Efficient connectivity testing of hypercubic networks with faults *

Tomáš Dvořák, Jiří Fink[†], Petr Gregor[‡], Václav Koubek
Tomasz Radzik

Abstract

Given a connected graph G and a set F of faulty vertices of G, let $G - F$ be the graph obtained from G by deletion of all vertices of F and edges incident with them. Is there an algorithm, whose running time may be bounded by a polynomial function of $|F|$ and $\log |V(G)|$, which decides whether $G - F$ is still connected? Even though the answer to this question is negative in general, we describe an algorithm which resolves this problem for the n-dimensional hypercube in time $O(|F|n^3)$. Furthermore, we sketch a more general algorithm that is efficient for graph classes with good vertex expansion properties.

1 Introduction

A study of interconnection networks, originally initiated by particular applications in telephone and computer networks, has become fairly pervasive in many different areas in the recent decade. In a whole avenue of problems that arise in the course of network design, a good deal of attention has been paid to the aspect of reliability: If some nodes of the network become overloaded or unavailable, can the network still preserve its functionality?

If interconnection networks are modeled as simple undirected graphs, our problem may be formulated as follows: Suppose we are given a class

*Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic,{dvorak@ksvi,fink@kam,gregor@ktiml,koubek@ktiml}.mff.cuni.cz, and Department of Computer Science, King's College London, United Kingdom,Tomasz.Radzik@kcl.ac.uk

†The Institute for Theoretical Computer Science (ITI) is supported by project 1M0545 of the Czech Ministry of Education.

‡Partially supported by the Czech Science Foundation Grant 201/08/P298.

$\mathcal{G}$ of graphs such that each graph $G \in \mathcal{G}$ has a property $\mathcal{P}$. Note that to describe an arbitrary vertex of G, we need a string of length $\Omega(\log |V(G)|)$ in the worst case. Let $n_G : V(G) \to 2^{V(G)}$ be an oracle which for a given vertex $v \in V(G)$ returns the set $N(v)$ of all neighbors of v in G in time $O(|N(v)| \cdot \log |V(G)|)$.

Problem 1. *Is there an algorithm which*

- *given an oracle n_G for a graph $G \in \mathcal{G}$ and a set F of faulty vertices of G,*
- *decides whether the graph $G - F$, obtained from G by deletion of all vertices of F and edges incident with them, still possesses the property $\mathcal{P}$,*
- *whose running time is bounded by a polynomial function of $|F|$ and $\log |V(G)|$?*

The requirement on the time complexity is motivated by practical considerations. Recall that to describe an arbitrary input F, a string of length $|F|\Omega(\log |V(G)|)$ is needed. It is plausible to presume that the number of nodes which may become faulty at the same time would be just a fraction of the total number of nodes of the network. A typical instance of this problem may be a network topology modeled by the graph of the n-dimensional hypercube with 2^n vertices, while the number of faults is bounded by $O(n^k)$ for some natural number k [5, 7]. In this case it would be useful to design an algorithm for Problem 1, running in time proportional to the length of the input F, possibly searching only some local neighborhood of F in G, rather than exploring the whole graph $G - F$.

A natural requirement imposed on each reasonable interconnection network is its connectivity. In this paper we therefore study an instance of Problem 1 where property $\mathcal{P}$ equals connectivity. To the best of our knowledge, no results on this problem have been reported previously.

It should be noted that although graph connectivity is a textbook example of an algorithmic problem that may be solved in linear time [3], our variant is more involved. In particular, we claim that if G may be an arbitrary connected graph, an algorithm testing the connectivity of $G - F$ in time $|F|^k$ for some natural number k does not exist. Indeed, suppose that $G_{x,y}$ and $G_{x,z}$ are two connected graphs containing distinct vertices $x \neq y$ and $x \neq z$, respectively. Let G_1 be the graph obtained from $G_{x,y}$ and $G_{x,z}$ by gluing together vertex x of $G_{x,y}$ with vertex x of $G_{x,z}$, and G_2

be the graph obtained from G_1 by adding edge yz. In order to verify the connectivity of $G_1 - \{x\}$ or $G_2 - \{x\}$, it is necessary to check the presence of edge yz, since only this edge distinguishes the connected graph $G_2 - \{x\}$ from disconnected $G_1 - \{x\}$. Since the choice of y and z was quite arbitrary, it follows that any algorithm that correctly decides on the connectivity of $G - F$ must necessarily read all edges of this graph. It follows that its running time is bounded from below by the size of the input graph, which need not be necessarily a polynomial in $|F|$ and $\log |V(G)|$.

This argument shows that it is necessary to restrict class $\mathcal{G}$ to some proper subclass of connected graphs. In this paper, we resolve our problem for the class of hypercubes, which has served for decades as a popular topology of interconnection networks for parallel or distributed computing [10]

It is worth mentioning that a fairly special instance of Problem 1 for $\mathcal{G}$ being the class of hypercubes and property $\mathcal{P}$ being the existence of

(i) Hamiltonian cycles and paths [4],

(ii) long cycles and paths [5, 7],

has been studied previously. There are positive results for a special case when the number of faults is bounded by a certain linear (i) or quadratic (ii) function of n. On the other hand, when the number of faults is not limited, the problems are NP-hard [1, 6].

The main results of this paper is an algorithm which verifies the connectivity of the n-dimensional hypercube with f faults in time $O(fn^3)$. We also describe a more general algorithm based on vertex-expansion properties that for the class of hypercubes works in time $O(f^2n^{2.5})$. The rest of the paper is laid out as follows. After introducing some necessary concepts and notations, we start with vertex-expansion approach in Section 3. In Section 4 we study walk transformations. This is our main technical tool, applied in Section 5 to derive a theorem relating connectivity of faulty hypercube with that of certain local neighborhood of the set of faults. Based on these theoretical results, in Section 6 we describe an algorithm for connectivity testing and analyze its time complexity. The paper is concluded with some open problems and directions for further research.

2 Preliminaries

The concepts used in this paper but undefined below may be found e. g. in [3]. In the rest of this text, n always denotes a positive integer while $[n]$ stands for the set $\{1, 2, \ldots, n\}$.

Vertex and edge sets of a graph G are denoted by $V(G)$ and $E(G)$, respectively. Given a set $V \subseteq V(G)$ let $G[V]$ denote the subgraph of G induced by V while $G - V$ stands for $G[V(G) \setminus V]$. The *distance* between vertices u, v in G is denoted by $d_G(u, v)$, the subscript being omitted if no ambiguity may arise. A *square* of the graph G, denoted by G^2, is the graph on vertices of G and edges between every two distinct vertices that are at distance at most two in G. Given a vertex u, an edge vw and sets $S, T \subseteq V(G)$, we define

$$\begin{aligned} d(u, S) &= \min\{d(u, v) \mid v \in S\}, \\ d(S, T) &= \min\{d(u, T) \mid u \in S\}, \\ d(u, vw) &= d(u, \{v, w\}), \\ N(u) &= \{v \in V(G) \mid d(u, v) = 1\}, \\ N(S) &= \{v \in V(G) \mid d(v, S) = 1\}. \end{aligned}$$

The *n-dimensional hypercube* Q_n is a graph with all binary vectors of length n as vertices, an edge joining two vertices whenever they differ in a single coordinate. For two vertices u, v of Q_n let $u \triangle v$ be the set of coordinates in which u and v differ. Note that $|u \triangle v| = d(u, v)$. The *direction* of an edge uv of Q_n is the integer $i \in [n]$ in which u and v differ; that is, $u \triangle v = \{i\}$.

3 Expansion approach

In this section we describe an algorithm for testing vertex-deleted connectivity which works efficiently for graph classes with good vertex expansion.

The set $N(S)$ contaning neighbors of vertices from $S \subseteq V(S)$ that are not in S is called the *boundary* of S. The graph G is said to have *vertex expansion* ε if $|N(S)| \geq \varepsilon \cdot |S|$ for every $S \subseteq V(G)$ with $|S| \leq |V(G)|/2$. Note that nonzero expansion implies connectedness; otherwise, a component of at most half of the vertices would have empty boundary.

Theorem 1. *Let $(G_n)_{n \in \mathbb{N}}$ be a sequence of graphs G_n with vertex expansion $\varepsilon_n > 0$ and maximal degree Δ_n. There is an algorithm that for input $n \in \mathbb{N}$,*

$F \subseteq V(G_n)$, $|V(G)|$, and $\varepsilon_n > 0$ tests the connectivity of $G_n - F$ in time

$$O\left(\frac{|F|^2 \cdot \Delta_n^2 \cdot \log(|V(G_n)|)}{\varepsilon_n}\right).$$

A sketch. A component of $G_n - F$ induced by vertices $S \subseteq V(G_n) \setminus F$ is said to be

- *major* if $|S| > |V(G_n)|/2$;
- *small* if $|S| \leq |F|/\varepsilon_n$.

Obviously, there is at most one major component. If $|F| > |V(G_n)|\varepsilon_n/2$, we can afford to run a standard search algorithm in $G_n - F$. Otherwise, it follows from vertex expansion of G_n that every component of $G_n - F$ is either major or small.

The key idea is that we can afford searching through small components completely. Furthermore, if we find more than $|F|/\varepsilon_n$ vertices in the same component, we know that we are in the major component (and thus we can stop our search).

Hence, the algorithm works as follows. We start searching $G_n - F$ from each (non-faulty) neighbor v of a faulty vertex u. If we find a component of more than $|F|/\varepsilon_n$ vertices, we stop the search from v with a remark that there exists a major component. If we have found a complete small component, we report that the graph $G_n - F$ is disconnected. Otherwise, we continue the search until we check all non-faulty neighbors v of all faulty vertices u. In this case, we report that the graph $G_n - F$ is connected.

The time complexity is obtained as follows. There are at most $|F| \cdot \Delta_n$ non-faulty neighbors of faulty vertices. From each of them we search for at most $|F|/\varepsilon_n$ vertices. For every vertex found we ask oracle for its neighbors, and each query takes $O(\Delta_n \cdot \log(|V(G_n)|)$ time. $\square$ $\square$

It follows from classical results of Harper [8] on isoperimetric problems that the hypercube Q_n has a vertex expansion $\frac{c}{\sqrt{n}}$ for some constant c.

Corollary 1. *There is an algorithm for testing connectivity of $Q_n - F$ that runs in $O(|F|^2 \cdot n^{2.5})$ time.*

4 Transformations of walks in hypercubes

In this section we introduce a useful concept of transformations of one walk to another walk of the hypercube. This is where the structure of the hypercube plays its role.

A *walk* in a simple graph G is a sequence $W = (v_0, v_1, \ldots, v_k)$ of vertices in G such that v_i and v_{i+1} are adjacent for all $0 \leq i < k$. If W starts with the vertex u and ends with the vertex v, we say that W is an *uv-walk*.

Let $W = (v_0, v_1, \ldots, v_k)$ be a walk in Q_n. Let d_i be the direction of the edge between v_{i-1} and v_i for every $i \in [k]$. Then the sequence $(d_1, d_2, \ldots, d_k)$ is called the *transitional sequence* of the walk W. For a sequence τ over $[n]$ and $i \in [n]$ let $\#(\tau, i)$ be the number of occurrences of i in τ. It is easy to see that a sequence τ over $[n]$ is a transitional sequence of some uv-walk in Q_n if and only if

$$u \triangle v = \{i \in [n];\ \#(\tau, i) \text{ is odd}\}. \tag{4.1}$$

Thus, we may identify uv-walks in Q_n with sequences over $[n]$ satisfying (4.1). We use both representations of an uv-walk as a sequence of vertices and as its transitional sequence, depending on what is more convenient.

Let τ be a transitional sequence of an uv-walk W. Consider the following three operations on τ:

$$\begin{aligned}
&\mathsf{swap}(\tau_1, i, j, \tau_2) = (\tau_1, j, i, \tau_2) && \text{for } \tau = (\tau_1, i, j, \tau_2),\\
&\mathsf{insert}_i(\tau_1, \tau_2) = (\tau_1, i, i, \tau_2) && \text{for } \tau = (\tau_1, \tau_2),\\
&\mathsf{delete}(\tau_1, i, i, \tau_2) = (\tau_1, \tau_2) && \text{for } \tau = (\tau_1, i, i, \tau_2),
\end{aligned}$$

where τ_1, τ_2 are contiguous subsequences of τ and $i, j \in [n]$. Since these operations preserve (4.1), their results are also transitional sequences of some uv-walk.

We say that two uv-walks σ and τ in Q_n are *equivalent* if $\#(\sigma, i) = \#(\tau, i)$ for all $i \in [n]$. Note that the operation swap transforms an uv-walk to an equivalent uv-walk. Conversely, the following proposition holds.

Proposition 1. *For every two equivalent uv-walks σ and τ in Q_n, there is a sequence of* swap*s that transforms σ into τ.*

Proof. Since σ and τ are equivalent, they have the same length k. Moreover, there is a permutation $f : [k] \to [k]$ such that $\sigma(i) = \tau(f(i))$ for all $i \in [k]$. An arbitrary decomposition of f into consecutive transpositions gives us a sequence of swaps that transforms σ into τ. □ □

Let $W = (v_0, v_1, \dots, v_k)$ be a walk in Q_n with a transitional sequence $\tau = (d_1, d_2, \dots, d_k)$. We say that $\mathsf{insert}_i(\tau_1, \tau_2)$ on τ is performed in the vertex v_i where $0 \le i \le k$ if $\tau_1 = (d_1, d_2, \dots, d_i)$ and $\tau_2 = (d_{i+1}, d_{i+2}, \dots, d_k)$.

Proposition 2. *For every two uv-walks σ and τ, there are two sequences of* inserts *that transform σ into σ' and τ into τ', respectively, such that σ' and τ' are equivalent. Moreover, these* inserts *can be performed in arbitrary vertices.*

Proof. For every direction $i \in [n]$ we perform the operations insert_i on σ if $\#(\sigma, i) < \#(\tau, i)$, or on τ if $\#(\sigma, i) > \#(\tau, i)$ until we obtain $\#(\sigma, i) = \#(\tau, i)$. These inserts can be performed on any position. □ □

Since delete is an inverse of insert, we obtain the following corollary.

Corollary 2. *For every two uv-walks σ and τ in Q_n there is a sequence of* inserts*,* swaps *and* deletes *(in this order) that turns σ into τ.*

5 Local connectivity

For a given set F of vertices in Q_n we define a subgraph $G(F) = (A \cup B \cup F, E)$ of Q_n by

$$A = N(F), \quad B = N(A) \setminus F, \quad E = \{uv \in E(Q_n);\, u \in A \cup F\}.$$

That is, $G(F)$ is the subgraph of Q_n on all vertices at distance at most 2 from F and with all edges at distance at most 1 from F. Our aim in this section is to show that if $G(F)$ is connected and $G(F) - F$ is disconnected, then $Q_n - F$ is also disconnected. Note that if $Q_n^2[F]$ is connected, then $G(F)$ is connected as well.

Let W be a walk in Q_n and let u be a vertex on W. We say that u is a *port* on W if $u \in A$ and exactly one of his neighbors on W is in F. Note that if u is a port on W and not an endvertex, then his second neighbor on W is in $A \cup B$. Furthermore, since u may have several occurrences on the walk W, the notion of ports is defined with respect to a particular occurrence of u on W, and not the vertex u itself.

For a connected component C of $G(F) - F$ let $p(C, W)$ denote the number of ports on the walk W from the component C. First, we show that swap performed on W preserves the parity of $p(C, W)$.

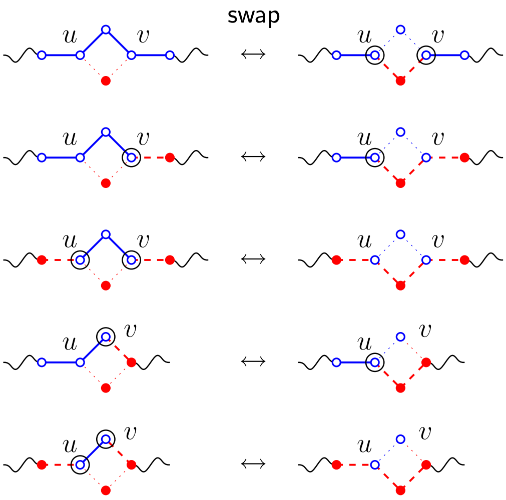

Figure 1: All possible swaps that change ports.

Lemma 1. *Let W_2 be a walk in Q_n obtained from a walk W_1 by a single* swap*, and let $F \subseteq V(Q_n)$. For every component C of $G(F)-F$, the numbers $p(C, W_1)$ and $p(C, W_2)$ differ by 0 or 2.*

Proof. See Figure 1 for all configurations of swaps that change ports. The vertices of F are full (red), the vertices of $A \cup B$ are empty (blue), the ports are circled. The edges of walks W_1, W_2 that are incident to F are dashed (red), the edges of W from $G(F) - F$ are full (blue).

Note that in each case, the ports change on the vertices u and v. Since u and v are connected by edges of $G(F) - F$, they are in the same component C of $G(F) - F$. In the first, third and last case, the numbers of ports of C changes by 2, whereas in the second and fourth case, it remains unchanged. □ □

Corollary 3. *For every $F \subseteq V(Q_n)$, every component C of $G(F) - F$, and every equivalent walks W_1 and W_2 in Q_n, the parity of $p(C, W_1)$ and $p(C, W_2)$ is the same.*

Now we show that global connectivity implies local connectivity. That is, disconnected $Q_n - F$ can be recognized locally on $G(F) - F$.

Lemma 2. *Let $F \subseteq V(Q_n)$ be such that $G(F)$ is connected. If $Q_n - F$ is connected, then $G(F) - F$ is also connected.*

Proof. Suppose for a contradiction that there are vertices $u, v \in A \cup B = V(G(F) - F)$ that are connected in $Q_n - F$ by a walk P but are disconnected in $G(F) - F$. Clearly, the walk P contains some vertex x that is not from A; otherwise, P is in $G(F) - F$.

Let C_u and C_v denote the components of $G(F) - F$ containing the vertices u and v, respectively. Since $G(F)$ is connected, there is an uv-walk R in $G(F)$. As u and v are disconnected in $G(F) - F$, the walk R contains some vertex $y \in F$, and an odd number of ports from each component C_u and C_v.

By Proposition 2, the walks P and R can be transformed by inserts to walks P' and R' in Q_n, respectively, such that P' and R' are equivalent. Moreover, inserts on P and on R can be performed at the vertices x and y, respectively. It follows that the sets of ports on P and R do not change by these transformations. In particular, $p(C, P') = p(C, P)$ and $p(C, R') = p(C, R)$ for every component C of $G(F) - F$.

However, from Corollary 3 it follows that $p(C_u, P)$ and $p(C_v, P)$ have odd parity. Hence, the walk P contains some port, and consequently, some vertex of F. This is a contradiction with the assumption that P is a walk in $Q_n - F$. □ □

Lemma 3. *Let F be a set of vertices of Q_n such that $G(C) - C$ is connected for every component C of $Q_n^2[F]$. Then $Q_n - F$ is connected as well.*

Proof. Let $u, v \in V(Q_n) \setminus F$ and P be an arbitrary uv-walk in Q_n. If P contains no vertex from F, we are done. Otherwise it contains a subwalk $S = (x, y_1, \ldots, y_m, z)$ whose all vertices except x and z are in F. Then $y_1, \ldots, y_m$ belong to the same component C of $Q_n^2[F]$. By our assumption, $G(C) - C$ contains an xz-walk T. Replacing the subwalk S of P with T, we obtain an uv-walk which contains less vertices from F than P. Repeating this process for every subwalk of P of the described type, we finally obtain a uv-walk in $Q_n - F$, and the desired conclusion follows. □ □

Theorem 2. *Let $F \subseteq V(Q_n)$. The graph $Q_n - F$ is connected if and only if $G(C) - C$ is connected for every component C of $Q_n^2[F]$.*

Proof. Let $Q_n - F$ be connected and C be a component of $Q_n^2[F]$. By Lemma 2 it suffices to prove that $G(C)$ is connected in order to prove that $G(C) - C$ is connected.

Let u, v be vertices of $G(C)$ and our aim is to prove that u and v belong into the same component of $G(C)$. There exist vertices $u', v' \in C$ such that $d(u, u') \leq 2$ and $d(v, v') \leq 2$. Since C is a component of $Q_n^2[F]$, there exists a sequence $u' = w_1, w_2, \ldots, w_k = v'$ of vertices of C such that $d(w_i, w_{i+1}) \leq 2$ for every $1 \leq i < k$. Therefore, vertices $w_1, w_2, \ldots, w_k$ belong to the same component of $G(C)$. Consequently, u and v are in the same component of $G(C)$ as well.

The other implication follows from Lemma 3. □ □

6 Algorithm

In this section we apply Theorem 2 to design an algorithm for testing the connectivity of $Q_n - F$. To accomplish this task, we employ the following data structures.

List F of faulty vertices of Q_n.

Disjoint-set data structure D [3, Chapter 22] with operations

- Make(v, D) creates a singleton set $\{v\}$,
- Find(v, D) returns a pointer to the set containing v,
- Union(u, v, D) unites the sets containing u and v,

whose amortized time complexity may be loosely bounded by $O(\log m)$, provided that Make(v, D) was executed m times. We use D to detect the connectivity of $G(C)$ where C is a component of $Q_n^2[F]$.

Binary trie T [9, Section 6.3] which stores information about some vertices of Q_n. Each vertex of Q_n stored in T is represented by a leaf of T, which we denote by v_T. Moreover, v_T includes the following additional information:

- a pointer to v in the disjoin-set data structure D
- a boolean variable indicating whether v is *healthy* or *faulty*,
- a boolean variable *visited* indicating that v has been visited and $v \in N(F) \cup F$.

Note that we mark as visited only faulty vertices and their neighbors, even though our algorithm inspects also vertices at distance 2 from F. Given a vertex v of Q_n,

- Insert(v, T) inserts v into T and returns v_T,
- Retrieve(v, T) returns v_T or NIL if it does not exist.

Both operations require $O(n)$ time.

Given a list F of faulty vertices of Q_n, Algorithm 6.1 finds all components of $Q_n^2[F]$ using a depth–first search (DFS), described as Procedure 6.2. For every $f \in F$, all vertices v at distance at most two from f are visited. If v is faulty, DFS is applied recursively on v, which ensures that the algorithm indeed finds the components of $Q_n^2[F]$. If v is healthy, then v is inserted into the disjoint-set data structure D. Furthermore, sets of D containing vertices u and v are united for every edge uv at distance one from f. In that way, after a call to DFS(f) (line 8 of Algorithm 6.1) is completed, disjoint sets of D represent components of the graph $G(C) - C$ for the component C of $Q_n^2[F]$ containing f. This verifies the condition of Theorem 2.

The trie T is used to store information about the vertices visited during the search. Note that due to the time and space constraints, T cannot contain each of 2^n vertices of Q_n. Faulty vertices are inserted into T during the initialization of Algorithm 6.1. Healthy vertices of $G(C)$ for a component C of $Q_n^2[F]$ are inserted into T during the DFS of C. When the whole component of $Q_n^2[F]$ is found, all healthy vertices are removed from T .

Removing all healthy vertices from trie T (line 10 of Algorithm 6.1) may be implemented using the depth-first search of T. Since the total number of calls to Insert($\cdot, T$) is bounded by $O(|F|n^2)$, the total time complexity of this clean-up is $O(|F|n^3)$.

To analyze the time complexity of our algorithm, observe that DFS(f) is called exactly once for each faulty vertex $f \in F$. Next, considering the code of Procedure 6.2, the outer for-loop (line 1) is executed for every neighbor of f, while the inner for-loop (line 8) is executed for some vertices at distance two from f. Therefore, the total number of the inner loop executions is bounded by $|F|n^2$. The time critical operation are Retrieve($\cdot, T$) and Insert($\cdot, T$), requiring $O(n)$ time for each call as noted above. Hence, the total running time of the algorithm is bounded by $O(|F|n^3)$.

Theorem 3. *Given an integer $n \geq 1$ and a set of vertices F of Q_n, the problem whether the graph $Q_n - F$ is connected can be decided in $O(n^3|F|)$ time.*

Algorithm 6.1: CONNECTIVITY(n, F)

Input: Positive integer n, a list F of faulty vertices of Q_n
Output: "$Q_n - F$ is connected" or "$Q_n - F$ is disconnected"

$T \leftarrow$ empty trie
foreach $f \in F$ **do** $f_T \leftarrow$ INSERT(f, T) ; mark f_T as faulty and non-visited
foreach $f \in F$ **do**
  $f_T \leftarrow$ RETRIEVE(f, T)
  if f_T *is not visited* **then**
    mark f_T as visited
    $D \leftarrow$ empty data structure for disjoint sets
    DFS(f) `// DFS of the component` C `of` $Q_n^2[F]$ `containing` f
    if D *contains more than one set* **then return** "$Q_n - F$ is disconnected"
    remove all healthy vertices from T and clean-up data structure D
return "$Q_n - F$ is connected".

Procedure 6.2: DFS(f)

Input: Faulty vertex f of Q_n `//` f `belongs to a component` C `of` $Q_n^2[F]$

Data: Binary trie T, disjoint-set data structure D

foreach $u \in N(f)$ **do**
  $u_T \leftarrow \textsc{Retrieve}(u, T)$
  if $u_T = \mathrm{NIL}$ **then**
    $u_T \leftarrow \textsc{Insert}(u, T)$; mark u_T as healthy and non-visited; $\textsc{Make}(u, D)$
  if u_T *is not visited* **then**
    mark u_T as visited
    if u_T *is healthy* **then**
      foreach $v \in N(u)$ **do**
        $v_T \leftarrow \textsc{Retrieve}(v, T)$
        if $v_T = \mathrm{NIL}$ **then**
          $v_T \leftarrow \textsc{Insert}(v, T)$; mark v_T as healthy and non-visited; $\textsc{Make}(v, D)$
        if v_T *is healthy* **then**
          if $\textsc{Find}(u, D) \neq \textsc{Find}(v, D)$ **then** $\textsc{Union}(u, v, D)$
          `// edge` uv `belongs to` $G(C)$
        else if v_T *is not visited* **then** mark v_T as visited; DFS(v)
          `// faulty vertex` v `belongs to` C
    else DFS(u)   `// faulty vertex` u `belongs to` C

7 Concluding remarks

In this paper we have described two algorithms for testing the connectivity of the n-dimensional hypercube with f faulty vertices. The (more general) expansion algorithm runs in $O(f^2 n^{2.5})$ time, whereas the local connectivity algorithm runs in $O(fn^3)$ time.

It is worth pointing out the following corollary: If $|F| = O(n^k)$ for some $k \in \mathbb{N}$, the size of $Q_n - F$ is exponential in n, but our algorithm still tests the connectivity of $Q_n - F$ in time which is polynomial in n.

We believe that it would be interesting to find other classes of graphs for which the connectivity instance of Problem 1 has a positive solution. Natural candidates are other hypercubic networks [10] whose fault-tolerance has been investigated previously [2, 11]. In some networks, transformations of walks are possible if we allow swaps on larger cycles (of bounded-size), e.g. hexagonal grids, n-dimensional torus C_d^n with fixed d, planar graphs with faces of bounded size. We think that the approach described in Section 5 works for such networks as well.

Another question is what other properties can be efficiently tested in vertex-deleted graphs. A biconnectivity can be defined such that a graph $G = (V, E)$ is biconnected if $G - \{x\}$ is connected for every vertex x. If F is a set of faulty vertices of a hypercube Q_n then every vertex x of distance at least 2 from F has a connected neighborhood of distance 2 in $Q_n - F$. Thus it suffices to verify connectedness for vertices from $A \cup B$. Hence there exists an algorithm deciding a biconnectivity for $Q_n - F$ which requires $O(|F|^2 \cdot n^5)$ time. An analogous idea can work for multidimensional meshes and also for multiconnectivity.

References

[1] M. Y. Chan, S-J. Lee, *On the existence of Hamiltonian circuits in faulty hypercubes*, SIAM J. Discrete Math. **4** (1991), 511–527.

[2] Y.-C. Chen, Y.-Z. Huang, L.-H. Hsu, and J. J. M. Tan, *A family of Hamiltonian and Hamiltonian connected graphs with fault tolerance.* J. Supercomput. **54** (2010), 229-238.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 2001.

[4] T. Dvořák and P. Gregor, *Partitions of faulty hypercubes into paths with prescribed endvertices*. SIAM J. Discrete Math. **22** (2008), 1448-1461.

[5] T. Dvořák, V. Koubek, *Long paths in hypercubes with a quadratic number of faults*, Inf. Sci. **179** (2009), 3763–3771.

[6] T. Dvořák, V. Koubek, *Computational complexity of long paths and cycles in faulty hypercubes*, Theor. Comput. Sci. **411** (2010), 3774–3786.

[7] J. Fink, P. Gregor, *Long paths and cycles in hypercubes with faulty vertices*, Inf. Sci. **179** (2009), 3634–3644.

[8] L. H. Harper, *Optimal Numberings and Isoperimetric Problems on Graphs*, J. Comb. Theory **1** (1966), 385-393.

[9] D. E. Knuth,*The Art of Computer Programming, Volume III: Sorting and Searching*, 2nd ed., Addison-Wesley, 1998.

[10] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. Morgan Kaufmann, San Mateo, CA 1992.

[11] J.-H. Park, H.-C. Kim and H.-S. Lim *Many-to-Many Disjoint Path Covers in the Presence of Faulty Elements*. IEEE Trans. Comput. **58** (2009), 528–540.