Effective Approaches for Constructing Fault-Tolerant Communication Trees in Hypercubes*

by

Po-Jen Chuang★, Shih-Yuan Chen, and Juei-Tang Chen

Department of Electrical Engineering
Tamkang University
Tamsui, Taipei Hsien
Taiwan 25137, R. O. C.
Tel: (+886)2-26215656 Ext. 2615
Fax: (+886)2-26216755 or 26221565
E-mail: pjchuang@ee.tku.edu.tw

---

*to be submitted to *Workshop on Computer Systems*

★the contact author

Effective Approaches for Constructing Fault-Tolerant Communication Trees in Hypercubes

Abstract

A communication tree is a binomial tree embedded in a hypercube, whose communication direction is from the leaves to the root. If a problem to be solved can be divided into independent subproblems, each subproblem can first be solved by one of the hypercube processors and all subresults can then be merged into the final results through tree communication. For this purpose, we propose using two random search techniques, the genetic algorithm (GA) and the simulated annealing (SA) approaches, to construct fault-tolerant communication trees with the lowest data transmission time. Experimental evaluation shows that with reasonably low search time, our proposed GA and SA approaches are able to find more desirable communication trees (i.e., trees with less data transmission time) than the minimal cost approach. We also introduce a distributed approach which, under the aid of two algorithms, is able to effectively reduce search time for the GA and SA approaches due to parallel search for communication subtrees in disjoint subcubes.

Key Words

Fault-tolerant communication trees, hypercubes, genetic algorithms, simulated annealing, data transmission time, search time.

1. Introduction

A communication tree is a binomial tree embedded in a hypercube, whose communication direction is from the leaves up to the root. If a problem to be solved in a hypercube can be divided into independent subproblems, each subproblem can first be solved by one of the hypercube processors and all subresults can then be merged into the final results through tree communication, that is, from the leaves to the root. The number of communication trees for a hypercube can be large. To obtain a communication tree, a sink (the node as the receiver of the final results) and the order of dimensions (to merge the subresults) should be determined in the first place because different sinks and dimension orders lead to different communication trees. Thus in an n-dimensional hypercube, there will be a total of $n! \times 2^n$ communication trees, each with n communication stages and $2^n - 1$ links.

As the number of communication trees in a hypercube can be large, it is therefore desirable to find a fault-free tree (i.e., a tree with no link failures), if possible, for utilization. (Note that only link failures are assumed to exist in the hypercube system where a faulty node is treated as a node with all links to its neighbors being faulty.) To find a fault-free tree in a faulty hypercube is nevertheless quite intricate because all $n! \times 2^n$ possibilities (trees) and $2^n - 1$ used links in each tree need to be checked. If a fault-free communication tree does not exist, it becomes essential to locate a tree with as few faulty links as possible and to reroute messages dynamically according to the fault pattern so that data transmission time (the time for data to be collected to the sink) can be held down as low as possible.

The *exhaustive search* approach can be used to search for an optimal communication tree, i.e., a tree with the fewest link failures. By checking all possible sinks and dimension orders one by one, the approach is able to find an optimal communication tree. The operation is nevertheless very inefficient as it has to check all $n! \times 2^n$ possibilities and $2^n - 1$ used links in a tree. Time complexity will be as much as $O(n! \times 2^n) \times O(2^n - 1) = O(n! \times 4^n)$. When the dimension is high,

the possibilities can be too large to handle.

In case a located optimal communication tree is not fault-free, it is necessary to reroute messages in such a way that data transmission time can be kept to the least. This is also the main goal for constructing communication trees. A *minimal cost* approach [1] has been designed to find fault-tolerant communication trees in hypercubes to meet this purpose. To find a communication tree with as few link failures as possible, its algorithm runs as follows. A node is first selected as the sink and the dimension order is determined according to the fault pattern. Then check from the selected sink node (the root) to the leaves sequentially, with as *low cost* (as few link failures passed) as possible. Similar to the exhaustive search approach, a communication tree so constructed may not be optimal in terms of data transmission time. Meanwhile, as the minimal cost approach does not allow the adjacent links of the sink to be faulty, the number of faulty links in a hypercube must be limited under 2^{n-1} [1].

For improvement, we propose to use some random search techniques, such as the genetic algorithm (GA) [2,3] and the simulated annealing (SA) [4] approaches, to find such a purposed communication tree. Simulation results show that both the GA and SA approaches are able to get better communication trees (in terms of data transmission time) with reasonably low time complexity. To further cut down such complexity, a distributed approach is also introduced. The distributed approach first partitions an n-dimensional hypercube into several disjoint k-dimensional subcubes. The search for communication subtrees with the lowest data transmission time is then simultaneously activated in these k-dimensional subcubes by using the two random search techniques. When the objective communication subtrees are found, they can be easily combined to form a final communication tree in the n-dimensional hypercube. Due to parallel search for communication subtrees in disjoint subcubes, time complexity for attaining the final communication tree in the hypercube can be effectively reduced. To partition the n-cube into the disjoint k-subcubes, a fault-free $(n-k)$-subcube of possibly the highest dimension should be found in the first place. For this purpose, two algorithms, one by bit comparison (a *top-down* approach)

and the other by dimension screening (a *bottom-up* approach), are presented to find the maximal fault-free subcube in a faulty hypercube.

The rest of this paper is organized as follows. Section 2 gives useful background. Section 3 presents our proposed approaches using the genetic algorithm and the simulated annealing. Experimental performance evaluation and comparison among various approaches is given in Section 4. Section 5 presents a distributed approach to trim down search time complexity. Two algorithms able to find the fault-free subcube of the highest dimension for the approach are also provided in this section. Section 6 Concludes the paper.

2. Background

An n-dimensional hypercube, briefed as an n-cube, comprises 2^n nodes, each with n adjacent links serving as communication channels directly to n neighbors. Nodes in an n-cube are numbered from 0 to 2^n-1, by n-bit binary numbers (x_{n-1} ... x_i ...x_0), as their addresses. For convenience, bit i (x_i) is referred to as dimension i of the binary representation of an address. A 4-cube is illustrated in Fig. 1. A node and any of its neighbors are exactly one bit different in their addresses. Two nodes with their address bits being different in only a dimension (say, j) are called "dimension-j neighbors" to each other. For example, in Fig. 1, nodes 0100 and 0110 are dimension-1 neighbors where the link between them is denoted as 01-0 (with dashed bit in bit 1). A k-dimensional subcube in an n-cube, briefed as a k-subcube, comprises 2^k nodes such that all nodes are exactly n-k bits identical in their addresses, with the rest k bits being *don't care*. For example, nodes 1001, 1011, 1101, and 1111 form a 2-subcube in the 4-cube (denoted as 1**1 — 2 identical bits 1's and 2 don't care bits *'s).

A tree communication scheme has been introduced to solve a problem that can be divided into independent subproblems [5]. Each subproblem is first solved by one of the hypercube processors and all subresults are then merged into the final results through tree communication. The communication tree used in [5] for a 4-cube with 4 communication *stages* is shown in Fig. 2. The

arrow lines represent directions of the data flow. The node at the starting point is a sending node and the one at the arrowhead is a receiving node. For instance, nodes 0001 and 0000 form a sender-receiver pair that sends and receives data at stage 0. After receiving data, the receiving node performs computation on the received data and its own data. Through $log_2 N$ ($N = 16$ in this case) stages of operation, the final results will be gathered in node 0000. Such a tree communication scheme, being able to merge the results of independent subproblems into the final results for an original problem, can be applied to various types of computation. An n-level binomial tree BT_n can be constructed recursively by connecting the roots of two BT_{n-1}'s and assigning one of them as the root of BT_n. The fault-free nodes and links involved in a broadcasting session also form a binomial tree with reversed communication directions, i.e., from the root down to the leaves.

There are different ways to generate new communication trees. With the receiver of the last stage being defined as the sink, a communication tree can be determined by selecting a node as the sink or by deciding on a dimension for the messages at a stage to traverse. Thus we have two ways to generate a new tree: Choosing another node as the sink or changing the order of dimensions, and totally there will be $n! \times 2^n$ feasible communication trees for an n-cube. A simple communication tree, with node 0^n as the sink and (0, 1, 2, ..., n-1) as the dimension order, can function efficiently and correctly in a fault-free system. But when hardware failures, such as faulty links, exist in the system, the tree communication scheme becomes defective. To find a fault-free communication tree in a faulty hypercube is indeed quite intricate since it has to check all $n! \times 2^n$ possibilities (trees) and also the $2^n - 1$ links used in each tree. Time complexity will be as much as $O(n! \times 2^n) \times O(2^n - 1) = O(n! \times 4^n)$. Besides, if a fault-free communication tree does not exist, it is necessary to find a communication tree with possibly the fewest faulty links and to reroute messages dynamically according to the fault pattern so that data transmission time can be trimmed down to the least.

3. Our Proposed Approaches

It is clear that retaining the shortest data transmission time is the main goal for constructing communication trees. As the communication trees found by the exhaustive search approach and the minimal cost approach are not necessarily the best in terms of data transmission time, we therefore propose using the genetic algorithm and the simulated annealing approaches to find more desirable communication trees, i.e., trees with possibly the lowest data transmission time. The two random search techniques are to be introduced below.

3.1 The Genetic Algorithm Approach

The genetic algorithm (GA) is an adaptive search technique, with distinctive ability to explore a large searching space. It provides an alternative to traditional optimization techniques by using directed random searches to locate optimal or near optimal solutions of complex problems and is rooted in the mechanisms of evolution and natural genetics. It operates on a pool of chromosomes which represent candidate solutions to the problem under investigation. Chromosomes are selected following "survival of the fittest" and are passed on to the next generation in a process called "reproduction". An objective function is supplied and used to weigh the fitness values of the chromosomes. Reproduction is realized by such genetic operators as selection, crossover and mutation to generate new points in the search space.

To employ the genetic algorithm to generate an optimal fault-tolerant communication tree in the hypercube, we first choose a node with the minimal number of faulty adjacent links as the sink and encode all dimensions into a string. (One possible string in an 8-cube, for example, is "20146357".) Then some strings (assume the number — the so-called ***population size*** — is η) are randomly generated as the initial population and the three genetic operators (selection, crossover and mutation) are followed to locate an excellent communication tree. That is, from the initial randomly generated strings (dimension orders of trees) to the formation of the final desired

(optimal or near optimal) communication tree, the search is all conducted by using the genetic algorithm:

1. Selection:

Three strings from the pool of strings (population) are arbitrarily selected and the best string (the one with the minimal data transmission time) is selected into the next generation. A new generation can be produced by repeating this process η times.

2. Crossover:

Two strings are arbitrarily chosen from the population generated in the above selection step. The exchange is carried out as follows. First, find the crossover points. (For instance, if the two chosen strings are 012435768 and 014352678, the crossover points will be positions 2 and 5, assuming the leftmost position is counted as position 0.) Produce two new strings by exchanging the dimensions of the two original strings between the two crossover points. (Following the above example, we get two new strings 014352768 and 012435678). Then select the best string (from the above 4 strings) into the next generation. Repeat the process η times to produce a new generation.

3. Mutation:

Select (in order from string 1 to string η) a string, say Str_i, from the population generated at the crossover step and decide whether to change Str_i or not with a probability of Pr_m. If Str_i is to be changed, randomly choose two positions (say, p_1 and p_2) and exchange the two dimensions at p_1 and p_2 in Str_i to get a new string; otherwise, maintain the string in the population.

Note that in the selection and crossover processes above, data transmission time is used as the fitness value to select the strings (chromosomes) into the next generation. Data transmission time has been defined as the time for data to be collected to the sink, following the dimension order of the communication tree. Assume that the data generated and to be collected from each node is 1

unit and the time for 1 unit of data to be transmitted through a fault-free link is counted as 1 unit in data transmission time. When the data to be transmitted from a node (say, node x) through a certain dimension (say, dimension j) at a stage encounter a faulty link, they will bypass the faulty link and be transmitted by way of all possible neighbors of node x. That is, to balance the load, node x divides data messages into equal partitions and sends one partition to one neighbor if the link inbetween and the dimension-j link from the neighbor are fault-free, as calculated in [1]. The incurred extra transmission time is then added into the needed transmission time of the data.

Take the 3-cube in Fig. 3(a) as an example. Assume there are 4 faulty links -00, 01-, 10-, and 1-1 in the cube and node 001 is chosen as the sink. As shown in the figure, there is 1 unit of data generated and to be collected from each node before transmission. The steps to calculate the data transmission time for string 012 (i.e., the dimension order of a tree for a 3-stage data collection) is given below.

Step 1: According to dimension order 012, the data are transmitted through dimension 0 at stage 0, i.e., from 000 to 001, 010 to 011, 110 to 111 and 100 to 101. Since dimension-0 links 01- and 10- are faulty, the data will bypass them and be transmitted by way of fault-free neighbors 000 and 110 respectively. The 1 unit of data generated in node 010 is then divided into 2 partitions which are respectively transmitted to 000 and 110, and the 1 unit of data generated in node 100 will be transmitted to 110. Transmitting these data to the 2 neighbors takes only 1 unit of time due to parallel transmission. Now node 000 will possess 1.5 units of data (1 unit generated plus 0.5 unit received) while node 110 possess 2.5 units of data (1 unit generated plus 1.5 units received), as demonstrated in Fig. 3(b).

Step 2: Because transmitting the data through dimension 0 from node 000 to 001 and from 110 to 111 takes respectively 1.5 units and 2.5 units of time, the data transmission time at this step will be 2.5 units. The receiving nodes 001 and 111 will now each possess 2.5 units and 3.5 units of data, as Fig. 3(c) shows.

Step 3: To transmit data through dimension 1 at stage 1 along the tree, it is necessary to bypass faulty link 1-1 and to move forward by way of neighbor 011. Data transmission time for data to go from node 111 to 011 will thus be 3.5 units. Fig. 3(d) shows that node 011 contains 4.5 units of data after this step.

Step 4: 4.5 units of time is needed at this step to transmit data through dimension 1 from node 011 to 001. Node 001 now possesses 7 units of data, as shown in Fig. 3(e).

Step 5: From Fig. 3(f), we see that transmitting data through dimension 2 at stage 2 from node 101 to 001 takes 1 unit of time. Now node 001 finally collects all 8 units of data and the data transmission time for string 012 is thus 1 + 2.5 + 3.5 + 4.5 + 1 = 12.5 units in total.

Based on the above steps, the possible minimal data transmission time (*DT*) for an *n*-cube can be derived as $2^n - 1 = (2^0+2^1+2^2+ \ldots +2^{n-1})$ units. (For example, in an 8-cube, the possible minimal *DT* is $2^8 - 1 = 255$.) Two conditions to terminate the algorithm are set as follows:

1. When the best possible solution (the solution with $DT = 2^n - 1$) is found, terminate the algorithm.
2. When DT_i is larger than $0.999 \times DT_{i-1}$, terminate the algorithm. (DT_i is the data transmission time of the best solution obtained from every 5 generations where *i* denotes the sequence of the 5-generation groups. For example, DT_1 results from the first 5 generations, DT_2 from the second 5 generations, and so on.)

3.2 The Simulated Annealing Approach

The simulated annealing (SA) approach is considered in this paper as another search technique to form the communication tree. The SA approach, another alternative to traditional optimization techniques, is basically an iterative random search procedure with adaptive moves to locate optimal or near optimal solutions of complex problems. As the name indicates, it needs an *annealing*

schedule of the temperatures besides a random generator of "moves" and an objective function. By permitting "uphill moves" under the control of probabilistic criterion (a Boltzmann machine-like mechanism), the temperature is able to keep the algorithm from getting stuck. With the higher the temperatures, the larger the probability to do "uphill moves", it tends to avoid the first local minima encountered. The approach has been successfully applied in different combinatorial optimizations, such as the Travel Salesman Problem [4].

The SA approach randomly generates one initial solution which then generates a new solution based on the neighborhood structure. The two solutions will compete by using the Boltzmann machine-like mechanism. In the process of minimization, if the objective function value of the new solution is lower than that of the initial one, the new solution is selected. If the new solution has a higher value, it can still be selected under some probability which is usually assumed to result from the Boltzmann probability distribution function of the objective function value difference between the two "competing" solutions. The selected solution will generate another new solution and the competing process is repeated again. The iterative process continues until convergence or for a specified length of times.

To employ the SA approach to generate an optimal communication tree in the hypercube, we let the solution be encoded in the same way as the chromosome in the GA approach (that is, the dimension-encoded string). Each dimension number in the string is called an ***element*** of the encoded solution. The objective function is defined to calculate data transmission time needed to collect data to the sink through the communication tree as illustrated in Section 3.1, therefore its value ought to be minimized. The objective function considered is denoted by $\rho(x)$ for solution x.

The iterative process of the SA approach in searching a desired communication tree is given in the following.

(1) Initialization:

Initialize the iteration count and the temperature. Generate one initial solution randomly.

Set the initial solution the selected solution (x).

(2) Iterative steps:

(a) Generation: Generate a new solution (x') from the selected solution (x) based on the neighborhood structure — that is, randomly choose two elements from the selected encoded solution and exchange them to generate a new solution. (For instance, if the initial solution in an 8-cube is 20146357, a generated new solution may be 20546317, obtained by exchanging 1 and 5 in the string.) Calculate the difference between the objective function values (i.e., the data transmission time) of the new and the selected solutions, say $\triangle\rho = \rho(x') - \rho(x)$. If the objective function value of the new solution is not higher than that of the selected one (that is, when $\triangle\rho \leqq 0$), the new solution becomes the selected solution. Otherwise, the new solution is selected with the probability $exp(-\triangle\rho / T) > r$, where T is the temperature at the iteration, r is a random number uniformly distributed between 0 and 1. Note that the above probability results from the Boltzmann probability distribution function to permit "uphill moves".

(b) Cooling (lowering the temperature to reduce the probability of "uphill moves")

(c) Convergence check (terminal check according to the same conditions for the GA approach)

4. Experimental Evaluation

Extensive simulation has been carried out to collect two performance measures of interest, ***the search time*** (for finding an objective communication tree) and ***the solution quality*** (based on the data transmission time of the found communication tree — less data transmission time indicates better solution quality) to evaluate performance of the exhaustive (EX) approach, the minimal cost (MC) approach and our proposed GA and SA approaches. As defined before, data transmission time is the time for data to be collected to the sink and in the presence of faults, it is calculated as demonstrated in Section 3.1.

Simulation is conducted in hypercubes with different dimensions (8, 9 and 10) and different percentages of faulty links (10%, 20% and 30%). Link failures are assumed to distribute evenly in the hypercubes and to be permanent before task initialization, i.e., no new failures will occur at the run-time. A total of 10 randomly generated hypercubes (each with a fixed number of faulty links) are used to evaluate the performance of the above approaches in finding the objective communication trees. Each approach is run ten times in each of the 10 faulty hypercubes, and the results are calculated to yield an averaged value. The 10 averaged values obtained from the 10 hypercubes are then averaged into a final result. (The population size and the probability of mutation for the GA approach are randomly chosen to be 20 and 0.2. Note that the choices of these values would not affect the resulting communication tree.)

As mentioned, the data generated and to be collected from each node is 1 unit and the time for 1 unit of data in a node to be transmitted through a fault-free link to another node is counted as 1 unit in data transmission time. Thus the possible minimal data transmission time for an ***n***-cube is $\xi = 2^n - 1$. In the presence of faults, however, the incurred extra amount of transmission time due to rerouting of messages (mentioned in Section 3.1) must be properly added. For more significant illustration, *additional data transmission time* (*data transmission time* $-\ \xi$) for the four approaches are also collected for comparison. It is collected from different hypercube dimensions and numbers of faulty links, and is depicted in Fig. 4. As Figs. 4(a), 4(b) and 4(c) demonstrate, additional data transmission time for our proposed GA and SA approaches is constantly less than that for the MC approach. The difference becomes even more apparent when the numbers of faulty links and the dimensions of the hypercubes increase. For instance, additional data transmission time for the MC, GA and SA approaches is indeed quite close in hypercubes of dimensions 8 and 9 with 10% of faulty links, as shown in Fig. 4(a) (where the GA and SA approaches display only slightly lower values). But when faulty links increase to 30%, additional data transmission time for the GA and SA approaches drops apparently (see Fig. 4(c)). It can also be observed that when the dimensions of the hypercubes grow, the difference between additional

data transmission time for the MC, GA and SA approaches also grow, with advantages going to the GA and SA approaches. That is, the proposed GA and SA approaches are able to find better communication trees (with less additional data transmission time) than the MC approach, and with increased number of faulty links and enlarged sizes of the hypercubes (which is the more practical situation), both of the proposed approaches perform even better. (The experimental performance of the EX approach is listed here for reference only because the search time involved in locating an optimal communication tree for the approach is very conspicuous, as mentioned in Section 1. In the simulation, the search by the EX approach is terminated when a communication tree better than that of the MC approach is obtained. The resulting data transmission time can be found in Fig. 4.)

In our simulation, search time for obtaining the objective communication trees for the four approaches is also recorded. To find a better communication tree than the MC approach, the GA and SA approaches take almost the same or slightly more search time than the MC approach. For example, the GA/SA approaches respectively take 0.11/0.09 and 0.97/0.48 second more search time to find a better communication tree than the MC approach in the 8- and 10-dimensional hypercubes with 10% of faulty links. As a matter of fact, the MC approach is shown to take the least search time in finding an objective tree (which is not as optimal as the trees found by the GA and SA approaches), while the search time for the EX approach (to find a communication tree that is only better than what the MC approach finds) is remarkably higher than the other approaches and it turns out even more conspicuous with the increase of dimensions and faulty links. Search time for the GA and SA approaches also grows with the increase of dimensions and faulty links, but the increased values are very small and insignificant. To give an example, in a 10-dimensional hypercube with 20% of faulty links, the time the EX approach takes to search for a better communication tree is 153 seconds more than the MC approach, whereas the GA and SA approaches take only 4.67 and 1.77 more seconds to locate even better communication trees. When the hypercube is with 30% of faulty links, search time for the EX, GA and SA approaches is respectively 242.6, 1.92 and 6.57 seconds more than that of the MC approach. As a result, the GA

and SA approaches may need slightly more search time (than the MC approach) in locating the objective communication trees, but the over time is negligible when compared to the optimum of the located trees, i.e., trees with minimal data transmission time.

5. A Distributed Approach

The above simulation results display that both our proposed GA and SA approaches are able to obtain better communication trees with reasonably low search time and the adjacent links of the chosen sink do not need to be all fault-free. To further trim down the search time for the proposed GA and SA approaches, a distributed approach is introduced as follows.

The idea of the distributed approach is simple but effective: The n-cube is first partitioned into several disjoint k-subcubes and the search for the objective communication subtrees with the lowest data transmission time is simultaneously activated in these k-subcubes by using the GA or SA approach. All objective communication subtrees so found are then combined into a final communication tree in the n-cube. As the search for the subtrees in the disjoint subcubes is parallel, search time for attaining a final communication tree can be effectively reduced.

To partition the n-cube into disjoint k-subcubes, a fault-free $(n\text{-}k)$-subcube of possibly the highest dimension should be found in the first place. Each node of the $(n\text{-}k)$-subcube is viewed as a subsink for each of the 2^{n-k} disjoint k-subcubes, i.e., as the root of the communication tree in each k-subcube. Then, the search for the communication subtrees with the lowest data transmission time are simultaneously activated in these disjoint k-subcubes. To attain the final results, the subresults in each k-subcube are collected into the subsink through each subtree and the results in all subsinks are collected into the sink through the tree in the fault-free $(n\text{-}k)$-subcube. Any tree (with any sink and dimension order) can be selected to collect the subresults (in the subsinks) into the final results (in the sink) as the $(n\text{-}k)$-subcube is fault-free.

Finding the maximal fault-free subcube, i.e., the subcube of the highest dimension, in a faulty hypercube has been investigated in the literature [6-8] (where [6,7] consider only node failures).

To serve our purpose, we present Algorithm I, which is derived from the *bit comparison* approach in [8], in the following.

Algorithm I

```
if (there exist no faulty links)    return (S_n);          /* S_n is the entire hypercube */
Choose a node m with minimal number of faulty adjacent links;
/* Search for an (n – i)-subcube which contains m and is fault-free */
i = 0;
do
{
   i = i + 1;
   for (any of C(n, i) distinct (n – i)-subcube S_{n-i} which contains node m)
        if (non don't care notation bits of S_{n-i} are not all the same as the corresponding notation bits
                of any faulty link)
            return (S_{n-i});
}
until (i = n – 1);
return (S_0);        /* S_0 contains only m */
```

Note that inside the *do loop*, each ($n – i$)-subcube containing node m is checked to see if it is fault-free by bit comparison with the faulty links. As can be easily derived, a subcube will be fault-free if non don't care notation bits of the subcube are not all the same as the corresponding notation bits of any faulty link. That is, if a subcube contains a faulty link, the non don't care notation bits of the subcube should be exactly the same as the corresponding notation bits of the faulty link. It is also clear that there are C(n, i) distinct ($n – i$)-subcubes which contain node m since the notation of a subcube containing a particular node can be obtained by changing any $n – i$ address bits of the node into don't care bits. To give an example, the steps to find the maximal fault-free subcube in a 4-cube with 8 faulty links (with notations 010-, 01-1, 0-11, 11-0, 11-1, 111-, 1-01, -110 as depicted in Fig. 5) by using Algorithm I are given below.

Step 1: Node 0001 is randomly chosen from the nodes with zero (minimal) faulty adjacent links.

Step 2: Subcube 0*** is a 3-cube which contains node 0001. Compared with the notation of the faulty links, it is found by bit comparison that the only non don't care notation bit of 0*** (0 in bit 3) is exactly the same as the corresponding bit (bit 3) of 010- , indicating subcube 0*** at least contains faulty link 010-. Then randomly select another subcube *0**.

Step 3: By bit comparison we find that the only non don't care notation bit of *0** (0 in bit 2) is not the same as the corresponding bit (bit 2) of any link (with 1 or - only in bit 2), meaning subcube *0** contains no faulty links. Therefore, subcube *0** is returned as the required maximal fault-free subcube.

Algorithm I, which checks all possible fault-free subcubes one by one by bit comparison from the large sized subcubes ((n–1)-subcubes) to the small sized subcubes (0-subcubes) to see if they contain faulty links, can be considered as a *top-down* approach. To be complete, we derive another algorithm, Algorithm II, which locates the required fault-free subucbe through a *dimension screening* approach.

Algorithm II

```
if (there exist no faulty links)    return (S_n);          /* S_n is the entire hypercube */
Choose a node m with minimal number of faulty adjacent links as a fault free 0-subcube S_0;
i = 0;
dim = φ;        /* empty set */
do
{
   i = i + 1;
   candidate_dim = φ;
   for (dimension j not in dim)
       if (S_{i-1} and all dimension-j neighbors to its nodes form a fault-free i-subcube S_i)
       {
          t = total number of faulty adjacent links of all the dimension-j neighbors;
          candidate_dim = candidate_dim ∪ {(j, t)};
       }
```

```
    if (candidate_dim ≠ φ )
    {
        select an element (j, t) from candidate_dim where t is the minimum value in the set;
        dim = dim ∪ {j};
        S_i = the i-subcube with all notation bits i (in dim) being don't care and the other bits being
                                the same as the corresponding address bits of node m;
    }
    else
        return (S_{i-1});
}
until (i = n);
```

As we can see, Algorithm II uses dimension screening to locate the required fault-free subcube, and it grows in the direction that the new dimension neighbors are with the minimal total number of faulty adjacent links. The steps of using Algorithm II to find the required fault-free subcube in the 4-cube (in Fig. 5) are given below for illustration.

Step 1: Node 0001 is randomly chosen as a fault-free 0-subcube S_0 from nodes with zero (minimal) faulty adjacent links.

Step 2: S_0 forms a fault-free 1-subcube S_1 respectively with neighbors in dimensions 0, 1, 2 and 3, i.e., 0000, 0011, 0101 and 1001. t is calculated for each dimension, resulting in *candidate_dim* = {(0, 0), (1,1), (2,2), (3,1)}. (0, 0) is selected, *dim* = {0}, and S_1 = 000*.

Step 3: S_1 forms a fault-free 2-subcube S_2 respectively with dimension-1 neighbors 0010, 0011 and dimension-3 neighbors 1000, 1001. t is calculated for each dimension, resulting in *candidate_dim* = {(1,1), (3,1)}. (3,1) is randomly selected since both dimensions are with t = 1. *dim* = {0,3} and S_2 = *00*.

Step 4: S_2 forms a fault-free 3-subcube S_3 only with dimension-1 neighbors 0010, 0011, 1010, 1011. *candidate_dim* = {(1,1)}. (1,1) is selected, *dim* = {0,3,1} and S_3 = *0**.

Step 5: S_4 is not fault-free, so return S_3.

Note that by dimension screening, Algorithm II searches fault-free subcubes from the small sized subcubes to the large sized subcubes. Such a ***bottom-up*** approach works better than the top-down approach of Algorithm I (in terms of search complexity) when the maximal fault-free subcube is smaller, i.e., when the hypercube is with more faulty links. (By contrast, when the maximal fault-free subcube is bigger, the top-down approach will work more desirably.)

As Algorithm II develops from the small sized subcubes to the large sized subcubes, the new dimension neighbors are always with the minimal total number of faulty adjacent links. If more than one direction can be chosen (i.e., more than one dimension is with the minimum t in *candidate_dim*, as in Step 3 above), one of them is randomly chosen. In this way, it may run into and get "local" maximal fault-free subcube in some rare situations since these unchosen directions are not checked any more. (Algorithm II would involve less complexity due to this, in comparison with Algorithm I which checks all possible subcubes on the way.) Whatever the situation, both approaches can be employed to serve our purpose.

The maximal fault-free subcube in the faulty 4-cube, as it turns out for both approaches, is S_3 (i.e., subcube *0**). Each node in S_3 (each black node shown in Fig. 5) can be viewed as a subsink for each of the 8 disjoint 1-subcubes (0*01, 1*01, 1*11, 0*11, 0*10, 1*10, 1*00 and 0*00 respectively). In this way the 4-cube is partitioned into 8 disjoint 1-subcubes and the search for the communication subtrees with the lowest data transmission time can be initiated in parallel in these disjoint 1-subcubes using the GA or SA approach. The objective communication subtrees so found can be handily combined into a final communication tree in the 4-cube. The subresults in each 1-subcube are then collected into the subsink through each subtree and the results in all subsinks are then collected to the sink through any tree in the fault-free 3-subcube *0**. Due to such a parallel search for communication subtrees in the disjoint subcubes, search time for attaining the final communication tree can be further reduced and the improvement is even more obvious with hypercubes of higher dimensions.

6. Conclusions

The search for a fault-tolerant communication tree with minimal data transmission time in a faulty hypercube is desirable as it can accelerate subresult collection and enhance the overall system performance accordingly. Various approaches, such as the exhaustive (EX) approach and the minimal cost (MC) approach can be employed to construct such purposed communication trees. The EX approach can find an optimal communication tree (with the fewest link failures) by checking all possible sinks and dimension orders one by one, but the operation involved is quite complicated and time consuming. The MC approach, in constructing a communication tree which may not be an optimal one in terms of data transmission time, requires all adjacent links of the sink to be fault-free. To meet this requirement, the number of faulty links in a hypercube must be limited under 2^{n-1}. For improvement we propose to use two random search techniques — the genetic algorithm (GA) and the simulated annealing (SA) approaches — to search for more desirable communication trees in the hypercubes. The genetic algorithm is an adaptive search technique with distinctive ability to explore a large searching space. It provides an alternative to traditional optimization techniques by using directed random searches to locate optimal or near optimal solutions of complex problems and is rooted in the mechanisms of evolution and natural genetics. The SA approach is another alternative to traditional optimization techniques and is basically an iterative random search procedure with adaptive moves to find the best solutions of intricate problems. The performance of the EX, MC, GA and SA approaches in locating a desired communication tree has been simulated for evaluation and comparison. Simulation results show that with negligibly more search time, the proposed GA and SA approaches are able to ensure better communication trees than the MC approach. When the sizes of hypercubes and the number of faulty links increase, the GA and SA approaches can work even better, that is, data transmission time of the constructed communication trees turns out even shorter. To trim down the search time for the GA and SA approaches, we also present a distributed approach which partitions an *n*-cube

into several disjoint *k*-subcubes so that the search for the objective communication subtrees with the lowest data transmission time can be simultaneously activated in these *k*-subcubes to bring down time complexity. In order to partition the *n*-cube into the disjoint *k*-subcubes, two algorithms (one by bit comparison; the other by dimension screening) are provided and employed to find a fault-free (*n-k*)-subcube of possibly the highest dimension in the first place. Due to such a parallel search for communication subtrees in the disjoint subcubes, the search time for attaining the final communication tree can be consequently shortened.

References

[1] Y.-R. Leu and S.-Y. Kuo, "A fault-tolerant tree communication scheme for hypercube systems," *IEEE Trans. on Computers*, Vol. 45, No. 45, pp. 641-650, June 1996.

[2] J. H. Holland, *Adaptation in natural and artificial systems*, Univ. of Michigan Press, Ann Arbor, Mich., 1975.

[3] M. Srinivas and L. M. Patnaik, "Genetic algorithms: a survey," *IEEE Computer*, Vol. 27, No. 6, pp. 17-26, June 1994.

[4] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by simulated annealing," S*cience*, Vol. 220, pp. 671-680, 1983.

[5] A. C. Elster and A. P. Reeves, "Block-matrix operations using orthogonal trees," *Proc. SIAM 3rd Int'l Conf. on Hypercube Multiprocessors*, Jan. 1988, pp. 1554-1561.

[6] M. A. Sridar and C. S. Raghavendra, "On finding maximal subcubes in residual hypercubes," *Proc. 2nd IEEE Symp. on Parallel and Distributed Processing*, Dec. 1990, pp. 870-873.

[7] H.-L. Chen and N.-F. Tzeng, "Quick determination of subcubes in a faulty hypercube," *Proc. 21st Int'l Conf. on Parallel Processing*, Aug. 1992, pp. 338-345.

[8] F. Ozguner and C. Aykanat, "A reconfiguration algorithm for fault tolerance in a hypercube multiprocessor," *Information Processing Letters*, Vol. 29, No. 5, pp. 247-254, Nov. 1988.

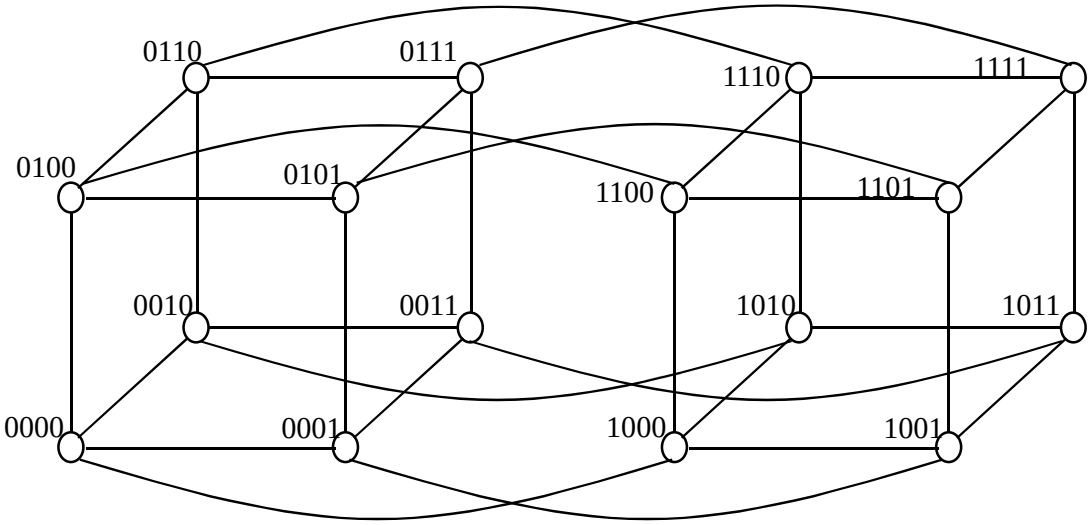


Fig. 1. A 4-cube.

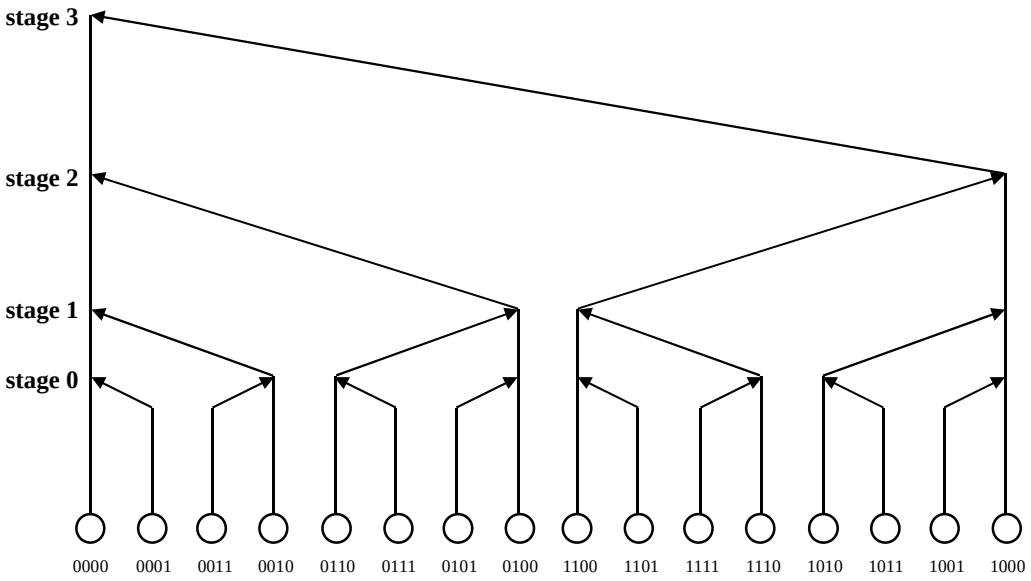


Fig. 2. A tree communication scheme for a 4-cube.

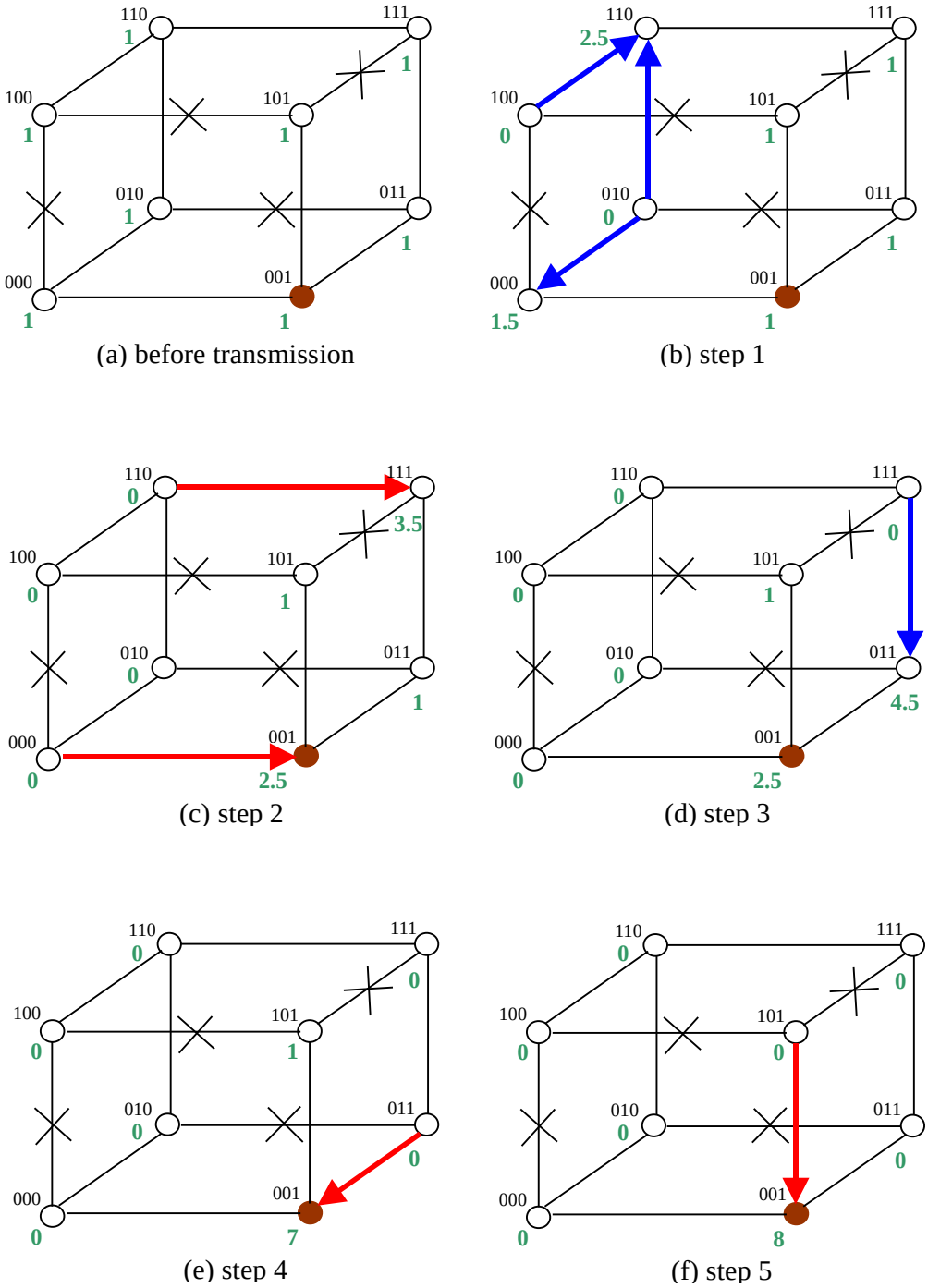
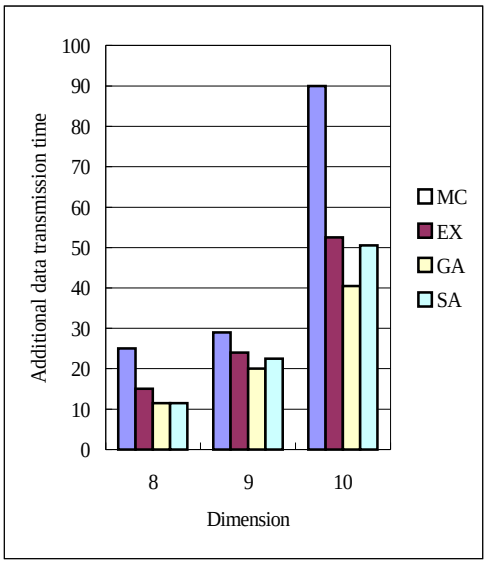
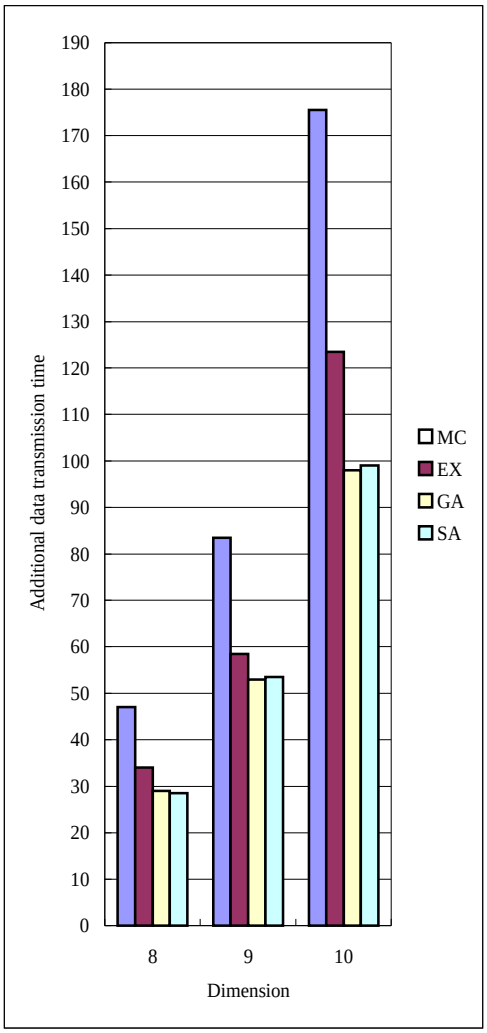


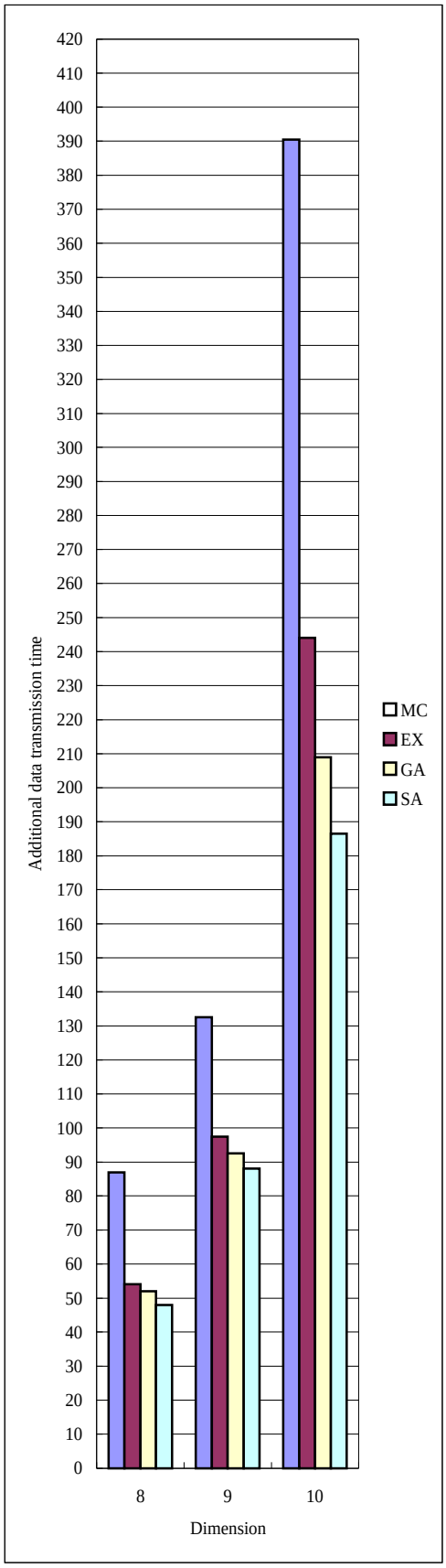
Fig. 3. Steps to illustrate the calculation of the data transmission time for the communication tree with sink 001 and dimension order 012.



(a) 10% faulty links.



(b) 20% faulty links.



(c) 30% faulty links.

Fig. 4. Additional data transmission time for various approaches in faulty hypercubes.

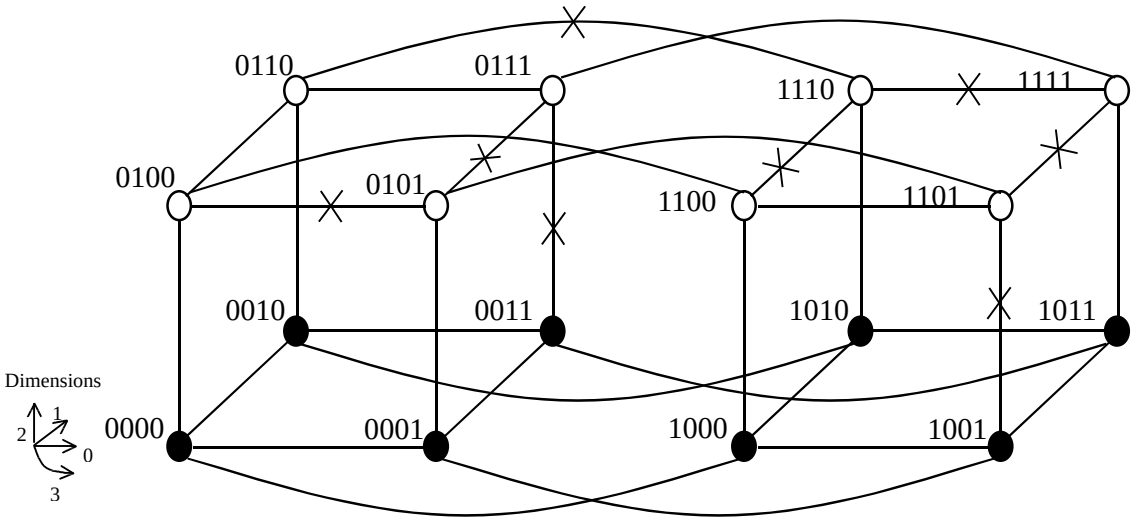


Fig. 5. A 4-cube with 8 faulty links.