One-to-many node-disjoint paths in (n, k)-star graphs

Yonghong Xiang and Iain A. Stewart
Department of Computer Science, Durham University,
Science Labs, South Road, Durham DH1 3LE, U.K.
e-mail: {yonghong.xiang,i.a.stewart}@durham.ac.uk

Abstract

We present an algorithm which given a source node and a set of $n-1$ target nodes in the (n, k)-star graph $S_{n,k}$, where all nodes are distinct, builds a collection of $n-1$ node-disjoint paths, one from each target node to the source. The collection of paths output from the algorithm is such that each path has length at most $6k-7$, and the algorithm has time complexity $O(k^2n^2)$.
keywords: interconnection networks; (n, k)-star graphs; one-to-many node-disjoint paths.

1 Introduction

Chiang and Chen [6] introduced (n, k)-star graphs, $S_{n,k}$, where $n > k \geq 1$, as alternatives to n-star graphs, for which the 'jump' from $n!$ nodes in an n-star graph to $(n+1)!$ nodes in an $(n+1)$-star graph is deemed excessive (n-star graphs were devised in [1] as rivals to hypercubes in that they can incorporate comparable numbers of nodes yet have smaller diameters and degrees). The two parameters, n and k, of (n, k)-star graphs allow much more precision with regard to incorporating more nodes, and allow fine tuning with regard to a degree/diameter trade-off.

Since their introduction in [6], (n, k)-star graphs have been well studied and their basic topological and algorithmic properties are well understood.

For example: they form a hierarchical family of graphs, each of which is node-symmetric [6]; they can be recursively decomposed in a number of ways [6]; they have a simple shortest-path routing algorithm [6]; the node-connectivity of $S_{n,k}$ is $n-1$ [7]; there is an exact formula for their diameters, and their fault-diameters are at most their fault-free-diameters plus 3 [7]; a cycle of length $\frac{n!}{(n-k)!} - f$ can be found in $S_{n,k}$ when the number of faulty nodes f is at most $n-3$ and $n-k \geq 2$ [4]; their Hamiltonicity and Hamiltonian-connectedness properties are well understood in the presence of a limited number of faulty nodes and edges [12]; and $S_{n,k}$ is super spanning connected if $n \geq 3$ and $n-k \geq 2$ [13].

As regards the node-connectivity of $S_{n,k}$, it was shown in [7] that there are $n-1$ node-disjoint paths joining any two distinct nodes of $S_{n,k}$ (with an implicit algorithm for construction) and that each of these paths has length at most the diameter, $\Delta(S_{n,k})$, of $S_{n,k}$ plus 3; that is, the *wide-diameter* of $S_{n,k}$ is at most $\Delta(S_{n,k}) + 3$. It was also shown in [7] that the diameter $\Delta(S_{n,k})$ is $2k-1$, if $1 \leq k \leq \lfloor \frac{n}{2} \rfloor$, and $k + \lfloor \frac{n-1}{2} \rfloor$, if $\lfloor \frac{n}{2} \rfloor + 1 \leq k < n$. The wide-diameter analysis was improved in [16, 17] to yield that: when $2 \leq k < \lfloor \frac{n}{2} \rfloor$ or $k = n-1$, the wide-diameter of $S_{n,k}$ is exactly $\Delta(S_{n,k}) + 2$; when $\lfloor \frac{n}{2} \rfloor + 1 \leq k < n-2$, the wide-diameter of $S_{n,k}$ is either $\Delta(S_{n,k}) + 1$ or $\Delta(S_{n,k}) + 2$; and the wide-diameter of $S_{n,1}$ is $\Delta(S_{n,1}) + 1$. Thus, the one-to-one node-disjoint paths problem for $S_{n,k}$ has been pretty much resolved (note that as $S_{n,k}$ is regular of degree $n-1$, there is no scope for incorporating more node-disjoint paths between two nodes). In this paper, we are concerned with the one-to-many node-disjoint paths problem for $S_{n,k}$; that is, we are given $S_{n,k}$, $n-1$ distinct target nodes in the set T, and a source node I, different from any target node, and we wish to find $n-1$ node-disjoint paths, one from the source node I to each target node of T.

The one-to-many node-disjoint paths problem is a fundamental problem in the design and implementation of parallel and distributed computing systems and it has been extensively studied for a variety of (families of) interconnection networks. Whilst Menger's Theorem [3] implies that, given a source node and $n-1$ distinct target nodes (different from the source) in a graph of node-connectivity $n-1$, there exist $n-1$ node-disjoint paths to each of the target nodes from the source, it is sometimes difficult to identify and actually construct the paths, especially if the paths are to be as short as possible. Suppose that G is a c-connected graph. We say that G has *c-Rabin number* r if r is the minimum number for which any $c+1$ distinct

nodes $s, t_1, t_2, \ldots, t_c$ are such that there are c node-disjoint paths from s to $t_1, t_2, \ldots, t_c$, each of length at most r. It was shown in [11] that given a c-connected graph G, it is NP-hard to compute the c-Rabin number of G. However, in many interconnection networks, which almost always have 'uniformity' properties such as recursive decomposability, node-symmetry, and degree regularity, the situation is much more acceptable (see, for example, [1, 2, 5, 8, 9, 10, 11, 14, 15, 18, 19]). We only highlight here two such studies of the one-to-many node-disjoint paths problem: in hypercubes and in n-star graphs (recall, n-star graphs were introduced as improvements to hypercubes, and (n, k)-star graphs as improvements to n-star graphs). In [19], Rabin studied the one-to-many node-disjoint paths problem in hypercubes where he showed that given a source node and n target nodes in an n-dimensional hypercube, there exist node-disjoint paths from the source to each of the target nodes such that each path has length at most 1 plus the diameter of the n-dimensional hypercube (this result was slightly improved in [8]). In [10], Gu and Peng showed that given a source and $n-1$ target nodes in an n-star graph, there is an algorithm of time complexity $O(n^2)$ that builds $n-1$ paths from the source to each of the target nodes such that the length of each path is at most the diameter of the n-star graph (that is, $\lfloor \frac{3(n-1)}{2} \rfloor$) plus 2.

In this paper, we prove the following theorem.

Theorem 1 *When T is a set of $n-1$ distinct nodes in $S_{n,k}$, where $n > k \geq 1$, and when I is a node not in T, there is an algorithm that finds $n-1$ node-disjoint paths in $S_{n,k}$ from the source I to each of the nodes in T. Furthermore, all paths found by this algorithm have length at most $6k-7$ and the time complexity of the algorithm is $O(k^2n^2)$.*

We also show that this result is optimal for the case when $k = 2$.

We present the basic definitions in Section 2 before dealing with the case when $k = 2$ in Section 3. In Section 4, we present the algorithm alluded to in Theorem 1 and its proof of correctness, and in Section 5 we consider the lengths of the paths constructed by our algorithm and also its time complexity. Our conclusions are presented in Section 6, where we comment on our results in comparison with analogous ones for n-star graphs and hypercubes. We remark that weaker results than those presented in this paper were claimed in [21]. However: the proof of the main theorem in [21] is incomplete in that the cases considered do not exhaust those that might arise,

with the consequence that more sophistication in the construction and the analysis is called for; no analysis of the lengths of the paths constructed in [21] was given; and the base case of the induction (see the proof of the main theorem in [21]) was merely stated as being self-evident when, as we shall see in Section 3, this is not the case.

2 Basic definitions and lemmas

It is worthwhile beginning with an n-star graph in order that we might understand why (n,k)-star graphs emerged. The *n-star graph* S_n has node set $V(S_n) = \{(u_1, u_2, \ldots, u_n) : \text{ each } u_i \in \{1, 2, \ldots, n\} \text{ and } u_i \neq u_j, \text{ for } i \neq j\}$, and there is an edge $((u_i, u_2, \ldots, u_n), (v_1, v_2, \ldots, v_n))$ if, and only if, $u_1 = v_i$ and $u_i = v_1$, for some $i \in \{2, 3, \ldots, n\}$, with $u_l = v_l$, for all $l \in \{2, 3, \ldots, n\} \setminus \{i\}$. In order to avoid the significant jump from $n!$ nodes in an n-star graph to $(n+1)!$ nodes in an $(n+1)$-star graph, (n,k)-star graphs were devised, as 'generalized' n-star graphs. Let $n > k \geq 1$. The *(n,k)-star graph*, denoted $S_{n,k}$, has node set $V(S_{n,k}) = \{(u_1, u_2, \ldots, u_k) : \text{ each } u_i \in \{1, 2, \ldots, n\} \text{ and } u_i \neq u_j, \text{ for } i \neq j\}$, and there is an edge $((u_1, u_2, \ldots, u_k), (v_1, v_2, \ldots, v_k))$ if, and only if, either:

- $u_i = v_i$, for $2 \leq i \leq k$, and $u_1 \neq v_1$ (a *1-edge*);
- $u_1 = v_i$ and $u_i = v_1$, for some $i \in \{2, 3, \ldots, k\}$, with $u_l = v_l$, for all $l \in \{2, 3, \ldots, k\} \setminus \{i\}$ (an *i-edge*).

In consequence, $S_{n,k}$ has $\frac{n!}{(n-k)!}$ nodes and $\frac{n-1}{2} \times \frac{n!}{(n-k)!}$ edges. Note that $S_{n,n-1}$ is isomorphic to the n-star S_n, and that $S_{n,1}$ is a clique on n nodes.

An important property of $S_{n,k}$, of which we make crucial use, is that it can be partitioned into n node-disjoint copies of $S_{n-1,k-1}$ over one of $k-1$ dimensions. In more detail, let $i \in \{2, 3, \ldots, k\}$ and partition the nodes of $S_{n,k}$ by fixing the ith component of each node. Thus, define $S^i_{n,k}(j) = \{(u_1, u_2, \ldots, u_k) \in V(S_{n,k}) : u_i = j\}$, for each $j \in \{1, 2, \ldots, n\}$. It is trivial to see that the set of nodes $S^i_{n,k}(j)$, for $j \in \{1, 2, \ldots, n\}$, induces a copy of $S_{n-1,k-1}$. Note that there are $k-1$ dimensions over which we can so partition $S_{n,k}$.

We adopt the following notation throughout this paper. Let $I = (u_1, u_2, \ldots, u_k)$ be an arbitrary node of $S_{n,k}$. Note that there are $k-1$ neighbours of I that are joined to I via an i-edge, and $n-k$ neighbours of I that are joined

to I by a 1-edge; each neighbour is characterized by its first component. We denote the neighbour of I whose first component is j by I^j. Paths are written explicitly as sequences of nodes, such as $(x_1, x_2, \ldots, x_m)$ (it will be clear from the context whether we are referring to a path of nodes or the components of a node), and we sometimes denote a specific path from node s to node t by $\rho(s,t)$. We write $x \in S_{n,k} \setminus X$, where X is a set of nodes of $S_{n,k}$, to denote that x is a node of $S_{n,k}$ different from any node in X.

We now give a lemma that shall be useful later.

Lemma 2 *Let $(x_1, x_2, \ldots, x_k)$ be some node of $S^k_{n,k}(x_k)$, where $x_1 \neq y \neq x_k$. There are $n-2$ distinct nodes of $S^k_{n,k}(x_k)$ each of whose first component is y and each of which is reachable by a path in $S^k_{n,k}(x_k)$ of length at most 3 from $(x_1, x_2, \ldots, x_k)$.*

Proof Suppose that $x_i = y$, where $1 \neq i \neq k$. The nodes are as follows. For each z where $z \notin \{x_1, x_2, \ldots, x_k\}$, define the path

$$(x_1, x_2, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_k), (z, x_2, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_k),$$
$$(y, x_2, \ldots, x_{i-1}, z, x_{i+1}, \ldots, x_k).$$

When $2 \leq j \leq k-1$ and $j \neq i$, define the path

$$(x_1, x_2, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_k),$$
$$(x_j, x_2, \ldots, x_{j-1}, x_1, x_{j+1}, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_k),$$
$$(y, x_2, \ldots, x_{j-1}, x_1, x_{j+1}, \ldots, x_{i-1}, x_j, x_{i+1}, \ldots, x_k),$$

and define the path

$$(x_1, x_2, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_k), (y, x_2, \ldots, x_{i-1}, x_1, x_{i+1}, \ldots, x_k).$$

Suppose that $y \notin \{x_1, x_2, \ldots, x_k\}$. The nodes are as follows. For each z where $z \notin \{x_1, x_2, \ldots, x_k, y\}$, define the path

$$(x_1, x_2, \ldots, x_k), (z, x_2, \ldots, x_k), (x_2, z, x_3, \ldots, x_k), (y, z, x_3, \ldots, x_k).$$

When $2 \leq j \leq k-1$, define the path

$$(x_1, x_2, \ldots, x_k), (x_j, x_2, \ldots, x_{j-1}, x_1, x_{j+1}, \ldots, x_k),$$
$$(y, x_2, \ldots, x_{j-1}, x_1, x_{j+1}, \ldots, x_k),$$

and define the path

$$(x_1, x_2, \ldots, x_k), (y, x_2, \ldots, x_k).$$

The result follows. □

Our intention is to build an algorithm to find $n-1$ node-disjoint paths from each of $n-1$ distinct target nodes, held in T, to a given source node I of $S_{n,k}$ (I is never a target node). Note that hitherto we have spoken of paths from the source to the target nodes whereas now we are speaking of paths from the target nodes to the source. We have swapped the orientation as it turns out that our paths will actually be constructed by starting at a target node and working towards the source. Of course, this is of no practical consequence.

Before we present our algorithm, we show that there are certain assumptions that we can make.

Lemma 3 *Let T be a set of $n-1$ target nodes in $S_{n,k}$, where $k \geq 3$. There exists a dimension $i \in \{2, 3, \ldots, k\}$ such that each of $S^i_{n,k}(1), S^i_{n,k}(2), \ldots, S^i_{n,k}(n)$ contains at most $n-2$ nodes of T.*

Proof Suppose that for every $j \in \{2, 3, \ldots, k\}$, when we partition $S_{n,k}$ over dimension j, we get that some $S^j_{n,k}(i_j)$ contains all the target nodes from T. Thus, all target nodes in T have the form $(u, i_2, i_3, \ldots, i_k)$, for some u. This yields a contradiction as there are only $n-(k-1)$ such nodes. □

Suppose that $k \geq 3$. By Lemma 3, we can choose a dimension, i, say (where $i \in \{2, 3, \ldots, k\}$), so that when we partition the (n,k)-star $S_{n,k}$ over dimension i to obtain the $(n-1, k-1)$-stars $S^i_{n,k}(1), S^i_{n,k}(2), \ldots, S^i_{n,k}(n)$, we can be sure that each $S^i_{n,k}(j)$ contains at most $n-2$ target nodes. Suppose that $i \neq k$. The automorphism of $S_{n,k}$ obtained by swapping the ith and kth components of every node is such that $S^i_{n,k}(j)$ is mapped to $S^k_{n,k}(j)$. Suppose that $I = (y_1, y_2, \ldots, y_k)$ and let σ be any permutation of $\{1, 2, \ldots, n\}$ for which $\sigma(y_j) = j$, for $j = 1, 2, \ldots, k$. The permutation σ yields an automorphism of $S_{n,k}$ by mapping each node $(x_1, x_2, \ldots, x_k)$ to $(\sigma(x_1), \sigma(x_2), \ldots, \sigma(x_k))$, so that each $S^k_{n,k}(j)$ is mapped to $S^k_{n,k}(\sigma(j))$. Thus, we may assume that our source node I is $I_k = (1, 2, \ldots, k)$ and that when we partition over dimension k, the resulting $(n-1, k-1)$-stars $S^k_{n,k}(1), S^k_{n,k}(2), \ldots, S^k_{n,k}(n)$ each contains at most $n-2$ target nodes. Note

that when $k = 2$, we can assume that our source is I_k but not that partitioning over dimension k results in $(n-1, k-1)$-stars each containing at most $n-2$ target nodes. Henceforth, for brevity, we denote $S^k_{n,k}(i)$ by S_i (with S_i not to be confused with the n-star graph of the same name).

For $i \in \{k+1, k+2, \ldots, n\}$, we define $I_i = (k, 2, 3, \ldots, k-1, i) \in S_i$; for $i \in \{2, 3, \ldots, k-1\}$, we define $I_i = (k, 2, 3, \ldots, i-1, 1, i+1, \ldots, k-1, i) \in S_i$; and we define $I_1 = (k, 2, 3, \ldots, k-1, 1) \in S_1$. For $i = 1, 2, \ldots, n$, we denote the set of target nodes of T which lie in S_i, that is, $T \cap S_i$, by T_i. The basic structure of $S_{n,k}$ can be visualized in Fig. 1.

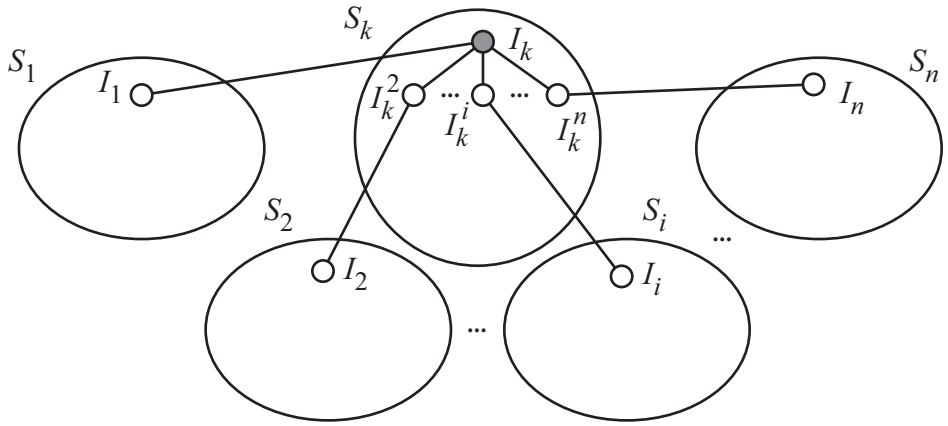

Figure 1. The basic partition of $S_{n,k}$.

3 The case for $k = 2$

In this section, we devise an algorithm `Disjoint_paths_when_`k`=2` which finds node-disjoint paths in $S_{n,2}$ from $n-1$ target nodes in T to the source node I_2 (which is not a target node). (Note that the one-to-many node-disjoint paths problem is trivial for $S_{n,1}$, an n-clique.) We begin with an overview of our algorithm and describe how it works in an abstract way before presenting a more detailed pseudo-code description (we link the abstract description with specific lines of pseudo-code). We refer the reader to Fig. 2 for a visual depiction of the general scenario.

Algorithm overview

1. Our algorithm begins by marking every edge from a node $I_2^j \in T_2$ to I_2 as a path to be output (thus, this deals with $|T_2|$ paths; recall, every S_i is a clique). (lines 2–4)

2. Consider some S_j, where $T_j = \emptyset$ and where if $j \neq 1$ then the node $I_2^j \notin T_2$. We might use such an S_j as a collection of 'transit' nodes; that is, we might construct a path passing through some of these nodes and then on from I_j to I_2^j and on to I_2 (or directly to I_2 if $j = 1$). We call such an S_j a transit set. (line 5)

3. Consider some S_i, where $i \neq 2$ and $T_i \neq \emptyset$ (that is, in which there is at least one target node). We need to build a path from every target node in S_i to I_2. (line 6)

4. If $i = 1$ or $I_2^i \notin T_2$ then we choose some target in T_i, giving preference to the node I_i, if it is a target node, and build a path through I_i, I_2^i (if $i \neq 1$) and on to I_2. This deals with 1 target node of S_i, though there may be others. (lines 7–17)

5. So, we have either $|T_i| - 1$ or $|T_i|$ remaining target nodes in S_i to deal with (depending upon whether we have just found a path to I_2 from a target node in 4 above).

6. We choose either $|T_i| - 1$ or $|T_i|$ (as appropriate) different transit sets (these transit sets will be used to build paths from the target nodes remaining in S_i to I_2; we shall prove below that we can always find the required number of transit sets). (lines 18–22)

7. For every remaining target node $I_i^j \in T_i$, we build a path to I_2, either directly through S_j (if S_j is one of our chosen transit sets) and on to I_2, or via some non-target node $I_i^l \in S_i$, through the chosen transit set S_l, and on to I_2 (if S_j is not one of our chosen transit sets). (lines 24–33)

8. We no longer regard the transit sets chosen as subsequently being transit sets; that is, available for use when we wish to construct other paths in future. (line 23)

9. We repeat the above process, from 3 onwards, for each S_i for which $i \neq 2$, $T_i \neq \emptyset$, and whose target nodes we have yet to deal with.

We now present our pseudo-code before elaborating on the description of our algorithm presented above.

```
1    Disjoint_paths_when_k=2(n,T,I_2,paths)
2    for every node I_2^j ∈ T_2 do
3      add the path ρ(I_2^j, I_2) = (I_2^j, I_2) to paths;
4    od
5    set transit := {S_j : j ∈ {1,3,4,...,n} and T_j = ∅, and
     if j ≠ 1 then I_2^j ∉ T_2};
6    for i = 1,2,...,n where i ≠ 2 and T_i ≠ ∅ do
7      if i = 1 or I_2^i ∉ T_2 then
8        if I_i ∈ T_i then
9          add the path ρ(I_1, I_2) = (I_1, I_2) (resp. ρ(I_i, I_2) =
           (I_i, I_2^i, I_2)) to paths if i = 1 (resp. i ≠ 1);
10         sorted_target := I_i;
11       else
12         choose some I_i^j ∈ T_i and add the path ρ(I_1^j, I_2) =
           (I_1^j, I_1, I_2) (resp. ρ(I_i^j, I_2) = (I_i^j, I_i, I_2^i, I_2)) to paths
           if i = 1 (resp. i ≠ 1);
13         sorted_target := I_i^j;
14       fi
15     else
16       sorted_target := ϵ;
17     fi
18     if sorted_target ≠ ϵ then
19       let good_trans ⊆ transit be of size |T_i| − 1;
20     else
21       let good_trans ⊆ transit be of size |T_i|;
22     fi
```

In the algorithm above, we write *sorted_target* := ϵ to denote that no node is associated with *sorted_target*.

We need to verify that such a subset *good_trans* exists, which we do now (see clause 6 in the description of our algorithm). Note that we need to verify this fact for every iteration of the for-loop spanning lines 6-34 and not just the first; thus, we deal with the general scenario below. Consider an iteration for some value of i. Suppose that $X = \{l : l = 1, 3, 4, \ldots, n, l < i, T_l \neq \emptyset\}$ with $Y \subseteq X$ defined as $Y = \{l : l \in X \setminus \{1\}, I_2^l \in T_2\}$, *i.e.*, X indexes the S_l's

(with target nodes) that have so far been dealt with in the for-loop in lines 6-34, and Y indexes those such S_l's for which I_2^l blocks direct paths from I_l to I_2 (so, every path from any target node of S_l must be routed through some $S_{l'}$ for which $T_{l'} = \emptyset$ and $I_2^{l'} \notin T$, if $l' \neq 1$). On an iteration of the for-loop for some i where $i \neq 2$ and $T_i \neq \emptyset$, any S_l from $\{S_1, S_2, \ldots, S_n\} \setminus \{S_2, S_i\}$ fails to be in *transit* for exactly one of six reasons:

1. S_l contains target nodes, $2 \neq l < i$, and ($l \neq 1$ and $I_2^l \in T_2$), *i.e.*, $l \in Y$;
2. S_l contains target nodes, $2 \neq l < i$, and ($l = 1$ or $I_2^l \notin T_2$), *i.e.*, $l \in X \setminus Y$;
3. S_l contains target nodes and $l > i$;
4. S_l contains no target nodes but is used as a set of transit nodes for a path from some target in S_j, where $j \in Y$;
5. S_l contains no target nodes but is used as a set of transit nodes for a path from some target in S_j, where $j \in X \setminus Y$;
6. S_l contains no target nodes, S_l is not used as a set of transit nodes for a path from some target node, $l \neq 1$, and $I_2^l \in T_2$.

Some of the different cases are illustrated in Fig. 2, where the target nodes are represented in black and where $i = 18$ (note that all S_j's are cliques even though they are not depicted as such). We can associate a target node with any S_l not in *transit* by choosing: the target node I_2^l in case 1; the (unique) target node t of S_l for which the path $\rho(t, I_2)$ passes through I_l in case 2; any target node of S_l in case 3; the unique target node t upon whose path $\rho(t, I_2)$ the nodes of S_l are used as transit nodes in cases 4 and 5; and the target node I_2^l in case 6. All such target nodes are distinct and are different from the target nodes in T_i. Thus, $|transit| \geq (n-2) - ((n-1) - |T_i|) = |T_i| - 1$. Furthermore, if $sorted\_target = \epsilon$ then $I_2^i \in T_2$ and $i \neq 1$, and this target node is distinct from all target nodes which were associated above; hence, in this case $|transit| \geq (n-2) - ((n-1) - |T_i| - 1) = |T_i|$ and our claim holds.

```
23      transit := transit \ good_trans;
24      for every I_i^j ∈ T_i \ {sorted_target} do
25        if S_j ∈ good_trans then
26          add the path ρ(I_i^1, I_2) = (I_i^1, I_1^i, I_1, I_2) (resp. ρ(I_i^j, I_2) =
```

```
            (I_i^j, I_j^i, I_j, I_2^j, I_2)) to paths if j = 1 (resp. j ≠ 1);
27          remove S_j from good_trans;
28        else
29          choose I_i^l ∉ T_i for which S_l ∈ good_trans;
30          add the path ρ(I_i^j, I_2) = (I_i^j, I_i^1, I_1^i, I_1, I_2) (resp. ρ(I_i^j, I_2)
            = (I_i^j, I_i^l, I_l^i, I_l, I_2^l, I_2)) to paths if l = 1 (resp. l ≠ 1);
31          remove S_l from good_trans;
32        fi
33      od
34    od
```

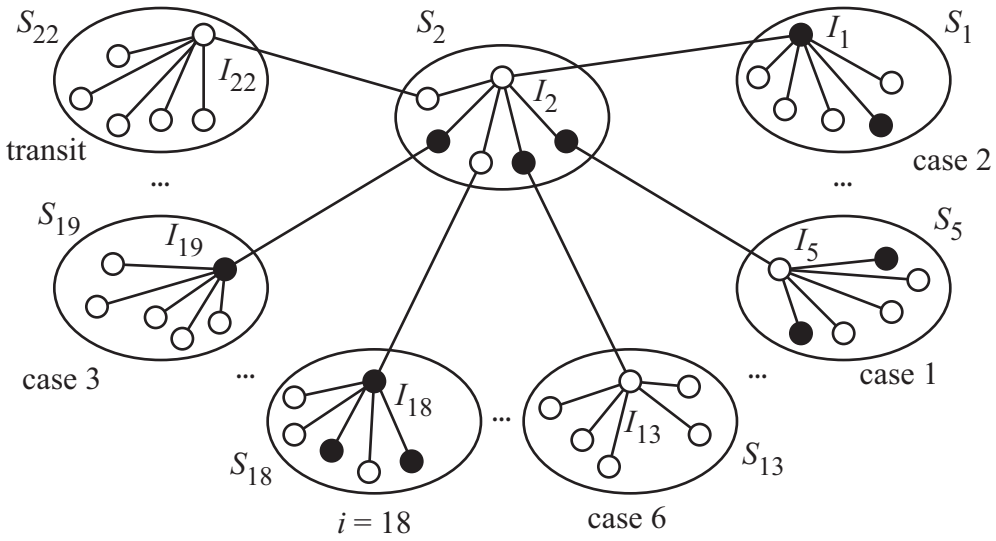

Figure 2. An illustration of different cases.

Consequently, `Disjoint_paths_when_`k=2 achieves its aims. Furthermore, all paths found by `Disjoint_paths_when_`k=2 have length at most 5 and the time complexity of `Disjoint_paths_when_`k=2 is $O(n^2)$.

Theorem 4 *When T is a set of $n-1$ distinct nodes in $S_{n,2}$ and when I is a node not in T, the algorithm* `Disjoint_paths_when_`k=2 *finds $n-1$ node-disjoint paths from the nodes in T to the node I. Furthermore, all paths found have length at most* 5 *and the time complexity of* `Disjoint_paths_when_`k=2 *is $O(n^2)$.*

Note that there are situations where at least one of the paths found in Theorem 4 necessarily has length 5. One such situation is when $n \geq 4$ and the target nodes include the nodes $(1,3)$, $(2,1)$ and $(2,3)$; for it is easy to see that any path from the node $(1,3)$ to the node $(1,2)$ (avoiding the nodes $(2,1)$ and $(2,3)$) must have length at least 5. Consequently, when $n \geq 4$, Theorem 4 is optimal in terms of the length of the longest path found. Of course, $S_{3,2}$ is a cycle of length 6 and so there is a configuration consisting of a source and two distinct targets where there is necessarily a path of length at least 3 joining the source and one of the targets.

4 Building node-disjoint paths when $k > 2$

We now detail a recursive algorithm to construct node-disjoint paths from $n-1$ distinct target nodes in $S_{n,k}$, given by the set of nodes T, to a source node (which is different from each target node). Recall that we may assume that our source is I_k and, for $k \geq 3$, when we partition over dimension k, none of the resulting copies of $S_{n-1,k-1}$ (namely $S_1, S_2, \ldots, S_n$) contains more than $n-2$ target nodes.

We remind the reader of some structural properties of $S_{n,k}$ and introduce some notation. Consider any node x of any S_i, where $i \in \{1, 2, \ldots, n\}$. The node x has $n-1$ neighbours, with 1 *external* neighbour outside S_i, in $S_{\iota(x)}$ (and so $\iota(x)$ is the first component of x), and $n-2$ *internal* neighbours in S_i. Any two neighbours y and y' of x within S_i are such that $\iota(x) \neq \iota(y) \neq \iota(y') \neq \iota(x)$. Denote the neighbour of x in $S_{\iota(x)}$ by $x_{\iota(x)}$ and call it x's neighbour of index $\iota(x)$, and for any neighbour y of x inside S_i, we say that y is x's neighbour of index $\iota(y)$ and refer to y as $x_{\iota(y)}$. Thus, x has a neighbour of every index from $\{1, 2, \ldots, n\} \setminus \{i\}$.

Our algorithm `Disjoint_paths` iterates through the subgraphs of $\{S_1, S_2, \ldots, S_n\} \setminus \{S_k, S_1\}$ finding paths in $S_{n,k}$ from each of the target nodes encountered. Actually, not the full paths are found: only the portions of the paths until the paths enter S_k, via some (distinct) entry nodes (that are not target nodes). Next, paths are found in the same way as above to deal with any target nodes in S_1 (if there are any). However, it is ensured that there is always one path from a target node, which may lie outside S_1, through the nodes of S_1, on to I_1, and then on to I_k. This accounts for one of our eventual output paths (note that this path does not contain any nodes of S_k, apart from I_k). Thus, we are left with finding $n-2$ node-disjoint paths from the

entry nodes in S_k and the target nodes of T_k to I_k, which we do recursively.

Here is our algorithm in more detail. Define $transit = \{S_j : T_j = \emptyset, j \neq k\}$. Consider some S_i, where $1 \neq i \neq k$ and $T_i \neq \emptyset$ (and so $S_i \notin transit$). A lower bound on the size of $transit$ is $(n-2) - ((n-1) - |T_k| - |T_i|) = |T_i| + |T_k| - 1$, and this lower bound is only reached when every S_j for which $S_j \notin transit$ and $i \neq j \neq k$ contains exactly one target node. For every target node $x \in T_i$, if $\iota(x) = k$ and $x_{\iota(x)} \notin T_k$ then place x in the set X_i (so, X_i contains every target node in T_i with first component k whose external neighbour is not a target node). We shall ultimately construct a path from each target node x in X_i to its neighbour in S_k, which we call x's entry node, and then on to I_k in S_k. Note that all entry nodes are distinct.

Define $Y_i' = \{x : x \in T_i \setminus X_i, \iota(x) \neq k, S_{\iota(x)} \in transit\}$; that is, Y_i' consists of those target nodes of $T_i \setminus X_i$ whose external neighbours lie in sets in $transit$. Note that it might be the case that for two distinct target nodes x and x' in Y_i', $\iota(x) = \iota(x')$. Choose a subset $Y_i \subseteq Y_i'$ that is maximal with respect to the property that for any two distinct target nodes x and x' in Y_i, $\iota(x) \neq \iota(x')$. If $x \in Y_i$ then remove $S_{\iota(x)}$ from $transit$. As we shall see, any S_j originally in $transit$ has its nodes used on at most one path from any target node in $T \setminus T_k$; moreover, any path we ultimately construct from some target node will use nodes from at most one S_j which originally appeared in $transit$. We shall ultimately construct a path from each target node x of Y_i to $x_{\iota(x)}$, then on through nodes in $S_{\iota(x)}$ to a node in S_k, and then on to I_k. We find an appropriate path through $S_{\iota(x)}$ and in to S_k, via an entry node in S_k, presently. Thus, we have $|T_i| - |X_i| - |Y_i|$ target nodes remaining to be dealt with in S_i and we have that $|transit|$ is at least $|T_i| - |Y_i| + |T_k| - 1$.

Suppose that $|T_i| - |X_i| - |Y_i| \leq |T_i| - |Y_i| + |T_k| - 1$, i.e., $0 \leq |X_i| + |T_k| - 1$, i.e., either $X_i \neq \emptyset$ or $T_k \neq \emptyset$. Each of the target nodes in $T_i \setminus (X_i \cup Y_i)$ can choose an internal neighbour so that the indices of the neighbours chosen are all different and are such that each chosen neighbour's external neighbour lies in some $S_j \in transit$ (note that all neighbours chosen are necessarily distinct and can not be target nodes, as otherwise they would have been included in Y_i on the grounds that Y_i is maximal). We shall ultimately construct a path from each target node in $T_i \setminus (X_i \cup Y_i)$ to its chosen internal neighbour, then on to the chosen neighbour's external neighbour, then through the S_j within which this external neighbour resides, then on to an entry node in S_k, and finally on to the node I_k. All S_j's to be used in these paths are removed from $transit$.

Suppose that $X_i = \emptyset$ and $T_k = \emptyset$, and so we have that $|transit| =$

$|T_i \setminus Y_i| - 1$. For all but one of the target nodes of $T_i \setminus Y_i$, we can proceed as in the previous paragraph and choose (distinct) internal neighbours through which we will construct paths that will ultimately lead to an entry node in S_k and on to I_k (we reiterate that none of these internal neighbours can be target nodes because of the maximality of Y_i). As above, all S_j's to be used on these paths are removed from *transit*. Thus, that leaves only 1 target node x of $T_i \setminus Y_i$ to deal with. Let y be the neighbour of x of index k. Note that as $X_i = \emptyset$ and $T_k = \emptyset$, y is an internal neighbour of x and can not be a target node. Moreover, by construction, y is not a chosen neighbour of one of the other target nodes in $T_i \setminus Y_i$, above (as $\iota(y) = k$ and all chosen neighbours of the other target nodes in $T_i \setminus Y_i$ have index different from k). We shall ultimately construct a path from x to y, then to y's neighbour in S_k (the entry node), and then on to I_k.

Consider now some $S_{i'}$ for which $T_{i'} \neq \emptyset$ and $i' \notin \{1, k, i\}$. Whilst dealing with S_i, above, we reduced the number of elements in *transit*; however, each time we removed some S_j from *transit*, above, we dealt with some target node in S_i. Thus, a lower bound on the size of *transit* now is $(n-3) - ((n-1) - |T_k| - |T_i| - |T_{i'}|) = |T_k| + |T_i| + |T_{i'}| - 2 \geq |T_k| + |T_{i'}| - 1$. Consequently, when we deal with $S_{i'}$ in exactly the same way that we dealt with S_i, the numeric arguments are identical and thus our path 'reservations' can be made for all target nodes in $S_{i'}$ too. Similarly, we 'reserve' paths in this way for all target nodes outside $S_k \cup S_1$.

Now consider S_1. We wish to ensure that we ultimately construct a path: from some target in S_1, if there is one, through S_1 to I_1, and then on to I_k; or, if $T_1 = \emptyset$, from some target outside $S_1 \cup S_k$, in to S_1, then on to I_1, and then on to I_k. We will deal with all other target nodes in S_1 as we have done above. As we have seen, immediately prior to dealing with S_1, we have that $|transit|$ is at least $|T_k| + |T_1| - 1$ and there are $|T_1|$ target nodes in S_1 to deal with.

Suppose that S_1 contains at least 1 target node. We proceed as we did above for other S_i's and deal with the target nodes in T_1. Having done so, we have reserved paths from all target nodes in T_1. Let D be the set of target nodes in T_1 whose distance to I_1 in S_1 is shortest (note that D might just consist of 1 element, and might, in fact, be $\{I_1\}$). Choose some $x \in D$ and consider a shortest path ρ in S_1 from x to I_1; denote by y the node on this path adjacent to x, if y exists (of course, if y exists then it is not a target node). If y does not exist then $D = \{I_1\}$; so, replace the path just reserved for I_1 with the path (I_1, I_k). If y exists and y does not appear on

any other reserved path from a target node in $T_1 \setminus \{x\}$ then replace the path reserved for x with the path ρ extended with the edge (I_1, I_k). Consequently, we may suppose that y appears on some reserved path from a target node x' in $T_1 \setminus \{x\}$. In particular, the node y must have been chosen as one of x''s internal neighbours because $x' \notin X_1 \cup Y_1$ and, at that point, $S_{\iota(x')} \notin \mathit{transit}$. We can now replace x''s reserved path with the path obtained by moving from x' to y, then along ρ to I_1, and then to I_k. Note that this path is node-disjoint from all of the reserved paths just constructed (which consist of at most 1 edge before they leave S_1).

Alternatively, suppose that S_1 contains no target nodes. If S_1 is no longer in $\mathit{transit}$ then we have indeed reserved a path from some target node outside S_1 through S_1 to I_1 and on to I_k. So, suppose that S_1 still resides in $\mathit{transit}$. By hypothesis, not all target nodes lie in S_k and so we can trivially ensure that for one target node outside S_k, we construct a path from this target node (possibly through one of its neighbours) in to S_1 and on to I_k through I_1 (note that if our initially chosen target node x is such that its internal neighbour y for which $y_{\iota(y)} = 1$ is a target node then we simply choose y as the target node whose corresponding path to I_k passes through S_1).

We have now made our path reservations: $n-2$ of these reservations are for paths to entry nodes in S_k; and 1 reservation is for a path, from some target node x_1, through S_1 and on to I_k directly. We can visualize these reservations in Fig. 3, where the black nodes are target nodes, the dark grey node is the source node, and the light grey nodes are entry nodes.

We now have to make concrete these path reservations. This means find paths through S_j's and on to entry nodes in S_k. However, we must ensure that all entry nodes in S_k are distinct from each other and also from any target nodes in S_k (as otherwise our resulting paths will not be vertex-disjoint).

We iterate through the target nodes of $T \setminus (T_k \cup \{x_1\})$ and make concrete our paths. Note that every such target node has an associated reserved path and that the number of target nodes and entry nodes ultimately chosen in S_k will be $n-2$. Some of our paths are already concrete. For example, any node x in some X_i has entry node $x_{\iota(x)}$ in S_k. Consider some target node x that has a reserved path through some S_j (where S_j was originally in $\mathit{transit}$). The path from x to the node $y \in S_j$ where the path enters S_j (y is the external neighbour of x or the external neighbour of an internal neighbour of x) is well-defined. By Lemma 2, there are $n-1$ distinct nodes of S_j, each at a distance of at most 3 from y, with the property that their external neighbours all lie in S_k. So, at least one of these external neighbours in S_k

has not yet been chosen as an entry node and is not a target node. Thus, we may choose such a node as the entry node corresponding to the target node x. Hence, we can find an entry node corresponding to every target node of $T \setminus (T_k \cup \{x_1\})$ so that all of these entry nodes are distinct and also distinct from all target nodes in S_k.

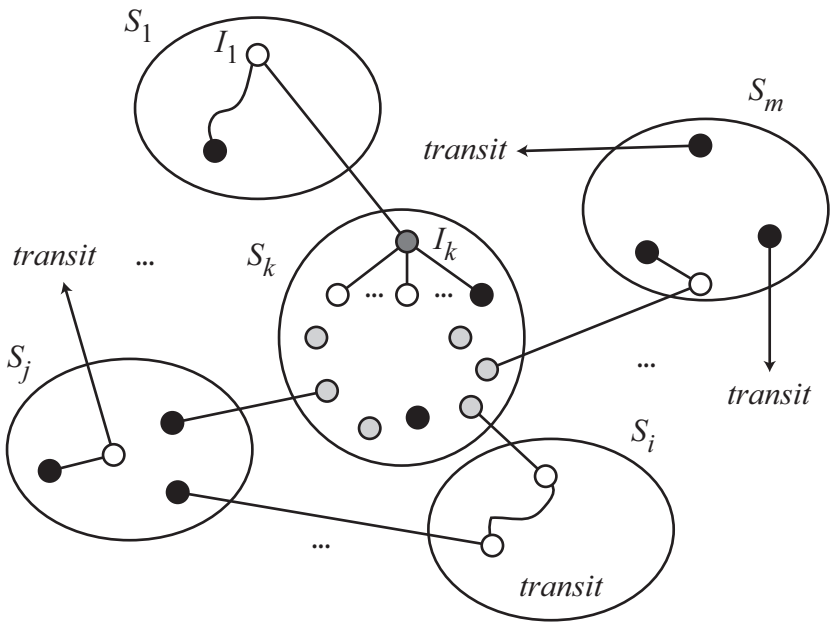

Figure 3. The path reservations.

Finally, having constructed our entry nodes, we recursively find $n-2$ vertex-disjoint paths in S_k from the target nodes of T_k and the entry nodes, and then we use (some of) these paths to extend our partially constructed paths from the target nodes of $T \setminus (T_k \cup \{x_1\})$ to their corresponding entry nodes to the source I_k. Hence, we have an algorithm to find node-disjoint paths in $S_{n,k}$ from each of any set T of $n-1$ target nodes to any given source node; that is, `Disjoint_paths` achieves its aims.

5 Path lengths and complexity

Having proved that our algorithm `Disjoint_paths` finds a collection of node-disjoint paths in $S_{n,k}$ from $n-1$ target nodes to a source node, we now turn to the lengths of the paths produced by the algorithm and the time complexity of the algorithm.

5.1 Path lengths

We derive below an upper bound on the length of any path constructed by `Disjoint_paths`; in the first instance, this upper bound is in the form of a recurrence relation. Let b_k be an upper bound on the length of any path produced by the algorithm `Disjoint_paths` applied in $S_{n,k}$, irrespective of n (at the moment, we have not shown that such an upper bound exists; however, we show, using induction, that it does and derive an estimate of it). By Theorem 4, $b_2 = 5$. As our induction hypothesis, we assume that b_{k-1} exists and is independent of n.

Clearly, any path produced by our algorithm `Disjoint_paths` which lies entirely within S_k has length at most b_{k-1}. By considering the construction of paths from target nodes in any S_i to I_k, where $i \neq k$, the length of any such path is at most the maximum of $\{\Delta(S_{n-1,k-1}) + 3, b_{k-1} + 6\}$. Solving this recurrence relation with $b_k = 5$ results in an upper bound on the length of any path constructed by `Disjoint_paths` of $b_k \leq 6k - 7$.

5.2 Time complexity

As regards the time complexity of our algorithm, we assume that our model of computation is such that when dealing with $S_{n,k}$, the integer n can be stored in one word of memory; thus, basic arithmetic operations involving n can be undertaken in constant time. This means that, for example, we can iterate through the neighbours of a given node in $O(n)$ time (the node is given as a k-tuple of integers between 1 and n). Note that we do not have an explicit representation for $S_{n,k}$, as its adjacency matrix, for example, as this would require an exponential amount of memory (in k); we simply start with n, k, a list of our target nodes, and our source, and we construct vertices of $S_{n,k}$ as and when they are required.

We begin by considering the computation undertaken by `Disjoint_paths` apart from the recursive call. We can register the S_i's in which target nodes reside and the number of target nodes in the S_i's in $O(kn)$ time. With reference to our remark in the first paragraph of the previous section, our assumptions as regards the number of target nodes in any S_i and the particular source node and partition in any recursive call result from an $O(kn)$ time computation. The initialization of *transit* can be undertaken in $O(kn)$ time. We can clearly deal with reserving paths corresponding to target nodes in some S_i in $O(|T_i|k)$ time, and we can make these paths concrete

in $O(|T_i|kn)$ time (note that the straightforward algorithm in [6] for finding a shortest path joining two nodes in $S_{n,k}$ is easily implementable with time complexity $O(kn)$). Thus, we obtain that the time taken by the algorithm `Disjoint_paths` on $S_{n,k}$, apart form the recursive call, is $O(kn^2)$ time. Consequently, by solving a simple recurrence relation, the time complexity of the algorithm `Disjoint_paths` is $O(k^2n^2)$.

6 Conclusions

In this paper, we have derived a polynomial-time algorithm to find node-disjoint paths from each of $n-1$ distinct target nodes in $S_{n,k}$ to a source node (different from any target node). The length of any path constructed is at most $6k-7$. This should be compared with the diameter of $S_{n,k}$ which is at most $2k-1$ (see the Introduction for an exact formula for the diameter of $S_{n,k}$, suffice it to say that when $1 \leq k \leq \lfloor \frac{n}{2} \rfloor$, the diameter is exactly $2k-1$).

The most appealing aspect of our construction is that we have managed to obtain an upper bound on the length of any of our paths from the source node to the target nodes in $S_{n,k}$ that is independent of n; thus, n can be increased arbitrarily and our upper bound remains the same. Also, our algorithm is efficient and in practice will be even faster as the scenarios whereby recursive calls are made will be much fewer than in our worst-case analysis. Furthermore, our algorithm should be easily and efficiently implementable on a distributed-memory machine where global knowledge of the source and target nodes is available but where the paths have to be constructed in a distributed fashion by the individual processors (see, for example, the underlying model in [20]). We say no more here concerning this claim but note that the structure of our algorithm lends itself to the incremental construction of paths.

However, our upper bound is roughly three times the diameter of $S_{n,k}$, whereas for n-star graphs the analogous upper bound is at most the diameter plus 2 and for hypercubes the analogous upper bound is at most the diameter plus 1. Thus, we conjecture that our result is not optimal. (Note that our result is optimal when $k = 2$.) It may be the case that a refined analysis of the possible distributions of sources and targets might yield that a recurrence of $b_k \leq b_{k-1} + 2$ (see the previous section) can be obtained (which is what would be required in order to get a maximal path length comparable with the scenarios in n-star graphs and hypercubes). As yet, we have been unable

to make progress in this direction ((n,k)-star graphs are conceptually much more complex than n-star graphs and hypercubes).

Of course, we can apply our algorithm to $S_{n,n-1}$, *i.e.*, the n-star. What results is an algorithm of time complexity $O(n^4)$ that finds node-disjoint paths, each of length at most $6n-13$. As might be expected, the algorithm from [5], designed specifically for n-stars, is better in that it has time complexity $O(n^2)$ and results in node-disjoint paths each of length at most $\frac{3n+9}{2}+2$. Similarly, we can apply our algorithm to produce a (u,v)-container, for distinct nodes u and v of $S_{n,k}$. Again, as expected, the resulting container is much worse than that produced by the (polynomial-time) algorithm in [16] (specifically designed for the purpose) where one of wide-diameter at most $2k+1$ is produced. Nevertheless, our algorithm gives a polynomial-time alternative for constructing node-disjoint paths in n-stars and containers in $S_{n,k}$.

References

[1] S.B. Akers, D. Horel and B. Krishnamurthy, The star graph: an attractive alternative to the n-cube, *Proc. of Internat. Conf. on Parallel Processing* (1987) 393–400.

[2] L.N. Bhuyan and D.P. Agrawal, Generalized hypercube and hyperbus structures for a computer network, *IEEE Transactions on Computers* **C-33** (1984) 323–333.

[3] B. Bollobas, *Extremal Graph Theory*, Dover (2004).

[4] J.-H. Chang and J. Kim, Ring embedding in faulty (n,k)-star graphs, *Proc. of 8th Internat. Conf. on Parallel and Distributed Systems* (*ICPADS'*01) (2001) 99–106.

[5] C.C. Chen and J. Chen, Nearly optimal one-to-many parallel routing in star networks, *IEEE Transactions on Parallel and Distributed Systems* **8** (1997) 1196–1202.

[6] W.-K. Chiang and R.-J. Chen, The (n,k)-star graph: a generalized star graph, *Information Processing Letters* **56** (1995) 259–264.

[7] W.-K. Chiang and R.-J. Chen, Topological properties of the (n, k)-star graph, *Internat. Journal of Foundations of Computer Science* **9** (1998) 235–248.

[8] S. Gao, B. Novick and K. Qiu, From Hall's matching theorem to optimal routing on hypercubes, *Journal of Combinatorial Theory Series B* **74** (1998) 291–301.

[9] Q.P. Gu and S. Peng, Efficient algorithms for disjoint paths in star graphs, *Proc. of 6th Transputer/Occam International Conf.* (1994) 53–65.

[10] Q.P. Gu and S. Peng, Node-to-set disjoint paths problem in star graphs, *Information Processing Letters* **62** (1997) 201–207.

[11] D.F. Hsu and Y.D. Lyuu, A graph-theoretical study of transmission delay and fault tolerance, *Internat. Journal of Mini and Microcomputers* **16** (1994) 35–42.

[12] H.-C. Hsu, Y.-L. Hsieh, J.J.M. Tan and L.-H. Hsu, Fault Hamiltonicity and fault Hamiltonian connectivity of the (n, k)-star graphs, *Networks* **42** (2003) 189–201.

[13] H.-C. Hsu, C.-K. Lin, H.-M. Hung and L.-H. Hsu, The spanning connectivity of the (n, k)-star graphs, *Internat. Journal of Foundations of Computer Science* **17** (2006) 415–434.

[14] C.-N. Lai, G.-H. Chen and D.-R. Duh, Constructing one-to-many disjoint paths in folded hypercubes, *IEEE Transactions on Computers* **51** (2002) 33–45.

[15] S.C. Liaw, G.J. Chang, F. Cao and D.F. Hsu, Fault-tolerant routing in circulant networks and cycle prefix networks, *Annals of Combinatorics* **2** (1998) 165–172.

[16] T.-C. Lin, D.-R. Duh and H.-C. Cheng, Wide diameters of (n, k)-star networks, *Internat. Conf. on Computing, Communications and Control Technologies* (*CCCT'*04) (2004) 160–165.

[17] T.-C. Lin and D.-R. Duh, Constructing vertex-disjoint paths in (n, k)-star graphs, *Information Sciences* **178** (2008) 788–801.

[18] J.-H. Park, One-to-many disjoint path covers in a graph with faulty elements, *Proc. of Computing and Combinatorics,* 10*th Ann. Internat. Conf.* (*COCOON'*04), Lecture Notes in Computer Science Vol. 3106 (ed. K.-Y. Chwa, J.I. Munro), Springer (2004) 392–401.

[19] M.O. Rabin, Efficient dispersal of information for security, load balancing, and fault tolerance, *Journal of the ACM* **36** (1989) 335–348.

[20] I.A. Stewart, Distributed algorithms for building Hamiltonian cycles in k-ary n-cubes and hypercubes with faulty links, *Journal of Interconnection Networks* **8** (2007) 253–284.

[21] Y. Xiang, Y. Lin, Z. Gu, C. Zhang and B. Cong, One-to-many node-disjoint paths problem in com-star interconnection networks, *Proc. of* 13*th IASTED Int. Conf. on Parallel and Distributed Computing and Systems* (*PDCS'*01) (2001) 248–253.