A Self-stabilizing One-To-Many Node Disjoint Paths Routing Algorithm in Star Networks

Rachid Hadid(✉) and Vincent Villain(✉)

MIS, Université de Picardie Jules Vernes, 33 rue Saint Leu, Cedex 1, 80039 Amiens, France
hadid.rashid@gmail.com, vincent.villain@u-picardie.fr

Abstract. The purpose of the paper is to present the first self-stabilizing algorithm for finding $n-1$ *one-to-many node-disjoint* paths in *message passing* model. Two paths in a network are said to be node disjoint if they do not share any nodes except for the endpoints. Our proposed algorithm works on n-dimensional star networks S_n. Given a source node s and a set of $D = \{d_1, d_2, ..., d_{n-1}\}$ of $n-1$ destination nodes in the n-dimensional star network, our algorithm constructs $n-1$ node-disjoints paths $P_1, P_2, ..., P_{n-1}$, where P_i is a path from s to d_i, $1 \leq i \leq n-1$. Since the proposed solution is self-stabilizing [7], it does not require initialization and withstands transient faults. The stabilization time of our algorithm is $O(n^2)$ rounds.

Keywords: Fault-tolerance · Self-stabilization · Distributed systems · Star networks · Node disjoint paths

1 Introduction

The concept of *self-stabilization* [7] is the most general technique to design a system to tolerate arbitrary transient (in other words, limited in time) faults. A self-stabilizing system, regardless of the initial states of the processors and initial messages in the links, is guaranteed to converge to the intended behaviour in finite time. We view a fault that perturbs the state of the system but not its program as a transient fault. The problem of finding disjoint paths in a network has been given much attention in the literature due to its theoretical as well as practical significance to many applications, such as layout design of integrated circuits [22], communication protocols [10], secure message transmission [24], survivable design of telecommunication networks [23] and reliable routing [15]. Node disjoint paths can be used for secure communication by breaking up data into several shares and sending them along the disjoint paths to make it difficult for an adversary with bounded eavesdropping capability to intercept a transmission or tamper with it. Network survivability reflects the ability of a network to maintain service continuity during and after failures. In practice, it is important to construct node disjoint paths in networks, because they can be used to enhance the transmission reliability. Alternatively, the same crucial message can be sent over multiple node disjoint paths in a network that is prone to message losses to avoid omission failures, or information on the re-routing of traffic along non-faulty disjoint paths can be provided in the

Published by Springer Nature Switzerland AG 2020
A. Remke and V. Schiavoni (Eds.): DAIS 2020, LNCS 12135, pp. 186–203, 2020.
https://doi.org/10.1007/978-3-030-50323-9_12

presence of faults in some disjoint paths. Routing is a process of transmitting messages among nodes, and its efficiency is crucial to the performance of a network. Efficient routing can be achieved by using internally node disjoint paths, because they can be used to avoid congestion, accelerate transmission rate, and provide alternative transmission routes. Moreover, node disjoint paths between two processes present additional benefits such as broadening the network bandwidth and load balancing of the network by allowing communicating pair of processes to distribute the communication load on node disjoint paths without congesting communication channels in the network. There are three paradigms for the study of node disjoint paths in interconnection networks: the *one-to-one*, and the *one-to-many*, and the *many-to-many* node disjoint paths [8]. The one-to-one node disjoint paths constructs the maximum number of node disjoint paths in the network between two given nodes, and the one-to-many node disjoint paths constructs node disjoint paths in the network from a given node to each of nodes in a given set. The one-to-many node disjoint paths problem are fundamental and extensively studied in graph theory. One-to-many node disjoint paths were first presented in [26] where the Information Dispersal Algorithm (IDA) was proposed on the hypercube. Some algorithms to find one-to-many node disjoint paths in a variety of networks are proposed in [5,6,8–10,14,17–19,21,25]. Let $G = (V, E)$ be a connected graph, where V and E represent the node set and edge set of G, respectively. Throughout this paper, we use network and graph, processor and node, and link and edge, interchangeably. The *n-dimensional star* network (*n-star* for short) [1–3] is one of most efficient and popular interconnection networks because of its attractive properties, including regularity, node symmetric, small diameter, and recursive construction. The *n-star* network, denoted as $S_n = (V, E)$, is a bidirected graph consisting of $n!$ nodes, each node is identified with a distinct *permutation* of n symbols $1, 2, \ldots, n$. There is a link between any two permutations (nodes) iff one can be reached from the other by

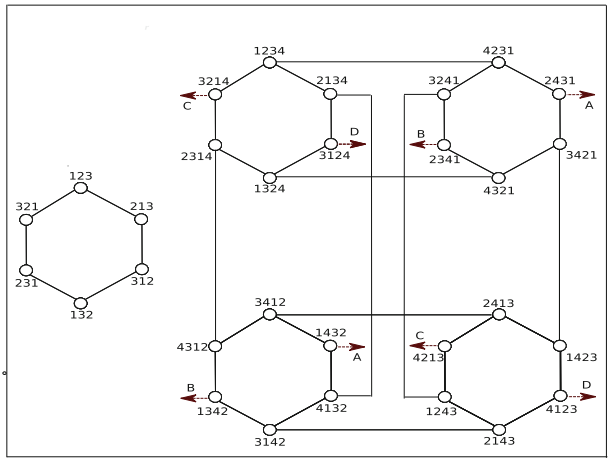

Fig. 1. An example 3-star and 4-star.

interchanging its first symbol with any other symbol. More precisely, the node representing permutation $a_1a_2 \dots a_{i-1}a_ia_{i+1} \dots a_n$ have links to n-1 other permutations (nodes) $a_ia_2 \dots a_{i-1}a_1a_{i+1} \dots a_n$, for some $2 \leq i \leq n$. The node with the permutation $123 \dots n$ will be called the *identity node*. Figure 1 illustrates the 3-*star* and 4-*star* systems. The construction of the one-to-many node disjoint paths in the n-star has also been considered by researchers in graph theory [5,8,29]. The disjointness of these paths is ensured by using different approaches. In a first approach [29], we can fix a particular symbol j, $1 \leq j \leq n$, at the last position of all the nodes on the path noted P_j. So, all the nodes of the path P_j (except for at most one) have a common symbol j at the last position of their permutations. Hence, each selection of the symbol j, $1 \leq j \leq n$, will make the path P_j node disjoint from the others paths. The following example illustrates the concepts (see example 1).

Example 1. Assume that we have 4-*star* system and let $s = 1234$ and $D = \{4321, 1342, 4123, 2134\}$. We construct 4 node disjoint paths P_1, P_2, P_3 and P_4 from s to each destination process $d_1 = 4321$, $d_2 = 1342$, $d_3 = 4123$, and $d_4 = 2134$, respectively, by fixing the symbols 1, 2, 3, and 4 at the last position of all the nodes (except for at most one) on P_1, P_2, P_3, and P_4, respectively. The node disjoint paths P_1, P_2, P_3, and P_4 may look as follows. (i) $P_1 = (s = 1234 \rightarrow, 423\underline{1} \rightarrow, 32\underline{4}1 \rightarrow, 2\underline{3}41 \rightarrow, 4\underline{3}21)$. (ii) $P_2 = (s = 1234 \rightarrow, 2\underline{1}34 \rightarrow, 413\underline{2} \rightarrow, 31\underline{4}2 \rightarrow, 1\underline{3}42)$. (iii) $P_3 = (s = 1234 \rightarrow, 32\underline{1}4 \rightarrow, 421\underline{3} \rightarrow, 2\underline{4}13 \rightarrow, 14\underline{2}3 \rightarrow, 4\underline{1}23)$. (iv) $P_4 = (s = 1234 \rightarrow, 2\underline{1}34)$. Where the underlined digit denotes the swapped one.

However, since n-star graphs are node symmetric, the position at which to fix the symbol j does not have to be the last one, as shown in [8]. The position at which we fix our symbols could be i for $2 \leq i \leq n$. Therefore, we have $n-1$ ways of fixing the symbol j in the path noted P_i^j, where i denotes the position where the symbol j is fixed. So, for each path P_i^j is designed a unique position i, $2 \leq i \leq n$, and a distinct symbol j, $1 \leq j \leq n$, such that i is the position of the symbol j in all the permutations (except for at most one) of the nodes on the path P_i^j. We can illustrate this approach by using the following example (see example 2).

Example 2. Assume that we have 4-*star* system and let $s = 1234$ and $D = \{4321, 1342, 4123\}$. We construct 3 node disjoint paths P_2^1, P_3^2, and P_4^3 by fixing the symbols 1, 2, and 3 at the second, third, and fourth position for each node on the path P_2^1, P_3^2, and P_4^3, respectively. (i) A path P_2^1 from s to $d_1 = 4321$ that keeps the symbol 1 at position 2 may look as follows: $P_2^1 = (s = 1234 \rightarrow, 2\underline{1}34 \rightarrow, 31\underline{2}4 \rightarrow, 1\underline{3}24 \rightarrow, 432\underline{1})$. (ii) A path P_3^2 from s to $d_2 = 1342$ that keeps the symbol 2 at position 3 may look as follows: $P_3^2 = (s = 1234 \rightarrow, 2\underline{1}34 \rightarrow, 31\underline{4}2 \rightarrow, 1\underline{3}42)$. (iii) A path P_4^3 from s to $d_3 = 4123$ that keeps the symbol 3 at position 4 may look as follows: $P_3^2 = (s = 1234 \rightarrow, 32\underline{1}4 \rightarrow, 421\underline{3} \rightarrow, 2\underline{4}13 \rightarrow, 14\underline{2}3 \rightarrow, 4\underline{1}23)$.

However, this solution can be simplified as follows [5]. We may choose to fix the same symbol j, $1 \leq j \leq n$, over all the node disjoint paths, but in different positions i for $2 \leq i \leq n$. So, for each path P_i, the same symbol j is fixed at the same position i in all processes on the path P_i, except for at most one. The following example illustrates the concepts (see example 3).

Example 3. Assume that we have 4-*star* system and let $s = 1234$ and $D = \{4321, 1342, 4123\}$. We construct 3 node disjoint paths P_2, P_3, and P_4 by fixing the symbol 1 ($j = 1$) at the second, third, and fourth position for each node on the path P_i, $2 \leq i \leq 4$, respectively, as follows. $P_2 = (s = 1234 \rightarrow, 2\underline{1}34 \rightarrow, 31\underline{2}4 \rightarrow, 1\underline{3}24 \rightarrow, 432\underline{1})$. $P_3 = (s = 1234 \rightarrow, 32\underline{1}4 \rightarrow, 2\underline{3}14 \rightarrow, 431\underline{2} \rightarrow, 1\underline{3}42)$. $P_4 = (s = 1234 \rightarrow, 423\underline{1} \rightarrow, 2\underline{4}31 \rightarrow, 34\underline{2}1 \rightarrow, 142\underline{3} \rightarrow, 4\underline{1}23)$.

Self-stabilizing algorithms to find node disjoint paths are proposed in [11,12,16,28]. Self-stabilizing algorithms for finding one-to-one node disjoint paths between two endpoints for hypercube and mesh networks have been proposed in [11,28], respectively. A new self-stabilizing algorithm for finding two one-to-one node disjoint paths problem in arbitrary network was proposed in [12]. The basis of the algorithm was outlined in [16] as a brief announcement. It has been shown that finding the node disjoint paths is NP-hard in general graphs [13]. For n-dimensional hypercubes H_n (which has diameter $d(H_n) = n$), it was proved that n disjoint paths for the one-to-one node disjoint paths paradigm [27] and n disjoint paths for the one-to-many node disjoint paths paradigm [26] can be found in $O(n^2)$ time. For n-dimensional star graphs, (which has diameter $d(S_n) = \lfloor \frac{3(n-1)}{2} \rfloor$), it was shown that $n-1$ disjoint paths for one-to-one node disjoint paths paradigm can be found in $O(n^2)$ time [20] and $n-1$ disjoint paths for one-to-many node disjoint paths can be found in $O(n^2)$ time [8]. The time complexity of the above self-stabilizing algorithms is as follows: $O(d)$ rounds for algorithm [11] (working in mesh network), whereas the time complexity of the algorithms [12,16,28] is $O(d^2)$ rounds, where d the diameter of the network.

1.1 Contributions

In this paper, we present the first self-stabilizing distributed algorithm for finding $n-1$ node-disjoint paths between the source process s and $n-1$ other destination processes $\{d_1, d_2, ..., d_{n-1}\}$ in the n-star network. While it takes the same polynomiale $O(n^2)$ rounds to solve the same problem by the result in [5,8,29]. We propose a method based on message-passing techniques to process global information, which is more approach to reality. We adapt the approach used in [5] to ensure the disjointness of these $n-1$ paths. Unlike previous solutions, our algorithm does not utilize the cycle presentation of the permutations, making it easy to understand. Our approach is different from the previous one [5] in that the disjoint paths are constructed from the source s to the $n-1$ processes in the n-star. This makes our solution more suitable to implement. In addition, it reveals interesting functions to implement the disjointness of the paths in a self-stabilizing distributed environment.

The rest of the paper is organised as follows. In Sect. 2, we describe the distributed system model used in this paper. Then, we present the one to many node disjoint paths algorithm in Sect. 3 and its correctness proof in Sect. 4. Finally, we make some concluding remarks in Sect. 5.

2 Distributed System and Programs

Our algorithm is designed to operate on an *asynchronous distributed system* modelled as an *n-star network*. A *transposition* $\pi[1, i]$ on a permutation p,

noted $\pi[1,i](p)$, is to exchange the positions of the first and the ith symbol in the permutation p. For example, if $p = a_1a_2...a_{i-1}a_ia_{i+1}... a_n$, then $\pi[1,i](p) = a_ia_2...a_{i-1}a_1a_{i+1}...a_n$. There is a link between any two processes p and q if and only if $\pi[1,i](p) = q$, for some $2 \leq i \leq n$. We will use a process identity (process name) in a star network to refer also to the permutation that labels the process. We consider the message-passing model where communication between neighboring processes is carried out by messages exchanged through bidirectional links, $i.e.$, each link can be seen as two channels in opposite directions. A $distributed\ protocol$ for such a message passing system consists of a collection of n $local\ programs$, one for each processor in the system. This local program provides the ability to the processor either to perform local computations, or to send and receive messages from each of its neighbours in the n-star network. More precisely, the program consists of a collection of actions. Overall, an action is of the form: $<guard>::<statements>$. A $<guard>$ is mainly triggered when an $input$ message is received. In addition, tools like $timer$ or $randomly$ and $spontaneous$ are used in the $<guard>$. $<statements>$, executed when a $<guard>$ is activated, is a sequence of assignments/computations, invoking functions/procedures, and/or message sending. Note that an action can be executed only if its guard is activated. A message is of the following form: $(type, value)$. A message may also contain more than one value. We define the $state$ of each process to be the state of its local memory and the contents of its incoming channels. The global state of the system, referred to as a $configuration$, is defined as the product of the states of processes. We denote by $\mathcal{C}$ the set of all possible configuration. An execution of a protocol $\mathcal{P}$ in a system $\mathcal{S}$ is an infinite sequence of configurations $\gamma_0, \gamma_1, ..., \gamma_i...$ such that in any transition $\gamma_i \mapsto \gamma_{i+1}$ either a process take a step. We assume that the message delivery time is finite but unbounded. We also consider a message in transit until it is processed by the receiving processor. Moreover, each link is assumed to be of bounded capacity, FIFO, and reliable (the messages are not lost and delivered UN-corrupted) during and after the stabilization phase.

3 Self-stabilizing Algorithm

In this section, we first present the basis description of the proposed solution. Given a process s and $n-1$ other distinct processes $D = \{d_1, d_2, \ldots, d_{n-1}\}$ in the n-star system, the proposed algorithm constructs $n-1$ node-disjoint paths $P_1, P_2, \ldots, P_{n-1}$, where P_h is the path from s to d_h, for $h = 1, 2, \ldots, n-1$. Note that the algorithm works also for $D = \{d_1, d_2, \ldots, d_m\}$ such that $m < n$. Our solution works in two phases referred to as $labeling\ phase$ (Algorithm 2) and $\mathcal{O}ne\text{-}\mathcal{T}o\text{-}\mathcal{M}any\ node\text{-}disjoint\ paths\ construction\ phase$ (Algorithm 4). The $\mathcal{O}ne\text{-}\mathcal{T}o\text{-}\mathcal{M}any$ node-disjoint paths construction phase is based on the labeling process phase, $i.e.$, the progress of this phase is ensured only after the labeling process terminates successfully. During the labeling phase, each destination process d_h, $1 \leq h \leq n-1$, in D should be labeled by a unique label j, $2 \leq j \leq n$, such that the index j is the reserved position for the symbol 1 for the processes on P_j connecting process s to process d_h. So, after this phase the set $D = \{d_1, d_2, \ldots, d_{n-1}\}$ is mapped to the set $DL = \{(d_1, j_1), (d_2, j_2), \ldots, (d_{n-1}, j_{n-1})\}$ where j_h, $2 \leq j_h \leq n$, is the label assigned to the process d_h, $1 \leq h \leq n-1$.

During the $\mathcal{O}ne\text{-}\mathcal{T}o\text{-}\mathcal{M}any$ node-disjoint paths construction phase, we construct $n-1$ node-disjoint paths $P_2, P_3, \ldots, P_n$ from the source process s to the new labeled destination processes $(d_1, j_1), (d_2, j_2), \ldots, (d_{n-1}, j_{n-1})$ such that each path P_j, $2 \leq j \leq n$, connects the source process s to a destination process $(d_h, j) \in DL$. For each path P_j, $2 \leq j \leq n$, we reserve a unique position j for the path, such that, for all processes on P_j (except for maximum two processes) the symbol 1 is at jth position in the permutations. In other words, we construct each path P_j, $2 \leq j \leq n$, from the source process s to the destination process $(d_h, j) \in DL$ and keeps the symbol 1 in a fixed position j along all the processes on P_j, except for maximum two processes.

Prior to the presentation of these two phases in details, we first present Algorithm 1 which computes the shortest distance between the source s and a destination d (Function $\mathcal{D}ist(s,d)$). This metric is needed during the labeling phase. A shortest path P from a process s in the n-star system S_n to a destination process d is given by the following two rules [3]. Assuming that the shortest path P is built up to the process $p \neq d$ (initially, $p = s$), then the successor of p on P is identified as follows (See Function $NextNeigh()$ in Algorithm 1). Let x be the first symbol in the permutation of p, then (r_1) If there exists a symbol y in the kth position, $2 \leq k \leq n$, such that $d[k] = x$ ($i.e.$, $Direct(p)$ is true), then exchange x with y ($d[k]$ denotes the symbol at the position k on a permutation d). In other words, x directly reaches its correct position in d.

(r_2) Otherwise, exchange x with the symbol y in the kth position such that y is not in a correct position in d, $y \neq d[k]$, if exists ($i.e.$, $ByPass_p \neq \emptyset$). If multiple such symbols exist, then choose the symbol y with the smallest position in the permutation d. In other words, y, which is in incorrect position, is temporarily placed in the first position. Thereafter, it is moved to its correct position by applying rule (r_1). The Function $\mathcal{D}ist(s,d)$ computes in $dist$ (using Function $NextNeigh()$) the number of steps needed to built a shortest path from the source s to the destination d. The following example illustrates the distance computing between s and d in 5-star system (see example 4).

Algorithm 1. Distance Computing Algorithm

Input Source s and destination d.
Function $NextNeigh(d : Process\text{-}ID) : Process\text{-}ID$
Predicate $\text{Direct}(p) \equiv (\exists k, 2 \leq k \leq n, :: (p[1] = d[k]))$
Macro $\text{ByPass}_p = \{k, 2 \leq k \leq n, :: p[k] \neq d[k]\}$
begin
(r_1) **if** $Direct(p)$ **then**
$k \leftarrow j, 2 \leq j \leq n, :: (p[1] = d[j])$
(r_2) **else**
$k \leftarrow min_k(ByPass_p)$
return$(\pi(1,k)(p))$
end

Output The distance $\mathcal{D}ist(s,d)$ between s and d.
Function $\mathcal{D}ist(s, d : Process\text{-}ID) : Integer$
Variables $dist \leftarrow 0$; $p \leftarrow s$;
begin
while $(p \neq d)$ **do**
$q \leftarrow NextNeigh(d)$;
$dist \leftarrow dist + 1$;
$p \leftarrow q$;
end while
return$(dist)$
end

Example 4. (i) The distance computing, using the path P created by the rules (r_1) and (r_2), from s to $d = 51243$ looks as follows: $P = (s = 12345\ [dist = 0]\ (\xrightarrow{r_1}), 2\underline{1}345\ [dist = 1]\ (\xrightarrow{r_1}), 31\underline{2}45\ [dist = 2]\ (\xrightarrow{r_1}), 5124\underline{3} = d)\ [dist = 3]$. (ii) The distance computing between s and $d = 14523$ looks as follows. $P = (s = 12345\ [dist = 0]\ (\xrightarrow{r_2}), 2\underline{1}345\ [dist = 1]\ (\xrightarrow{r_1}), 413\underline{2}5\ [dist = 2]\ (\xrightarrow{r_1}), 1\underline{4}325\ [dist = 3]\ (\xrightarrow{r_2}), 34\underline{1}25\ [dist = 4]\ (\xrightarrow{r_1}), 5412\underline{3}\ [dist = 5]\ (\xrightarrow{r_1}), 14\underline{5}23 = d)\ [dist = 6]$. Where the

value in brackets indicate the value of $dist$ at a process, and the rule used to swap to the next process is indicated between parentheses on the arrow.

It is shown in [3] that this algorithm will always find the shortest path from s to d in S_n. It has been also shown in [3] that the diameter of S_n is equal to $\lceil 3(n-1)/2 \rceil$.

3.1 Labeling Process

The labeling phase is handled by the source process s (Algorithm 2 and Function $\mathcal{L}abeling()$). Let $D = \{d_1, d_2...., d_{n-1}\}$ be a set of $n-1$ distinct processes in the n-star network S_n. During this phase, each destination process d_h, $1 \leq h \leq n-1$, in the set D is labeled by a unique label j, $2 \leq j \leq n$. Thereafter, we denote by DL the set of the new labeled processes (d_h, j), $2 \leq j \leq n$. The index j, $2 \leq j \leq n$, associated to each destination d_h, refers to the unique fixed position of the symbol 1 along all the processes on the path P_j (except for maximum two processes) connecting s and d_h. Process p in the n-star network S_n is a (1)-$process$ if 1 is the first symbol in the permutation p. Process p in S_n is a (i_1)-$process$, where $2 \leq i \leq n$, if 1 is the ith symbol in the permutation p. The set DL is implemented using an array of structure where each element of the array in the position pos, $1 \leq pos \leq n-1$, contains the couple (id, lab) where id and lab represent the permutation (d_h) and the label (j) associated to the processs d_h, respectively. The labeling process is implemented by repeating the following three simple rules (L_1), (L_2), and (L_3). Note that the index pos is initiated to 1 and is increased each time a new element is added to DL.

(L_1) Let $D_i \subseteq D$ be the subset of all (i_1)-$processes$, $2 \leq i \leq n$, such that $D_i \neq \varnothing$; then, pick a process $d_h \in D_i$, $1 \leq h \leq n-1$ such that $\mathcal{D}ist(s, d_h)$ is the smallest among all the processes in D_i, label d_h by i and call (d_h, i) the $representative\ process$ of the set D_i (see Function $Representative(D_i)$). If multiple processes have the same smallest distance, then choose the process with the smallest id number among them as representative process. Thus, for each $D_i \neq \varnothing$, the representative process (d_h, i) gets the position i. The process d_h is deleted from D and (d_h, i) is inserted into DL. So, $DL[pos].id$ and $DL[pos].lab$ are seted to d_h and i, respectively (See Procedure $Affect\text{-}label(Representative(D_i), i, pos)$.

(L_2) Then, for each process d_h in the set D such that d_h is a (i_1)-$process$, but not a representative process for a subset D_i, reserve a position j for d_h with $2 \leq j \leq n$, that is not already assigned to any process in D, then d_h is labeled by j. Similarly to (L_1), the process d_h is deleted from D and (d_h, j) is inserted into DL.

(L_3) Finally, for each d_h in the set D such that d_h is a (1)-$process$ reserve a position j for d_h with $2 \leq j \leq n$, that is not already assigned to any process in D, then d_h is labeled by j. Similarly to (L_1), the process d_h is deleted from D and (d_h, j) inserted into DL. Note that all (i_1)-$processes$ are labeled before (1)-$processes$. In order to illustrate the above concepts, we provide the following example (see example 5)

Example 5. Assume that we have 5-star system and the set $D = \{d_1, d_2, d_3, d_4\}$ such that $d_1 = 52143$, $d_2 = 43152$, $d_3 = 32541$, and $d_4 = 23451$. During this phase, each destination process d_h, $1 \leq h \leq 4$, in the set D will be labeled by a unique label j, $2 \leq j \leq 5$. So, by applying the Rule (L_1), we have $D_3 = \{d_1, d_2\}$ and

$D_5 = \{d_3, d_4\}$. The distances from s to d_1 and d_2 can be computed using Algorithm 1 and $\mathcal{D}ist(s, d_1) = 2$ and $\mathcal{D}ist(s, d_2) = 4$. So, the representative process of D_3 is d_1, and hence, the label 3 is assigned to the process d_1. Similarly, the distances from s to d_3 and d_4 are $\mathcal{D}ist(s, d_3) = 2$ and $\mathcal{D}ist(s, d_4) = 4$. So, the representative process of D_5 is d_3, and hence labeled by 5. Thus, after applying the rule (L_1), we have $D = \{d_2, d_4\}$ and $DL = [(d_1, 3), (d_3, 5)]$. Then, from the rule (L_2), the processes d_2 and d_4 are labeled by 2 and 4, respectively. Thus, after the labeling phase, the array structure $DL = [(d_1, 3), (d_3, 5), (d_2, 2), (d_4, 4)]$.

From the above, it is clear that all representative processes appear before all other processes in the array DL. In addition, the (i_1)-*processes* appear after the representative processes and before other remainder processes, *i.e.*, (1)-*processes*. In the sequel, we assume that all the processes in DL are identified based on their positions in the array DL. Thus, the process in the position one is denoted by d_1 and the process in the second position is denoted by d_2 and so on. For the sake of simplicity, each element (d_h, j) in DL is denoted by $d_h.j$, where d_h refers to the process at the position h, $1 \leq h \leq n-1$, and j, $2 \leq j \leq n$, is the label assigned to d_h. Moreover, we use the notation $d.j$, when the rank h of the process d in DL is omitted, to refer simply to a process d in DL indexed by a label j.

Algorithm 2. Labeling Process Algorithm

Input $D = \{d_1, d_2, ..., d_{n-1}\}$ be a set of $n-1$ distinct processes in the n-star network S_n.
Output DL is an array of structure of $n-1$ *labeled processes* in the n-star network S_n.
Struct D_s { id, lab }
Variables DL array [1,...,$n-1$] of D_s
Predicates (i_1)-Process$(p) \equiv (p[i] = 1) \wedge (i \neq 1)$
(1)-Process$(p) \equiv (p[1] = 1)$
Function *Representative*$(D_i : Set)$: *Process-ID*
Begin
return$(min_{id}\{p \in D_i :: \mathcal{D}ist(s, p) = min_{q \in D_i}(\{\mathcal{D}ist(s, q)\})\})$
end
Procedure $Affect\text{-}label(d, i, pos : integer)$
Begin
$DL[pos].id \leftarrow d$; $DL[pos].lab \leftarrow i$;
$D \leftarrow D \backslash \{d\}$; $Labels \leftarrow Labels \backslash \{i\}$
$pos \leftarrow pos + 1$;
end

Function $\mathcal{L}abeling(D : (Set)\ \{d_1, d_2 \ldots, d_{n-1}\})$:
(Struct) DL array [1,...,n-1] of D_s)
Variables **Set** $Labels = \{2, \ldots, n\}$
Begin $pos \leftarrow 1$
(L_1) **for each** $i, i \in \{2, 3, \ldots, n\}$ **do**
$D_i \leftarrow \{p \in D :: (i\_1)\text{-}Process(p)\}$
if $(D_i \neq \emptyset)$ **then**
$Affect\text{-}label(Representative(D_i), i, pos)$
end if
end do
(L_2) **if** $(D \neq \emptyset)$ **then**
for each $d \in \{p \in D :: \neg(1)\text{-}Process(p)\}$ **do**
pick up a label $l \in Labels$ **then**
$Affect\text{-}label(d, l, pos)$
end do
end if
(L_3) **if** $(D \neq \emptyset)$ **then**
for each $d \in \{p \in D :: (1)\text{-}Process(p)\}$ **do**
pick up a label $l \in Labels$ **then**
$Affect\text{-}label(d, l, j)$
end do
end if
return(DL)
end

3.2 $\mathcal{O}$ne-$\mathcal{T}$o-$\mathcal{M}$any Node-Disjoint Paths Construction

Upon completion of the labeling phase, each process $d.j$ in DL is assigned a unique label j, $2 \leq j \leq n$, such that j is the position of the symbol 1 in all the permutations (except for at most two) of the processes on the path P_j connecting s to $d.j$. During this second phase, $n-1$ node-disjoint paths, noted by $P_2, P_3, \ldots, P_n$, are constructed from the source s to the destination processes in DL such that each path P_j, $2 \leq j \leq n$, connects the source process s to the destination process $d.j$. In order to carry out this task, we need to solve the following two problems: (i) We need a procedure that

constructs a path from the process s to a destination process $d.j$, $2 \leq j \leq n$, and keeps the symbol 1 in a fixed position j along all the processes on the path P_j, except for at most one process. (ii) All these paths $P_2, P_3, ..., P_n$ must be node-disjoint.

The basic construction (property (i)) is referred to as $elementary\ construction$ and is implemented basically using Function *NextNeighFix1*() (see Algorithm 3, namely the $\mathcal{O}ne$-$\mathcal{T}o$-$\mathcal{O}ne\ \mathcal{F}ix$-1 path construction algorithm) and the $message\ (d, j)$ containing two parameters, the destination d and the position j of the symbol 1 in all the processes on the path P_j (see Actions a_{2_1} and a_3, Algorithm 4). Observe that, under certain circumstances ($i.e.$, a non-elementary construction), where the destination d is said to be $marked$ (Function $Marked()$), Algorithm $\mathcal{O}$ne-$\mathcal{T}$o-$\mathcal{M}$any disjoint paths uses another message containing three parameters (see Actions a_{2_2} and a_4, Algorithm 4). Now, we describe the elementary construction and the purpose of the second one (non-elementary construction) is discussed later. The processes in this construction exchange one type of message containing two parameters: destination d and the fixed position j, $2 \leq j \leq n$, of the symbol 1 along the processes on P_j. Once the elementary construction is started, the source process s initiates the construction of the path P_j from s to $d.j$ by sending the message (d, j) to its successor on P_j (Action a_{2_1}). Subsequently, each process p ($p \neq d$), upon receipt of this message, transmits the message to its successor process on the path P_j (Action a_3). Each process uses the function *NextNeighFix1*() to identify the successor process on the path P_j that kept the symbol 1 in the same fixed position j. The function $NextNeighFix1(d, j)$ contains also two parameters, the destination d and the position j of the symbol 1 in its successor process on the path P_j. This function is implemented by executing one of the following six rules: (r_0), (r_1),..., and (r_5) (see Algorithm 3 and Function $NextNeighFix1()$). The rules are executed in the order they are written. So, if the first rule is not applicable, then we try the next rule and so on. Assuming that the shortest path from s to d is built up to the process $p \neq d$ (initially, $p = s$), then the successor of p is identified as follows. Let x be the first symbol in the permutation of p, then (r_0) This rule is executed one time and only by the source process s, during the initialization phase. So, p is the identity process s (Predicate $Source(p)$), exchange the first symbol 1 with the jth symbol in s. In this rule, the symbol 1 is moved to the desired position j in the destination d.

Algorithm 3. Self-stabilizing $\mathcal{O}ne$-$\mathcal{T}o$-$\mathcal{O}ne\ \mathcal{F}ix$-1 path construction algorithm

Input a source process s and a target process $d.j$ in the n-star network S_n.

Output a Path connecting the processes s and $d.j$.

Function *NextNeighFix*1(d : *Process-ID*, $j \in \{2, \ldots, n\}$) : *Process-ID*
  Predicate
    Source(p) $\equiv$ ($p = s$)
    DirectP(p) $\equiv$ ($\mid Place_p \mid = 1$)
    ByPass1(p) $=$ ($\mid Swap_p \mid \geq 1$)
    ByPass2(p) $\equiv$ (($\mid Diff \mid = 2$) $\wedge$ (($Diff = \{1, i\}$)$\vee$($Diff = \{1, j\}$)))
    ByPass3(p) $\equiv$ (($\mid Diff \mid = 2$) $\wedge$ ($Diff = \{i, j\}$))
    ByPass4(p) $\equiv$ (($\mid Diff \mid = 3$) $\wedge$ ($Diff = \{1, i, j\}$)
  Macro
    $\text{Diff}_p = \{k, 1 \leq k \leq n, :: (p[k] \neq d[k])\}$
    $\text{Place}_p = \{k, 2 \leq k \leq n, :: ((p[1] = d[k]) \wedge (p[k] \neq 1))\}$
    $\text{Swap}_p = \{k, 2 \leq k \leq n, :: ((p[k] \neq d[k]) \wedge (p[k] \notin \{1, d[1], d[j]\}))\}$
  Function *Position*-1(p : *Process-ID*) : *Integer*
    Begin
      Return($k, 1 \leq k \leq n, :: p[k] = 1$)
  end

Begin
  $i \leftarrow$ *Position*-1(d)
  Case of ::
    (r_0) *Source*(p) :: $r \leftarrow j$
    (r_1) *DirectP*(p) :: $r \leftarrow k :: k \in Place_p$
    (r_2) *ByPass1*(p) :: $r \leftarrow min_k(Swap_p)$)
    (r_3) *ByPass2*(p) :: $r \leftarrow k :: (k \in Diff_p \wedge k \neq 1)$
    (r_4) *ByPass3*(p) :: $r \leftarrow i$
    (r_5) *ByPass4*(p) :: $r \leftarrow j$
  end Case of
  Return($\pi[1, r](p)$)
end

(r_1) If there exists a symbol y, $y \neq 1$, in the kth position, $2 \leq k \leq n$, such that x occupies a correct position in d, *i.e.*, $d[k] = x$ (Predicate $DirectP(p)$), then exchange x with y. In this rule, x directly reaches its correct position k in the destination d. However, in order to keep the symbol 1 in a fixed position, y should not be equal to 1, if possible.

(r_2) Otherwise, exchange x with the symbol y, $y \notin \{1, d[1], d[j]\}$, in the kth position such that y does not occupy a correct position in d, *i.e.*, $d[k] \neq y$, if exists (Predicate $ByPass1(p)$). In this rule, we move x to a position k not occupied by a correct symbol, *i.e.*, $p[k](= y) \neq d[k]$. If multiple positions are not occupied by a correct symbol, then choose the symbol with the smallest position. Similarly to the case (r_1), to maintain the symbol 1 in a fixed position, y should not be equal to 1. In addition, to maintain the path P_j as shortest as possible, y should be different than $d[1]$ and $d[j]$.

(r_3) If all the symbols are in correct positions except the first symbol x and the symbol y in the j-th or i-th position (in this case the destination d is an (i_1)-$process$ and the i-th position is the position of the symbol 1 in d) such that $y = d[1]$ (Predicate $ByPass2(p)$), then exchange x with the symbol y. In this situation, we consider two cases. (a) First, $x = d[j]$, in this case d is a (1)-$process$. Then, exchange x with the symbol y, $y = 1$, in the jth position. (b) Second, $x = 1$, in this case d is an (i_1)-$process$. Then, exchange x with the symbol y, $y = d[1]$, in the ith position. In both cases, the successor process of p on P_j is the destination process d_i.

(r_4) If all the symbols are in correct positions except the ith symbol $p[i]$ $(= d[j])$ and the jth symbol $p[j](= d[i] = 1)$ (Predicate $ByPass3(p)$), then exchange x, the first symbol in p, with the symbol y in the ith position.

(r_5) If all the symbols are in correct positions except the first symbol x $(x = d[j])$, the ith symbol $p[i]$ $(= d[1])$, and the jth symbol $p[j]$ $(= d[i] = 1)$ (Predicate $ByPass4(p)$), then exchange x with the symbol y, $y = 1$, in the jth position. In order to illustrate the elementary construction concept, we provide the following example with 8-star system and a destination $d.5$ (example 6).

Example 6. We consider the three following possible cases of the destination process $d.5$: $d.5$ $(= 53241876)$ is a representative process, $d.5$ $(= 13245876)$ is a (1)-$process$, and $d.5$ $(= 23145876)$ is a (i_1)-$process$ (in our case $i = 3$). A path from s to $d.5$ that keeps the symbol 1 at position 5 created by the elementary construction may look as follows. (a) Let $d.5 = 53241876$, $P_5 = (s = 12345678(\xrightarrow{r_0}), 52341678(\xrightarrow{r_2}), 25341678(\xrightarrow{r_1}), 35241678(\xrightarrow{r_1}), 53241678(\xrightarrow{r_2}), 63241578(\xrightarrow{r_1}), 83241576(\xrightarrow{r_1}), 53241876)$. ($b$) Let $d.5 = 13245876$, $P_5 = (s = 12345678(\xrightarrow{r_0}), 52341678(\xrightarrow{r_2}), 25341678(\xrightarrow{r_1}), 35241678(\xrightarrow{r_1}), 53241678(\xrightarrow{r_2}), \quad 63241578(\xrightarrow{r_1}), 83241576(\xrightarrow{r_1}), 53241876(\xrightarrow{r_3}), 13245876)$. ($c$) Let $d.5 = 23145876$, $P_5 = (s = 12345678\ (\xrightarrow{r_0}), 52341678(\xrightarrow{r_2}), 32541678(\xrightarrow{r_1}), 23541678(\xrightarrow{r_2}), 63541278(\xrightarrow{r_1}), 83541276(\xrightarrow{r_1}), 23541876(\xrightarrow{r_4}), 53241876(\xrightarrow{r_5}), 13245876(\xrightarrow{r_3}), 23145876)$.

A (1)-*process associated with* the process $d.j, 2 \leq j \leq n$, such that $d.j$ is an (i_1)-$process$, is the process obtained from $d.j$ by swapping its first symbol with the symbol 1 located at position i (*i.e.*, $\pi[1, i](d.j)$).

Remark 1. From the above elementary construction we deduce the following remarks. (i) If $d.j$ is a representative process of a set D_j, then all the processes of the path P_j

constructed from s to $d.j$ are (j_1)-*processes*, except s (see example 6 case (a)).
(ii) If $d.j$ is a (1)-*process*, then all the processes of the path P_j are (j_1)-*processes*, except the endpoints s and $d.j$ (see example 6 case (b)).
(iii) If $d.j$ is a (i_1)-*process* $(i \neq j)$, then all the processes of the path P_j are (j_1)-*processes*, except the (1)-*process associated with* the process $d.j$ and its endpoints s and $d.j$ (see example 6 case (c)).

Now, we are ready to present the remainder of the $\mathcal{O}ne\text{-}\mathcal{T}o\text{-}\mathcal{M}any$ node-disjoint paths algorithm (*i.e.*, the non-elementary construction). Observe that from Remark 1, if all the processes in DL are representative processes (j_1)-*processes* and/or (1)-*processes*, then the $\mathcal{O}ne\text{-}\mathcal{T}o\text{-}\mathcal{M}any$ node-disjoint paths algorithm is obviously obtained by applying the elementary construction from the source s to each destination process $d.j$, $2 \leq j \leq n$ in DL. Since for each destination process $d.j$, $2 \leq j \leq n$, there exists a distinct position j which is reserved for the symbol 1 for all the processes on the path P_j. So, each path P_j, $2 \leq j \leq n$, connecting s to $d.j$ is node disjoint from the other paths. This is illustrated in the following example (see example 7).

Example 7. Let us consider the set $DL = [(d_1 = 43152, 3), (d_2 = 23451, 5), (d_3 = 14352, 2), (d_4 = 15432, 4)]$, where $d_1.3$ and $d_2.5$ are representative processes, $d_3.2$ and $d_4.4$ are (1)-*processes*. The node disjoint paths are built using the elementary construction as follows. (i) $P_3 = (s = 12345\ (\xrightarrow{r_0}), 32\underline{1}45\ (\xrightarrow{r_1}), 2\underline{3}145\ (\xrightarrow{r_1}), 5314\underline{2}\ (\xrightarrow{r_1}), 4315\underline{2})$. (ii) $P_5 = (s = 12345\ (\xrightarrow{r_0}), 5234\underline{1}\ (\xrightarrow{r_1}), 423\underline{5}1\ (\xrightarrow{r_1}), 32\underline{4}51\ (\xrightarrow{r_1}), 2\underline{3}451)$. (iii) $P_2 = (s = 12345\ (\xrightarrow{r_0}), 2\underline{1}345\ (\xrightarrow{r_1}), 5134\underline{2}\ (\xrightarrow{r_1}), 413\underline{5}2\ (\xrightarrow{r_1}), 1\underline{4}352)$. (iv) $P_4 = (s = 12345\ (\xrightarrow{r_0}), 423\underline{1}5\ (\xrightarrow{r_1}), 32\underline{4}15\ (\xrightarrow{r_2}), 2\underline{3}415\ (\xrightarrow{r_1}), 5341\underline{2}\ (\xrightarrow{r_1}), 3\underline{5}412\ (\xrightarrow{r_1}), 1543\underline{2})$.

A (1)-*process* associated with the (i_1)-*process* $d_h \in DL$, $2 < h \leq n$, (that is not a representative process) is said *marked process* (Function $Marked()$, Algorithm 4) if it is equal to the (1)-*process* associated with another (i_1)-*process* $d_{h'}$ such that $1 \leq h' < h$, *i.e.*, the path to the destination $d_{h'}$ is constructed before the path to the destination d_h. A critical step in applying only the elementary construction to construct the $n-1$ $\mathcal{O}ne\text{-}\mathcal{T}o\text{-}\mathcal{M}any$ node-disjoint paths is the *case* 5 (rule (r_5)) where for a given (i_1)-*process* $d_h \in DL$, $2 < h \leq n$, that is not a representative process of the set D_i, the (1)-*process* associated with d_h is *marked*. So, in this case, there already exists a destination process $d_{h'} \in DL$, $1 \leq h' < h$, such that its (1)-*process* is identical to the (1)-*process* associated with d_h. In this situation, the (1)-*process* associated with d_h already belongs to a previously constructed path, say $P_{j'}$, initiated by s to process $d_{h'}$ before s initiates the construction of the path, say P_j, to d_h. Therefore, the two paths P_j and $P_{j'}$ are not node-disjoint, since they intersect at the (1)-*process* associated with the two processes d_h and $d_{h'}$. We can illustrate this situation by using again the example 3 presented in Sect. 3.1 (example 8).

Algorithm 4. Self-stabilizing $\mathcal{O}ne\text{-}\mathcal{T}o\text{-}\mathcal{M}any$ node-disjoint paths algorithm

Input $D = \{d_1, d_2 \ldots, d_{n-1}\}$ be a set of $n-1$ distinct processes in the n-star network S_n.
Output $n-1$ node-disjoint paths connecting the identity process s and the processes in D.
Predicate (i_1)-$Process(t) \equiv ((\pi[1,i](t) = 1) \wedge (i \neq 1))$
Function (1)-$Process(p : Process\text{-}ID) : Process\text{-}ID$
Return $(\pi[1,k](p) :: 2 \leq k \leq n, p[k] = 1)$
end
Function $Marked(t,h)$: $Boolean$
Return$(\exists d_k \in D :: (2 \leq k < h) \wedge ((1)\text{-}Process(d_k) = t))$
end
Function $UMNeigh(t,h) : Process\text{-}ID$
Return$(\pi[1,k](t) :: (2 \leq k \leq n) \wedge (\pi[1,k](t) \notin \{D, (1)\text{-}Process(t)\}) \wedge \neg Marked((1)\text{-}Process(\pi[1,k](t)), h)))$
end

For source process (identity process) $p = s$

Algorithm $One\text{-}To\text{-}Many(D : set)$
Begin
• (a_1) **Spontaneously**
$DL = \mathcal{L}abeling(D)$
• (a_2) **Timeout[]**
for each process $d_h.j \in DL$ $(h = 1$ **to** $n-1)$ **do**
(a_{2_1}) **if** $((i\_1)\text{-}Process(d_h) \Rightarrow \neg Marked(1\text{-}Process(d_h), h))$ **then**
$Send(d_h.j)$ **to** $NextNeighFix1(d_h, j)$
(a_{2_2}) **else**
$Send(UMNeigh(d_h, h), d_h.j)$ **to**
$NextNeighFix1(UMNeigh(d_h, h), j)$
end if
end do
end

For process $p \neq s$

Algorithm $One\text{-}To\text{-}Many()$
Begin
• (a_3) **Upon Receipt**(d, j)
if $(p \neq d)$ **then**
$Send(d, j)$ **to** $NextNeighFix1(d, j)$
end if
• (a_4) **Upon Receipt**(d', d, j)
if $(p \neq d')$ **then**
$Send(d', d, j)$ **to** $NextNeighFix1(d', j)$
else
$Send(d, j)$ **to** d
end if
end

Example 8. Assume that we have 5-star system and let $DL = [(d_1 = 52143, 3), (d_3 = 32541, 5), (d_2 = 43152, 2), (d_4 = 23451, 4)]$. By Algorithm 4 and the elementary construction, the source s initiates the construction of the paths following this order, P_3, P_5, P_2, and P_4 (see example 3). (i) A path P_3 from s to $d_1 = 52143$ that keeps the symbol 1 at position 3 may look as follows. $P_3 = (s = 12345\ (\xrightarrow{r0}), 32\underline{1}45\ (\xrightarrow{r1}), 5214\underline{3})$. (ii) A path P_5 from s to $d_3 = 32541$ that keeps the symbol 1 at position 5 may look as follows. $P_5 = (s = 12345\ (\xrightarrow{r0}), 5234\underline{1}\ (\xrightarrow{r1}), 32\underline{5}41)$. (iii) A path P_2 from s to $d_2 = 43152$ that keeps the symbol 1 at position 2 may look as follows. $P_2 = (s = 12345(\xrightarrow{r0}), 2\underline{1}345(\xrightarrow{r1}), 5134\underline{2}(\xrightarrow{r1}), 413\underline{5}2(\xrightarrow{r4}), 31\underline{4}52(\xrightarrow{r5}), 1\underline{3}452(\xrightarrow{r3}), 43\underline{1}52)$. (iv) A path P_4 from s to $d_4 = 23451$ that keeps the symbol 1 at position 4 may look as follows. $P_4 = (s = 12345(\xrightarrow{r0}), 423\underline{1}5(\xrightarrow{r1}), 32\underline{4}15(\xrightarrow{r1}), 2\underline{3}415(\xrightarrow{r3}), 5341\underline{2}(\xrightarrow{r5}), 1345\underline{2}\ (\xrightarrow{r3}), 2345\underline{1})$.

Observe that, in the above example, the (1)-$process$ 13452 associated with d_4 is marked, since it is equal to the (1)-$process$ associated with d_2. So, the paths P_2 and P_4 are not node-disjoint paths. To alleviate such a problem, the following scheme is introduced in the $\mathcal{O}$ne-$\mathcal{T}$o-$\mathcal{M}$any disjoint algorithm. During the construction of node-disjoint paths $P_2, \ldots, P_n$ (such that each path P_j, $2 \leq j \leq n$, connects the source s to the destination process $d_h.j$, $1 \leq h \leq n-1$), each time a marked process associated with a (i_1)-$process$ $d_h.j$ is identified (Function $Marked(d,h)$), the path P_j from s to d_h is constructed as follows ($i.e.$, the non-elementary construction). First, we need to identify a neighbour $d'.j$ of the process d_h such that the process $d'.j$ is not in DL and the (1)-$process$ associated with $d'.j$ is not marked. Note that each marked process is a (1)-$process$ contained in a previously constructed paths. Then, the node-disjoint path P_j from s to $d_h.j$ is constructed in the following two steps. First, the path P_j is constructed from s to $d'.j$. Then, the construction continues from $d'.j$ to $d_h.j$. This is

implemented by the introduction of the message (d', d, j) containing three parameters d', d, and j. Hence, the (1)-$process$ associated with $d_h.j$ is not included in the path P_j, whereas it includes the not marked (1)-$process$ associated with $d'.j$. We can illustrate this approach by using again the example 8 (see example 9).

Example 9. From example 8, the paths P_2 and P_4 are not node-disjoint paths, since the 1-$process$ associated with $d.2$ and $d.4$ are identical and equal to 13452. Let $d'.4 = 43251$ be the selected neighbour of the process $d.4$ and the (1)-$process$ associated with $d'.4$ is 13254. Observe that $d'.4 = 43251$ is not in DL and the (1)-$process = 13254$ associated with $d'.4$ is not marked. A path from s to $d.4 = 23451$ that keeps the symbol 1 at position 4 created by the above approach may look as follows. First we create the path from s to $d'.4 = 43251$ as follows: $s = 12345 \xrightarrow{r_0} 42\underline{3}15 \xrightarrow{r_2} 2\underline{4}315 \xrightarrow{r_1} 34\underline{2}15 \xrightarrow{r_1} 4\underline{3}215 \xrightarrow{r_4} 5\underline{3}214 \xrightarrow{r_4} 132\underline{5}4 \xrightarrow{r_5} 4325\underline{1}$. Then, the construction continues from $d'.4$ to $d.4$ as follows: $4325\underline{1} \xrightarrow{r_3} 2345\underline{1}$

The $\mathcal{O}$ne-$\mathcal{T}$o-$\mathcal{M}$any node disjoint paths algorithm works as follows. After the labeling phase (Action a_1), each process in DL is assigned a unique label j, $2 \leq j \leq n$. The source process s initiates the construction of the $n-1$ node-disjoint paths $P_2, P_3, \ldots, P_n$ such that each path P_j, $2 \leq j \leq n$, connects the source process s to the destination $d_h.j \in DL$, $1 \leq h \leq n-1$ (Action a_2). Then, we need to consider two cases.
(a) In the simplest case, when the destination process $d_h.j$ is in one of the three following situations ($i.e.$, the (1)-$process$ associated to d_h is not marked): either it is a representative process of a set D_i or, a (1)-$process$, an (i_1)-$process$ and its associated (1)-$process$ is not marked. In this case, the construction is elementary and is handled by the message (d_h, j) containing two parameters, d_h and j. So, once the source process s initiates this construction, it transmits the (d_h, j) message to its successor on the path P_j using function *NextNeighFix1*(). Analogously, when a process p $(p \neq d_h)$ receives this message, it transmits the message to its successor on the path P_j. This is repeated until the destination d_h is reached. This is implemented using Actions (a_{2_1}) and (a_3).
(b) However, when $d_h.j$ is an (i_1)-$process$ and the (1)-$process$ associated with the process $d_h.j$ is marked, then, as explained before (non elementary construction), we first identify a neighbour $d'.j$ of the process $d_h.j$ such that $d'.j$ is not in DL and the (1)-$process$ associated with $d'.j$ is not marked. This is implemented using Function *UMNeigh*(). In this case, the construction is handled by using the message (d', d_h, j) containing three parameters: the first and the second destinations $d'.j$ and $d_h.j$ to reach, respectively, and the position j of the symbol 1 along all the processes on the path P_j. So, the source process s initiates the construction of the path P_j from s to d' by sending the message (d', d, j) to its successor on P_j (Action (a_{2_2}). Then, subsequently, upon a process p receives this message, we need to consider two cases (Action a_4). (i) If the process p is the first destination d', meaning that the first destination d' is reached, then p sends the message (d_h, j) with the second destination d_h to reach to its successor; (ii) Otherwise, the first destination d' is still not reached, then p transmits the message to its successor. Each process, upon receipt of a message, uses the function *NextNeighFix1*() to identify the successor process on a path P_j that kept the symbol 1 in the same position j.

4 Proof of Correctness

We will show that Algorithm 4 constructs $n-1$ one-to-many node-disjoint paths. From Algorithm 4, each destination process $d.j$ in DL such that $d.j$ is a representative process of a set D_j, (1)-$process$, or a (i_1)-$process$ and its associated 1-$process$ is not marked (Function $\neg Marked()$), the elementary construction is sufficient to construct a path P_j from s to $d.j$ (Actions a_{2_1} and a_3). However, if $d.j$ is a (i_1)-$process$ and its associated (1)-$process$ is marked, then we need to identify a neighbour process of $d.j$ (say $d'.j$) such that $d'.j$ is not in DL and its associated (1)-$process$ is not marked (Function $UMNeigh()$). Then, the path P_j is constructed in two steps: first a path is constructed from s to $d'.j$, then from $d'.j$ to $d.j$ (Action a_{2_2} and a_4). From Algorithm 4, we can claim the following lemma.

Lemma 1. *Let $d.j \in DL$ a destination process, then Algorithm 4 constructs a path P_j from s to $d.j$ with the following properties.*
(i) If $d.j$ is a $representative$ process, then all the processes on P_j keeps the symbol 1 at position j, except s.
(ii) If $d.j$ is a (1)-process, then all the processes on P_j keeps the symbol 1 at position j, except its endpoints s and $d.j$.
(iii) If $d.j$ is a (i_1)-process ($i \neq j$), then all the processes on P_j keeps the symbol 1 at position j, except the (1)-process associated with $d.j$, and at most two (i_1)-processes ($i.e$, $d.j$ and one of its neighbour $d'.j$ in $\mathcal{N}_j$), and s.

In the sequel, we need the following definition. We say that a process sequence $(u_1, u_2, ..., u_s)$ in the n-star S_n is a *simple cycle* if all processes are distinct and (u_s, u_1), (u_i, u_{i+1}), $1 \leq i \leq s-1$, are all edges in S_n. From [4], we can claim the following result.

Lemma 2. *There is no simple cycle of the length less than six in the n-star S_n.*

Let $\mathcal{P}revP_j$ denote the set of all the paths built before the path P_j. We introduce the definition of the sets $\mathcal{N}_j$ and (1)-$\mathcal{N}_j$ associated with each destination process $d.j \in DL$ to facilitate the proof. Observe that each process $d.j$ has $n-1$ neighbours, one is a (1)-$process$ and the others are (i_1)-$processes$. Let the $n-1$ neighbours of $d.j$ be $d_2 = \pi[1,2](d.j), d_3 = \pi[1,3](d.j), d_4 = \pi[1,4](d.j), ..., d_n = \pi[1,n](d.j)$ where $d_i = \pi[1,i](d.j)$ is the (1)-$process$ associated with $d.j$ and $\pi[1,k](d.j), 2 \leq k \leq n$ and $k \neq i$, are the (i_1)-$processes$ neighbours of $d.j$. Let $\mathcal{N}_j = \{d_k, 2 \leq k \leq n$ and $k \neq i\}$ be the set of all the (i_1)-$processes$ neighbours of $d.j$. Let (1)-$\mathcal{N}_j = \{(1)$-$d_k, 2 \leq k \leq n\}$ be the set of all (1)-$processes$ associated with processes in $\{d.j\} \cup \mathcal{N}_j$ such that (1)-d_k is the (1)-$process$ associated with d_k.

Lemma 3. *Let $d.j \in DL$ be a (i_1)-process, but not the representative process, of set D_i and let the sets $\mathcal{N}_j$ and (1)-$\mathcal{N}_j$ be defined as above. Then, each path P_t of the paths $\mathcal{P}revP_j$ contains, at most, one process in $\mathcal{N}_j$ and, at most, one process in (1)-$\mathcal{N}_j$.*

Proof. Let $P_t \in \mathcal{P}revP_j$ be the path that connects the source process s to the destination process $d.t$ such that t is the position reserved for the symbol 1 for all the processes on P_t (except for maximum one).

1. If $d.t$ ($t \neq j$) is a (1)-*process*, then by Lemma 1 case (ii), all internal processes of P_t are (t_1)-*processes* and $t \neq j$. Similarly, by Lemmas 1 case (iii), the path P_j contains only (j_1)-*processes*, one (1)-*process*, and at most two (i_1)-*processes* and $t \neq i$, since the position i is reserved for the representative process of the set D_i. Thus, the path P_t contains no process in the set $\mathcal{N}_j$ and contains at most one process, *i.e.*, the process $d.t$, in (1)-$\mathcal{N}_j$.

2. If $d.t$ is a (k_1)-*process* for $k \neq t$ and $k \neq i$, then by Lemmas 1 case (iii), the path P_t contains only (t_1)-*processes*, one (1)-*process*, and at most two (k_1)-*processes* and $t \neq i$. Since $k \neq j$ (because there exists a representative process for each subset D_k and D_j) and $t \neq j$ (from the labeling process), the path P_t contains no process in the set $\mathcal{N}_j$ and contains at most one process in (1)-$\mathcal{N}_j$, *i.e.*, the (1)-*process* associated with $d.t$.

3. Assume that the process $d.t$ is the representative process of the set D_i. Then, by Lemma 1 case (i), all processes except the first process s on the path P_t are (i_1)-*processes*. Thus, the path P_t contains no process in the set (1)-$\mathcal{N}_j$. To prove that the path P_t contains, at most, one process in $\mathcal{N}_j$, assume the contrary, that P_t contains two (i_1)-*processes* in the set $\mathcal{N}_j$ (*i.e.*, $d.t$ and a neighbour of $d.t$). But, the path P_t cannot contain the process $d.t$, otherwise, $\mathcal{D}ist(s, d.j) < \mathcal{D}ist(s, d.t)$, contradicts the selection of the representative process node $d.t$ in the set D_i. Thus, P_t contains, at most, one process in $\mathcal{N}_j$, the neighbour of $d.t$.

4. Finally, suppose that the process $d.t$ is a (i_1)-*process*, but not the representative process, of the set D_i. We need to consider two cases.

a. The path P_t contains only one (i_1)-process that is the process $d.t$ and one (1)-*process* that is the (1)-*process* associated with $d.t$, and the rest of the processes on P_t are all (t_1)-*processes*, $t \neq j$. So, the path P_t contains at most one process in the set $\mathcal{N}_j$ (*i.e.*, the process $d.t$) and contains, at most, one process in (1)-$\mathcal{N}_j$ (*i.e.*, the (1)-*process* associated with $d.t$).

b. The path P_t contains two adjacent (i_1)-*processes* that is the process $d.t$ and a not marked neighbour $d_k \in \mathcal{N}_t$ of $d.t$, and one (1)-*process* that is the (1)-*process* associated with d_k (*i.e.*, (1)-d_k), and the rest of the processes on P_t are all (t_1)-*processes*, $t \neq j$. In this case, not both of $d.t$ and d_k can be in the set $\mathcal{N}_j$; otherwise, the star system S_n would have a simple cycle ($d.t$, d_k, $d.j$), a contradiction with Lemma 2. So, the path P_t contains at most one process in the set $\mathcal{N}_j$ (*i.e.*, the process $d.t$ or d_k) and contains, at most, one process in (1)-$\mathcal{N}_j$ (*i.e.*, the (1)-*process* associated with d_k).

Lemma 4. *Let $d.j$ be a (i_1)-process, but not the representative process, of set D_i and let the sets $\mathcal{N}_j$ and (1)-$\mathcal{N}_j$ be defined as above. Then, for each path P_t of the paths $\mathcal{P}revP_j$, if the path P_t contains a process d_k in the set $\mathcal{N}_j$ and a (1)-process (1)-d_l in the set (1)-$\mathcal{N}_j$, then the (1)-d_l process must be the (1)-process associated with the process d_k.*

Proof. From Lemma 3, the path P_t contains a process in $\mathcal{N}_j$ and a process in (1)-$\mathcal{N}_j$ only in the case 4.*b*. The process $d.t$ is a (i_1)-*process*, but not the representative process, of the set D_i. The path P_t contains two adjacent (i_1)-*processes* that is the process $d.t$ and a not marked neighbour $d_k \in \mathcal{N}_t$ of $d.t$, and one (1)-*process* that is the (1)-*process* associated with d_k (*i.e.*,(1)-d_k), and the rest of the processes on P_t are all

(t_1)-$processes$, $t \neq j$. The path P_t contains at most one process in the set $\mathcal{N}_j$ ($i.e.$, the process $d.t$ or d_k) and contains, at most, one process in (1)-$\mathcal{N}_j$ ($i.e.$, the (1)-$process$ associated with d_k). If the (1)-$process$ (1)-d_k is in the set (1)-$\mathcal{N}_j$, then the process $d.t$ cannot be in the set $\mathcal{N}_j$–otherwise, the process d_k is not in the set $\mathcal{N}_j$ and the process sequence $(d_t, d_k, (1)\text{-}d_k, d'.j, d.j)$ would form a simple cycle of length 5 in the n-star ($d'.j$ is is a neighbour $d.j$ on P_j).

Lemma 5. *Let $d.j$ be a (i_1)-process, but not the representative process, of set D_i. If the (1)-process associated with the process $d.j$ is marked, then there is a neighbour d_k of $d.j$ such that the process d_k and the (1)-process associated with d_k are not marked.*

Proof. By Lemmas 3 and 4, each previously constructed path $P_t \in \mathcal{P}rev P_j$ contains, at most, one process in $\mathcal{N}_j$ (say, d_k), and, at most, one (1)-$process$ in (1)-$\mathcal{N}_j$, the (1)-$process$ associated with d_k. Let $|\mathcal{P}rev P_j| = x \leq n-2$. So, there exists x marked processes $\mathcal{N}_j$ and x (1)-$processes$ marked processes in the set (1)-$\mathcal{N}_j$. Thus, there exists $(n-x-1) \geq 1$ unmarked processes in the set $\mathcal{N}_j$ along with their $(n-x-1)$ unmarked in the set (1)-$\mathcal{N}_j$, hence the result.

By Algorithms 2, 4 and Lemmas 1, 5, we can show this result by induction on the number of the node-disjoint paths previously constructed, $i.e.$, $|\mathcal{P}rev P_j|$, such that $0 \leq |\mathcal{P}rev P_j| \leq n-2$.

Lemma 6. *Let P_j $(2 \leq j \leq n)$ be the path constructed from s to $d.j$ by Algorithms 4, then the path P_j is node-disjoint with all paths $\mathcal{P}rev P_j$ previously constructed by the algorithm.*

Theorem 1. *Given a source s and a set D=$\{d_1, d_2, ..., d_{n-1}\}$ of $n-1$ distinct processes in the n-star, Algorithm 4 is a self-stabilizing one-to-many node-disjoint paths algorithm and construct $n-1$ one-to-many node-disjoint paths in at most $O(n^2)$ rounds, such that each path connects s to a process in D.*

Proof. From Algorithm 4 and Action (a_1), the source process s initiates the labeling process infinitely often. The labeling process of the set $D = \{d_1, d_2, ..., d_{n-1}\}$ uses the distance computing algorithm (Algorithm 2). In the worst situation, we have $D_2, D_3, ..., D_{\lceil n/2 \rceil} \subseteq D$ subsets such that each subset D_i $(i \in 2, 3, ..., \lceil n/2 \rceil)$ contains two (i_1)-$processes$. So, in order to find the representative process of each subset D_i (see rule (L_1), Algorithm 2), we need to compute the distance from s to each process in D_i, $i.e.$, $2(\lceil n/2 \rceil - 1)$ processes. Since, each distance computing needs at most $\lceil 3(n-1)/2 \rceil$ rounds, where $\lceil 3(n-1)/2 \rceil$ is the diameter of the S_n, so the labling phases requieres $2(\lceil n/2 \rceil - 1)(\lceil 3(n-1)/2 \rceil)$ rounds ($O(n^2)$ rounds). Then, from action (a_2), the source s also initiates the construction of the $n-1$ node disjoint paths infinitely often. Then, from actions a_3 and a_4 each process participates in the construction of the paths infinitely often. From Lemma 4, the $n-1$ node-disjoint paths are constructed in at most $\lceil 3(n-1)/2 \rceil$ rounds. Thus, the stabilization time is $O(n^2)$ rounds.

5 Conclusion and Future Work

In this paper, we presented the first distributed self-stabilizing algorithm for finding $one\text{-}to\text{-}many$ node-disjoint paths algorithm in message passing model. Two paths in

a network are said to be node disjoint if they do not share any nodes except for the endpoints. Our algorithm works on n-star networks S_n. Due to being self-stabilizing, it tolerates transient faults, and does not require initial configuration. The stabilization time of our algorithm is $O(n^2)$ rounds. In this work, we merely provided an algorithm to find one-to-many disjoint paths in n-star networks. Devising distributed and self-stabilizing algorithms for hypecube is open problem that we consider as future work.

References

1. Akers, S.B., Krishnamurthy, B.: Group graphs as interconnection networks. In: 14th International Conference on Fault Tolerant Computing, Trans. pp. 422–427 (1984)
2. Akers, S.B., Krishnamurthy, B.: A group theoretic model for symmetric interconnection networks. IEEE Trans. Comput. **4**(38), 555–565 (1989)
3. Akers, S.B., Harel, D., Krishnamurthy, B.: The star graph: an attractive alternative to the n-cube. In: Proceedings International Conference on Parallel Processing, St. Charles, Illinois, pp. 393–400 (1987)
4. Chen, C.C.: Combinatorial and algebraic methods in star and bruijn networks. Department of Computer Science, Texas A and M university, Ph.D. dissertion (1995)
5. Chen, C.C., Chen, J.: Nearly optimal one-to-many parallel routing in star networks. IEEE Trans. Parallel Distributed Syst. **8**(12), 1196–1202 (1997)
6. Cheng, E., Gao, S., Qiu, K., Shen, Z.: On disjoint shortest paths routing on the hypercube. In: Du, D.-Z., Hu, X., Pardalos, P.M. (eds.) COCOA 2009. LNCS, vol. 5573, pp. 375–383. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02026-1_35
7. Dijkstra, E.W.: Self-stabilizing in spite of distributed control. Commun. Assoc. Comput. Mach. **17**(11), 643–644 (1974)
8. Dietzfelbinger, M., Madhavapeddy, S., Sudborough, I.H.: Three disjoint path paradigms in star networks. In: Proceedings of the Third IEEE Symposium on Parallel and Distributed Processing, pp. 400–406 (1991)
9. Gao, S., Hsu, D.F.: Short containers in Cayley graphs. Discrete Appl. Math. **157**, 1354–1363 (2009)
10. Gu, Q.P., Peng, S.: Node-to-set and set-to-set cluster fault tolerant routing in hypercubes. Parallel Comput. **24**, 1245–1261 (1998)
11. Hadid, R., Karaata, M.H.: An adaptive stabilizing algorithm for finding all disjoint paths in anonymous mesh networks. Comput. Commun. **32**(5), 858–866 (2009)
12. Hadid, R., Karaata, M.H., Villain, V.: A stabilizing algorithm for finding two node-disjoint paths in arbitrary networks. Int. J. Found. Comput. Sci. **28**(4), 411–435 (2017)
13. Hsu, D.F.: A graph theoretical study of transmission delay and fault tolerance. In: Proceedings 4th International Conference on Parallel and Distributed Computing and Systems, pp. 20–24 (1991)
14. Hsu, D.F.: On container width and length in graphs, groups, and networks. IEICE Trans. Fundam. Electron. Commun. Comput. Sci. **E77-A**(4), 668–680 (1994)
15. Hsu, C.C.: A genetic algorithm for maximum edge-disjoint paths problem and its extension to routing and wavelength assignment problem. Ph.D. thesis, NC State University (2013)
16. Karaata, M.H., Hadid, R.: Briefannouncement: a stabilizing algorithm for finding two disjoint paths in arbitrary networks. In: Stabilization, Safety, and Security of Distributed Systems, pp. 789–790 (2009)
17. Lai, C.N.: One-to-Many disjoint paths in the hypercube and folded hypercube. Ph.D. thesis, Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan (2001)

18. Lai, C.N.: Two conditions for reducing the maximal length of node-disjoint paths in hypercubes. Theoret. Comput. Sci. **418**, 82–91 (2012)
19. Lai, C.N.: Optimal construction of all shortest node-disjoint paths in hypercubes with applications. IEEE Trans. Parallel Distrib. Syst. **23**(6), 1129–1134 (2012)
20. Latifi, S.: On the fault-diameter of the star graph. Inf. Process. Lett. **46**, 143–150 (1993)
21. Latifi, S., Ko, H., Srimani, P.K.: Node-to-set vertex disjoint paths in hypercube networks. Technical report CS-98-107, Colorado State University (1998)
22. Lengauer, T.: Combinatorial Algorithms for Integrated Circuit Layout. Wiley, New York (1990)
23. Ma, C., et al.: p-MDP structure against multi-failures in high-degree node based optical networks. In: Communications and Networking in China, pp. 756–760 (2013)
24. Murthy, S., Souzaand, R.J.D., Varaprasad, G.: Digital signature-based secure node disjoint multipath routing protocol for wireless sensor networks. IEEE Sensors J. **12**(10), 2941–2949 (2012)
25. Qiu, K.: An efficient disjoint shortest paths routing algorithm for the the hypercube. In: Proceedings of the 14th IEEE International Conference on Parallel and Distributed Systems (ICPADS 2008), IEEE Computer Society Press, Vol. 4, pp. 371–384 (1999)
26. Rabin, M.A.: Efficient dispersal of information for security, load balancing, and fault tolerance. J. ACM **36**, 335–348 (1989)
27. Saad, Y., Shultz, M.H.: Topological properties of hypercubes. IEEE Trans. Comput. **37**, 867–872 (1988)
28. Sinanoglu, O., Karaata, M.H., AlBdaiwi, B.: An inherently stabilizing algorithm for node-to-node routing over all shortest node-disjoint paths in hypercube networks. IEEE Trans. Comput. **59**(7), 995–999 (2010)
29. Sur, S., Srimani, P.K.: Topological properties of star grahs. Comput. Math. Applic. **25**(12), 87–98 (1993)