Fault Diagnosis in a Flash

Richard Beigel *
Computer Science Department
Yale University
New Haven, CT 06520-8285
rjb@cs.yale.edu

William Hurwood*
Mathematics Department
Yale University
New Haven, CT 06520-8283
will@math.yale.edu

Nabil Kahale†
XEROX PARC
Palo Alto, CA 94304
kahale@parc.xerox.com

Abstract

Consider a set of n processors that can communicate with each other. Assume that each processor can be either "good" or "faulty". Also assume that the processors can test each other. We consider how to use parallel testing rounds to identify the faulty processors, given an upper bound t on their number. We prove that 4 rounds are necessary and sufficient when $2\sqrt{2n} \leq t \leq 0.03n$ (for n sufficiently large). Furthermore, at least 5 rounds are necessary when $t \geq 0.49n$ (for n sufficiently large), and 10 rounds are sufficient when $t < 0.5n$ (for all n). (It is well known that no general solution is possible when $t \geq 0.5n$.)

1 Introduction

Imagine a robot surveying a nuclear accident. As it collects data in radioactive hot spots, its processors tend to get fried. Before too many processors fail, the robot retreats to a safe spot in order to heal. During the healing process, the robot's processors test each other and communicate test results (some correct, some incorrect) to an external controller in a very safe location as far as possible from the accident. The controller determines which processors are really faulty and sends instructions to deactivate them. Then the robot may return to its mission.

The robot's "healing" process is called system level fault diagnosis, as proposed initially by Preparata, Metze, and Chien [11]. When a good processor tests another processor it reports the testee's status correctly to the controller. But when a faulty processor tests another processor it may report the testee's status correctly or incorrectly. Such testing protocols have been usefully implemented, see [4, 3].

Fault diagnosis is not possible in general unless a strict majority of the processors are good [11]. We will assume that throughout this paper.

In *one step* fault diagnosis (see, for example, [13]), every processor tests every processor that it is connected to and the results are communicated to the external controller. Unfortunately, a degree-d interconnect withstands at most $d-1$ faults and requires d tests per processor. Because we prefer an algorithm that is both very fast and also highly fault tolerant, we will not consider one step fault diagnosis in this paper.

In *adaptive serial* fault diagnosis (proposed by Nakajima [9] as an extension to Preparata et al's model), we assume a complete interconnect like an ethernet, so any processor can test any other. The testing process is divided into rounds, during which exactly one test may be performed. The controller dictates which test is performed in each round, and then receives the results of that test. After a finite number of rounds, the controller must determine which processors are really faulty. We assume that the fault status of processors is constant throughout the testing (in our example the robot remains in a safe haven throughout the testing process). The controller is, of course, assumed to be reliable as well.

This led naturally to *adaptive parallel* fault diagnosis, where each processor may participate in at most one test per round. In [5, 7, 6, 1] the number of rounds of testing as a function of n was eventually reduced from a linear bound, through logarithmic and doubly logarithmic bounds to a large constant bound. In [2] explicit small values were found for the number of rounds needed. With deterministic preprocessing, it was shown that 84 rounds suffice for infinitely many values of n. With probabilistic preprocessing, it was shown that for sufficiently large n there existed a testing procedure that would complete diagnosis in 32 rounds. The best lower bound previously known [1] is 3.

This paper substantially closes the gap. We will show that complete diagnosis can be performed in 13 rounds of testing using deterministic preprocessing, and in 10 rounds using randomized preprocessing. In contrast with [2], these algorithms work for all values of n.

We will raise the lower bound from 3 to 5 rounds. So the bounds on the general problem are closed from 3–32 rounds to 5–10 rounds.

If we soften the requirements by assuming that at most 3% of the processors have failed, then we obtain striking tight results: 4 rounds of testing are necessary and sufficient to determine exactly which processors are good and which have failed.

*Supported in part by NSF grants CCR–8958528 and CCR–9415410.

†Partially supported by NSF under grant CCR–9404113.

2 Definitions

We formalize the *parallel model* of fault diagnosis below:

- The computing system is considered to be partitioned into n complex indivisible units called *processors* or *nodes*.
- The object of the algorithm is to perform complete diagnosis, that is to determine the *status* of all the processors. A processor may be either *good* or *faulty*.
- t is a given upper bound on the number of faulty nodes.
- A *test* is an interaction between two processors, a *tester* and a *testee*. The tester claims to determine the status of the testee. If the tester is good then it will correctly diagnose the testee. But if the tester is faulty then it has the option of incorrectly diagnosing the testee. A processor is not permitted to test itself.
- We call a test *good* if and only if the tester reports that the testee was good. Alternatively we will say that the tester liked the testee.
- In a *round* of testing, several tests may be performed simultaneously, but each processor can participate in at most one test. (A round of tests is a directed matching on the processors, but not necessarily a perfect matching.)
- It is permitted for the tests performed in a particular round to be chosen in the light of the results of tests from former rounds. A round is called *adaptive*, if more information is used to select its tests, than was used to select the tests of any previous round.

At times we will find it convenient to view the testing process as a contest between a deterministic *controller* and an omniscient *adversary*. The controller decides which tests to perform, and the adversary places the faulty nodes in such a way as to maximize the number of rounds needed to complete the diagnosis.

Its easy to see [11] that $t < n/2$ is a necessary condition for diagnosis to be possible at all. We shall refer to the case $t = \lceil \frac{n}{2} \rceil - 1$ as the *general* problem.

3 At most 3% faulty processors

In this section we shall show that if $t \leq 0.03n$ then four rounds are necessary and sufficient to perform the diagnosis. We show sufficiency by giving an algorithm that takes 4 rounds, and necessity by giving a strategy for the adversary that prevents diagnosis in 3 rounds.

In [2] an algorithm is presented which works in two phases. In the first phase a fixed test graph H_d, obtained as the union of d Hamiltonian cycles, is applied to the processors. Let G_i denote the directed graph of all good tests performed in rounds 1 to i.

It takes $2d$ rounds of non adaptive tests to apply H_d, assuming that n is even. [2] shows that graphs of this form can be found so that no matter how the faulty processors are arranged, G_{2d} will contain a large strongly connected component of good processors.

A second phase of non adaptive tests is carried out in which several simultaneous tasks are performed. First each large component tests every other large component. Second every large component tests all the processors which are not in a large component. Finally every processor not in a large component tests each large component. This provides sufficient information to complete the diagnosis.

In order to obtain algorithms which work in fewer rounds we will introduce the concept of *supernodes*. A subset U of the processors, can be viewed as a supernode in round $i+1$, if G_i induces a strongly connected graph on U. This couldn't happen if U contained both good and faulty processors, since the strongly connected condition means that there would have to be some test of a faulty processor by a good processor.

A supernode can be viewed as a single processor (because all the processors in it have the same status) that can participate in $|U|$ tests in a single round (because all the processors in it can test or be tested independently). We shall refer to a supernode as good if its elements are all good, as faulty if all its elements are faulty.

Our new algorithms work in 3 stages. First we build supernodes. Then we apply the existing algorithm from [2] to them, viewing each supernode as a single vertex. If the supernodes are large enough this need only take one or two rounds. Finally for the third stage we have to diagnose processors which were not placed in any of the supernodes.

3.1 Complete diagnosis with four rounds

The following algorithm will always succeed if $t \leq 0.03n$, for sufficiently large n.

Rounds 1–2 Divide the nodes up into cycles of size $2d$, setting aside at most $2d-1$ nodes. Have each processor test the status of its successor in its cycle. Since there is an even number of processors in each cycle this can be done in two rounds.

Any cycle in which every test was good can be viewed as a supernode.

Round 3 Each supernode can participate in $2d$ testing operations during this round. So we can apply a d-regular testing graph, H_d, to the supernodes.

H_d will induce a strongly connected component on some subset of the good supernodes.

Round 4 This good component now tests every processor whose status is still in doubt. (We assume that it is large enough for this to only take one round.)

We have to show that d, ϵ and H_d can be chosen in such a way to be certain that this procedure will always work, that is so that the strongly connected good component contains more than half of the original nodes. This can be done using a result similar to that in [2], which is a natural extension if a lemma of Erdős and Rényi, (see for example [10, p. 45]).

Lemma 1 *Let $G = (V, E)$ be a directed graph of order m. Let $0 < \lambda, \gamma < 1$. Suppose that for every pair of sets $A, B \subseteq V$, $A \cap B = \emptyset$, $|A| + |B| = \lambda m$ and $|A|, |B| \leq \frac{1+\gamma}{2}\lambda m$, there are edges in G directed from A to B and from B to A. Then G induces a strongly connected component of size at least $\gamma\lambda m$ on any subgraph of order λm.*

Proof: Let H be the subgraph induced by C on a set of $r = \lambda m$ vertices. Let $C_1, C_2, \ldots C_l$ be the strongly connected components of H.

We may order these components so that if there is an edge $C_i \longrightarrow C_j$ then $i \leq j$.

Let C_k be the largest component, and assume that $|C_k| < \gamma r$. Then we have either

$$\sum_{i=1}^{k-1} |C_i| \leq \frac{1-\gamma}{2} r \quad \text{or} \quad \sum_{i=k+1}^{l} |C_i| \leq \frac{1-\gamma}{2} r$$

Hence taking A to be either $\cup_{i=1}^{k} C_i$ or $\cup_{i=k}^{l} C_i$, as appropriate, and letting B be the remainder of the vertices of H, contradicts the premise. So we must have a strongly connected component of the required size. □

The techniques of [2] show how to select H_d so that it will satisfy the conditions of the lemma. We find that a suitable H_d exists for all sufficiently large n when $\epsilon = .03$ and $d = 4$.

3.2 Lower bound for $t \geq 2\sqrt{2n}$

We give a strategy by which the adversary can prevent diagnosis from being completed in 3 rounds. Initially the adversary will consider every processor to be good. As the testing proceeds the adversary may announce that some processors are in fact faulty. Every tester will always report that any processor that has been announced to be faulty, is in fact faulty, and it will claim that every other processor is good.

The adversary will ensure that whenever a processor is announced to be faulty, all the processors which tested it in previous rounds have also been declared as faulty. At the end of the third round it will ensure that there is at least one processor not declared faulty which has never been tested by a good processor. So there will be no way that the controller can deduce the status of this single processor.

Let T_i denote the directed graph which has an edge from node a to node b iff processor a tested processor b at some point during the first i rounds. Let $\overline{T}_i$ denote the corresponding undirected simple graph. Let G_i and $\overline{G}_i$ be the subgraphs obtained from T_i and $\overline{T}_i$ by restricting the edge set to consist solely of those edges that correspond to good tests. Then the adversary's strategy is as follows:

Round 1 The adversary reports that all the tests were good.

Round 2 The undirected graph, $\overline{T}_2$ of all tests performed has maximum degree 2. So its components are all paths or cycles.

The adversary chooses an integer x. It wants to make sure that $\overline{G}_2$ has no components larger than x. It provisionally places as few faulty processors as possible, so that the longest path in $\overline{T}_2$ containing no faulty processors is $x - 1$. Let F be the set of faulty processors chosen.

If $a \in F$ and a was in fact tested in round 1 the adversary replaces a with the processor which tested it. This ensures that no processor in F was tested earlier but means that now the longest path in $\overline{T}_2$ with no faulty processors is at most $x + 1$.

The adversary wants to ensure that there is at least one processor b, not declared to be faulty, which has either not been tested by the end of round 2, or that was tested by a faulty node. If necessary it will add one more processor to F, to ensure that there is a processor available with these properties.

The adversary announces test results consistent with F being the set of faulty nodes.

Round 3 If the testing graph for round 3 doesn't test b, or tests b with a processor that has been declared to be faulty then the controller cannot diagnose b.

If b is tested by some node c then the adversary decides that c is faulty, and also that all the processors in the component of $\overline{T}_2$ that includes c are faulty. It announces the results of all new tests truthfully with respect to this selection of faulty nodes.

Complete diagnosis will be impossible, since all the processors that tested b are declared faulty, and so the controller cannot trust them.

Theorem 2 *Complete diagnosis cannot be performed in three rounds if $t \geq 2\sqrt{2n}$.*

Proof: We just count up the number of faulty processors that the adversary could need to carry out the algorithm just described.

In round 2 it is in general only necessary to place one faulty processor for every x processors along a path. But two faulty processors are needed for a cycle of length $x + 1$. So at most $2\frac{n}{x+1} + 1$ processors are needed at this stage.

In round 3 the adversary may have to place a further $x + 1$ faulty nodes, since this is the size of the largest possible component of $\overline{G}_2$. There must be at least one more possible faulty node, so that the controller can't deduce the status of b by a counting argument. So the adversary needs

$$2\frac{n}{x+1} + 1 + (x + 1) + 1 \leq t$$

Choosing $x + 1$ to be $\lceil t/2 \rceil$ we get

$$8n \leq t^2 - 4t$$

So provided $t \geq 2\sqrt{2n}$ the adversary can prevent three round diagnosis. □

In particular if $t = 0.03n$ then diagnosis is impossible in 3 rounds for sufficiently large n. In fact the $2\sqrt{2n}$ bound is asymptotically tight. It is not hard to show that if $t \leq \sqrt{n/3}$ then complete diagnosis can be performed in 3 rounds (for large enough n).

4 General problem

The results in this section apply to the general problem. That is we are told only that $t < n/2$. We give an algorithm for complete diagnosis which substantially lowers the constant number of testing rounds needed from 32 given in [2], to 10.

As in section 3 the algorithm works by building supernodes. We will proceed as follows: First we build up supernodes, using a pairing operation, that are large enough to quickly perform the existing algorithms, and find some known good nodes. In the process we will have to set aside processors which failed to join together in supernodes. We shall show that we can exploit information gained during the construction phase to quickly diagnose all the set aside processors using the known good processors, even though most of the processors may have been set aside.

4.1 Building the supernodes

First we pair the nodes off, possibly having to leave one unpaired node out. Suppose that $\{a, b\}$ is one of the pairs of nodes. Then for round 1 processor a will test processor b and for round 2 processor b will test processor a. If both tests were good then $\{a, b\}$ is a supernode of order 2. If this doesn't happen then one or other of a and b is a faulty processor, possibly both of them are. We set the pair a to b aside for the time being. Later we will show what to do with these discards.

Since a strict majority of the original processors were good, and at least half the discarded nodes were faulty, a strict majority of the remaining nodes (the supernodes and possibly an unpaired node) are good. So at least half the supernodes are good.

In round 3 the supernodes are paired together, again leaving possibly one unpaired supernode. Since the supernodes can engage in two testing operations per round, only one round is needed to test both ways within each pair. As before it either yields a new larger supernode of order 4, or the pair must contain at least one faulty supernode. Again we set aside the pairs that didn't form order 4 supernodes.

We proceed similarly in the following rounds. By the end of round 6 the remaining supernodes will contain 32 ordinary nodes each. Let $\mathcal{D}$ denote the set of processors that we have kept, $\mathcal{C}$ be the discarded processors. A strict majority of $\mathcal{D}$ consists of good processors.

4.2 Diagnosing the supernodes

The object of this section is to show how we can diagnose the processors in $\mathcal{D}$ using only two rounds of testing. We are given a set $\mathcal{D}$ of N supernodes, each of order 32.

$\mathcal{D}$ might also contain a single supernode of order 16, one of order 8, etc. Let J denote the largest supernode, if any, in $\mathcal{D}$ whose order is less than 32. Since a strict majority of the nodes in $\mathcal{D}$ are good, either a strict majority of the order 32 supernodes are good; or exactly half of the order 32 supernodes are good, whence J must exist and must be good.

We start by diagnosing the order 32 supernodes in $\mathcal{D}$, assuming that a strict majority of them are good. If in fact exactly half of them are good, they will be divided into two classes, one good and one faulty. Since J is then good, this situation is resolved by having J test a single processor from some order 32 supernode. If J likes that processor, it must be good, and so it belongs to the good class. Otherwise it must belong to the faulty class. We say that we are using J as a *judge*.

We distinguish between two cases. First suppose $N \leq 16$. Then in a single round, every order 32 supernode can test, and be tested by every other order 32 supernode. Simultaneously, J can test all the smaller unpaired supernodes, if there were any.

Since all possible tests between the order 32 supernodes have been carried out, and since the good nodes like each other, they will form a new giant supernode containing at least half of the order 32 supernodes. If there is only one such giant supernode, it must contain the good processors, and we can use it in the second round to test the unpaired supernodes in $\mathcal{D}$. If there are two giant supernodes we can use J as a judge as we have just described. Either way the diagnosis of $\mathcal{D}$ is finished after two rounds.

Otherwise we have $N > 16$. In this case we diagnose the supernodes by adapting the algorithm from [2]. This algorithm has two phases. First it performs the tests given by a fixed regular testing graph. The graph chosen is guaranteed to induce precisely one *big* component (a component of size larger than $N/6$) on the good nodes, but it may also induce several big components on the faulty nodes. Second these components test each other, and also test any processors that are not in any big components.

Both of these phases can be done with non adaptive tests. In our case we shall perform each phase as a single round of tests on our supernodes. So we must show that the original algorithm can be done with at most 32 rounds of non adaptive tests for each phase.

For the first phase [2, theorem 3] shows that for all $N > 16$ there is a graph of the form H_{16} that has the required properties. Since H_{16} is obtained by superimposing 16 Hamiltonian cycles it can be tested in 32 rounds of non adaptive tests.

To perform the second phase of [2] in one round we use the following lemma:

Lemma 3 *Suppose that N processors, of which a strict majority are good, have been divided into components such that there is precisely one good component of size greater than $N/6$. Then the diagnosis can be completed in eight non adaptive rounds.*

Proof: Let $A_1, A_2, \ldots A_r$ denote the strongly connected components of size greater than $N/6$ induced by the testing graph. We have $r \leq 3$ since there are at most two faulty components of size greater than $N/6$.

Let B be the set of processors not in any of the A_i. Since $|B| < \frac{5N}{6}$ it is clear that 5 rounds of non adaptive testing suffice for a component A_i to test each node in B. Another 3 rounds will let each processor in B test some processor in A_i. Since we can partition B into five parts, each part smaller than $N/6$, we can carry out all these tests simultaneously.

Let B_i be the processors in B which A_i considers to be good, and which consider A_i to be good as well. Then the processors in B_i have the same status as A_i. So there will only be one i for which $|A_i \cup B_i| > \frac{N}{2}$. This set must be precisely the set of good processors. □

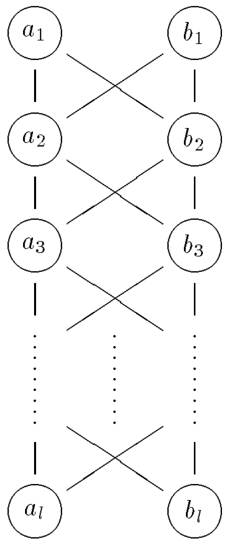

Figure 1: Chain testing pattern

This completes the diagnosis of $\mathcal{D}$. Since this second round requires at most 8 testing operations per order 32 supernode, the supernodes can simultaneously test the unpaired nodes in $\mathcal{D}$. Each of the (at most 3) big components can test unpaired supernodes of orders 4, 8 and 16, ensuring that they have been tested by a good node. We also require J to test some order 32 supernode, in case it is needed to be used as a judge.

But if there are unpaired supernodes of order 1 or 2, it is not possible to ensure that they are tested by all the big components and we might have to leave them for another round. However if there is only one big component, it has to be good and it can test them.

In summary, if $|\mathcal{D}| \leq 16{\cdot}32{+}31 = 543$, or if less than 1/6 of the processors are faulty then we can completely identify all the processors in $\mathcal{D}$ in two rounds. But in any case we can identify all of the processors, except possibly for three of them. Two tests are needed to identify these last three.

4.3 Chains

We now have some order 32 supernodes that have been diagnosed as good. Now we wish to use those nodes to diagnose the pairs in $\mathcal{C}$. We shall improve on an idea from [1].

The key idea is to use the chain testing pattern shown in figure 1. Each pair $\{a_i, b_i\}$ in the chain is a pair of nodes (or supernodes) which were discarded together. If two processors in the figure are connected by an edge then they should test each other, both ways.

The construction guarantees that at least one node in every pair is faulty. Suppose for a given chain we have that exactly one node from every pair is good. Each good node will determine which of the two nodes in the subsequent level is good. So in this case we can diagnose the entire chain, just by diagnosing the top two nodes, a_1 and b_1.

Even if the chain did contain a few faulty–faulty pairs the good nodes in it would form large strongly connected components. So if we tested the large components first we would quite likely find several good components.

Let $m = |\mathcal{D}|$, and $k = |\mathcal{C}|$. Suppose all the pairs in $\mathcal{C}$ have been arranged into one long chain, and that all the edges in the chain have been tested both ways. The chain is divided into a number of strongly connected components, which we shall refer to as *fragments.*

Define g to be the *surplus* good nodes in $\mathcal{D}$, and b to be the *surplus* faulty nodes in $\mathcal{C}$, where g and b are defined by

$$\begin{aligned} g &= |\{a \in \mathcal{D} : a \text{ good}\}| - |\{a \in \mathcal{D} : a \text{ faulty}\}| \\ b &= |\{a \in \mathcal{C} : a \text{ faulty}\}| - |\{a \in \mathcal{C} : a \text{ good}\}| \end{aligned}$$

Since a strict majority of the nodes are good $g > b$.

Since each faulty–faulty pair in the chain contributes two surplus faulty nodes, there are exactly $b/2$ faulty–faulty pairs. Now consider the good fragments in the chain. Each good fragment must be separated from the following one by a faulty–faulty pair. Since $g > b$ there are at most $\frac{1}{2}(g+1)$ good fragments.

The number of good processors available from $\mathcal{D}$ is $\frac{1}{2}(m+g) \geq g$. Spend round 9 testing the g biggest fragments. Let r be the number of nodes in the smallest fragment tested. Let α be the number of these fragments that are faulty, so $g - \alpha$ are good.

Thus there are at most $\frac{1}{2}(g+1) - (g - \alpha) = \alpha + \frac{1}{2}(1-g)$ good fragments that are not tested, and these contain at most $r\alpha + \frac{r}{2}(1-g)$ good processors, since the largest untested fragment has size r. There are $\frac{1}{2}(k-b)$ good processors in the chain, so we must have discovered at least $\frac{1}{2}(k - b - 2r\alpha - r + rg)$ good processors.

Since there are $\frac{1}{2}(k+b)$ faulty nodes, and we've already found at least $r\alpha$ faulty nodes, there remain at most $\frac{1}{2}(k+b) - r\alpha$ faulty nodes to test. In total the number of untested nodes (good and faulty) is at most $\frac{1}{2}(k + b + r - gr)$.

In round 10 we can use the good processors found in round 9 to examine the untested nodes. By subtracting, we find that the number of nodes left over is at most $b + r(\alpha - g + 1)$.

Provided that $\alpha < g$ (that is, we found at least one good fragment in round 9) the multiplier of r is negative or zero. But we have at least g good processors in $\mathcal{D}$, which we can also employ in round 10. Since $g > b$ we have enough good processors to test all the untested nodes. So complete diagnosis is finished in 10 rounds. (We cover the case $\alpha = g$ at the end of subsection 4.4.)

4.4 Building the chain

We show how to construct the chain so that it is ready for use in round 9.

Since each node in a chain is involved in eight tests, the chain could be built in eight rounds. Unfortunately eight rounds are not available.

In round 1 we test the processors in pairs. If one processor doesn't like it pair we know immediately that this pair will go into $\mathcal{C}$. There is no need to carry out the test the other way.

So after round 1 we have some number of pairs in $\mathcal{C}$ and we have seven rounds to construct a chain from them. In four rounds we can carry out all the forward tests down the chain. We only need to test the reverse direction of those forward tests that were good. If a processor likes both of its successors we know it is faulty, (since one of its successors must be faulty), and so there is no need to carry out any more tests involving this processor. So the edges which need to be reverse tested form a binary tree, and so are the union of three matchings. Thus all the reverse tests can be done in only three more rounds.

After round 2 there may be some more pairs $\{a, b\}$ for $\mathcal{C}$. But in each of these pairs a liked b in round 1 and b didn't like a in round 2. So a must be faulty. Its easy to carry out all the remaining tests to form a second chain in only four rounds. We can also link this chain in with the first chain by ensuring that the bottom processors from the first chain don't attempt to test there successors until round 3 by which time we know which processors are in the second chain.

Hence by round 9 we can construct a single chain containing all the simple pairs in $\mathcal{C}$. Unfortunately although the processors added to $\mathcal{C}$ in later rounds are grouped into supernodes and so can carry out more tests per round, we are unable to link these chains in with this single chain.

So we cannot avoid having two chains. This causes a problem because the split between the chains means that there could be one more good fragment than in the above analysis, and this prevents the diagnosis from working.

But if we make both chains into closed loops (that is have tests between the top and bottom pairs in the chain as well) then we get exactly as many possible good fragments as before, and the analysis works.

Provided that there are an even number of pairs (after both round 1 and round 2) assigned to $\mathcal{C}$ the construction above can be used to make a loop instead of a chain. So we can make a single chain, but maybe four processors are left out of it.

Meanwhile after round 3 we are given some pairs of order 2 supernodes to add to $\mathcal{C}$. Again setting one pair aside if necessary we can suppose that there are an even number of pairs.

Since these nodes can carry out two tests per round, we can perform all the necessary tests in four rounds. Since there are an even number of pairs in the chain we can ensure that both the top and bottom pairs in the chain are not needed to perform any tests in either round 7 or round 8.

Similarly after round 4 we have order 4 supernodes. In two rounds we can test all the edges between them to get a chain whose top and bottom pairs are free in round 7 and round 8.

The order 8 and order 16 supernodes only need one round to construct the chain from them, so they cause no problem.

So in round 7 we have up to 4 lengths of chain, and each length has at least two free processors available in its top and bottom supernodes. Thus in two rounds we can link the lengths together, to construct a closed loop for round 9.

So it is possible to construct our two closed loop chains, but up to eight processors may have been set aside.

There are still a few exceptional cases to consider. First it is possible that we didn't manage to completely diagnose $\mathcal{D}$ by the start of round 9. But we have seen this can only happen if at most $5/6$ of the processors in $\mathcal{D}$ are good, and $|\mathcal{D}| > 543$. Thus $g \leq \frac{2}{3}m$. Since we only need g good processors in $\mathcal{D}$ for the algorithm and there really are $\frac{1}{2}(g+m)$ good processors we have at least 100 extra processors. We can spare ten processors in round 9 to complete the diagnosis of $\mathcal{D}$.

Second we have to test any processors from $\mathcal{C}$ that didn't get into the chain. If we have the extra processors described in the last paragraph, we have enough spare good processors to test these as well. If this doesn't occur, then examination of the algorithm which diagnosed $\mathcal{D}$ shows that some extra good processors have been diagnosed by round 7 (or even earlier if $m < 32$) and we can use them in round 8.

Finally, it is possible that no good fragments are found in round 9 at all. (In the analysis we assumed that one good fragment was found.) In this case it is easy to show that $r = 1$ and conclude that there are enough good processors in $\mathcal{D}$ to diagnose all except possibly two processors from $\mathcal{C}$ even if every fragment contains one processor. The previous paragraph shows that we can do a few extra tests before round 9. All that is required is that two processors from the chain are tested by good nodes.

4.5 Constructive algorithm

The previous sections showed that for all n there exists an algorithm which will complete the diagnosis in few rounds. But the algorithm wasn't explicitly found, since it depended on finding H_{16}. Here we will show that given a finite amount of pre-processing, a suitable test graph can be found in polynomial time for all n.

As in [2] the procedure will be basically the same, only instead of using H_d, we shall use the Cayley graphs G_n of Lubotzky, Phillips and Sarnak, see [8]. As before we will build supernodes and chains, apply the test graph to them, and recover some known good processors from them. These will be used to diagnose the remaining processors.

The problem is that G_n doesn't exist for every value of n. We fix a prime p, with p congruent to 1 modulo 4. Then there is a $(p+1)$-regular graph G_n with $q(q^2-1)/2$ vertices, for any prime q satisfying $(\frac{q}{p}) = 1$, q congruent to 1 modulo 4. The second largest (in absolute value) scaled eigenvalue of this graph is at most $2\sqrt{p}/(p+1)$.

If there is no suitable q define G_n to be the graph $G_{n'}$ along with $n-n'$ isolated vertices, where n' is chosen to be the largest integer smaller than n for which a suitable graph exists.

Lemma 4

$$\lim_{n\to\infty} \frac{n-n'}{n} = 0$$

Proof: The lemma follows from considering the distribution of primes in an arithmetic progression. See eg [12, page 214] for more details. □

So the procedure is as follows. We choose n_0 large enough that for $n > n_0$ the fraction of isolated vertices in G_n is less than ϵ. In this case at least $1/2-\epsilon$ fraction of the non-isolated vertices must be good nodes.

[2] shows that provided

$$\frac{\sqrt{ab}}{\sqrt{(1-a)(1-b)}} > \frac{2\sqrt{p}}{p+1}$$

then $G_{n'}$ will induce an edge between any sets of vertices A and B where $|A| = an'$ and $|B| = bn'$.

Lemma 1 shows how we can use this condition to induce a large component. A suitable choice of values is to take $\epsilon = 1/30$, and let the extreme values for a and b be $(7/30)+(1/30)$ and $(7/30)-(1/30)$ respectively. (This corresponds to $\lambda = 7/15$ and $\gamma = 1/7$ in the lemma.) Then G_n induces a component with at least $(1/15)n'$ good nodes, provided $p \geq 37$.

Since we need to test both ways, to test the undirected degree-38 graph G_n we will need order 128 supernodes. Complete diagnosis can be performed in $8+2+3 = 13$ rounds.

If we want to perform diagnosis for a set of supernodes with $n \leq n_0$ we use the method described in the proceeding sections. Since this is only a finite number of cases, we can use preprocessing to find suitable H_{16}.

4.6 In general, diagnosis requires at least five rounds

In this section we shall extend the ideas of section 3.2 to show that it is impossible to perform complete diagnosis in four rounds, given only that $t < \lceil n/2 \rceil$.

The adversary will place faulty nodes so as to ensure that $\overline{G}_3$ has a constant bound, independent of n, on the number of vertices in a component. (Recall that G_i ($\overline{G_i}$) denotes the (undirected) graph of all the good tests performed in rounds 1 to i.)

Unfortunately we need to delete many more vertices from a max degree 3 graph, than we did from a max degree 2 graph to disconnect it. Also we have to ensure that those processors that are declared faulty in round 3 were only tested by other faulty nodes.

The adversary places faulty nodes in round 2 to ensure that the maximum size of a connected component in $\overline{G}_2$ is 3 vertices. Let $C_1, C_2, \ldots C_r$ be the components of $\overline{G}_2$ that do not consist of declared faulty nodes.

The adversary then treats round 3 as being a max degree 3 testing graph, T, on the C_i. Since no test was performed between the C_i in rounds 1–2, the adversary is free to declare any individual C_i to be faulty.

First we need some graph theoretic theorems, which show that we can separate a max degree 3 graph into bounded size components by deleting at most $\frac{49}{100}$ of the vertices.

Theorem 5 *If G is an undirected graph, of maximum degree 3, then G can be separated into components that each contain at most one cycle by deleting an $\frac{8}{17}$ fraction of its vertices.*

Proof: The proof is by induction on the order of the graph. Consider the shortest cycle in the graph, and delete the vertices adjacent to it, and in some cases a few additional vertices, to get the smaller graph to which the induction can be applied.

A complete proof of this theorem will be given in the journal version of this paper. □

Lemma 6 *Let $G = (V, E)$ be a tree with n vertices, and r be some positive integer. Then there exists a subset Z of the vertices of G such that $|Z| \leq n/r$ and the subgraph induced by G on the vertices in $V \setminus Z$ has no components with more than $r-1$ vertices.*

Proof: Let G_U denote the subgraph induced by G on U, for any $U \subseteq V$.

The proof will be by induction on n. It will suffice to show that there exists a *suitable* vertex $x \in V$, where a vertex x is said to be suitable if at least $r-1$ of the vertices of $G_{V-\{x\}}$ are contained in components of size at most $r-1$.

We will proceed by defining a set $\mathcal{L}$ of pairs (L, v) associated with G, where $L \subseteq V$ and $v \in V \setminus L$. Each pair is required to satisfy that the vertices of L form a connected component of $G_{V-\{v\}}$ and that $|L| \leq r-1$. Let

$$U(\mathcal{L}) = \bigcup_{(L,v)\in\mathcal{L}} L$$

Initially set $\mathcal{L} = \emptyset$. Now extend $\mathcal{L}$ as follows. Since G is a tree, $G_{V\setminus U(\mathcal{L})}$ is a forest. Let us suppose that we can find $x \in V$, a non-isolated leaf of this forest. Then there is a single vertex y adjacent to x in $G_{V\setminus U(\mathcal{L})}$.

Let $(L_1, v_1), \ldots, (L_p, v_p)$ be the set of pairs in $\mathcal{L}$ with $v_i = x$. It is possible that $p = 0$ if there are no such pairs. Let $L = L_1 \cup \ldots \cup L_p$ and set $a = |L|$.

If $a \geq r-1$ then x is suitable and we can complete the lemma by erasing x and applying the induction hypothesis. Otherwise $a < r-1$ and so $|L \cup \{x\}| \leq r-1$. Hence we can extend $\mathcal{L}$ by adjoining $(L \cup \{x\}, y)$ to it.

We claim that this method of extending $\mathcal{L}$ will in fact always result in finding a suitable x.

By induction of $|U(\mathcal{L})|$ we can see that $G_{V\setminus U(\mathcal{L})}$ will be connected. Initially $\mathcal{L} = \emptyset$, so $U(\mathcal{L}) = \emptyset$ and so $G_{V\setminus U(\mathcal{L})} = G$ which is connected. The extension of $\mathcal{L}$ by $(L \cup \{x\}, y)$ will always have the effect of adding x to $U(\mathcal{L})$, where x was a leaf of $G_{V\setminus U(L)}$. So $G_{V\setminus U(\mathcal{L})}$ will remain connected.

Since extending $\mathcal{L}$ in this manner increases $|U(\mathcal{L})|$ by one, eventually we will have that $G_{V\setminus U(\mathcal{L})}$ is a single vertex x. Then either x is suitable, or $|V| \leq r-1$. In either case the lemma is completed. □

Suppose a graph G has n vertices and G has at most one cycle. We want to erase vertices of G so that each component has size at most $r-1$. By lemma 6 if G is a tree then we can do this be erasing at most k/r vertices. If $n \leq r-1$ then no vertices need to be erased at all.

Otherwise we have $n \geq r$, and G contains a cycle. Delete one vertex from the cycle in G to get a tree G'. By the lemma we can reduce G' to components of size at most $r-1$ by deleting at most $(n-1)/r$ vertices.

Hence the total number of vertices deleted is at most $\frac{n-1}{r}+1 \leq \frac{n-1}{r}+\frac{n}{r} \leq \frac{2n}{r}$. In conclusion we have

Lemma 7 *Let $G=(V,E)$ be a graph with at most one cycle, $|V|=n$, and r be some positive integer. Then there exists a subset Z of the vertices of G such that $|Z| \leq 2n/r$ and the subgraph induced by G on the vertices in $V \setminus Z$ has no components with more than $r-1$ vertices.*

Corollary 8 *A max degree 3 graph K which contains at most one cycle can be separated into components of size at most 199 vertices each, by deleting a $\frac{1}{100}$ fraction of its vertices. None of the vertices removed had degree 1 in K.*

Proof: This is a restatement of the lemma with $r=200$. □

Combining theorem 5 and corollary 8 we obtain

Theorem 9 *An undirected max degree 3 graph G can be separated into components of maximum size 400 vertices by deleting at most $\frac{49}{100}$ fraction of the vertices.*

The $\frac{49}{100}$ fraction can certainly be improved, but for our purposes any number smaller than $\frac{1}{2}$ is sufficient.

The following adversary argument proves the lower bound. (Recall that T_i ($\overline{T_i}$) denotes the (undirected) graph of all tests made in rounds 1 to i.)

Round 1 The adversary reports that all tests were good. Since the controller cannot possibly gain by not carrying out as many tests as possible on each round, we shall assume that $\overline{T}_1$ is a complete matching. (The possibility that n is odd causes no extra difficulty, except that it complicates the explanation.)

Round 2 The adversary considers all the connected components of $\overline{T}_2$. Each component is either a cycle or a path with an even number of vertices. A case by case examination shows that in each component it is able to place faulty nodes so that

- None of these faulty nodes were tested in round 1.
- At most $\frac{1}{3}$ of the nodes of each component are declared faulty.

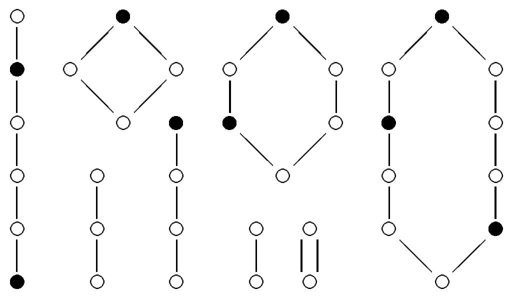

Figure 2: Sample arrangements of faulty nodes in a component

- The faulty nodes separate the apparently good nodes, so that each connected component C_i of $\overline{G}_2$ has size at most three.
- Any C_i of size 3, take the form of a path of length 3, not a 3-cycle.
- The adversary could declare one of the nodes in each C_i of size 2 or 3 to be faulty, and it still would have declared at most half the vertices of T_2 to be faulty.

Some example arrangements that satisfy these conditions are shown in figure 2. Note that no C_i can be a cycle of length 3 because $\overline{G}_2$ has no cycles of odd length.

The adversary then announces its choice F_2, of faulty nodes and gives test results consistent with that set.

Round 3 The adversary constructs a graph $\overline{G}$, on the C_i, by connecting C_i to C_j if and only if there is a test between them in T_3. So $\overline{G}$ has maximum degree 3. By theorem 9 the adversary selects a set W from the vertices of $\overline{G}$, where $|W| \leq \frac{49}{100}|\overline{G}|$ and deleting W separates $\overline{G}$ into components of at most a constant size.

Corollary 8 shows that without loss of generality, all $C_i \in W$ satisfy $|C_i|>1$. We can also assume that no processor in a $C_i \in W$ tested another processor in the same C_i in round 3.

The adversary now forms F_3, the set of additional processors it declares to be faulty in round 3. It cannot take F_3 to be all the processors making up W, since this might mean that more than half the processors are declared to be faulty.

Instead if $C_i \in W$ has $|C_i|=2$ then both processors in C_i are included in F_3. If $C_i \in W$ has $|C_i|=3$ then two processors from C_i are selected for F_3. Since C_i isn't a 3-cycle, we can ensure that the processor in C_i which isn't declared to be faulty has never tested either of the other two processors. Thus consistency in maintained.

The adversary announces test results consistent with $F_2 \cup F_3$ being the set of faulty processors.

Round 4 As with the previous algorithm, the adversary can easily ensure the existence of a node, b, which was only tested by declared faulty nodes in rounds 1–3. Suppose b is tested in round 4, by a node c which hadn't been declared faulty.

Then the adversary declares all the nodes in the component C of $\overline{T}_3$ which contains c to be faulty. C has bounded size, independent of n.

Let K be the component of $\overline{G} \setminus W$ which contains the C_k which includes c. Then K contains at most 199 of the C_i each of which contains at most 3 processors.

Since for $C_i \in W$ with $|C_i| = 3$ we only declared two of the three processors to be faulty, there could be a processor in C for each element of W adjacent to K in $\overline{G}$. This leads to at most 201 extra processors in C. In total we have $|C| < 798$.

The adversary adds C to its set of declared faulty processors, and returns test results consistent with this.

Complete diagnosis will be impossible if n is sufficiently large that we can declare all of C to be faulty, and still have declared less than one half of the processors to be faulty. In rounds 1–3 at most $\frac{49}{100}$ of the processors are declared to be faulty. Thus, if $798 < \frac{1}{100}n$, i.e., if $n > 79800$, complete diagnosis requires at least five testing rounds.

5 Bibliography

References

[1] R. Beigel, S. R. Kosaraju, and G. F. Sullivan. Locating faults in a constant number of testing rounds. In *Proceedings of the 1st Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 189–198, 1989.

[2] R. Beigel, G. Margulis, and D. A. Spielman. Fault diagnosis in a small constant number of parallel testing rounds. In *Proceedings of the 5th Annual ACM Symposium on Parallel Algorithms and Architectures*, 1993.

[3] R. P. Bianchini, Jr. and R. Buskens. An adaptive distributed system level-diagnosis algorithm and its implementation. In *Proceedings of 21st Int. Symp. on Fault Tolerant Computing*, pages 222–229, 1991.

[4] R. P. Bianchini, Jr., K. Goodwin, and D. S. Nydick. Practical application and implementation of distributed system-level diagnosis theory. In *Proceedings of 20th Int. Symp. on Fault Tolerant Computing*, pages 332–339, June 1990.

[5] S. L. Hakimi and K. Nakajima. On adaptive system diagnosis. *IEEE Transactions on Computers*, C-33(3):234–240, Mar. 1984.

[6] S. L. Hakimi, M. Otsuka, E. F. Schmeichel, and G. F. Sullivan. A parallel fault identification algorithm. *Journal of Algorithms*, 11:231–241, 1990.

[7] S. L. Hakimi and E. F. Schmeichel. An adaptive algorithm for system level diagnosis. *Journal of Algorithms*, 5:526–530, 1984.

[8] A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988.

[9] K. Nakajima. A new approach to system diagnosis. In *Proc. 19th Annu. Allerton Conf. Commun. Contr. and Comput.*, pages 697–706, Sept. 1981.

[10] E. M. Palmer. *Graphical Evolution*. John Wiley & Sons, New York, 1985.

[11] F. Preparata, G. Metze, and R. Chien. On the connection assignment problem of diagnosable systems. *IEEE Transactions on Electronic Computers*, EC-16(6):848–853, Dec. 1967.

[12] P. Ribenboim. *The Book of Prime Number Records*. Springer-Verlag, second edition, 1989.

[13] G. F. Sullivan. A polynomial time algorithm for fault diagnosability. In *Proceedings of the 25th IEEE Symposium on Foundations of Computer Science*, pages 148–156, 1984.