Tight Fault Locality

Shay Kutten [*] David Peleg [†]

Abstract

This paper lays a theoretical foundation for scaling fault tolerant tasks to large and diversified networks, such as the Internet. In such networks, there are always parts of the network that failed. On the other hand, various subtasks interest only parts of the network, and it is desirable that those parts, if nonfaulty, do not suffer from faults in other parts. Our approach is to refine the previously suggested notion of fault local algorithms (that was best suited for global tasks) for which the complexity of recovering was proportional to the number of faults. We refine this notion by introducing the concept of *tight fault locality* to deal with problems whose complexity (in the absence of faults) is sublinear in the size of the network. For a problem whose time complexity on an n-node network is $T(n)$ (where possibly $T(n) = o(n)$), a tightly fault local algorithm recovers a legal global state in $O(T(x))$ time when the (unknown) number of faults is x.

This concept is illustrated by presenting a general transformation for MIS algorithms to make them tightly fault local. In particular, our transformation yields an $O(\log x)$ randomized mending algorithm and an $O(\exp(O(\sqrt{\log x})))$ deterministic mending algorithm for MIS. The methods used in the transformation may be of interest by themselves.

[*]I.B.M. T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598, USA, and Dept. of IE&M, Technion, Haifa 32000, Israel. kutten@ie.technion.ac.il.

[†]The Norman D. Cohen Professorial Chair of Computer Sciences, Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot, 76100 Israel. peleg@wisdom.weizmann.ac.il. Supported in part by a Walter and Elise Haas Career Development Award and by a grant from the Israel Science Foundation. Part of the work was done while visiting IBM T.J. Watson Research Center.

1 Introduction

1.1 The problem

Many parallel models assume that information can be collected from all processors quickly. In real computer networks, however, communication is local, hence it takes at least diameter time to collect all information, leading to *global* (i.e. linear in the number of nodes) running times on worst case topologies. Consequently, Linial and others (cf. [L92, AGLP]) promoted the notion of *local* algorithms, namely, algorithms that require the collection of data only from small neighborhoods. We refer to these algorithms also as *sublinear*, since their running time is sublinear in the diameter (or the number of nodes).

The use of sublinear or local algorithms is becoming more and more essential in order for solutions to scale to the emerging huge networks of today, e.g., the Internet. However, one striking characteristic of the research on distributed fault tolerance is that usually faults are corrected *globally*, i.e., by algorithms involving the entire system. In other cases, the running time of the correction algorithm is the same as that of the algorithm for recomputing the function from scratch (cf. [ACS94, AGLP]). Fast and local fault correction is the topic of the current paper.

We first describe the types of faults dealt with in this paper. The global state of a distributed system can be represented as a vector whose components are the local states of the various nodes of the system. A transient fault in one node may change the node's state to some other local state that still looks legal. However the global state may no longer be legal. Intensive research efforts were invested in dealing with this situation, and bringing the system to a correct legal state. A common methodology is based on separating the task into two distinct subtasks, *detection* and *correction*. In this paper we concentrate on the development of fault-local variants for the correction phase.

The study of fault-local algorithms was initiated in [KP95], where the following basic question was raised. Consider a problem Π on graphs, whose solutions are representable as a function $\mathcal{V}$ from the inputs of the network vertices to their outputs. The set of legal solutions of Π on a given graph G is denoted by $\Pi(G)$. Consider a distributed network, whose nodes collectively store the representation of some solution $\mathcal{V} \in \Pi(G)$ of the problem Π on graph G. Suppose that at time t_0, the memory content stored at some subset F of the network nodes is distorted due to some transient failures. As a result, while the stored values still look locally legal, the global representation of $\mathcal{V}$ has changed into some inconsistent function $\mathcal{V}'$ that is no longer valid.

It is clear that, if the problem Π is computable, then investing sufficient computational efforts it is possible to *mend* the function, namely, change the values at some of (or all) the nodes, and reconstruct a valid representation of a (possibly different) solution of the same type, $\mathcal{V}' \in \Pi(G)$. The question raised in [KP95] was whether it is possible to take advantage of the relative rarity of faults and distributively mend the function in time complexity dependent on the number of *failed nodes*, $|F|$, rather than on the size of the entire network, n. This operation (if and when possible) was termed *fault-local mending.* The problem Π is *fault locally T-mendable* if, following the occurrence of faults in a set F of nodes, the solution can be mended in $O(T(|F|))$ time. A problem Π is *fault-locally mendable* if there exists some complexity function T such that Π is fault-locally T-mendable.

Note, though, that this definition of [KP95] makes no special requirements of the function T. For example, an easy-to-solve problem Π (say, solvable from scratch by an algorithm of complexity log-logarithmic in n) may be considered fault-locally mendable if it is fault-locally T-mendable for a highly costly function T (say, doubly-exponential in $|F|$).

In [KP95] it is shown that the actual situation is not all that bad for global (linear) functions. First, *every* (computable) problem is fault locally mendable. Moreover, every problem is fault locally $c|F|\log|F|$-mendable for a small constant c. Hence the complexity of mending is close to linear in $|F|$.

Fault-local mending may make a lot of sense *especially* for local functions, since faults are very often of extremely *local* nature, and involve only a small number of hosts. (For example, the famous crash of the ARPANET, Internet's former incarnation, was caused by a single node giving all other nodes wrong routing information [R81].) Moreover, systems reliability has been increasing, so the number of faults grows much more slowly than the network sizes, especially in domains (or localities) of the network that employ high security and quality standards.

Yet, the motivating observation of the current paper is that for easy-to-solve (or sublinear) local problems, fault-local mending can still be very bad in the worst-case. As our illustrative example, we take Π to be the problem of computing the *maximal independent set (MIS)* of the graph. MIS enjoys a randomized logarithmic time algorithm [L86]. Recomputing the MIS from scratch using Luby's algorithm will only take logarithmic time, whereas the running time of the mending algorithm of [KP95] will be exponentially higher if $|F|$ is large. Hence for such sublinear problems we may hope to be able to do much better, although this may require a more careful definition, as well as a more elaborate algorithm.

Towards this goal, we now introduce the new notion of *tight* fault-local mending. If the cost of computing Π from scratch on an n-vertex network is $\Omega(T(n))$ and Π is fault locally T-mendable, then we say that Π is *tightly locally mendable* (or simply *tightly mendable*). If Π is only fault locally *poly*(T)-mendable then we say that Π is *near-tightly locally mendable.* For randomized algorithms we use analogous terminology. In particular, if $T(n)$ is the complexity of a randomized algorithm for computing Π, then we say that Π is randomly locally T-mendable. The notion of tightness is defined similarly.

We should point out two inherent limitations of any kind of mending, or recovery in general (including, of course, the fault-local approach). First, note that the precise value of the original function $\mathcal{V}$ may not be recoverable, since we do not necessarily know the fault configuration, namely, which nodes suffered faults, or even how many faults occurred. (In fact, it is possible for the faults to change the global state to look precisely as some other legal solution $\mathcal{V}'$, in which case no problem will ever be detected!) Note that this limitation applies, of course, to any kind of correction.

While considered here as a limitation, it should be pointed out that viewed from another angle, our definition of *tight* fault locality implies a property that looks stronger than the one stated. This property can serve as another way to view the difference between fault locality and tight fault locality.

Consider a fault locally T-mendable problem Π. Suppose that we started from a legal solution $\mathcal{V}$, and through failures at a set of nodes F reached an illegal function $\mathcal{V}'$. As discussed above, there is no way for the mending algorithm to know which nodes suffered faults, or what the original solution was. It follows that the mending algorithm is forced to mend $\mathcal{V}'$ to the legal solution $\mathcal{V}''$ *closest* to $\mathcal{V}'$, otherwise it might be too costly if $|F|$ were small. This may in fact be cheaper in some cases than returning to the original $\mathcal{V}$. The definition of a "close" solution for tightly fault local algorithm is different (stronger) than previous definitions.

Put more formally, let $d(\mathcal{V},\mathcal{V}')$ denote the Hamming distance between the two solutions $\mathcal{V}$ and $\mathcal{V}'$, namely, the number of nodes on which they disagree,

$$d(\mathcal{V},\mathcal{V}') = |\{v \mid \mathcal{V}(v) \neq \mathcal{V}'(v)\}|.$$

For a problem Π, a graph G and a function $\mathcal{V}'$, let $\mathit{diff}(\mathcal{V}',\Pi,G)$ denote the distance of $\mathcal{V}'$ from $\Pi(G)$,

$$\mathit{diff}(\mathcal{V}',\Pi,G) = \min\{d(\mathcal{V}',\mathcal{V}) \mid \mathcal{V} \in \Pi(G)\}.$$

Clearly, if $\mathcal{V}'$ is obtained from $\mathcal{V} \in \Pi(G)$ through failures in a set of nodes F, then $\mathit{diff}(\mathcal{V}',\Pi,G)$ is bounded from above by $|F|$. Our observation implies that the complexity of mending a given faulty solution $\mathcal{V}'$ of Π must thus be $O(T(\mathit{diff}(\mathcal{V}',\Pi,G)))$, even if, in fact, the number of faults $|F|$ is much larger than $O(T(\mathit{diff}(\mathcal{V}',\Pi,G)))$. This is due to the fact that we cannot eliminate the possibility that indeed $\mathcal{V}''$ was the original solution, rather than $\mathcal{V}$, so we must assume the case that allows us the smallest cost for correction. Note that this does not always imply that the number of nodes whose state can be changed by the mending algorithm is $O(T(\mathit{diff}(\mathcal{V}',\Pi,G)))$, since changing the states of a large number of nodes may not take a large amount of time, e.g., in the case that they are "packed" in a small neighborhood.

Considered from this point of view, our tightly fault local mending techniques can be thought of as facilitating an approach defined formally and promoted in [DH95] as a basic goal of any algorithm for adjusting to topological changes and faults. That paper suggests the idea that the recovery algorithm should bring the system from its currently faulty global state to a "closest" global state. However, the definition of closeness in [DH95] is somewhat different; they count the *number* of nodes by which the states differ, rather than the *complexity* (as a function of that number) of reaching from one state to the other. Note also, that achieving the goal of [DH95] does not imply a fault local algorithm and thus, of course, it does not imply a tightly fault local algorithm. For example, the algorithm presented there for achieving closeness, performs a global computation even in response to a single change, and is thus *not* fault local.

The second inherent limitation of fault-local mending has to do with the specific representation of the solution function $\mathcal{V}$ in the network. A crucial observation is that if the function is represented *minimally*, then fault-local mending may be impossible (see [KP95]). Hence any solution approach must be based on making use of additional data structures at the various nodes, typically storing information about the local states of their neighbors. It should be clear that the use of such data structures does not by itself suffice to solve the problem. In fact, it may increase its complexity, since any additional data stored at the nodes of the system is just as prone to erasure or distortion due to faults as is the basic data.

1.2 Contributions

In this paper, we define the concept of *tight fault locality*, to better capture the desired performance of a mending algorithm for sublinear functions. We focus mainly on the maximal independent set (MIS) problem and examine its fault-locality properties. An MIS of G is a maximal set $M \subseteq V$ of nodes such that no two nodes in M are neighbors. This is a degenerate case of an input/output function, which has no input. I.e., a solution $\mathcal{V}$ is actually a Boolean function of the vertices whose value is $\mathcal{V}(v) = 1$ if v belongs to the MIS, and 0 otherwise, and given a graph G, the set of legal solutions $MIS(G)$ contains precisely those functions $\mathcal{V}$ that represent an MIS of G. Transient faults may change some 0's to 1's (such that the resulting set $\mathcal{V}'$ is no longer independent), and some 1's to 0's (such that the resulting set is no longer maximal).

The MIS problem has been intensively studied before in the context of understanding the nature of sublinear distributed algorithms. A number of sublinear algorithms exist for it, and hence it is a natural candidate for demonstrating the concepts and transformation technique presented in this paper. Moreover, MIS is of special interest in the context of fault local mending due to the fact that it is used as a paradigm for several purposes in a distributed system. One example is for scheduling access to nearby resources [L80] (a generalization of the Drinking Philosophers problem [CM84]), e.g. a communication channel. It models a setting in which whenever one node accesses a resource, its neighbors are prevented from using that resource.

Our results concerning MIS are the following. We present a generic mending algorithm for MIS, named *Mend*. This algorithm employs an MIS construction procedure P as a subroutine, and we denote the resulting algorithm by $Mend[P]$. The properties of this algorithm depend on the particular procedure chosen. First, employing the randomized algorithm of [L86], MIS_L, as our MIS procedure, we demonstrate that the MIS problem is randomly locally $\log |F|$-mendable. That is, if only $|F|$ faults occurred then the expected running time of the randomized mending algorithm, $T_{Mend[L]}$, is only $O(\log |F|)$ rather than $O(\log n)$. Thus the (randomized expected) complexity of locally mending MIS is as good as that of the best (randomized expected) algorithm known for distributively computing MIS from scratch.

Since we have no nontrivial lower bound on the (randomized or deterministic) complexity of distributed MIS, the above result serves only to show that MIS is randomly *near*-tightly locally mendable. Actually, we show a more general result, namely, that the existence of any randomized MIS algorithm whose time complexity is a "reasonably nice" function T (to be defined precisely later) implies that MIS is randomly locally T-mendable. This implies, in particular, that if the true randomized complexity of MIS is such a nice function, then MIS is randomly tightly locally mendable.

As for deterministic mending we show similar results. Specifically, relying on the deterministic algorithm of [PS92], MIS_{PS}, as our MIS procedure, we show that MIS is fault locally $2^{O(\sqrt{\log|F|})}$-mendable, i.e., the expected running time of the mending algorithm with this procedure, $Mend[PS]$, is $T_{Mend[PS]} = O(2^{O(\sqrt{\log|F|})})$. Again, we show a general result stating that the existence of any deterministic MIS algorithm with "reasonably nice" time complexity T implies that MIS is fault locally $poly(T)$-mendable. This implies, in particular, that if the true deterministic complexity of MIS is such a nice function, then MIS is near-tightly locally mendable.

The methods proposed here for mending MIS can be applied also for obtaining near-tight mending algorithms for a number of other problems, such as coloring and scheduling problems.

Note that the results listed above establish with certainty only the existence of *near*-tightly locally mendable problems. (Of course, problems that can be checked and computed in $O(1)$ time are trivially tightly mendable. Such problems are presented in [NS93].)

1.3 Related Work

The problem of distributed fault correction has been the subject of much research in the area of *dynamic networks*, where the most common type of faults is the crash of a communication link. An approach with mostly theoretical appeal was to run a global "reset" protocol, that enables restarting the computation from scratch [F79, AAG87]. Another approach, more common in practice, is to disseminate every piece of local information globally, so that any global function (e.g., routing) can be corrected (when faults or changes occur) by every node, simply by computing the new value from the known global information [MRR80, CGKK95]. Note that in both approaches, every node must participate in the modification process for every topological change, e.g., the addition or the crash of a single node, hence their worst case complexity is not very attractive. However, when the number of changes is small, the update approach may lead to considerable savings in communication for many problems, since most of the information does not need to be redistributed. It is argued [ACK90] that the reset approach can be avoided, since other approaches can be made more efficient even in the worst case.

In [ACK90], a spanning tree is maintained rather than recomputed. When a tree edge fails, the algorithm replaces it by a single other tree edge (if possible), keeping the rest of the tree intact. This property is called there *path preservation*, and it is argued that it keeps the routing on the tree intact for every route that does not use the failed edge. Similar examples appear in [CP87, CK85, NS93, MNS95, DH95]. In [DH95] this idea was generalized and formalized as a requirement that an illegal global state is corrected to a global state that is the "closest". A general global algorithm (*superstabilization*) is presented there to perform this task for any problem, if the number of detectable faults is precisely 1. This correctness notion, however, is not related to complexity: the *superstabilizing* general algorithm given therein has an $\Omega(diameter)$ running time even when only one fault occurs (hence it is not fault local). Thus, the intuition captured by [DH95] may prove a good generalization for the *path preservation* (i.e. formalizing the intuitive notion of small disruption), and fault locality formalizes another intuitive notion – that of efficiency. On the other hand, the definition of *tight fault locality*, introduced in the current paper, attempts to capture both of these intuitive notions.

All of the previous approaches mentioned above are more suitable for global (linear) problems; and even for those problems they (with the exception of the fault local approach of [KP95]) consume at least $\Omega(diameter)$ time, even for a small number of faults. This means that even one fault can cause a global computation. Moreover, these approaches (especially reset) are mainly suitable for a distributed system that cannot produce "useful work" when some of its nodes suffer a transient fault. A more dynamic solution to the MIS problem, in which the system corrects itself as locally as possible, letting undamaged regions of the system operate as usual in the meantime, appears in [ACS94]. However, the running time of that correction algorithm is the same as that of recomputing the function from scratch. On the other hand, the algorithm in [ACS94] is based on less synchrony assumptions than those used in this paper; in particular, it is not assumed there that all the nodes start the algorithm at the same time, or that the network is synchronous.

A generalization of dynamic networks to deal with self stabilization is proposed in [AKY90]. The reset algorithms in this context (e.g. [AKY90, APV91, AV91, AK93]), suffer from the same disadvantages mentioned above. Other models deal with other (sometimes even stronger) types of faults, or try to treat several kinds of faults simultaneously [GP93, DH95]. Other models studied in the literature allow efficient algorithms through imposing various restrictions, e.g., only one fault may occur at a time, or the network is a complete graph, etc.

2 Preliminaries

2.1 Model and Definitions

We model a distributed system as a graph $G = (V, E)$ where V is the set of the system (or network) nodes, $|V| = n$, and E is the set of links. Nodes communicate by sending messages over the links. The network is synchronous, that is, communication proceeds in rounds, where in every round each node receives all the messages sent to it from its neighbors in the previous round, and sends messages to some of its neighbors. Moreover, we assume that all faults occur simultaneously at time t_0. To focus on time complexity, we adopt the model employed in previous studies of locality issues [GPS87, L92], in which message complexity is abstracted away by allowing the transmission of arbitrary size messages in a single time unit. (Our messages are nevertheless not very large.)

We consider the problem of mending an MIS of a given graph G. The MIS is represented distributively by a vector of bits $\mathcal{M} = (\mathcal{M}_{v_1}, \ldots, \mathcal{M}_{v_n})$, with each node v storing its own bit $\mathcal{M}_v$, such that $\mathcal{M}_v = 1$ if'f v is in the MIS. Denote by $\mathcal{S}(\mathcal{M}) = \{v \mid \mathcal{M}_v = 1\}$ the set induced by $\mathcal{M}$. Hereafter, we shall occasionally use $\mathcal{M}$ for the MIS $\mathcal{S}(\mathcal{M})$, where the intention is clear from the context.

Our approach allows a more general representation, in which in addition to the output bit $\mathcal{M}_v$, each node v may maintain an additional data structure $\mathcal{D}_v$, which is a function of $\mathcal{M}$ and the particular algorithm. Note that $\mathcal{M}$ is a part of the definition of the problem, whereas different algorithms may use different definitions (and different value ranges) for $\mathcal{D}_v$.

We distinguish two important global states: the one just before the faults, and the one just after. Let $\mathcal{M}_v^*$ be the representation of the MIS before the faults, and let $\mathcal{M}_v$ be the current representation of the MIS (after the faults). Likewise, let $\mathcal{D}_v^*$ be the data structure stored at v before the faults, and let $\mathcal{D}_v$ be the current structure stored at v. Thus $\mathcal{M}_v$ and $\mathcal{D}_v$ are the input for the mending task. (Clearly, $\mathcal{M}_v^*$ and $\mathcal{D}_v^*$ are not known to v.)

Formally, the set F of faulty nodes and the set H of healthy nodes are defined, respectively, as

$$F = \{v \mid \mathcal{M}_v \neq \mathcal{M}_v^* \text{ or } \mathcal{D}_v \neq \mathcal{D}_v^*\} \qquad \text{and} \qquad H = \{v \mid \mathcal{M}_v = \mathcal{M}_v^* \text{ and } \mathcal{D}_v = \mathcal{D}_v^*\}$$

If the resulting set $\mathcal{S}(\mathcal{M})$ is not an MIS on a given graph, then there are two possible types of violations. A node v is said to be *uncovered* if neither v nor any of its neighbors are in $\mathcal{S}(\mathcal{M})$. A node v is said to be *collided* if both v and one of its neighbors are in $\mathcal{S}(\mathcal{M})$.

Definition 2.1 The *conflict set* of the assignment $\mathcal{M}$, denoted $\mathcal{C}(\mathcal{M})$, is the set of nodes that can detect locally that $\mathcal{S}(\mathcal{M})$ is not a legal MIS, namely, the uncovered nodes and the collided nodes.

Finally, we need a few definitions concerning the immediate surroundings of sets of nodes. Let $\Gamma_i(v)$ be the set of all nodes whose distance from v in G is less than or equal to i. For a set of nodes W, let $\Gamma_i(W) = \bigcup_{w \in W} \Gamma_i(w)$. The *complement* of a set of nodes $U \subseteq V$ is defined to be

$$Co(U) = V \setminus U.$$

The *i-border* of the set U is defined to be the set of nodes in the i most external layers of U, or formally,

$$Border_i(U) = \Gamma_i(Co(U)) \cap U.$$

$Border_1(U)$ is denoted simply $Border(U)$.

2.2 Utilizing MIS procedures

To date, the precise complexity of distributed MIS is not known. Hence to cast our results in as general a form as we can, we describe our mending algorithm using any distributed MIS procedure as a black box. We are interested in showing that MIS is tightly locally mendable (or at least near-tightly locally mendable). We can prove that subject to the assumption that the complexity of the optimal algorithm is a "reasonably nice" function. To establish that, we show that the existence of any MIS algorithm whose time complexity is a "reasonably nice" function T implies that MIS is fault locally T-mendable in the randomized case, and fault locally $poly(T)$-mendable in the deterministic case.

More precisely, by a "reasonably nice" complexity function we mean the following. A function $T : N \mapsto N$ is said to be *sub-logarithmic* if it is nondecreasing and positive for $n \geq 1$, and in addition it satisfies the following condition:

(SL1) $\forall x, y, \quad T(xy) \leq T(x) + T(y)$,

The function T is *super-logarithmic* if instead of (SL1), it satisfies

(SL2) $\forall x, y, \quad T(xy) > T(x) + T(y)$.

Note that the log function is sub-logarithmic, and in fact, most complexity functions likely to be used are either sub-logarithmic or super-logarithmic. (Our results may possibly hold also for functions that are neither sub-logarithmic nor super-logarithmic, but this is left for further research.)

First consider deterministic MIS procedures. Our definition of "reasonable" complexity allows any deterministic MIS procedure MIS_D whose complexity $T_{MIS_D}(n)$ is either sub-logarithmic or super-logarithmic. This applies, in particular, to the currently best known procedure, due to [PS92] (whose complexity is super-logarithmic).

Lemma 2.2 **[PS92]** There exists a deterministic distributed algorithm MIS_{PS} for computing an MIS on an n-vertex graph in time $T_{MIS_{PS}}(n) = O(2^{O(\sqrt{\log n})})$.

(Note that $2^{O(\sqrt{\log n})}$ is asymptotically larger than any polylogarithmic function in n, but smaller than n^ϵ for any $\epsilon > 0$.)

In our mending algorithm, we invoke an MIS procedure only for a prescribed number of steps τ, on some region G' of the graph, of size n'. If $\tau \geq T_{MIS_D}(n')$, then the procedure will indeed halt with the correct output. If this requirement does not hold, then we assume that the output is arbitrary.

We now turn to randomized MIS procedures. Since the best known randomized MIS algorithm is of expected time complexity $O(\log n)$, we need only consider procedures of sub-logarithmic complexity. Consequently, we shall concentrate on randomized Las-Vegas type MIS procedures MIS_R

whose expected time complexity $T_{MIS_R}(n)$ is sub-logarithmic. Relying on Markov's inequality, we will in fact make use of a corresponding Monte-Carlo variant of the procedure, $MIS_R(\tau)$, in which the procedure is halted after τ steps. Based on well known techniques, it is easy to show the following properties.

Lemma 2.3 Given a randomized Las-Vegas MIS procedure MIS_R of expected time complexity $T_{MIS_R}(n)$, there exists a constant $\tilde{c} \geq 1$, such that whenever executing the corresponding Monte-Carlo variant $MIS_R(\tau)$ with $\tau \geq \tilde{c} \cdot T_{MIS_R}(n)$, the output is a legal MIS with probability $3/4$.

We point out that our randomized mending algorithm is nevertheless a Las Vegas algorithm. Also, note that again our requirement applies in particular to the currently best known randomized procedure, due to [L86].

Lemma 2.4 **[L86]** There exists a randomized distributed algorithm MIS_L for computing an MIS on an n-vertex graph in expected time $T_{MIS_L}(n) = O(\log n)$.

3 Overview of the Solution

A first naive solution one may suggest for the problem of mending an MIS assignment is to have the nodes in the conflict set $\mathcal{C}(\mathcal{M})$ run an MIS protocol, without involving the rest of the nodes of the graph. The problem with this approach is that, even assuming it is possible to efficiently correct $\mathcal{M}$ in such a way[1], this alone cannot guarantee that the complexity of the mending process is bounded as a function of the number of corrupted nodes, since the conflict set might be much larger than the set of corrupted nodes. Consequently, in our solution every node stores information about its state at neighboring nodes. The obvious difficulty is that the data stored at a node for its neighbors might get corrupted just as easily as its own data. Moreover, simple voting schemes (if the node consults many neighbors) run into difficulties. See [KP95, LPRS93, P95].

We start by presenting a simplified solution for the case that $|F|$ is known in advance. First we describe the local data structures $\mathcal{D}_v$ employed in our solution. In addition to the output bit $\mathcal{M}_v$, each node v will store a bit $\mathcal{D}_v(u) = \mathcal{M}_u$ for each node u whose distance from v in the graph is 2 or less. Formally, $\mathcal{D}_v = \{\mathcal{D}_v(u) \in \{0,1\} \mid u \in \Gamma_2(v)\}$.

Given $|F|$, partition the nodes into two sets: a set `Big` consisting of the nodes v whose distance-2 neighborhood $\Gamma_2(v)$ contains more than $2|F|$ nodes, and a set `Small` containing the rest. Each of these sets will be treated separately.

The output of a node v in `Big` is determined by a *majority vote* among its 2-neighbors (including itself), based on the value stored for v in their data structures. If the result of this vote is different than $\mathcal{M}_v$ then v changes its $\mathcal{M}_v$ accordingly. (The reasons for which we look at the distance-2

[1]It is probably not, since it entails solving the problem of MIS completion, which is NP-Hard.

neighborhood of nodes rather than simply their direct neighborhoods will become clear later.) Note that following this vote the original correct value of $\mathcal{M}_v^*$ for any v in `Big` is recovered.

Now consider all the nodes in `Small`. Some of them may now belong to the conflict set $\mathcal{C}$. Denote this subset by $\hat{C} = \mathcal{C} \cap \texttt{Small}$. Notice that it is not enough to run an MIS algorithm for the nodes in $\hat{C}$ only, since some of them have neighbors in the complement $Co(\hat{C})$, and a combination of an arbitrary new MIS in $\hat{C}$ and the original MIS $\mathcal{M}$ in $Co(\hat{C})$ does not necessarily form a legal MIS. One would thus like to select a special kind of MIS in $\hat{C}$, namely, one that is constrained by the values of the original MIS $\mathcal{M}$ in $Co(\hat{C})$. More specifically, a node v in $\hat{C}$ whose neighbor w in $Co(\hat{C})$ has $\mathcal{M}_w = 1$ is not allowed to output $\mathcal{M}_v = 1$, and likewise, uncovered nodes in $Co(\hat{C})$ may impose a 1 value on their neighbors in $\hat{C}$.

Unfortunately, solving such a constrained MIS even by a sequential algorithm is NP-Hard. To overcome this difficulty, we resort to a more relaxed method of fusing the new MIS constructed for $\hat{C}$ to the existing one on $Co(\hat{C})$. We first employ a procedure that selects a subset L of some of the nodes in $Co(\hat{C})$ that are at distance 1 or 2 from $\hat{C}$. For L produced by this procedure, the original MIS $\mathcal{M}$ restricted to the resulting set $Co(\hat{C} \cup L)$ has the nice property that it is "shielded" from the influence of fusing to it a new MIS assignment on $\hat{C} \cup L$, no matter what MIS values are chosen for $\hat{C} \cup L$. More specifically, consider a node v in $Co(\hat{C} \cup L)$ that borders with $\hat{C} \cup L$. Then in the MIS $\mathcal{M}$ defined on $Co(\hat{C} \cup L)$, the value of v is $\mathcal{M}_v = 0$, and, moreover, v has a neighbor u in $Co(\hat{C} \cup L)$ such that $\mathcal{M}_u = 1$. We say that $\mathcal{M}$ is a *shielded MIS* on $Co(\hat{C} \cup L)$. Therefore, as we now compute a new MIS assignment on $\hat{C} \cup L$, it is easy to see that no matter what this MIS assignment is, its fusing together with $\mathcal{M}$ in $Co(\hat{C} \cup L)$ yields a legal MIS in G.

As for time complexity, the execution time of this protocol is bounded by ensuring that the size of $\hat{C} \cup L$ is $O(|F|)$. (The dependence on the particular MIS procedure used for computing the new MIS assignment on this region is discussed in the next subsection.) For bounding $|L|$, we rely on the fact that L consists of nodes that are in the 2-neighborhood of nodes in $\hat{C}$. This ensures that $|L| = O(|F|)$ as a result of our choice to include in $\hat{C}$ only nodes with a small 2-neighborhoods. This is an additional reason for treating nodes with large 2-neighborhoods separately, through direct majority vote.

We now turn to the general case, where it is not assumed that $|F|$ is known. In this case, our algorithm runs in phases, performing a search on the number of faults $|F|$. In phase i it is "guessed" that the number $|F|$ of faults is not larger than some λ_i. The precise definition of λ_i, given later, depends on the complexity of the MIS procedure used, and is aimed at preventing the number of phases from appearing as a multiplicative factor in the complexity of the algorithm.

Several things can go wrong when the guess is too small. One minor problem is that since F is larger than guessed (in the first few phases), the size of the conflict set $\mathcal{C}_i$ may also be larger than anticipated; hence, the MIS computation on $\mathcal{C}_i \cup L_i$ may be too long. An easy way to prevent that from happening is to execute the MIS computation in the i's phase only for the time that would have been needed for this computation had the size of $\mathcal{C}_i$ indeed been λ_i. That is, phase i is run for $O(T_{MIS_R}(\lambda_i))$ time in the randomized case, or $O(T_{MIS_D}(\lambda_i))$ in the deterministic case.

A more severe problem is that the computations carried out in the first phases may increase the number of "currently faulty" nodes. That is, denoting by F_i the set of nodes v whose MIS value $\mathcal{M}_v$ at the beginning of the ith phase is different from their original value $\mathcal{M}_v^*$, while $F_1 = F$, the set F_2 is affected by the outcome of phase 1: The assignment of new MIS values to the nodes of the conflict set $\mathcal{C}_1$ in phase 1 may cause many nodes whose original value was correct to erroneously change their value, thus making the set F_2 much larger than F, and causing many new conflicts to show up in phase 2, and so on. This is the case even if phase 1 is terminated after the fixed time bound.

The only way to completely wipe out the influence of the previous phase would be to make all participating nodes return to their initial values. Indeed, this operation is applied at nodes that decide at the end of a phase to participate in subsequent phases (using the initial values, saved in variables $\mathcal{M}_v^{sav}$). But applying such a re-initialization operation globally may be highly time consuming, since it may happen that the assignment in the neighborhood of some nodes in the conflict set $\mathcal{C}_i$ is consistent at the end of the phase, and hence these nodes are ready to terminate, while some other nodes observe inconsistencies, implying the necessity of another phase. Therefore re-initialization of all nodes requires broadcast of a message from the latter nodes to the former, which may be time consuming.

It is therefore necessary to ensure two things: first, that the algorithm is able to detect that the guess was wrong, and secondly, that the operations of the algorithm under the wrong guess did not cause "too much damage". We achieve this behavior by attempting to prevent seemingly faulty nodes from participating in the voting process and gaining undue influence. Towards that end, we introduce one additional "screening" step before performing our votes. The idea is that nodes with a suspiciously high number of conflicts with neighbors (where "high" here is relative to the phase) will be "guessed" faulty, and consequently will be barred from participating in the voting. It is important to note that despite the possibility of non-voters, the votes carried at the nodes of `Big` will still require a strict majority, namely, more than half of the entire 2-neighborhood (not just of the voters).

4 MIS completions

Formally, the problem of *relaxed MIS completion* requires us to compute an MIS, given a partial MIS assignment. It is permitted to change any part of the partial MIS assignment, as long as the end result is a legal MIS. Note that the only difference between this problem and the problem of MIS is that the complexity of this problem may be lower, due to the given partial information. We comment that in our specific solution, the only values of the given partial MIS assignment U that may be changed are at the set $Border_2(U)$.

4.1 Shielded MIS, Fringed MIS and Shielded Kernels

We now describe a distributed procedure REL_COMP$(G, W, \mathcal{M}, \tau)$, that given a graph G, a subset W of the vertices, a partial MIS assignment $\mathcal{M}$ on W (possibly with some uncovered nodes on the border of W), and a time bound τ, computes a relaxed completion of $\mathcal{M}$ on the rest of the graph (with minimal penetration to the region of W). The procedure makes use of an MIS procedure, which can be either deterministic or randomized, and is applied only for τ steps.

Definition 4.1 Given a graph $G = (V, E)$, a subset $U \subseteq V$ and an MIS $\mathcal{R}$ on the subgraph induced by U, we say that $\mathcal{R}$ is a *Shielded MIS* for U if it contains no nodes from $Border(U)$.

Definition 4.2 Given a graph $G = (V, E)$, a set $W = \{v_1, \ldots, v_k\} \subset V$ and an *independent set* assignment $\mathcal{M} = (\mathcal{M}_{v_1}, \ldots, \mathcal{M}_{v_k})$ on W, the set $\mathcal{S}(\mathcal{M})$ is said to be a *fringed-MIS* of W if it is maximal everywhere except possibly on its border, namely, the only violations that prevent it from being a legal MIS involve nodes v in $Border(W)$ that are uncovered.

This definition is motivated by the fact that our mending problem leads to situations where some of the MIS was ruined, yet other segments remain intact. These segments now form a fringed-MIS, since although no changes have occurred on these nodes themselves, their border nodes may become uncovered due to changes in their neighbors.

Definition 4.3 Given a graph $G = (V, E)$ and a subset $W = \{v_1, \ldots, v_k\} \subset V$, a *shielded kernel* for W is a pair of subsets $(\mathcal{R}, U)$ such that

1. $\mathcal{R} \subseteq U \subseteq W$,
2. U contains (at least) all the internal layers of W up to the last two layers, namely, $W \setminus Border_2(W) \subseteq U$, and,
3. $\mathcal{R}$ is a Shielded MIS for U.

Fig. 1 describes a procedure SHIELD_MIS, that given a graph G, a set of nodes W and a fringed-MIS $\mathcal{M}$ on W, generates a shielded kernel $(\mathcal{R}, U)$ for W. Fig. 2 illustrates the operation of the procedure. (The procedure is sequential; we discuss its distributed implementation later.)

Lemma 4.4 Procedure SHIELD_MIS provides a shielded MIS for the given problem, and requires $O(1)$ time for a distributed implementation.

Proof: Property 1 in the definition of shielded MIS is guaranteed trivially by the procedure. For Property 2, we need to argue that every node that is eliminated from U belongs to the external two layers of W. For the first elimination step, step 2, this is immediate. For the second elimination

```
1. Initialize $U$ to be $W$.
2. Let $X = \{v \in Border(U) \mid \mathcal{M}_v = 1\}$, and eliminate the nodes of $X$ from $U$.
3. Let $Y = \{v \in U \mid v$ is uncovered$\}$, and eliminate the nodes of $Y$ from $U$.
4. Define $\mathcal{R} = \{v \in U \mid \mathcal{M}_v = 1\}$.
5. Output $(\mathcal{R}, U)$.
```

Figure 1: Procedure Shield_MIS$(G, W, \mathcal{M})$.

step, step 3, the claim follows from the fact that $\mathcal{M}$ is a fringed-MIS, hence initially it could have uncovered nodes only on its border, and step 2 could introduce new uncovered nodes only in one of the two external layers of W.

It remain to prove property 3. We first argue that $\mathcal{R}$ forms an MIS on U. Since $\mathcal{S}(\mathcal{M})$ is a fringed-MIS on W, and no new 1-valued nodes were introduced, there are no collision violations in $\mathcal{R}$. Step 3 directly guarantees that U has no uncovered nodes. Hence $\mathcal{R}$ is an MIS on U.

Finally, we need to argue that $\mathcal{R}$ contains no nodes from U's border. Assume, to the contrary, that $\mathcal{R}$ contains some node v from the border of U. If v was also on the border of W, then it should have been erased from U in step 2. Therefore, necessarily v was an internal node of W, and was brought to the border of U due to the elimination of some of its neighbors. But step 2 eliminates only nodes v with $\mathcal{M}_v = 1$, none of which could have been a neighbor of v (since otherwise $\mathcal{M}$ contains a collision), and step 3 eliminates only nodes whose neighbors w in U all have $\mathcal{M}_w = 0$, hence again none of them could be a neighbor of v; contradiction. ▮

4.2 Using shielded MIS for relaxed MIS completion

We now use the procedure given in the previous section for deriving a procedure Rel_Comp for relaxed completion of a given fringed-MIS. We are given a graph $G = (V, E)$, a subset $W = \{v_1, \ldots, v_k\} \subset V$ and a fringed-MIS assignment $\mathcal{M} = (\mathcal{M}_{v_1}, \ldots, \mathcal{M}_{v_k})$ on W. Procedure Rel_Comp is described in Figure 3, and its properties are stated in the following lemma.

Lemma 4.5 1. Procedure Rel_Comp reassigns MIS values only at the nodes of a set Q restricted to $Co(W) \cup Border_2(W)$.

2. If a deterministic MIS procedure MIS_D is used, and the time limit τ satisfies $\tau \geq T_{MIS_D}(|Q|)$, then Procedure Rel_Comp provides a relaxed MIS completion for $\mathcal{M}$.

3. If a randomized MIS procedure $MIS_R(\tau)$ is invoked, and the time limit τ satisfies $\tau \geq \tilde{c} \cdot T_{MIS_R}(|Q|)$ (for the constant $\tilde{c}$ of Lemma 2.3), then Procedure Rel_Comp provides a relaxed

Figure 2: The operation of Procedure SHIELD_MIS on a set W with a fringed-MIS $\mathcal{M}$. ($\mathcal{M}$ is not an MIS because the upper left corner node is uncovered.)

1. Applying Procedure SHIELD_MIS$(G, W, \mathcal{M})$, obtain a shielded kernel $(\mathcal{R}, U)$ for W.
2. Let $Q = Co(U)$.
3. Apply an MIS procedure to the subgraph induced by Q for τ steps, and get an MIS $\mathcal{R}'$. This can be a deterministic procedure MIS_D or a Monte-Carlo type procedure $MIS_R(\tau)$.
4. Output $\mathcal{R} \cup \mathcal{R}'$ as an MIS for the entire graph.

Figure 3: Procedure REL_COMP$(G, W, \mathcal{M}, \tau)$.

MIS completion with probability $3/4$.

4. The time complexity of Procedure REL_COMP is $O(\tau)$.

Proof: The fact that $\mathcal{R}$ is a shielded MIS for U guarantees that it can be combined without conflicts with the MIS $\mathcal{R}'$ computed for the complement of U, to yield an MIS for the entire graph. The time complexity of the procedure is composed of that of Procedure SHIELD_MIS, which is $O(1)$, and the time complexity of the MIS procedure on Q, which is limited to τ steps. The bound on the failure probability in the randomized case follows from Lemma 2.3. ▮

4.3 Distributed implementation

While the description of Procedure REL_COMP and its subprocedure SHIELD_MIS is given in "sequential" form, it is clear that both procedures have a straightforward distributed implementation. The only delicate point is that these procedures are to be initiated (as we shall see later) simultaneously not only by the nodes of the set W, but by some nodes of its complement, $Co(W)$. This does not create any special difficulties, though, since the exact time duration for all the computations is known to all the nodes in the graph. Moreover, these initiators are neighbors of W, and thus the nodes in W which perform the procedure get the initiation messages within a constant time.

5 MIS correction algorithm

We are now ready to describe our mending algorithm for the MIS problem. The algorithm, $Mend$, is given below in Figure 4. The algorithm starts by saving the input values of $\mathcal{M}$ and the data structure $\mathcal{D}$ in variables $\mathcal{M}^{sav}$ and $\mathcal{D}^{sav}$. Let $\mathcal{D}_v^{sav}(v) = \mathcal{M}_v^{sav}$. These input values will be used from time to time throughout the execution. The variables $\mathcal{M}$ and $\mathcal{D}$ will be used for recording the output of the computation. Specifically, whenever a node v selects a new value for its MIS assignment, it stores it in $\mathcal{M}_v$ as well as in $\mathcal{D}_w(v)$ at every node $w \in \Gamma_2(v)$.

The algorithm makes use of a *record inconsistency graph* G_{RI} defined on V, whose edges identify pairs of nodes of which one must be faulty. The edges of this graph are determined on the basis of an inconsistency between the $\mathcal{M}$ value of a node and the recorded value for that node in the data structure stored at a neighbor. Formally, this graph is defined as follows.

Definition 5.1 The (undirected) *record inconsistency* graph G_{RI} consists of the following edges. For every two nodes $u, w \in V$, the edge (u, w) is included in G_{RI} if one of the following conditions holds:

1. $dist(u, w) \leq 2$ and $\mathcal{M}_u^{sav} \neq \mathcal{D}_w^{sav}(u)$ or vice versa,
2. $\mathcal{D}_u^{sav}(x) \neq \mathcal{D}_w^{sav}(x)$ for some common distance-2 neighbor x of u and w (i.e., $x \in \Gamma_2(u) \cap \Gamma_2(w)$).

Note that G_{RI} can be constructed distributively in constant time, as each node can find out which other nodes neighbor it in G_{RI}. For every node v, let $deg_{RI}(v)$ denote v's degree in G_{RI}.

As outlined earlier, the algorithm conducts majority votes to determine the output of some nodes, using their neighbors' data structures. However, a node v with high $deg_{RI}(v)$ (where "high" here is w.r.t. the current phase) is "guessed" to be faulty, and is not allowed to vote. In addition, the algorithm maintains a set `Detected` of nodes that detected a local inconsistency in their own initial output (i.e. every node v such that $\mathcal{M}_v^{sav} = 1$ and $\mathcal{D}_v^{sav}(w) = 1$ for some neighbor w, as well as every node v such that $\mathcal{M}_v^{sav} = 0$ and $\mathcal{D}_v^{sav}(w) = 0$ for every neighbor w). Since the nodes in `Detected` are known to be faulty, they too do not vote.

In each phase i, the algorithm separates nodes with large neighborhoods from ones with small neighborhoods using a threshold λ_i. The sequence $\{\lambda_i\}$ will be fixed later, as a function of the specific MIS procedure used. The only requirement imposed on the choice of this sequence is that it must be *quadratically growing*, namely, satisfy

$$\lambda_{i+1} \geq \lambda_i^2$$

(thus the length of the sequence is smaller than $\log\log n$). This choice also determines the value of J_{max}, the number of phases in the algorithm. (J_{max} is taken to be the first i such that $\lambda_i \geq n$.)

For the algorithm and analysis, we use the following definitions. Let

$$\mathtt{NV}_i = \{v \mid deg_{RI}(v) > \lambda_i\} \cup \mathtt{Detected} \qquad \text{and} \qquad \mathtt{Vot}_i(v) = \Gamma_2(v) \setminus \mathtt{NV}_i$$

(standing, respectively, for "Non-Voting" and "Voters"). For every node v, let $\gamma(v) = |\Gamma_2(v)|$. Let

$$\mathtt{Big}_i = \{v \in V \mid \gamma(v) > 2\lambda_i\} \qquad \text{and} \qquad \mathtt{Small}_i = \{v \in V \mid \gamma(v) \leq 2\lambda_i\}.$$

6 Correctness and analysis of the main algorithm

We first describe the general structure of the proof. In Lemma 6.5 (using Lemma 6.4) we show that for a sufficiently large i, phase i computes a valid MIS. However, the complexity depends on the size of the set F_i of "currently faulty" nodes whose output $\mathcal{M}_v$ at the beginning of phase i is different than the original value $\mathcal{M}_v^*$. Note that these could be either faulty nodes, or healthy nodes that erroneously computed a new $\mathcal{M}_v$ during previous phases. In Lemmas 6.1 and 6.2 we show that the number of such nodes F_i in phase i is not much larger than the number of faults F. First, in Lemma 6.1 we show that the number of nodes participating in computing an MIS in phase i is not much larger than the number of (low-degree) nodes in F_i. Then, in Lemma 6.2 we show that the number of (low-degree) nodes in F_{i+1} is not much larger than the number of nodes that participated in the MIS in phase i.

Lemma 6.1 In any phase i, the set $\mathcal{C}_i$ satisfies $|\mathcal{C}_i| = O(|F| + |F_i \cap \mathtt{Small}_i| \cdot \lambda_i)$.

Proof: Consider a phase i. Partition the set $\mathcal{C}_i$ defined in step 2d into $\mathcal{C}_F = (F \cup F_i) \cap \mathcal{C}_i$ and $\mathcal{C}_H = (H \cap H_i) \cap \mathcal{C}_i$. Clearly, $|\mathcal{C}_F|$ is bounded by $|F| + |F_i \cap \mathtt{Small}_i|$, since $\mathcal{C}_i \subseteq \mathtt{Small}_i$. It is therefore sufficient to bound $|\mathcal{C}_H|$.

The notion of *accusation* is defined as follows. Each node in conflict must accuse some node. In particular, for every $v \in \mathcal{C}_i$, the set of nodes accused by v, denoted $\mathtt{Accuse}(v)$, is $\{w \mid (v,w) \in E, \mathcal{M}_v = \mathcal{M}_w = 1\}$ if v is collided, and $\{w \mid \mathcal{D}_v^{sav}(w) = 1\}$ if v is uncovered.

We now argue that $\emptyset \neq \mathtt{Accuse}(v) \subseteq F_i$ for every node $v \in \mathcal{C}_H$. To see that $\mathtt{Accuse}(v)$ is nonempty, for an uncovered v, note that since v is in both H and H_i, necessarily $\mathcal{M}_v^* = \mathcal{M}_v^{sav} =$

```
on a global start signal do
  1. For every node $v$, set $\mathcal{M}_v^{sav} \leftarrow \mathcal{M}_v$ and $\mathcal{D}_u^{sav}(v) \leftarrow \mathcal{D}_u(v)$ for every $u \in \Gamma_2(v)$.
  2. For $i = 1$ to $J_{max}$ do (* i.e., when $\lambda_i \le n$ *)
    (a) For every node $v \in \mathtt{Big}_i$ that has a collision or uncoverage conflict with some neighbors
        (with the current $\mathcal{M}$ values), perform a vote on the value of $\mathcal{M}_v$ among the voting nodes
        in its 2-neighborhood as follows:
        i. Let $\mathcal{M}_v \leftarrow 0$ if $|\{w \in \mathtt{Vot}_i(v) \mid \mathcal{D}_w^{sav}(v) = 0\}| > \gamma(v)/2$.
        ii. Let $\mathcal{M}_v \leftarrow 1$ if $|\{w \in \mathtt{Vot}_i(v) \mid \mathcal{D}_w^{sav}(v) = 1\}| > \gamma(v)/2$.
        iii. If neither of the two votes has the majority        (* due to non-voters *)
             $v$ marks itself and its neighbors in $\mathtt{Vot}_i(v)$ "nonparticipating"
             (and leaves $\mathcal{M}_v$ unchanged).
    (b) Let $\mathtt{NP}_i$ be the set of nonparticipating nodes.
    (c) for every participating node $v \in \mathtt{Small}_i \setminus \mathtt{NP}_i$ do:
        If $v$ has a collision with a "participating" neighbor $w \in \mathtt{Big}_i$
        (namely, $\mathcal{M}_v = \mathcal{M}_w = 1$), then let $\mathcal{M}_v = \mathcal{M}_v^{sav}$.
    (d) Let $\mathcal{C}_i = (\mathcal{C}(\mathcal{M}) \cap \mathtt{Small}_i) \setminus \mathtt{NP}_i$.
        (* Participating nodes that are in $\mathtt{Small}_i$ and in the conflict set induced by the current
        $\mathcal{M}$. *)
    (e) If $\mathcal{C}_i \neq \emptyset$ then the nodes of $\mathcal{C}_i$ invoke Procedure REL_COMP$(G, V \setminus \mathcal{C}_i, \mathcal{M}, \tau_i)$ for $\tau_i =$
        $T_{MIS_D}(\lambda_{i+3})$ in the deterministic case, and $\tau_i = \tilde{c} \cdot T_{MIS_R}(\lambda_{i+3})$ in the randomized case.
    (f) for every node $v$ whose $\mathcal{M}_v$ has changed, let $\mathcal{D}_u(v) = \mathcal{M}_v$ for every $u \in \Gamma_2(v)$.
  3. (* Handling the case that all phases failed. *)
     In the randomized case only: if any conflict still exists then invoke a Las Vegas variant of
     Algorithm $MIS_R$.
```

Figure 4: Algorithm *Mend*.

$\mathcal{M}_v = 0$, and v's records $\mathcal{D}_v^{sav}$ are identical to the original $\mathcal{D}_v^*$. Hence these records must show a neighbor w such that $\mathcal{D}_v^{sav}(w) = 1$, hence $w \in \mathtt{Accuse}(v)$. The argument for a collided v is similar. To see that each node w accused by v is in F_i, note that since $v \in H \cap H_i$, $\mathcal{D}_v^{sav}(w)$ agrees with $\mathcal{M}_w^*$, and hence $\mathcal{M}_w \neq \mathcal{M}_w^*$, hence $w \in F_i$.

Consequently, partition the set $\mathcal{C}_H$ into two sets, the set ACC_BIG_i of nodes $v \in \mathcal{C}_H$ that accuse some neighbor $w \in F_i \cap \mathtt{Big}_i$, and the set ACC_SMALL_i of nodes $v \in \mathcal{C}_H$ for which $\mathtt{Accuse}(v) \subseteq F_i \cap \mathtt{Small}_i$. To bound $|\mathcal{C}_H|$, we separately bound $|\text{ACC\_BIG}_i|$ and $|\text{ACC\_SMALL}_i|$.

Claim A: $|\text{ACC\_BIG}_i| \leq |F|$.

Proof: Consider a node $v \in \text{ACC\_BIG}_i$, and arbitrarily associate with it one particular node $w_v \in F_i \cap \mathtt{Big}_i \cap \mathtt{Accuse}(v)$. Observe that w_v cannot be a nonparticipating node, since if $w_v \in \mathtt{NP}_i$ then v should have been nonparticipating as well. (See step 2(a)iii in the algorithm.) We use this observation later on to show that w_v has many faulty voting neighbors.

Consider a node $w \in \mathtt{Big}_i$. Group the neighbors of w into the following sets. Let $\Gamma^F(w)$ denote the set of 2-neighbors x of w such that $x \in F \setminus \mathtt{NV}_i$ and $\mathcal{D}_x^{sav}(w) = \mathcal{M}_w$. Note that $\Gamma^F(w) \subseteq \Gamma_2(w) \cap (F \setminus \mathtt{NV}_i)$. Let $\Gamma^{AB}(w)$ denote the nodes $v \in \text{ACC\_BIG}_i$ for which $w = w_v$. Note that by the definition of ACC_BIG and of w, $\Gamma^{AB}(w) \subseteq H$. Let $\gamma^Z(w) = |\Gamma^Z(w)|$ for any appropriate Z.

Since the majority vote carried by w was won by the nodes of $\Gamma^F(w)$, the voting rule used to determine $\mathcal{M}_w$ (Step 2a), combined with the fact that $w \in \mathtt{Big}_i$, imply that

$$\gamma^F(w) \ > \ \gamma(w)/2 \ \geq \ \lambda_i \ . \qquad (1)$$

Observe that each node of $\Gamma^F(w)$ is connected to each node of $\Gamma^{AB}(w)$ in G_{RI}. This follows from the fact that by the definitions of $\Gamma^F(w)$ and $\Gamma^{AB}(w)$, each node $y \in \Gamma^F(w)$ is at distance 4 or less from each node $z \in \Gamma^{AB}(w)$ (since both y and z are 2-neighbors of w), and in addition, y's record regarding w's bit is erroneous while z's record is correct, hence $\mathcal{D}_y(w) \neq \mathcal{M}_w^* = \mathcal{D}_z(w)$, so y and z are connected by an edge in G_{RI}.

Let $c_F(w)$ denote the number of edges connecting the nodes of $\Gamma^F(w)$ to the nodes of $\Gamma^{AB}(w)$ in G_{RI}. As $c_F(w)$ is bounded (for w to be a voting node), we conclude that not too many nodes can be corrupted, since each corruption "costs" distinct λ_i edges in G_{RI}. More precisely,

$$c_F(w) \geq \gamma^F(w) \cdot \gamma^{AB}(w).$$

Combined with (1), it follows that $c_F(w) \geq \lambda_i \cdot \gamma^{AB}(w)$. Hence

$$\begin{aligned} |\text{ACC\_BIG}_i| \ &= \ \sum_{w \in \mathtt{Big}_i} \gamma^{AB}(w) \ \leq \ \frac{1}{\lambda_i} \sum_{w \in \mathtt{Big}_i} c_F(w) \\ &\leq \ \frac{1}{\lambda_i} \sum_{x \in F \setminus \mathtt{NV}_i} deg_{RI}(x) \ \leq \ \frac{1}{\lambda_i} \cdot (\lambda_i \cdot |F \setminus \mathtt{NV}_i|) \ \leq \ |F|. \end{aligned}$$

(The second inequality holds since the sets $\Gamma^{AB}(w)$ are distinct for different w's, and hence the numbers $c_F(w)$ count distinct edges. The third inequality relies on the definition of $\mathtt{NV}$.) ▮

Claim B: $|\text{ACC\_SMALL}_i| \;\leq\; 2\lambda_i|F_i \cap \mathtt{Small}_i|$.

Proof: By definition of ACC_SMALL_i, each $v \in \text{ACC\_SMALL}_i$ has a neighbor in $F_i \cap \mathtt{Small}_i$ (namely, the node it accuses). It follows that $\text{ACC\_SMALL}_i \subseteq \Gamma(F_i \cap \mathtt{Small}_i)$. But the neighborhood of every $v \in F_i \cap \mathtt{Small}_i$ is of size $|\Gamma(v)| \leq 2\lambda_i$ by definition of $\mathtt{Small}_i$. Hence

$$|\text{ACC\_SMALL}_i| \;\leq\; |\Gamma(F_i \cap \mathtt{Small}_i)| \;\leq\; 2\lambda_i|F_i \cap \mathtt{Small}_i|. \qquad ▮$$

The lemma now follows immediately from Claims A and B. ▮

Lemma 6.2 In any phase i, the set F_i satisfies $|F_i \cap \mathtt{Small}_i| = O(|F| \cdot \lambda_{i+1})$, and the set $\mathcal{C}_i$ satisfies $|\mathcal{C}_i| = O(|F| \cdot \lambda_i \cdot \lambda_{i+1})$.

Proof: The second claim follows from the first claim and the previous lemma, as

$$|\mathcal{C}_i| \;\leq\; O(|F| + |F_i \cap \mathtt{Small}_i| \cdot \lambda_i) \;\leq\; O(|F| \cdot \lambda_{i+1} \cdot \lambda_i).$$

The first claim is proved by induction on i. The claim is clear for the beginning of phase $i = 1$. Now assume the claim for phase i and look at the beginning of phase $i+1$. Nodes enter $F_{i+1} \cap \mathtt{Small}_{i+1}$ (if they are not in F_i) either from $\Gamma_2(\mathcal{C}_i)$ (as a result of performing MIS), or as a result of moving from $\mathtt{Big}_i$ to $\mathtt{Small}_{i+1}$. The first of those sources is bounded by noting that $\mathcal{C}_i \subseteq \mathtt{Small}_i$, so

$$|\Gamma_2(\mathcal{C}_i)| \;\leq\; 2\lambda_i|\mathcal{C}_i| \;\leq\; O(\lambda_i \cdot (|F| \cdot \lambda_{i+1} \cdot \lambda_i)) \;\leq\; O(|F| \cdot \lambda_{i+2}).$$

(The last inequality follows from the assumption that the sequence $\{\lambda_i\}$ is quadratically growing.)

As for the second, note that a node $v \in \mathtt{Big}_i$ that was not participating in the previous phase, enters phase $i+1$ with its initial value $\mathcal{M}_v^{sav}$ (Note that it was a nonparticipating node in all the previous phases.) Hence the number of such nodes joining $F_{i+1} \cap \mathtt{Small}_i$ is bounded by $|F|$. Also, the number of nodes in $F \cap \mathtt{Big}_i$ that join F_{i+1} is also bounded by $|F|$.

It remains to consider nodes $v \in H \cap \mathtt{Big}_i$ that have participated in the previous phase i. For such a node v, getting the wrong value (causing it to enter F_{i+1}) necessitates the influence of at least $\gamma(v)/2$ neighbors from $F \setminus \mathtt{NV}_i$, and thus incurs a contribution of at least λ_i to $\sum_{x \in F \setminus \mathtt{NV}_i} deg_{RI}(x)$ in G_{RI}. But the above sum is bounded from above by $\lambda_i \cdot |F \setminus \mathtt{NV}_i|$. Thus the number of nodes in $\mathtt{Big}_i$ that can be influenced in such a way is bounded by $|F \setminus \mathtt{NV}_i| \leq |F|$. ▮

Lemma 6.3 Let i be such that $\lambda_i \geq |F|$. Then $\mathtt{NV}_i$ contains only nodes from F.

Proof: A node $v \in H$ can be adjacent only to nodes of F in G_{RI}, hence its degree in this graph is at most $deg_{RI}(v) \leq |F|$. Thus $\mathtt{NV}_i$ does not contain it by definition. ▮

Lemma 6.4 Let i be such that $\lambda_i \geq |F|$. Then,

(a) every node w in $\texttt{Big}_i$ is participating; and

(b) The voted value for $\mathcal{M}_w$ equals $\mathcal{M}^*_w$.

Proof: Such a node w has at least $\gamma(w) \geq 2\lambda_i + 1$ neighbors (including itself), out of which at most λ_i are in F. By the previous lemma, none of its healthy neighbors from H can be in $\texttt{NV}_i$. Thus, at least $\gamma(v) - \lambda_i > \gamma(v)/2$ do vote for a value for $\mathcal{M}_w$ which is equal to $\mathcal{M}^*_w$. The lemma follows. ▮

Lemma 6.5 If $|F| \leq \lambda_i$, then at the end of phase i, if the algorithm uses a deterministic MIS procedure MIS_D then $\mathcal{M}$ is a valid MIS assignment, and if it uses a randomized MIS procedure MIS_R then $\mathcal{M}$ is a valid MIS assignment with probability $3/4$.

Proof: Suppose that $|F| \leq \lambda_i$. By Lemma 6.4 all the nodes in $\texttt{Big}_i$, as well as their neighbors, are participating nodes. Moreover the nodes in $\texttt{Big}_i$ start then with their values from $\mathcal{M}^*$. As for the nodes in $\texttt{Small}_i$, since they all are participating, a valid MIS is reached if Procedure REL_COMP computes a valid MIS.

By Lemma 6.1, the set $\mathcal{C}_i$ of nodes involved in MIS conflicts after step 2a of the algorithm satisfies $|\mathcal{C}_i| = O(|F| \cdot \lambda_i \cdot \lambda_{i+1})$. Since $\mathcal{C}_i \subseteq \texttt{Small}_i$ and $Q \subseteq \Gamma_2(\mathcal{C}_i)$, (where Q is the set defined in step 2 of Procedure REL_COMP, namely, the expansion of the set $\mathcal{C}_i$) it follows that

$$|Q| \leq |\mathcal{C}_i| \cdot 2\lambda_i \leq |F| \cdot 2\lambda_i \cdot \lambda_i \cdot \lambda_{i+1} \leq 2\lambda_i^3 \lambda_{i+1} \leq \lambda_{i+3}.$$

Hence the time required to compute MIS on Q by MIS_D is $T_{MIS_D}(|Q|) \leq T_{MIS_D}(\lambda_{i+3})$, and hence the expected time required to compute MIS on Q by MIS_R is $T_{MIS_R}(|Q|) \leq T_{MIS_R}(\lambda_{i+3})$. Given the choice of time limit prescribed for the ith phase in Step 2e, Lemma 4.5 guarantees that a valid MIS assignment is produced with certainty in the deterministic case, and with probability $3/4$ in the randomized case. ▮

We are now ready to give the main results.

Lemma 6.6 Given a deterministic distributed procedure MIS_D, if $T_{MIS_D}(n)$ is sub-logarithmic or super-logarithmic then for a suitable choice of the sequence $\{\lambda_i\}$, Algorithm $Mend[MIS_D]$ reaches a legal MIS assignment in time $T_{Mend[MIS_D]} = O(T_{MIS_D}(|F|^c))$ for some constant $c \geq 1$ (with $c = 1$ for the sub-logarithmic case).

Proof: Each of the two cases is handled separately. Consider first the case that $T_{MIS_D}(n)$ is super-logarithmic. Then we fix $\lambda_i = 2^{2^i}$. Note that this choice satisfies the quadratic growth requirement.

Let J be the first phase satisfying $\lambda_J \geq |F|$. By the previous lemma, the algorithm will halt on phase J. The total time consumed by phases 1 through J is bounded by

$$T_{Mend[MIS_D]} = \sum_{j=1}^{J} T_{MIS_D}(\lambda_{j+3}) = \sum_{j=1}^{J} T_{MIS_D}\left(2^{2^{j+3}}\right).$$

By the super-logarithmic assumption (SL2),

$$T_{Mend[MIS_D]} \;\leq\; T_{MIS_D}\left(\prod_{j=1}^{J} 2^{2^{j+3}}\right) \;=\; T_{MIS_D}\left(2^{\sum_{j=1}^{J} 2^{j+3}}\right) \;\leq\; T_{MIS_D}\left(2^{2^{J+4}}\right).$$

By the choice of J it follows that $|F| \geq 2^{2^{J-1}}$, and hence

$$T_{Mend[MIS_D]} \;\leq\; T_{MIS_D}\left(2^{2^{J-1}\cdot 2^5}\right) \;\leq\; T_{MIS_D}(|F|^{32}).$$

Next, consider the case that $T_{MIS_D}(n)$ is sub-logarithmic. Then we fix $\lambda_i = T^{-1}_{MIS_D}(2^i)$. This choice satisfies the quadratic growth requirement as well. To show this, we rely on the fact that since T_{MIS_D} is sub-logarithmic, its inverse $T \equiv T^{-1}_{MIS_D}$ is *super-exponential*, i.e., it satisfies

(SE) $\quad T(x+y) \geq T(x)T(y)$.

Therefore $\lambda_{i+1} = f(2^{i+1}) \geq f(2^i)^2 = \lambda_i^2$.

Letting J be the first phase satisfying $\lambda_J \geq |F|$, the total time consumed by the algorithm (that again halts on phase J) is bounded by

$$T_{Mend[MIS_D]} \;=\; \sum_{j=1}^{J} T_{MIS_D}(\lambda_{j+3}) \;=\; \sum_{j=1}^{J} T_{MIS_D}(T^{-1}_{MIS_D}(2^{j+3})) \;=\; \sum_{j=1}^{J} 2^{j+3} \;\leq\; 2^{J+4}.$$

By the choice of J it follows that $|F| \geq \lambda_{J-1}$, and hence by choice of λ_i and the fact that T_{MIS_D} is nondecreasing, $T_{MIS_D}(|F|) \geq T_{MIS_D}(\lambda_{J-1}) = 2^{J-1}$. Consequently

$$T_{Mend[MIS_D]} \leq 32 \cdot 2^{J-1} \leq 32 \cdot T_{MIS_D}(|F|). \qquad \blacksquare$$

Corollary 6.7 The MIS problem is fault locally $2^{O(\sqrt{\log |F|})}$-mendable.

Proof: Use Procedure MIS_{PS} of [PS92] in the mending algorithm. By Lemma 2.2 and Lemma 6.6, Algorithm $Mend[PS]$ mends MIS in time

$$T_{Mend[PS]} \;=\; O(T_{MIS_{PS}}(|F|^c)) \;=\; O(2^{\sqrt{\alpha \log |F|^c}}) \;=\; O(2^{O(\sqrt{\log |F|})})$$

(using Constant c from Lemma 6.6 and for some constant α). $\blacksquare$

Corollary 6.8 Let the time complexity of the MIS problem be $T^*_D(n)$. If $T^*_D(n)$ is sub-logarithmic then MIS is tightly locally mendable. If $T^*_D(n)$ is super-logarithmic then MIS is near-tightly locally mendable. $\blacksquare$

Lemma 6.9 Given a randomized distributed procedure MIS_R, if $T_{MIS_R}(n)$ is sub-logarithmic then for a suitable choice of the sequence $\{\lambda_i\}$, Algorithm $Mend[MIS_R]$ reaches a legal MIS assignment in expected time $O(T_{MIS_R}(|F|))$.

Proof: As in the sub-logarithmic case of lemma 6.6, we fix $\lambda_i = T^{-1}_{MIS_R}(2^i)$. Letting J be the first phase satisfying $\lambda_J \geq |F|$, the time consumed by the first J phases is bounded in the same way as in that proof, yielding a bound of

$$T_1 \leq 32\tilde{c} \cdot T_{MIS_R}(|F|).$$

By Lemma 6.5, the algorithm will halt on any phase $i \geq J$ with probability 3/4. Hence the algorithm will reach the execution of phase $J+k$ with probability $1/4^k$. The total expected time consumed by phases $J+1$ and on is therefore bounded by

$$T_2 \;=\; \sum_{k>0} \frac{1}{4^k} \cdot \tilde{c} \cdot T_{MIS_R}(\lambda_{J+k+3}) \;=\; \sum_{k>0} \frac{1}{4^k} \cdot \tilde{c} \cdot 2^{J+k+3} \;=\; \tilde{c} \cdot 2^{J+3} \sum_{k>0} \frac{1}{4^k} \cdot 2^k \;\leq\; \tilde{c} \cdot 2^{J+3}.$$

Again, by the choice of J it follows that $|F| \geq \lambda_{J-1}$, and hence by choice of λ_i and the fact that T_{MIS_R} is nondecreasing, $T_{MIS_R}(|F|) \geq T_{MIS_R}(\lambda_{J-1}) = 2^{J-1}$. Consequently,

$$T_2 \;\leq\; 16\tilde{c} \cdot 2^{J-1} \leq 16\tilde{c} \cdot T_{MIS_R}(|F|).$$

Finally, the algorithm will get to the final step of executing the Las Vegas procedure MIS_R only if all phases failed. This will happen with probability $1/4^k$ for $k = J_{max} - J$, and in that case, the algorithm will require an additional time $T_{MIS_R}(n)$, hence the contribution of this step to the expected time complexity is

$$T_3 \;\leq\; \frac{1}{4^k} \cdot T_{MIS_R}(n) \;\leq\; \frac{1}{4^k} \cdot T_{MIS_R}(\lambda_{J_{max}}) \;\leq\; \frac{1}{4^k} \cdot 2^{J_{max}} \;=\; \frac{2^J}{2^k} \;\leq\; \frac{T_{MIS_R}(|F|)}{2^{k-1}}.$$

Overall, the expected time complexity of the algorithm is bounded by

$$T_{Mend[MIS_R]} = T_1 + T_2 + T_3 = O(T_{MIS_R}(|F|))$$

as required. ∎

Corollary 6.10 The MIS problem is randomly locally $\log |F|$-mendable.

Proof: Use Procedure MIS_L of [L86] in the mending algorithm. By Lemma 2.4 and Lemma 6.9, Algorithm $Mend[L]$ mends MIS in time $T_{Mend[L]} = O(T_{MIS_L}(|F|)) = O(\log |F|)$. ∎

Corollary 6.11 Let the randomized expected time complexity of the MIS problem be $T^*_R(n)$. If $T^*_R(n)$ is sub-logarithmic[2] then MIS is randomly tightly locally mendable. ∎

[2]Recall that it is at most logarithmic.

Space and communication complexity: The dominating factor in the size of the storage required is the data structure $\mathcal{D}_v(u) = \mathcal{M}_u$, storing, at each node v, a bit for each node u whose distance from v in the graph is 2 or less. This storage requirement decreases with the decreased density of the graph. In the worst case it implies $O(n)$ bits per node. The dominating factor in the communication is the voting according to $\mathcal{D}_v(u)$, where, in the worst case, each node may receive a vote from every other node. Every vote travels at most over two edges, leading to a total bit communication complexity of $O(n^2)$.

7 Discussion

7.1 Distributed Termination of the Algorithm

Lemmas 6.6 and 6.9 discuss the time until a legal MIS assignment is reached. It may seem as if even after reaching a legal state, nodes still need to test in subsequent phases that they are not in conflict. Even had this been the case, we would still consider it a reasonable requirement, since in many contexts one needs to address the problem of *fault detection*, and not just fault *correction.* For instance, in the approach of self-stabilization, correction is said to have been achieved once a legal state is reached, yet all nodes must continue to periodically test the legality of their state.

Still, for completeness, we mention that our algorithm can be implemented in such a way that nodes that do not detect a conflict at the end of some phase, do not need to take any further steps in the algorithm, unless notified of conflicts by other nodes which are still active. Thus if the nodes which take steps at some phase i are not in any conflict, and have no neighbors in any conflict, at the end of phase i, then no node will take steps in any phase larger than i. (One should not confuse the action of taking steps, though, with the status of "non-participating" determined in Step 2(a)iii in the algorithm; "Non-participating" nodes nevertheless took some steps to determine that they are "non-participating".)

The required modification to the algorithm is described and analyzed in detail in [KP94].

7.2 Summary and Open Problems

The results presented in this paper are mainly of theoretical nature, since the solution is limited in a number of ways. In particular, it is not asynchronous, and, moreover, nodes are not allowed to start at arbitrary (and different) times. Even more desirable would be to have a self-stabilizing solution. Some initial steps in this direction have been taken in [KP96]. Finally, tight mendability is proved only under the assumption that the complexity of MIS is a function that "behaves nicely". (Nevertheless, even without this assumption it is known that MIS is at least nearly tightly mendable.) It would be interesting to remove that assumption.

Despite these limitations of our specific solution, it is our belief that the proposed general *approach* may be of considerable practical significance. Towards that goal, a number of necessary steps must be taken, to remove the limitations discussed above. In addition, although demonstrated only for MIS, questions related to mendability and tight mendability can be asked for other distributed problems. In particular, it is desirable to apply our approach to real life protocols.

Of course, one can think also of generalizations of our work to other models, defined by different characteristics; e.g., one can study tight fault locality in shared memory with e.g. atomic registers. Related questions can be asked in the context of interactive tasks (as opposed to input-output tasks). These are tasks that run forever, guaranteeing certain properties (e.g. that each node enters a critical section often).

Finally, questions similar to the one asked in this paper can be raised in *non-distributed* contexts as well. E.g., tight mending meaningful also for sequential data structures. There, it forms a generalization of the area of dynamic data structures [Fre83a, T83], in which one assumes that F (rather than 1 or n) changes occurred, and studies the complexity of updating the data structure.

Acknowledgments: It is a pleasure to thank Hadas Shachnai and Moti Yung for helpful discussions, and Esther Jennings for her useful comments on an earlier draft. Thanks are also due to two anonymous referees, whose remarks have helped to improve the presentation of the paper.

Summary of main notation

$G = (V, E)$ - a graph representing the system, with node set V, $|V| = n$, and link set E

$Mend[P]$ - a mending algorithm for MIS, employing an MIS construction procedure P

T_{Mend} - the running time of algorithm $Mend$

$\mathcal{M} = (\mathcal{M}_{v_1}, \ldots, \mathcal{M}_{v_n})$ - a given MIS vector

$\mathcal{S}(\mathcal{M})$ - the set induced by $\mathcal{M}$

$\mathcal{D}_v$ - an additional data structure maintained by node v

$\mathcal{M}^*_v$, $\mathcal{D}^*_v$ - the MIS bit and the data structure stored at v before the faults

$\mathcal{M}_v$, $\mathcal{D}_v$ - the MIS bit and the data structure stored at v after the faults

F, H - the sets of faulty and healthy nodes

$\mathcal{C}(\mathcal{M})$ - the conflict set

$\Gamma_i(W)$ - the i-neighborhood of W

$Co(U)$ - the complement of U (w.r.t. V)

$Border_i(U)$ - the i-border of U

$\texttt{Big}$, $\texttt{Small}$ - the sets of nodes v s.t. $|\Gamma_2(v)| > 2|F|$ (respectively, $|\Gamma_2(v)| \leq 2|F|$) for a given $|F|$

$\hat{C}$ - the set of conflicting nodes in $\texttt{Small}$

F_i - the set of faulty nodes at the beginning of the ith phase

G_{RI} - the record inconsistency graph

$deg_{RI}(v)$ - v's degree in G_{RI}

$\texttt{Detected}$ - the set of nodes that detected a local inconsistency in their initial output

λ_i - a quadratically growing sequence

$\texttt{NV}_i$ - the non-voting nodes in phase i

$\texttt{Vot}_i(v)$ - the phase i voters in $\Gamma_2(v)$

$\gamma(v) = |\Gamma_2(v)|$

$\texttt{Big}_i$, $\texttt{Small}_i$ - the sets of nodes with $\gamma(v) > 2\lambda_i$ (resp., $\gamma(v) \leq 2\lambda_i$)

$\mathcal{C}_F$ - the conflicting nodes which are faulty (now or at the beginning)

$\mathcal{C}_H$ - the conflicting nodes which are healthy (now and at the beginning)

$\texttt{Accuse}(v)$ - the nodes accused by v

w_v - the faulty accused node associated with v

$\textsc{Acc\_Big}_i$ - the set of nodes in $\mathcal{C}_H$ that accuse some neighbor in $F_i \cap \texttt{Big}_i$

$\textsc{Acc\_Small}_i$ - the set of nodes in $\mathcal{C}_H$ that accuse only neighbors from $F_i \cap \texttt{Small}_i$

$\Gamma^F(w)$ - the set of 2-neighbors x of w such that $x \in F \setminus \texttt{NV}_i$ and $\mathcal{D}^{sav}_x(w) = \mathcal{M}_w$.

$\Gamma^{AB}(w)$ - the nodes $v \in \textsc{Acc\_Big}_i$ for which $w = w_v$.

$\gamma^Z(w) = |\Gamma^Z(w)|$ (for any appropriate Z)

$c_F(w)$ - the number of edges connecting the nodes of $\Gamma^F(w)$ to the nodes of $\Gamma^{AB}(w)$ in G_{RI}

References

[AAG87] Yehuda Afek, Baruch Awerbuch, and Eli Gafni. Applying static network protocols to dynamic networks. In *Proc. 28th IEEE Symp. on Foundations of Computer Science*, October 1987.

[ACS94] B. Awerbuch, L. J. Cowen and Mark A. Smith. Efficient asynchronous distributed symmetry breaking. In *Proc. 26th ACM Symp. on the Theory of Computing*, Montreal, Canada, May 1994.

[ACK90] Baruch Awerbuch, Israel Cidon, and Shay Kutten. Optimal maintenance of replicated information. In *Proc. 31st IEEE Symp. on Foundations of Computer Science*, 1990.

[AKY90] Y. Afek, S. Kutten, and M. Yung. Memory-efficient self stabilizing protocols for general networks. In *Proc. 4th Workshop on Distributed Algorithms*, Springer-Verlag, September 1990.

[AGLP] B. Awerbuch, A. Goldberg, M. Luby and S. Plotkin. Network decomposition and locality in distributed computation, *Proc. 30th Symp. on Foundations of Computer Science*, pp. 364–375, October 1989.

[AK93] S. Aggarwal and Shay Kutten. Time-Optimal Self Stabilizing Spanning Tree Algorithms, *Proc. 13th Conf. on Foundations of Software Technology and Theoretical Computer Science*, Bombay, India, December 1993.

[APV91] B. Awerbuch, B. Patt-Shamir, and G. Varghese. Self-stabilization by local checking and correction. In *Proc. 32nd IEEE Symp. on Foundation of Computer Science*, pages 268–277, October 1991.

[APVD94] B. Awerbuch, B. Patt-Shamir, G. Varghese, and S. Dolev. Self stabilization by local checking and global reset, In *Proc. 8th Workshop on Distributed Algorithms*, Springer-Verlag, pages 226–239, October 1994.

[AV91] B. Awerbuch, , and G. Varghese. Distributed program checking: a paradigm for building self-stabilizing distributed protocols, in *Proc. 32nd IEEE Symp. on Foundation of Computer Science*, pages 258–267, October 1991.

[C91] D. E. Comer. Internetworking with TCP/IP. Prantice Hall, New Jersey, US, 1991.

[CGKK95] Israel Cidon, Inder Gopal, Marc Kaplan, and Shay Kutten. Distributed Control for Fast Networks, *IEEE Trans. Communications*, Vol. 43, No. 5, May 95, pp. 1950–1960.

[CK85] I. Chlamtac and S. Kutten. A Spatial Reuse TDMA/FDMA for Mobile Multihop Radio Networks. *IEEE INFOCOM 85*, Washington, DC, USA, March 1985.

[CM84] K. Chandy and J. Misra, The drinking philosophers problem, *ACM TOPLAS*, Vol. 6, No. 4, October 1984, pp. 632-646.

[CP87] I. Chlamtac, and S. Pinter. Distributed Node Organization Algorithm for Channel Access in a Multihop Dynamic Radio Network. *IEEE Transactions on Computers*, VOL. C-36, NO 6, June 1987, pp. 728-737.

[D74] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Comm. ACM*, 17:643–644, November 1974.

[DH95] S. Dolev and T. Herman. SuperStabilizing Protocols for Dynamic Distributed Systems. In *2nd Workshop on Self-Stabilizing Systems*. Las Vegas, Nevada, May 1995.

[F79] Steven G. Finn. Resynch Procedures and a Fail- Safe Network Protocols. *IEEE Trans. Communications*, Vol. COM-27, No. 6, June 1979.

[Fre83a] Greg N. Frederickson. Data structures for on-line updating of minimum spanning trees. In *Proc. 15th ACM Symp. on Theory of Computing*, April 1983.

[GP93] A.S. Gopal and K.J. Perry, Unifying Self Stabilization and Fault Tolerance, In *Proc. 12th ACM Symposium on Principles of Distributed Computing*, Ithaca, NY, August 1993, pp. 195-206.

[GPS87] A. V. Goldberg, S. Plotkin, and G. Shannon, Parallel symmetry breaking in sparse graphs, *Proc. 19th ACM Symp. on Theory of Computing*, May 1987.

[KP94] S. Kutten and D. Peleg. Tight Fault Locality. Technical Report CS94-24, The Weizmann Institute of Science, 1994.

[KP95] S. Kutten and D. Peleg. Fault-Local Distributed Mending, In *Proc. 15th ACM Symp. on Principles of Distributed Computing*, August 1995, 20–27.

[KP96] Shay Kutten and Boaz Patt-Shamir. Time- adaptive self stabilization. Manuscript.

[L92] N. Linial, Locality in distributed graph algorithms, *SIAM J. Comput.*, 21:193–201, 1992.

[LPRS93] N. Linial, D. Peleg, Y. Rabinovich and M. Saks, Sphere Packing and Local Majorities in Graphs, *Proc. 2nd Israel Symp. on Theory of Computing and Systems*, Natanya, Israel, June 1993, IEEE Comp. Soc. Press, 141–149.

[L86] Michael Luby, A simple parallel algorithm for the maximal independent set problem, *SIAM J. Comput.*, 15(4):1036–1053, November 1986.

[L80] Nancy A. Lynch, Fast Allocation of Nearby Resources in a Distributed System, *Proc. 12th ACM Symp. on Theory of Computing*, April 1980.

[MNS95] A. Mayer, M. Naor and L. Stockmeyer, Local Computations on Static and Dynamic Graphs, *Proc. 3rd Israel Symp. on Theory of Computing and Systems*, 1995.

[MRR80] J. M. McQuillan, I. Richer and E. C. Rosen, The New Routing Algorithm for the ARPANET. *IEEE Trans. Communications* Vol. COM-28, May 1980, pp. 711-719.

[NS93] M. Naor and L. Stockmeyer, What Can Be Computed Locally? In *Proc. 25th ACM Symp. on the Theory of Computing*, San Diego, California, May 1993, pp. 184-193.

[PS92] A. Panconesi and A. Srinivasan, Improved distributed algorithms for coloring and network decomposition problems, *Proc. 33rd Symp. on Theory of Computing*, 1992, 581–592.

[P95] D. Peleg, The power of small Coalitions in graphs, Technical Report CS95-12, the Weizmann Institute of Science, 1995. (To appear in *Proc. 2nd Colloq. on Structural Information & Communication Complexity*, 1995.)

[R81] E. C. Rosen, Vulnerability of Network Control Protocols: an Example. Computer Communications Review, July 1981. Also appeared as Internet RFC 789.

[T83] R.E. Tarjan, *Data Structures and Network Algorithms*, SIAM, Philadelphia, 1983.