belong to the same subset in the collection $\mathcal{S}$, for all $i = 1, 2, \ldots, m$. The element pre-matching $P$ is *maximum* if $m$ is the largest among all element pre-matchings of $\mathcal{S}$.

The following lemma follows directly from the definition.

**Lemma 2.1** *Let $\{a_1, a_2, \ldots, a_m\}$ and $\{b_1, b_2, \ldots, b_m\}$ be a maximum element pre-matching of $\mathcal{S}$. If the subsets $\{a_1, a_2, \ldots, a_m\}$ and $\{b_1, b_2, \ldots, b_m\}$ also form a partition matching $M$ of $\mathcal{S}$, then $M$ is a maximum partition matching.*

We show in this section that when the number of subsets in the collection $\mathcal{S}$ is sufficiently large, a maximum partition matching in $\mathcal{S}$ can be constructed via a maximum element pre-matching. Consider the algorithm PRE-MATCHING in Figure 1. We say that a set is *singular* if it consists of a single element.

The rest of this section is for a proof of correctness and complexity analysis of the algorithm PRE-MATCHING.

**Lemma 2.2** *Let $P = \{(a_i, b_i) \mid 1 \le i \le m\}$ be the set of pairs constructed by the algorithm* Pre-Matching, *then $P$ forms a maximum element pre-matching for the collection $\mathcal{S}$.*

Proof. Since for each pair $(a_i, b_i)$ constructed by the algorithm Pre-Matching, the elements $a_i$ and $b_i$ are from different sets in $\mathcal{S}$, the set $P$ forms an element pre-matching. We show that this element pre-matching $P$ is maximum.

Denote by $|C_i|$ the cardinality of the set $C_i$. Without loss of generality, we assume $|C_1| \ge |C_2| \ge \cdots \ge |C_k|$. There are two cases.

**Case 1.** $|C_1| > n/2$. Because in an element pre-matching, each pair $(a, b)$ must have either $a \in \cup_{i=2}^k C_i$ or $b \in \cup_{i=2}^k C_i$, a maximum element pre-matching can have at most $2\sum_{i=2}^k |C_i|$ pairs.

On the other hand, in case $|C_1| > n/2$, the algorithm Pre-Matching actually constructs $2\sum_{i=2}^k |C_i|$ pairs: since $|C_1| > \sum_{i=2}^k |C_i|$, in each execution of the body of the **while** loop in Step 2, the set $C_1$ always has the largest cardinality. Thus, the algorithm always picks an element $a$ in $C_1$ and an element $b$ in $\bigcup_{i=2}^k C_i$, then makes the pairs $(a, b)$ and $(b, a)$. The loop will stop when all elements in $\bigcup_{i=2}^k C_i$ have been used, which results in exactly $2\sum_{i=2}^k |C_i|$ pairs. Thus, the lemma is true for this case.

**Case 2.** $|C_1| \le n/2$. In this case we show that the algorithm Pre-Matching always constructs an element pre-matching with $n$ pairs (thus maximum).

If $\mathcal{S}$ consists of only two sets $C_1$ and $C_2$, then since $|C_1| \ge |C_2|$, $|C_1| \le n/2$, and $|C_1| + |C_2| = n$, we must have $|C_1| = |C_2|$. Now it becomes trivial to verify in this case that the algorithm Pre-Matching constructs a maximum element pre-matching with $n$ pairs.

Thus we assume that the collection $\mathcal{S}$ contains at least three sets. We prove the lemma for this case by induction on the size $n$ of the universal set $U$. When $n = 3$, the collection $\mathcal{S}$ consists of exactly three singular sets, and Step 3 of the algorithm shows how an element pre-matching with three pairs can be constructed. Now assume that $n > 3$. Note that after the first execution of the body of the **while** loop in Step 2, the collection $\mathcal{T}$ becomes

$$\mathcal{T}' = \{C_1', C_2', C_3, \ldots, C_k\}$$

where $C_1' = C_1 - \{a\}$, $C_2' = C_2 - \{a'\}$, and $|C_1'| + |C_2'| + |C_3| + \cdots + |C_k| = n - 2$. We show that the largest set in $\mathcal{T}'$ contains at most $(n-2)/2$ elements.

If $C_1'$ is still the largest set, then from $|C_1| \le n/2$, we have $|C_1| \le \sum_{i=2}^k |C_i|$. Consequently, $|C_1'| \le |C_2'| + |C_3| + \cdots + |C_k|$, i.e., $|C_1'| \le (n-2)/2$. On the other hand, if $C_1'$ is no longer the largest set, then $C_3$ must be the largest set in $\mathcal{T}'$, and before the first execution of the body of the **while** loop in Step 2, we have $|C_1| = |C_2| = |C_3|$. Thus, $|C_3| \le n/3$. Note that $|C_3|$ is an integer, thus $|C_3| \le n/3$ implies $|C_3| \le (n-2)/2$ for all $n > 3$.

Thus, the collection $\mathcal{T}'$ consists of $k$ subsets of the universal set $U - \{a, a'\}$ of $n-2$ elements, and the largest set in $\mathcal{T}'$ contains no more than $(n-2)/2$ elements. Note that the algorithm Pre-Matching applies the same strategy on the collection $\mathcal{T}'$. By the inductive hypothesis, the algorithm constructs an element pre-matching $P'$ with $n-2$ pairs for the collection $\mathcal{T}'$. Combining this with the pairs $(a, a')$ and $(a', a)$ constructed in the first execution of the body of the **while** loop in Step 2, we obtain an element pre-matching with $n$ pairs for the collection $\mathcal{S}$.

Combining all these analyses together, we conclude with the lemma. □

To complete the correctness proof for the algorithm Pre-Matching, we only need to show that for each pair $(a, a')$ constructed by the algorithm, there is always a distinct partition of $\mathcal{S}$ that implements the pairing. By the assumption of the algorithm, we have $2^k \ge 2n$.

We first consider Step 2 of the algorithm Pre-Matching. Suppose that at some moment, the algorithm needs a partition to left-pair an element $a \in C$ and right-pair an element $a' \in C'$. Note that each execution of the body of the **while** loop uses two partitions of the forms $(\mathcal{A}, \mathcal{B})$ and $(\mathcal{B}, \mathcal{A})$. Thus each execution of the body of the **while** loop can use at most one partition whose left-collection contains $C$ and whose right-collection contains $C'$. Consequently, less than $\lfloor n/2 \rfloor$ partitions whose left-collection contains $C$ and whose right-collection contains $C'$ have been used. Since $2^k \geq 2n$ and there are $2^{k-2} \geq n/2$ partitions of $\mathcal{S}$ whose left-collection contains $C$ and whose right-collection contains $C'$, we conclude that there is always an unused partition $P$ that can be used to left-pair the element $a$ and right-pair the element $b$.

The proof goes similarly for Step 3. For example, suppose we want to left-pair the element $a_3 \in C_3$ and right-pair the element $a_1 \in C_1$. There are totally $2^{k-2} \geq n/2$ partitions of $\mathcal{S}$ whose left-collection contains $C_3$ and whose right-collection contains $C_1$, of which at most $(n-3)/2$ have been used (by Step 2). Also note that no partition that is used to left-pair $a_1$ and right-pair $a_2$, or left-pair $a_2$ and right-pair $a_3$ in Step 3 have $C_3$ in its left-collection and $C_1$ in its right-collection. Therefore, there is always an unused partition that can be used to left-pair $a_3$ and right-pair $a_1$.

This shows that the algorithm Pre-Matching constructs a partition matching in the collection $\mathcal{S}$. Combining this with Lemma 2.1 and Lemma 2.2, we have the following theorem.

**Theorem 2.3** *Let $\mathcal{S} = \{C_1, C_2, \ldots, C_k\}$ be a collection of nonempty subsets of the universal set $U = \{1, 2, \ldots, n\}$ such that $\bigcup_{i=1}^{k} C_i = U$ and $C_i \cap C_j = \phi$, for $i \neq j$. If $2^k \geq 2n$, then the algorithm* Pre-Matching *constructs a maximum partition matching in $\mathcal{S}$.*

The algorithm Pre-Matching can be easily implemented to have running time $O(n^2)$. For this, we keep a list of used partitions of $\mathcal{S}$ (each partition can be represented by a binary number of $k$ bits). When a new partition is needed to pair $(C, C')$, we go through the list of used partitions to identify those partitions whose left-collection contains $C$ and whose right-collection contains $C'$. Then we systematically generate the "next" partition whose left-collection contains $C$ and whose right-collection contains $C'$, use it to pair $(C, C')$, and add the partition to the list of used partitions. Therefore, in time $O(n)$, we can find an unused partition to pair two elements and update our record. We conclude that the algorithm Pre-Matching runs in time $O(n^2)$.

## 3 Maximum partition matching: general case

According to Theorem 2.3, we only have to investigate maximum partition matchings for collections of $k$ subsets in $U$ such that $2^k < 2n$. We shall show in this section that a maximum partition matching for such collections can be constructed by a greedy strategy.

Suppose we have constructed a partition matching $M = M[(a_1, b_1), \ldots, (a_h, b_h)]$ and want to expand this matching. The partitions of the collection $\mathcal{S}$ then can be classified into two classes: $h$ of the partitions are used to pair the $h$ pairs $(a_i, b_i)$, $i = 1, \ldots, h$, and the rest $2^k - h$ partitions are unused. Now if there is an unused partition $P = (\mathcal{A}, \mathcal{B})$ such that there is a left-unpaired element $a$ in $\mathcal{A}$ and a right-unpaired element $b$ in $\mathcal{B}$, then we simply pair the element $a$ with the element $b$ using the partition $P$, thus expanding the partition matching $M$.

Now suppose that there is no such an unused partition, i.e., for all unused partitions $(\mathcal{A}, \mathcal{B})$, either $\mathcal{A}$ contains no left-unpaired elements or $\mathcal{B}$ contains no right-unpaired elements. This case may not necessarily imply that the current partition matching is the maximum. For example, suppose that $(\mathcal{A}, \mathcal{B})$ is an unused partition such that there is a left-unpaired element $a$ in $\mathcal{A}$ but no right-unpaired elements in $\mathcal{B}$. Assume further that there is a *used* partition $(\mathcal{A}', \mathcal{B}')$ that pairs elements $(a', b')$, such that the element $b'$ is in $\mathcal{B}$ and there is a right-unpaired element $b$ in $\mathcal{B}'$. Then we can let the partition $(\mathcal{A}', \mathcal{B}')$ pair the elements $(a', b)$, and then let the partition $(\mathcal{A}, \mathcal{B})$ pair the elements $(a, b')$, thus expanding the partition matching $M$. An explanation of this process is that the used partitions have been incorrectly used to pair elements, thus in order to construct a maximum partition matching, we must re-pair some of the elements. To further investigate this relation, we need to introduce a few notations.

For a used partition $P$ of $\mathcal{S}$, we put an underline on a set in the left-collection (resp. the right-collection) of $P$ to indicate that an element in the set is left-paired (resp. right-paired) by the partition $P$. The sets will be called the *left-paired set* and the *right-paired set* of the partition $P$, respectively.

**Definition** A used partition $P$ is *directly left-reachable* from a partition $P_1 = (\mathcal{A}_1, \mathcal{B}_1)$ if the left-paired set of $P$ is contained in $\mathcal{A}_1$ (the partition $P_1$ can be either used or unused). The partition $P$ is *directly right-reachable* from a partition $P_2 = (\mathcal{A}_2, \mathcal{B}_2)$ if the right-paired set of $P$ is contained in $\mathcal{B}_2$. A partition $P_s$ is *left-reachable* (resp. *right-reachable*) from a partion $P_1$ if there are partitions $P_2$, ..., $P_{s-1}$ such that $P_i$ is directly left-reachable (resp. directly right-reachable) from $P_{i-1}$, for all $i = 2, \ldots, s$.

The left-reachability and the right-reachability are transitive relations.

Let $P_1 = (\mathcal{A}_1, \mathcal{B}_1)$ be an unused partition such that there is no left-unpaired elements in $\mathcal{A}_1$, and let $P_s = (\mathcal{A}_s, \mathcal{B}_s)$ be a partition left-reachable from $P_1$ and there is a left-unpaired element $a_s$ in $\mathcal{A}_s$. We show how we can use a *chain justification* to make a left-unpaired element for the collection $\mathcal{A}_1$.

By the definition, there are used partitions $P_2$, ..., $P_{s-1}$ such that $P_i$ is directly left-reachable from $P_{i-1}$, for $i = 2, \ldots, s$. We can further assume that $P_i$ is *not* directly left-reachable from $P_{i-2}$ for $i = 3, \ldots, s$ — otherwise we simply delete the partition $P_{i-1}$ from the sequence. Thus, these partitions can be written as

$$P_1 = (\{C_1\} \cup \mathcal{A}_1', \mathcal{B}_1), \quad P_2 = (\{\underline{C_1}, C_2\} \cup \mathcal{A}_2', \mathcal{B}_2), \quad P_3 = (\{\underline{C_2}, C_3\} \cup \mathcal{A}_3', \mathcal{B}_3),$$
$$\ldots, \quad P_{s-1} = (\{\underline{C_{s-2}}, C_{s-1}\} \cup \mathcal{A}_{s-1}', \mathcal{B}_{s-1}), \quad P_s = (\{\underline{C_{s-1}}, C_s\} \cup \mathcal{A}_s', \mathcal{B}_s)$$

where $\mathcal{A}_1'$, ..., $\mathcal{A}_s'$ are subcollections of $\mathcal{S}$ without an underlined set.

We can assume that the left-unpaired element $a_s$ in $\mathcal{A}_s = \{\underline{C_{s-1}}, C_s\} \cup \mathcal{A}_s'$ is in a non-underlined set $C_s$ in $\mathcal{A}_s$ — otherwise we consider the sequence $P_1$, $P_2$, ..., $P_{s-1}$ instead. We modify the partition sequence into

$$P_1 = (\{C_1\} \cup \mathcal{A}_1', \mathcal{B}_1), \quad P_2 = (\{C_1, \underline{C_2}\} \cup \mathcal{A}_2', \mathcal{B}_2), \quad P_3 = (\{C_2, \underline{C_3}\} \cup \mathcal{A}_3', \mathcal{B}_3),$$
$$\ldots, \quad P_{s-1} = (\{C_{s-2}, \underline{C_{s-1}}\} \cup \mathcal{A}_{s-1}', \mathcal{B}_{s-1}), \quad P_s = (\{C_{s-1}, \underline{C_s}\} \cup \mathcal{A}_s', \mathcal{B}_s)$$

The interpretation is as follows: we use the partition $P_s$ to left-pair the left-unpaired element $a_s$ (the right-paired element in the right-collection $\mathcal{B}_s$ is unchanged). Thus, the element $a_{s-1}$ in the set $C_{s-1}$ the partition $P_s$ used to left-pair becomes left-unpaired. We then use the partition $P_{s-1}$ to left-pair the element $a_{s-1}$ and leave an element $a_{s-2}$ in the set $C_{s-2}$ left-unpaired, then we use the partition $P_{s-2}$ to left-pair $a_{s-2}$, $\cdots\cdots$. At the end, we use the partition $P_2$ to left-pair an element $a_2$ in the set $C_2$ and leave an element $a_1$ in the set $C_1$ left-unpaired. Therefore, this process makes an element in the left-collection $\mathcal{A}_1 = \{C_1\} \cup \mathcal{A}_1'$ of the partition $P_1$ left-unpaired.

The above process will be called a *left-chain justification*. Thus, given an unused partition $P_1 = (\mathcal{A}_1, \mathcal{B}_1)$ in which the left-collection $\mathcal{A}_1$ has no left-unpaired elements and given a used

**Algorithm.** GREEDY EXPANDING

INPUT: A collection $\mathcal{S} = \{C_1, \ldots, C_k\}$ of pairwise disjoint subsets of $U$, with $2^k < 2n$.
OUTPUT: A partition matching $M_{exp}$ in $\mathcal{S}$

Start with an empty partition matching $M_{exp}$ and repeat the following steps until no more changes.

1. **case 1.** There is an unused partition $P = (\mathcal{A}, \mathcal{B})$ that has a left-unpaired element $a$ in $\mathcal{A}$ and a right-unpaired element $b$ in $\mathcal{B}$

   Then pair the two elements $(a, b)$ by the partition $P$ and add $P$ to the matching $M_{exp}$;
2. **case 2.** There is an unused partition $P = (\mathcal{A}, \mathcal{B})$ that has a right-unpaired element $b$ in $\mathcal{B}$ but no left-unpaired elements in $\mathcal{A}$, and there is a used partition $P' = (\mathcal{A}', \mathcal{B}')$ left-reachable from $P$ and the collection $\mathcal{A}'$ has a left-unpaired element

   Then use the left-chain justification to make a left-unpaired element $a$ for the collection $\mathcal{A}$, pair the two elements $(a, b)$ by the partition $P$, and add $P$ to the matching $M_{exp}$;
3. **case 3.** There is an unused partition $P = (\mathcal{A}, \mathcal{B})$ that has a left-unpaired element $a$ in $\mathcal{A}$ but no right-unpaired elements in $\mathcal{B}$, and there is a used partition $P' = (\mathcal{A}', \mathcal{B}')$ right-reachable from $P$ and the collection $\mathcal{B}'$ has a right-unpaired element

   Then use the right-chain justification to make a right-unpaired element $b$ for the collection $\mathcal{B}$, pair the two elements $(a, b)$ by the partition $P$, and add $P$ to the matching $M_{exp}$;
4. **case 4.** There is an unused partition $P = (\mathcal{A}, \mathcal{B})$ that has no left-unpaired elements in $\mathcal{A}$ and no right-unpaired elements in $\mathcal{B}$, and there are two used partitions $P' = (\mathcal{A}', \mathcal{B}')$ left-reachable from $P$ and $P'' = (\mathcal{A}'', \mathcal{B}'')$ right-reachable from $P$ such that $\mathcal{A}'$ has a left-unpaired element and $\mathcal{B}''$ has a right-unpaired element

   Then use the left-chain justification to make a left-unpaired element $a$ for $\mathcal{A}$, and use the right-chain justification to make a right-unpaired element $b$ for $\mathcal{B}$, then pair the two elements $(a, b)$ by the partition $P$, and add $P$ to the matching $M_{exp}$;

Figure 2: The algorithm GREEDY EXPANDING

partition $P_s = (\mathcal{A}_s, \mathcal{B}_s)$ left-reachable from $P_1$ such that the left-collection $\mathcal{A}_s$ of $P_s$ has a left-unpaired element, we can apply the left-chain justification that keeps all used partitions in the partition matching $M$ and makes a left-unpaired element for the partition $P_1$. A process called *right-chain justification* for right-collections of the partitions can be described similarly. This motivates the algorithm GREEDY EXPANDING in Figure 2.

**Theorem 3.1** *The algorithm* GREEDY EXPANDING *runs in time* $O(n^2 \log n)$.

PROOF. For each set $C_i$, we keep a counter for the number of left-unpaired elements in $C_i$ and a counter for the number of right-unpaired elements in $C_i$ so that checking whether a partition has an unpaired element can be done in time $O(k)$.

To test left-reachabilities in each execution of Steps 1-4, we construct a directed graph $G$ whose vertex set consists of all used partitions and all the sets in $\mathcal{S}$. The edges of $G$ are defined by the direct left-reachability. Moreover, there is an edge from a set $C_i$ in $\mathcal{S}$ to a used partition $P$ if and only if $C_i$ is the left-paired set of $P$. Now for each set $C_i$ in $\mathcal{S}$, we check in the graph $G$ in time $O(|G|) = O(n)$ whether there is a path from the node $C_i$ to a used partition $P$ such that $P$ has a left-unpaired element in its left-collection. If such a used partition $P$ exists, we associate the set $C_i$ with the partition $P$.

Now to check for each unused partition $P' = (\mathcal{A}', \mathcal{B}')$ whether there is a used partition that is left-reachable from $P'$ and has a left-unpaired element, we only need to check whether the left-collection $\mathcal{A}'$ of $P'$ contains a set $C_i$ that is associated with a used partition.

The right-reachability can be handled similarly.

Therefore, after the $O(kn)$ time preprocessing that associates each set $C_i$ in $\mathcal{S}$ with a possible used partition, the conditions in cases 2-4 in the algorithm Greedy Expanding can be tested in time $O(k)$ for each unused partition.

Since there are at most $2^k < 2n$ partitions of $\mathcal{S}$ and each execution of the loop (steps 1-4) of the algorithm Greedy Expanding adds at least one partition to the partition matching $M_{exp}$, we conclude that the algorithm Greedy Expanding runs in time $O(kn^2) = O(n^2 \log n)$. □

After execution of the algorithm Greedy Expanding, we obtain a partition matching $M_{exp}$. For each partition $P = (\mathcal{A}, \mathcal{B})$ not included in $M_{exp}$, either $\mathcal{A}$ has no left-unpaired elements and no used partition left-reachable from $P$ has a left-unpaired element in its left-collection, or $\mathcal{B}$ has no right-unpaired elements and no used partition right-reachable from $P$ has a right-unpaired element in its right-collection.

**Definition** Define $L_{free}$ to be the set of partitions $P$ not used by $M_{exp}$ such that the left-collection of $P$ has no left-unpaired elements and no used partition left-reachable from $P$ has a left-unpaired element in its left-collection, and define $R_{free}$ to be the set of partitions $P'$ not used by $M_{exp}$ such that the right-collection of $P'$ has no right-unpaired elements and no used partition right-reachable from $P'$ has a right-unpaired element in its right-collection.

According to the algorithm Greedy Matching, each partition not used by $M_{exp}$ is either in the set $L_{free}$ or in the set $R_{free}$. The sets $L_{free}$ and $R_{free}$ may not be disjoint.

**Definition** Define $L_{reac}$ to be the set of partitions in $M_{exp}$ that is left-reachable from a partition in $L_{free}$, define $R_{reac}$ to be the set of partitions in $M_{exp}$ that is right-reachable from a partition in $R_{reac}$.

According to the definitions, if a used partition $P$ is in the set $L_{reac}$, then all elements in its left-collection are left-paired, and if a used partition $P$ is in the set $R_{reac}$, then all elements in its right-collection are right-paired.

We first show that if $L_{reac}$ and $R_{reac}$ are not disjoint, then we can construct a maximum partition matching from the partition matching $M_{exp}$ constructed by the algorithm Greedy Expanding.

**Lemma 3.2** *If the sets $L_{reac}$ and $R_{reac}$ contain a common partition and the partition matching $M_{exp}$ has less than n pairs, then there is a set $C_0$ in $\mathcal{S}$, $|C_0| \leq n/2$, such that either all elements in each set $C \neq C_0$ are left-paired and every used partition whose left-paired set is not $C_0$ is contained in $L_{reac}$, or all elements in each set $C \neq C_0$ are right-paired and every used partition whose right-paired set is not $C_0$ is contained in $R_{reac}$.*

Proof. Let the partition $P = (\mathcal{A}, \mathcal{B})$ be in the intersection of $L_{reac}$ and $R_{reac}$. Therefore, all elements in $\mathcal{A}$ are left-paired and all elements in $\mathcal{B}$ are right-paired. Since there are totally $n$ elements in $\mathcal{A} \cup \mathcal{B}$, one of $\mathcal{A}$ and $\mathcal{B}$ has at least $n/2$ elements.

Suppose that $\mathcal{A}$ has at least $n/2$ elements, which are all left-paired. Since $M_{exp}$ has less than $n$ pairs, there is a set $C_0$ that contains left-unpaired elements. In particular, $C_0$ is not contained in the collection $\mathcal{A}$. Thus, $|C_0| \leq n/2$.

Let $P_1$, …, $P_t$ be the partitions in $M_{exp}$ that are used to left-pair the elements in $\mathcal{A}$. Thus, $t \geq n/2$. Since the left-paired set of each $P_i$ is also contained in $\mathcal{A}$, by the definition, $P_i$ is directly left-reachable from $P = (\mathcal{A}, \mathcal{B})$. Now $P \in L_{reac}$. Thus, we also have $P_i \in L_{reac}$, for $i = 1, \ldots, t$. In particular, all elements in the left-collection of each $P_i$ have been left-paired. Consequently, the set $C_0$ is not contained in the left-collection of any of these partitions $P_1$, ..., $P_t$.

Suppose that there is another set $C \neq C_0$ in $\mathcal{S}$ that is also not contained in the left-collection of any of the partitions $P_1$, ..., $P_t$. Then since the total number of partitions whose left-collections do not contain the sets $C_0$ and $C$ is $2^{k-2}$, we get $2^{k-2} \geq t$. However, this would contradict the facts that $t \geq n/2$ and $2^k < 2n$.

Therefore, the set $C_0$ is the only set in $\mathcal{S}$ that is not contained in the left-collection of any of the partitions $P_1$, ..., $P_t$. In particular, the set $C_0$ is the only set whose left-collection contains left-unpaired elements. All elements in each set $C \neq C_0$ are left-paired.

Now let $P'$ be a used partition whose left-paired set $C \neq C_0$. Since the set $C$ must be contained in the left-collection of some partition $P_i$ among $\{P_1, \ldots, P_t\}$, by the definition of the left-reachability, the partition $P'$ is left-reachable from the partition $P_i$. Since the partition $P_i$ is in $L_{reac}$, we conclude that the partition $P'$ is also in $L_{reac}$.

Thus, we have proved that if the left-collection $\mathcal{A}$ of the partition $P = (\mathcal{A}, \mathcal{B})$ in the intersection of $L_{reac}$ and $R_{reac}$ has at least $n/2$ elements, then there is a set $C_0$ in $\mathcal{S}$, $|C_0| \leq n/2$, such that all elements in a partition $C \neq C_0$ are left-paired and every used partition whose left-paired set is not $C_0$ is in the set $L_{reac}$.

In case the right-collection $\mathcal{B}$ of the partition $P$ has at least $n/2$ elements, we can prove in a completely similar way that there is a set $C_0$ in $\mathcal{S}$, $|C_0| \leq n/2$, such that all elements in a partition $C \neq C_0$ are right-paired and every used partition whose right-paired set is not $C_0$ is in $R_{reac}$. $\square$

Now we are ready for the following theorem.

**Theorem 3.3** *If $L_{reac}$ and $R_{reac}$ have a common partition, then the collection $\mathcal{S}$ has a maximum partition matching of $n$ pairs, which can be constructed in linear time from the partition matching $M_{exp}$ constructed by the algorithm* Greedy Expanding.

Proof. If $M_{exp}$ has $n$ pairs, then $M_{exp}$ is already a maximum partition matching. Thus we assume that $M_{exp}$ has less than $n$ pairs. According to Lemma 3.2, we can assume, without loss of generality, that all elements in each set $C_i$, $i = 2, \ldots, k$, are left-paired, and that every used partition whose left-paired set is not $C_1$ is in $L_{reac}$. Moreover, $|C_1| \leq \sum_{i=2}^{k} |C_i|$.

Let $t = \sum_{i=2}^{k} |C_i|$ and $d = |C_1|$. Then we can assume that $M_{exp}$ consists of the partitions

$$P_1, \ldots, P_t, P_{t+1}, \ldots, P_{t+h}$$

where $P_1$, ..., $P_t$ are used by $M_{exp}$ to left-pair the elements in $\cup_{i=2}^{k} C_i$, and $P_{t+1}$, ..., $P_{t+h}$ are used by $M_{exp}$ to left-pair the elements in $C_1$, $h < d$. Moreover, all partitions $P_1$, ..., $P_t$ are in the set $L_{reac}$. Thus, the set $C_1$ must be contained in the right-collection in each of the partitions $P_1$, ..., $P_t$.

We ignore the partitions $P_{t+1}$, ..., $P_{t+h}$ and use the partitions $P_1$, ..., $P_t$ to construct a maximum partition matching of $n$ pairs. Note that $\{P_1, \ldots, P_t\}$ also forms a partition matching in the collection $\mathcal{S}$.

Consider the algorithm Partition Flipping given in Figure 3, where *flipping* a partition $(\mathcal{A}, \mathcal{B})$ gives the partition $(\mathcal{B}, \mathcal{A})$.

We must prove that the algorithm Partition Flipping correctly constructs a partition matching with $n$ pairs.

**Algorithm.** PARTITION FLIPPING

INPUT: a partition matching $\{P_1, \ldots, P_t\}$ that left-pairs all elements in $\cup_{i=2}^{k} C_i$, $t = \sum_{i=2}^{k} |C_i|$, and the set $C_1$ is contained in the right-collection of each partition $P_i$, $i = 1, \ldots, t$, $d = |C_1| \le t$.

OUTPUT: A maximum partition matching in $\mathcal{S}$ with $n$ pairs.

1. If not all elements in the set $C_1$ are right-paired by $P_1$, …, $P_t$, replace a proper number of right-paired elements in $\cup_{i=2}^{k} C_i$ by the right-unpaired elements in $C_1$ so that all elements in $C_1$ are right-paired by $P_1$, …, $P_t$;
2. Suppose that the partitions $P_1$, …, $P_{t-d}$ right-pair $t-d$ elements $b_1$, …, $b_{t-d}$ in $\cup_{i=2}^{k} C_i$, and that $P_{t-d+1}$, …, $P_t$ right pair the $d$ elements in $C_1$;
3. Suppose that $\overline{P}_1$, …, $\overline{P}_{t-d}$ are the $t-d$ partitions in $\{P_1, \ldots, P_t\}$ that left-pair the elements $b_1$, …, $b_{t-d}$;
4. Flip each of the $d$ partitions in $\{P_1, \ldots, P_t\} - \{\overline{P}_1, \ldots, \overline{P}_{t-d}\}$ to get $d$ partitions $P'_1$, …, $P'_d$ to left-pair the $d$ elements in $C_1$. The right-paired element of each $P'_i$ is the left-paired element before the flipping;
5. $\{P_1, \ldots, P_t, P'_1, \ldots, P'_d\}$ is a partition matching in $\mathcal{S}$ with $n$ pairs.

Figure 3: The algorithm PARTITION FLIPPING

Step 1 of the algorithm is always possible: since $C_1$ is contained in the right-collection of each partition $P_i$, $i = 1, \ldots, t$, and $t \ge d$, for each right-unpaired element $b$ in $C_1$, we can always pick a partition $P_i$ that right-pairs an element in $\cup_{i=2}^{k} C_i$, and let $P_i$ right-pair the element $b$. We keep doing this replacement until all $d$ elements in $C_1$ get right-paired. At this point, the number of partitions in $\{P_1, \ldots, P_t\}$ that right-pair elements in $\cup_{i=2}^{k} C_i$ is exactly $t - d$.

Step 3 is always possible since the partitions $P_1$, …, $P_t$ left-pair all elements in $\cup_{i=2}^{k} C_i$.

Now we verify that the sequence $\{P_1, \ldots, P_t, P'_1, \ldots, P'_d\}$ is a partition matching in $\mathcal{S}$.

No two partitions $P_i$ and $P_j$ can be identical since $\{P_1, \ldots, P_t\}$ is supposed to be a partition matching in $\mathcal{S}$. No two partitions $P'_i$ and $P'_j$ can be identical since they are obtained by flipping two different partitions in $\{P_1, \ldots, P_t\}$. No partition $P_i$ is identical to a partition $P'_j$ because $P_i$ has $C_1$ in its right-collection while $P'_j$ has $C_1$ in its left-collection. Therefore, the partitions $P_1$, …, $P_t$, $P'_1$, …, $P'_d$ are all distinct.

Each of the partitions $P_1$, …, $P_t$ left-pairs an element in $\cup_{i=2}^{k} C_i$, and each of the partitions $P'_1$, …, $P'_d$ left-pairs an element in $C_1$. Thus, all elements in the universal set $U$ get left-paired in $\{P_1, \ldots, P_t, P'_1, \ldots, P'_d\}$.

Finally, the partitions $P_1$, …, $P_t$ right-pair all elements in $C_1$ and the elements $b_1$, …, $b_{t-d}$ in $\cup_{i=2}^{k} C_i$. Now by our selection of the partitions, the partitions $P'_1$, …, $P'_d$ precisely right-pair all the elements in $\cup_{i=2}^{k} C_i - \{b_1, \ldots, b_{t-d}\}$. Thus, all elements in $U$ also get right-paired in $\{P_1, \ldots, P_t, P'_1, \ldots, P'_d\}$.

This concludes that the sequence $\{P_1, \ldots, P_t, P'_1, \ldots, P'_d\}$ is a maximum partition matching in the collection $\mathcal{S}$.

The running time of the algorithm PARTITION FLIPPING is obviously linear. □

Now we consider the case when $L_{reac}$ and $R_{reac}$ have no common partitions.

**Theorem 3.4** *If $L_{reac}$ and $R_{reac}$ have no common partitions, then the partition matching $M_{exp}$ constructed by the algorithm* GREEDY EXPANDING *is a maximum partition matching.*

PROOF. Let $W_{other}$ be the set of used partitions in $M_{exp}$ that belong to neither $L_{reac}$ nor $R_{reac}$.

Then

$$L_{free} \cup R_{free} \cup L_{reac} \cup R_{reac} \cup W_{other}$$

is the set of all partitions of the collection $\mathcal{S}$, and

$$L_{reac} \cup R_{reac} \cup W_{other}$$

is the set of partitions contained in the partition matching $M_{exp}$. Since all sets $L_{reac}$, $R_{reac}$, and $W_{other}$ are pairwise disjoint, the number of partitions in $M_{exp}$ is precisely

$$|L_{reac}| + |R_{reac}| + |W_{other}|$$

Now consider the set $W_L = L_{free} \cup L_{reac}$. Let $U_L$ be the set of elements that appears in the left-collection of a partition in $W_L$. We have

1. Every $P \in L_{reac}$ left-pairs an element in $U_L$;
2. Every element in $U_L$ is left-paired;
3. If an element $a$ in $U_L$ is left-paired by a partition $P$, then $P \in L_{reac}$. (Proof: let the element $a$ be in the set $C$, then the set $C$ is the left-paired set of the partition $P$. Since the element $a$ is in $U_L$, the set $C$ is also contained in the left-collection of a partition $P'$ that is in either $L_{free}$ or $L_{reac}$. The partition $P$ is left-reachable from $P'$, thus, must be in $L_{reac}$.)

Therefore, the partitions in $L_{reac}$ precisely left-pair the elements in $U_L$. This gives $|L_{reac}| = |U_L|$. Since there are only $|U_L|$ elements that appear in the left-collections in partitions in $L_{free} \cup L_{reac}$, we conclude that the partitions in $W_L = L_{free} \cup L_{reac}$ can be used to left-pair at most $|U_L| = |L_{reac}|$ elements in *any partition matching* in $\mathcal{S}$.

Similarly, the partitions in the set $W_R = R_{free} \cup R_{reac}$ can be used to right-pair at most $|R_{reac}|$ elements in *any partition matching* in $\mathcal{S}$.

Therefore, any partition matching in the collection $\mathcal{S}$ can include at most $|L_{reac}|$ partitions in the set $W_L$, at most $|R_{reac}|$ partitions in the set $W_R$, and at most all partitions in the set $W_{other}$. Consequently, a maximum partition matching in $\mathcal{S}$ consists of at most

$$|L_{reac}| + |R_{reac}| + |W_{other}|$$

partitions. Since the partition matching $M_{exp}$ constructed by the algorithm GREEDY EXPANDING contains just this many partitions, $M_{exp}$ is a maximum partition matching in the collection $\mathcal{S}$. □

We summarize all the discussions above into the following theorem.

**Theorem 3.5** *The maximum partition matching problem can be solved in time $O(n^2 \log n)$.*

PROOF. The problem is solved by the algorithm GENERAL MAXIMUM PARTITION MATCHING given in Figure 4. The correctness of the algorithm has been proved by Theorem 2.3, Theorem 3.3, and Theorem 3.4. To construct the sets $L_{reac}$ and $R_{reac}$, we use an algorithm similar to the one described for the algorithm GREEDY EXPANDING. That is, we first compute all the used partitions reachable from each set $C_i$, then use this information to examine each unused partition. This can be done in time $O(n^2 \log n)$ and we leave the detailed verification to interested readers. □

**Algorithm.** GENERAL MAXIMUM PARTITION MATCHING

INPUT: A collection $\mathcal{S} = \{C_1, \ldots, C_k\}$ of pairwise disjoint subsets of $U = \{1, 2, \ldots, n\}$.
OUTPUT: A maximum partition matching $M$ in $\mathcal{S}$

1. **case 1.** $2^k \geq 2n$
   call the algorithm PRE-MATCHING to construct a maximum partition matching;
2. **case 2.** $2^k < 2n$
   call the algorithm GREEDY EXPANDING to construct a partition matching $M_{exp}$; Compute $L_{reac}$ and $R_{reac}$;
   **if** $L_{reac}$ and $R_{reac}$ have a common partition
   **then** call the algorithm PARTITION FLIPPING to construct a maximum partition matching
   **else** $M_{exp}$ is a maximum partition matching.

Figure 4: The algorithm GENERAL MAXIMUM PARTITION MATCHING

# 4 Parallel routing in star networks

In this section, we show an application of maximum partition matching in parallel routing in star networks.

The star network [1] has received considerable attention recently by researchers as a graph model for interconnection network. It has been shown that it is an attractive alternative to the widely used hypercube model. Like the hypercube, the star network has rich structural and symmetric properties as well as fault tolerant characteristics. Moreover, it has a smaller diameter and degree while comparing with hypercube of comparable number of vertices.

Formally, the *$n$-dimensional star network* (or simply *the $n$-star network*) is an undirected graph consisting of $n!$ nodes labeled with the $n!$ permutations on symbols $1, 2, \ldots, n$. There is an edge between a node $u$ to a node $v$ if and only if there is a transposition $\pi[1, i]$, $2 \leq i \leq n$, such that $\pi[1, i](u) = v$. The $n$-star network is an $(n-1)$-connected and vertex symmetric Cayley graph [1].

Parallel routing, i.e., finding parallel node-disjoint paths between two nodes in a star network has been investigated recently [5, 10, 12, 13]. Since the $n$-star network is vertex symmetric, a parallel routing between two arbitrary nodes can be mapped to a parallel routing between a node and the identity node $\varepsilon$. Let $dist(u)$ define the distance from the node $u$ to $\varepsilon$. Day and Tripathi [5] and Jwo, Lakshmivarahan, and Dhall [10] have shown that for any node $u$, there are $n-1$ node-disjoint paths connecting $u$ and $\varepsilon$ such that no path has length larger than $dist(u) + 4$. An algorithm was described in [10] to construct the maximum number of node-disjoint paths of length $dist(u)$ between the nodes $u$ and $\varepsilon$. Unfortunately, the algorithm runs in exponential time in the worse case. Moreover, the algorithm seems to contain a serious bug. For example, the algorithm always constructs an even number of shortest paths from a node $u$ to $\varepsilon$, while a star network may have an odd number of node-disjoint shortest paths between a node $u$ and $\varepsilon$ (see our discussion in next section).

Let $u = a_1 a_2 \cdots a_n$ be a node in the $n$-star network (i.e., $u$ is a permutation on 1, ..., $n$). Suppose that the cycle structure of $u$ is $u = c_1 \cdots c_k e_1 \cdots e_m$, where $c_i$ are cycles of length at least 2, and $e_j$ are cycles of length 1. If we further let $l = \sum_{i=1}^{k} |c_i|$, then the shortest distance from $u$ to the identity node $\varepsilon$ is given by the following formula [1].

$$dist(u) = \begin{cases} l + k & \text{if } a_1 = 1 \\ l + k - 2 & \text{if } a_1 \neq 1 \end{cases}$$

From this formula, we can derive an upper bound on the number of node-disjoint shortest paths from $u$ to $\varepsilon$. We distinguish two cases. Recall that $\pi[1,i]$ is the transposition on permutations that exchanges the positions of the first symbol and the $i$th symbol, and that $\pi[a]$ is the transposition on permutations that exchanges the positions of the first symbol and the symbol $a$.

Suppose $a_1 = 1$ in the node $u = a_1 a_2 \cdots a_n$ with cycle structure $c_1 \cdots c_k e_1 \cdots e_m$. If $a \neq 1$ is in a single symbol cycle $e_i$, then by the discussion in Section 1 and the above formula, it is not hard to show that $dist(\pi[a](u)) = dist(u) + 1$, i.e., the edge from $u$ to $\pi[a](u)$ does not lead to a shortest path from $u$ to $\varepsilon$. Thus, in this case the total number of node-disjoint shortest paths from $u$ to $\varepsilon$ is bounded by $l = \sum_{i=1}^{k} |c_i|$. It is also easy to see that if $a$ is in a cycle $c_i$ of length at least 2, then $dist(\pi[a](u)) = dist(u) - 1$.

Suppose $a_1 \neq 1$. For this, we further assume that in the cycle structure $c_1 \cdots c_k e_1 \cdots e_m$ of $u$, we have $c_1 = (a_1^{(1)} a_2^{(1)} \cdots a_d^{(1)})$, where $a_1 = a_1^{(1)}$, and $a_d^{(1)} = 1$. Let $a$ be an element in the cycle $c_1$ such that $a \neq a_1$, $a \neq a_2^{(1)}$. Then by the discussion in Section 1 and the above formula, $dist(\pi[a](u)) = dist(u) + 1$, i.e., the edge from $u$ to $\pi[a](u)$ does not lead to a shortest path from $u$ to $\varepsilon$. Similarly, if $a$ is in a single symbol cycle $e_j$, then the edge from $u$ to $\pi[a](u)$ does not lead to a shortest path from $u$ to $\varepsilon$. Thus, in this case the total number of node-disjoint shortest paths from $u$ to $\varepsilon$ is bounded by $1 + \sum_{i=2}^{k} |c_i|$. It is also easy to see that if $a = a_2^{(1)}$ or if $a$ is in a cycle $c_i$ of length at least 2, $i > 1$, then $dist(\pi[a](u)) = dist(u) - 1$.

We summarize the above discussion into the following lemma.

**Lemma 4.1** *Let $u = a_1 a_2 \cdots a_n$ be a node in the $n$-star network with a cycle structure $u = c_1 \cdots c_k e_1 \cdots e_m$, where $c_i$ are cycles of length at least 2, and $e_j$ are cycles of length 1.*

*If $a_1 = 1$, then the number of node-disjoint paths of length $dist(u)$ from $u$ to $\varepsilon$ is bounded by $\sum_{i=1}^{k} |c_i|$;*

*If $a_1 \neq 1$, then the number of node-disjoint paths of length $dist(u)$ from $u$ to $\varepsilon$ is bounded by $1 + \sum_{i=2}^{k} |c_i|$ (we assume that the symbol 1 is contained in cycle $c_1$).*

The above discussion also tells us that on a shortest path from a node $v$ to $\varepsilon$, from each node $u = a_1 a_2 \cdots a_n$ on the path, with a cycle structure $u = c_1 \cdots c_k e_1 \cdots e_m$, to the next node on the path, we must perform one of the following two operations:

**(Rule 1)** If $a_1 = 1$, then merge the single symbol cycle $\{1\}$ into a cycle $c_i$ of length at least 2. This corresponds to applying a transposition $\pi[a]$ with $a \in c_i$;

**(Rule 2)** In case $a_1 \neq 1$, assume that $c_1 = (a_1^{(1)} a_2^{(1)} \cdots a_d^{(1)})$, where $a_1 = a_1^{(1)}$, and $a_d^{(1)} = 1$. Then either merge the cycle $c_1$ with a cycle $c_i$, $i > 1$ (this corresponds to applying the transposition $\pi[a]$, where $a \in c_i$), or delete the symbol $a_1$ from cycle $c_1$ (this corresponds to applying the transposition $\pi[a_2^{(1)}]$ and putting $a_1$ in a single symbol cycle).

**Remark.** Thus, a transposition $\pi[a]$ is never applied along the shortest path, if $a \neq 1$ is in a single symbol cycle. Consequently, once a symbol $a \neq 1$ is in a single symbol cycle, it will stay in the single symbol cycle forever along the shortest path.

Now we are ready to discuss parallel routing on the $n$-star network. Again suppose that $u = a_1 a_2 \cdots a_n$ is a node of the $n$-star network, and we want to construct a maximum number of node-disjoint shortest paths from the node $u$ to the identity node $\varepsilon$ in the $n$-star network.

If $a_1 = 1$, then by the above analysis, there are at most $l = \sum_{i=1}^{k} |c_i|$ node-disjoint shortest paths. In fact, it is not very hard to construct $l$ node-disjoint shortest paths from such a node $u$ to $\varepsilon$ [5, 10]. We will not discuss this case here. Interested readers are referred to [5, 10].

We will concentrate on the other case, which is more difficult.

**Problem A. Parallel Routing (Hard Case)**

Given a node $u = a_1 a_2 \cdots a_n$ in the $n$-star network, where $a_1 \neq 1$, with a cycle structure $c_1 \cdots c_k e_1 \cdots e_m$, where $c_1 = (a_1^{(1)} \cdots a_d^{(1)})$, $a_1^{(1)} = a_1$ and $a_d^{(1)} = 1$, construct a maximum number of node-disjoint shortest paths (of length $dist(u)$) in the $n$-star network from the node $u$ to the identity node $\varepsilon$.

We first derive another upper bound for the number of node-disjoint shortest paths from the node $u$ to $\varepsilon$ in terms of the maximum partition matching of the cycles $c_2, \ldots, c_k$, regarding $c_2, \ldots, c_k$ as sets of symbols.

**Lemma 4.2** *Let $u$ be the node described in Problem A. Then the number of node-disjoint shortest paths from $u$ to $\varepsilon$ cannot be larger than 2 plus the number of partitions of a maximum partition matching in $\mathcal{S} = \{c_2, \ldots, c_k\}$.*

Proof. Let $P_1, \ldots, P_s$ be $s$ node-disjoint shortest paths from $u$ to $\varepsilon$. For each path $P_i$, let $u_i$ be the first node on $P_i$ such that the symbol 1 is in the first position in the permutation $u_i$. The node $u_i$ is obtained by repeatedly applying **Rule 2**, starting from the node $u$. It is easy to prove, by induction, that for any node $v$ from $u$ to $u_i$ on the path $P_i$, the only possible cycle of length larger than 1 in $v$ that is not in $\{c_2, \ldots, c_k\}$ is the one that contains the symbol 1. In particular, the node $u_i$ must have a cycle structure of the form

$$u_i = c'_1 \cdots c'_{k_i} e'_1 \cdots e'_{m_i}$$

where $c'_i$ are cycles of length at least 2, $e'_j$ are cycles of length 1, and $\mathcal{B}_i = \{c'_1, \ldots, c'_{k_i}\}$ is a subcollection of $\mathcal{S} = \{c_1, \ldots, c_k\}$.

Assume that the first edge on $P_i$ is from $u$ to $\pi[b_i](u)$. By the discussion of Lemma 4.1, $b_i$ is either $a_2^{(1)}$ or one of the symbols in $\cup_{i=2}^{k} c_i$. Moreover, by **Rule 2**, once $b_i$ is contained in the cycle containing the symbol 1, it will stay in the cycle containing the symbol 1 until it is put into a single symbol cycle. In particular, the symbol $b_i$ is not in the set $\cup_{j=1}^{k_i} c'_j$.

Now consider the last edge on the path $P_i$, which must be from a node $w_i$ with cycle structure $(d_i 1)$ to the identity node $\varepsilon$. Since the symbol $d_i$ is in a cycle of length larger than 1 in $w_i$, by the **Remark**, $d_i$ is also in a cycle of length larger than 1 in the node $u_i$, that is $d_i \in \cup_{j=1}^{k_i} c'_j$. The only exception is $d_i = a_{d-1}^{(1)}$ (in this case $u_i = \varepsilon$).

Now we let $\mathcal{A}_i = \mathcal{S} - \mathcal{B}_i$. Then we can conclude that except at most two paths $P_1$ and $P_2$, each $P_i$, $i \geq 3$, of the other paths $P_3, \ldots, P_s$ must start with an edge $\{u, \pi[b_i](u)\}$, where $b_i$ is in $\mathcal{A}_i$, and end with an edge $\{w_i, \varepsilon\}$, where $w_i$ has a cycle structure $(d_i 1)$, where $d_i \in \mathcal{B}_i$ (the two exceptional paths $P_1$ and $P_2$ may start with the edge $\{u, \pi[a_2^{(1)}](u)\}$, or end with the edge $\{w, \varepsilon\}$, where $w$ has a cycle structure $(a_{d-1}^{(1)} 1)$).

Now since the $s$ paths $P_1, \ldots, P_s$ are node-disjoint, the symbols $b_3, \ldots, b_s$ are all pairwise distinct, and the symbols $d_3, \ldots, d_s$ are also pairwise distinct. Moreover, since all nodes $u_3, \ldots, u_s$ are pairwise distinct, the collections $\mathcal{B}_3, \ldots, \mathcal{B}_s$ of cycles are also pairwise distinct. Consequently, the partitions $(\mathcal{A}_3, \mathcal{B}_3), \ldots, (\mathcal{A}_s, \mathcal{B}_s)$ form a partition matching of the collection $\mathcal{S} = \{c_2, \ldots, c_k\}$.

This concludes that $s$ cannot be larger than 2 plus the number of partitions in a maximum partition matching of the collection $\mathcal{S} = \{c_2, \ldots, c_k\}$, thus proves the lemma. $\square$

**Algorithm.** SINGLE ROUTING

INPUT: A partition $(\mathcal{A}, \mathcal{B})$ of $\mathcal{S} = \{c_2, \ldots, c_k\}$, and two symbols $b$ in $\mathcal{A}$ and $d$ in $\mathcal{B}$. $b$ can be $a_2^{(1)}$ with $\mathcal{A} = \phi$, and $d$ can be $a_{d-1}^{(1)}$ with $\mathcal{B} = \phi$.

OUTPUT: A shortest path from $u$ to $\varepsilon$ starting with the edge $\{u, \pi[b](u)\}$ and ending with the edge $\{w, \varepsilon\}$, where $w$ has a cycle structure $(d1)$.

1. **if** $b \neq a_2^{(1)}$
   apply $\pi[b]$ to $u$ to merge the cycle in $\mathcal{A}$ that contains $b$ into the cycle $c_1$; then merge in an arbitrary order the rest of the cycles in $\mathcal{A}$ into the cycle containing the symbol 1;
2. repeatedly delete symbols in the cycle containing the symbol 1 until the symbol 1 is put into a single symbol cycle;
3. **if** $d \neq a_{d-1}^{(1)}$
   suppose that the cycle $C$ containing $d$ in $\mathcal{B}$ in $(d' \cdots d)$. Apply $\pi[d']$ to merge $(d' \cdots d)$ with the single symbol cycle $\{1\}$; then merge in an arbitrary order the rest of the cycles in $\mathcal{B}$ into the cycle containing the symbol 1;
4. repeatedly delete symbols in the cycle containing the symbol 1 until reach the node $\varepsilon$;

Figure 5: The algorithm OPTIMAL PARALLEL ROUTING

Now we show how we construct a maximum number of node-disjoint shortest paths from the node $u$ to the identity node $\varepsilon$ in the $n$-star network. We first show how to route a single shortest path from $u$ to $\varepsilon$, given a partition $(\mathcal{A}, \mathcal{B})$ of the collection $\mathcal{S} = \{c_2, \ldots, c_k\}$, and a pair of symbols $b$ and $d$, where $b$ is in $\mathcal{A}$ and $d$ is in $\mathcal{B}$. We also allow $b$ to be $a_2^{(1)}$ — in this case $d$ must be in $\cup_{i=2}^k c_i$ and $\mathcal{A} = \phi$, and $\mathcal{B} = \mathcal{S}$. Similarly, we allow $d$ to be $a_{d-1}^{(1)}$ — in this case $b$ must be in $\cup_{i=2}^k c_i$ and $\mathcal{B} = \phi$, and $\mathcal{A} = \mathcal{S}$. Consider the algorithm SINGLE ROUTING given in Figure 5. Since the algorithm SINGLE ROUTING starts with the node $u$ and applies only transpositions described in **Rule 1** and **Rule 2**, we conclude that the algorithm SINGLE ROUTING constructs a shortest path from the node $u$ to the node $\varepsilon$.

Now we are ready for describing the final algorithm. Consider the algorithm OPTIMAL PARALLEL ROUTING given in Figure 6.

**Theorem 4.3** *Algorithm* OPTIMAL PARALLEL ROUTING *constructs a maximum number of node-disjoint shortest paths from the node $u$ to the identity node $\varepsilon$ in time $O(n^2 \log n)$.*

PROOF. From Lemma 4.1 and Lemma 4.2, we know that the number of shortest paths constructed by the algorithm OPTIMAL PARALLEL ROUTING matches the maximum number of node-disjoint shortest paths from $u$ to $\varepsilon$. What remains is to show that all these paths are node-disjoint.

Suppose that the algorithm OPTIMAL PARALLEL ROUTING constructs $h$ shortest paths $P_1$, $P_2$, …, $P_h$ from node $u$ to node $\varepsilon$, here $h = s+1$ or $h = s+2$ depending on whether $s = \sum_{i=2}^k |c_i|$, and suppose that the path $P_i$ is constructed by calling the algorithm SINGLE ROUTING on partition $(\mathcal{A}_i, \mathcal{B}_i)$ of $\mathcal{S}$ and symbol pair $(b_i, d_i)$ for all $i = 1, \ldots, h$. Note that if $\mathcal{A}_i = \phi$ then we have $b_i = a_2^{(1)}$, and if $\mathcal{B}_i = \phi$ then we have $d_i = a_{d-1}^{(1)}$. Now fix an $i$ and consider the path $P_i$, which is constructed from the partition $(\mathcal{A}_i, \mathcal{B}_i)$ and the symbol pair $(b_i, d_i)$. Let $\mathcal{A}_i = \{c_2^{(i)}, \ldots, c_l^{(i)}\}$ and $\mathcal{B}_i = \{c_{l+1}^{(i)}, \ldots, c_k^{(i)}\}$, where if $\mathcal{A}_i \neq \phi$ then the cycle $c_2^{(i)}$ is of form $c_2^{(i)} = (b_i * * b_i')$ and if $\mathcal{B}_i \neq \phi$ then the cycle $c_{l+1}^{(i)}$ is of form $c_{l+1}^{(i)} = (* *\ d_i)$, here we use "$* *$" to represent irrelevant part of a cycle structure. Finally, recall that the cycle $c_1$ has the form $c_1 = (a_1^{(1)} * * a_{d-1}^{(1)} 1)$.

**Algorithm.** OPTIMAL PARALLEL ROUTING

INPUT: The node $u$ in the $n$-star network, as described in **Problem A**.

OUTPUT: A maximum number of node-disjoint shortest paths from $u$ to $\varepsilon$.

1. Construct a maximum partition matching $M[(b_1, d_1), \ldots, (b_s, d_s)]$ in the collection $\mathcal{S} = \{c_2, \ldots, c_k\}$ with the partitions $(\mathcal{A}_1, \mathcal{B}_1)$, ..., $(\mathcal{A}_s, \mathcal{B}_s)$ of $\mathcal{S}$;
2. **if** $s = \sum_{i=2}^{k} |c_i|$ **then** construct $s+1$ node-disjoint shortest paths as follows.
   2.1. Call the algorithm SINGLE ROUTING with the partition $(\phi, \mathcal{S})$ of $\mathcal{S}$ and the symbol pair $(a_2^{(1)}, d_1)$;
   2.2. Call the algorithm SINGLE ROUTING with the partition $(\mathcal{S}, \phi)$ of $\mathcal{S}$ and the symbol pair $(b_1, a_{d-1}^{(1)})$;
   2.3. For $i = 2$ to $s$, call the algorithm SINGLE ROUTING with the partition $(\mathcal{A}_i, \mathcal{B}_i)$ of $\mathcal{S}$ and the symbol pair $(b_i, d_i)$;
3. **if** $s < \sum_{i=2}^{k} |c_i|$ **then** construct $s+2$ node-disjoint shortest paths as follows.
   3.1. Let $b_0$ be a left-unpaired symbol and $d_0$ be a right-unpaired symbol in the partition matching constructed in Step 1.
   3.2. Call the algorithm SINGLE ROUTING with the partition $(\phi, \mathcal{S})$ of $\mathcal{S}$ and the symbol pair $(a_2^{(1)}, d_0)$;
   3.3. Call the algorithm SINGLE ROUTING with the partition $(\mathcal{S}, \phi)$ of $\mathcal{S}$ and the symbol pair $(b_0, a_{d-1}^{(1)})$;
   3.4. For $i = 1$ to $s$, call the algorithm SINGLE ROUTING with the partition $(\mathcal{A}_i, \mathcal{B}_i)$ of $\mathcal{S}$ and the symbol pair $(b_i, d_i)$;

Figure 6: The algorithm OPTIMAL PARALLEL ROUTING

The interior nodes of the path $P_i$ can be split into three segments $I_1^{(i)}$, $I_2^{(i)}$, and $I_3^{(i)}$. The first segment $I_1^{(i)}$ corresponds to nodes constructed in step 1 of the algorithm SINGLE ROUTING that first merges cycle $c_2^{(i)}$ into cycle $c_1$, obtaining a cycle of form $(b_i ** b_i' a_1^{(1)} ** a_{d-1}^{(1)} 1)$, then merges cycles $c_3^{(i)}$, ..., $c_l^{(i)}$ into the cycle containing symbol 1. Therefore, all nodes in this segment are of form

$$(** \ b_i ** b_i' a_1^{(1)} ** a_{d-1}^{(1)} 1) **$$

The second segment $I_2^{(i)}$ corresponds to the nodes constructed by step 2 of the algorithm SINGLE ROUTING that deletes symbols in the cycle containing the symbol 1. All nodes in this segment are of form

$$(** \ 1) c_{l+1}^{(i)} \cdots c_k^{(i)} **$$

The third segment $I_3^{(i)}$ corresponds to the nodes constructed by steps 3 and 4 of the algorithm SINGLE ROUTING, which first merges the cycle $c_{l+1}^{(i)}$ into the single symbol cycle $\{1\}$, obtaining a cycle of form $(** \ d_i 1)$, then merges the cycles $c_{l+2}^{(i)}$, ..., $c_k^{(i)}$, and then deletes symbols in the cycle containing the symbol 1. Therefore, all nodes in this segment should have the form

$$(** \ d_i 1) **$$

In case $\mathcal{A}_i = \phi$, we have $b_i = a_2^{(1)}$ and the segment $I_1^{(i)}$ is empty, and in case $\mathcal{B}_i = \phi$, we have $d_i = a_{d-1}^{(1)}$ and the segment $I_3^{(i)}$ is empty.

We now show that any two shortest paths $P_i$ and $P_j$, $i \neq j$, constructed by the algorithm OPTIMAL PARALLEL ROUTING are node-disjoint. Let $v$ be a node on the path $P_i$.

Suppose that $v = (**\ b_i ** b'_i a_1^{(1)} ** a_{d-1}^{(1)} 1) **$ is a node on the first segment $I_1^{(i)}$ of the path $P_i$. The node cannot be on the first segment $I_1^{(j)}$ of the path $P_j$ since all nodes on $I_1^{(j)}$ are of form $(**\ b_j ** b'_j a_1^{(1)} ** a_{d-1}^{(1)} 1) **$ and $b_i \neq b_j$ (thus $b'_i \neq b'_j$). Moreover, the node $v$ cannot be on the second or the third segment of $P_j$ since the cycle structure of a node on the second or the third segment of $P_j$ has more single symbol cycles (note that each execution of step 2 of the algorithm Single Routing creates a new single symbol cycle in the cycle structure).

If $v = (**\ 1) c_{l+1}^{(i)} \cdots c_k^{(i)} **$ is on the second segment $I_2^{(i)}$ of the path $P_i$, then $v$ cannot be on the second segment $I_2^{(j)}$ of $P_j$ since each node on $I_2^{(j)}$ is of form $(**\ 1) c_{f+1}^{(j)} \cdots c_k^{(j)} **$ and

$$\mathcal{B}_i = \{c_{l+1}^{(i)}, \ldots, c_k^{(i)}\} \neq \mathcal{B}_j = \{c_{f+1}^{(j)}, \ldots c_k^{(j)}\}$$

The node $v$ can neither be on the third segment $I_3^{(j)}$ of $P_j$ since each node on the segment $I_3^{(j)}$ is of form $(**\ d_j 1) **$, where $d_j \in \cup_{i=2}^{k} c_i$ while the cycle containing the symbol 1 in the node $v$ is either a single symbol cycle or of form $(**\ a_{d-1}^{(1)} 1)$ where $a_{d-1}^{(1)}$ is in $c_1$.

Finally, if $v = (**\ d_i 1) **$ is on the third segment of the path $P_i$, then $v$ cannot be on the third segment of $P_j$ because $d_i \neq d_j$.

By symmetry, the above analysis shows that the two shortest paths $P_i$ and $P_j$ constructed by the algorithm Optimal Parallel Routing must be node-disjoint.

The running time of the algorithm Optimal Parallel Routing is dominated by step 1 of the algorithm, which takes time $O(n^2 \log n)$ according to Theorem 3.5. Thus, the algorithm Optimal Parallel Routing runs in time $O(n^2 \log n)$. □

# 5 Conclusion and remarks

We have presented an efficient algorithm for the maximum partition matching problem. By a nontrivial reduction, we have shown that finding the maximum number of node-disjoint shortest paths between two given nodes in the star networks can be reduced to the maximum partition matching problem. This gives the first correct and efficient algorithm for constructing the maximum number of node-disjoint shortest paths between two given nodes in the star networks.

The problem of constructing the maximum number of node-disjoint shortest paths between two given nodes in the star networks was previously investigated in [10], which presents an algorithm (Algorithm 3.2 in [10]) that claims to find the maximum number of node-disjoint shortest paths between two given nodes in the star networks. For each node $u = c_1 \cdots c_k e_1 \cdots e_m$, the algorithm in [10] runs in time exponential in $k$, thus in time exponential in $n$ in the worst case (when $k = \Theta(n)$). More seriously, the algorithm seems based on an incorrect observation that claims that when $k > 1$ the maximum number of node-disjoint shortest paths from $u$ to $\varepsilon$ is always even (see the paragraph following Lemma 3.11 in [10]). Therefore, in case $k > 1$, the algorithm in [10] always produces an even number of node-disjoint shortest paths from the node $u$ to $\varepsilon$. The incorrectness of this can be shown as follows. Consider a node $u = c_1 \cdots c_k e_1 \cdots e_m$ in the $n$-star network, where $k > 1$. It is easy to make the node $u$ satisfy the following conditions: (1) $|c_2| \geq |c_i|$ for $i = 3, \ldots, k$ and $|c_2| \leq \sum_{i=3}^{k} |c_i|$; (2) the number $s = \sum_{i=2}^{k} |c_i|$ is even; and (3) $k - 1$ is at least as large as $\log(2s)$. Now according to the discussion in Section 2, we can construct a partition matching of order $s$ for the collection $\{c_2, \ldots, c_k\}$. Moreover, by step 2 of the algorithm Optimal Parallel Routing, the maximum number of node-disjoint shortest paths from $u$ to $\varepsilon$ is $s+1$, which is an odd number. A concrete example of this kind of nodes in the $n$-star network can be found in [3].

Finally, we describe how the algorithm Optimal Parallel Routing can be used to construct between two nodes in the $n$-star network $n-1$ node-disjoint paths of minimum bulk length. Let $G$ be a $h$-connected graph. By Menger's theorem [11], for any pair of nodes $u$ and $v$ in $G$, there are $h$ node-disjoint paths connecting $u$ and $v$. The *bulk length* of $h$ node-disjoint paths connecting $u$ and $v$ in $G$ is defined to be the length of the longest path among the $h$ paths. The *bulk distance* between the two nodes $u$ and $v$ is defined to be the minimum bulk length over all groups of $h$ node-disjoint paths connecting $u$ and $v$. Clearly, the bulk distance between two nodes $u$ and $v$ is at least as large as the distance between $u$ and $v$, which is defined to be the length of the shortest path connecting $u$ and $v$. In general, the problem of computing the bulk distance between two given nodes in a graph is NP-hard [7, 8].

The bulk distance problem on the star networks has been studied recently [4, 5, 10, 12, 13]. Since the $n$-star network is $(n-1)$-connected and vertex symmetric [1], the bulk distance problem on two arbitrary nodes in the $n$-star network can be converted to the problem of finding $n-1$ node-disjoint paths of minimum bulk length from the node $u$ to the identity node $\varepsilon$. Let $dist(u)$ and $Bdist(u)$ be the distance and bulk distance, respectively, between the node $u$ and the identity node $\varepsilon$ in the $n$-star network. It has been shown that $Bdist(u)$ is equal to $dist(u)$ plus an even number [5]. Day and Tripathi [5] have developed an algorithm that constructs $n-1$ node-disjoint paths between $u$ and $\varepsilon$ with bulk length $dist(u)+4$. Thus, we always have $Bdist(u) \leq dist(u)+4$. The authors of the present paper [4] have established a sufficient and necessary condition for the node $u$ to have bulk distance $dist(u)+4$, and developed an $O(n^2 \log n)$ time algorithm to construct $n-1$ node-disjoint paths between $u$ and $\varepsilon$ of bulk length $dist(u)+2$ when the bulk distance of $u$ is less than $dist(u)+4$.

Combining these results and the results in the present paper, we obtain an $O(n^2 \log n)$ time algorithm that constructs $n-1$ node-disjoint paths of bulk length $Bdist(u)$ between any node $u$ and the identity node $\varepsilon$ in the $n$-star network, as follows. We first check whether the bulk distance $Bdist(u)$ is $dist(u)+4$, using the formula given in [4]. If $Bdist(u) = dist(u)+4$, then we apply the algorithm given in [5] to construct $n-1$ node-disjoint paths of bulk length $dist(u)+4 = Bdist(u)$ from $u$ to $\varepsilon$. If the bulk distance of $u$ is less than $dist(u)+4$, then we apply the algorithm Optimal Parallel Routing in the present paper to find the maximum number of node-disjoint shortest paths. If the algorithm returns $n-1$ such shortest paths, then these paths are the $n-1$ node-disjoint paths of bulk distance $dist(u)$ between $u$ and $\varepsilon$. If the algorithm returns less than $n-1$ such shortest paths, then we know that the bulk distance of $u$ is $dist(u)+2$ so the algorithm developed in [4] can be applied to construct $n-1$ node-disjoint paths of bulk length $dist(u)+2$. This completes the description of the algorithm that always constructs $n-1$ node-disjoint paths of bulk length $Bdist(u)$ between a node $u$ and the identity node $\varepsilon$ in the $n$-star network. The running time of the algorithm is clearly $O(n^2 \log n)$.

We would like to make a few remarks on the complexity of the above algorithm. The bulk distance problem on general graphs is NP-hard [8]. Thus, it is very unlikely that the bulk distance problem can be solved in time polynomial in the size of the input graph. On the other hand, our algorithm solves the bulk distance problem in time $O(n^2 \log n)$ on the $n$-star network. Note that the $n$-star network has $n!$ nodes. Therefore, the running time of our algorithm is actually a polynomial of the logarithm of the size of the input star network. Moreover, our algorithm is almost optimal (differs at most by a $\log n$ factor) since the following lower bound can be easily observed — the distance $dist(u)$ from $u$ to $\varepsilon$ can be as large as $\Theta(n)$. Thus, constructing $n-1$ node-disjoint paths from $u$ to $\varepsilon$ takes time at least $\Theta(n^2)$ in the worst case.

**Acknowledgement.** The authors would like thank Professor Eva Tardos for her thorough and valuable comments and useful discussion on a version of this paper. The authors are grateful to Professor Jonathan Gross for informing them the recent discovery by Galil and Yu on graph bulk distance [7]. The authors also thank Professors Laxmi Bhuyan, Don Friesen, and Mi Lu and Dr. Xiangdong Yu for their comments and discussion.

## References

[1] S. B. Akers and B. Krishnamurthy, A group-theoretic model for symmetric interconnection networks, *IEEE Trans. on Computers 38*, (1989), pp. 555-565.

[2] G. Birkhoff and S. MacLane, *A Survey of Modern Algebra*, The Macmillan Company, New York, 1965.

[3] C. C. Chen, *Combinatorial and algebraic methods in star and de Bruijn networks*, Ph.D. dissertation, Dept. of Computer Science, Texas A&M University, 1995.

[4] C. C. Chen and J. Chen, Vertex-disjoint routings in star graphs, *Proceedings of IEEE 1st International Conference on Algorithms and Architectures for Parallel Processing*, (1995), pp. 460-464.

[5] K. Day and A. Tripathi, A comparative study of topological properties of hypercubes and star graphs, *IEEE Trans. on Parallel and Distributed Systems 5*, (1994), pp. 31-38.

[6] M. L. Furst, J. L. Gross, and L. A. McGeoch, Finding a maximum-genus graph imbedding, *Journal of ACM 35*, (1988), pp. 523-534.

[7] Z. Galil and X. Yu, Short length versions of Menger's theorem, *Proceedings of 27th Annual ACM Symp. on Theory of Computing*, (1995), pp. 499-508.

[8] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, CA, 1979.

[9] D. Gusfield and R. W. Irving, *The Stable Marriage Problem: Structure and Algorithms*, MIT Press Ser. Found. Comput., MIT Press, Cambridge, MA, 1989.

[10] J. Jwo, S. Lakshmivarahan, and S. K. Dhall, Characterization of node disjoint (parallel) path in star graphs, *Proc. 5th International Parallel Processing Symposium*, (1991), pp. 404-409.

[11] K. Menger, Zur allgemeinen kurventheorie, *Fund. Math. 10*, (1927), pp. 96-115.

[12] J. Misic and Z. Jovanovic, Routing function and deadlock avoidance in a star graph interconnection network, *Journal of Parallel and Distributed Computing 22*, (1994), pp. 216-228.

[13] S. Sur and P. K. Srimani, Topological properties of star graphs, *Computers Math. Applic. 25*, (1993), pp. 87-98.

[14] R. E. Tarjan, *Data Structures and Network Algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1983.