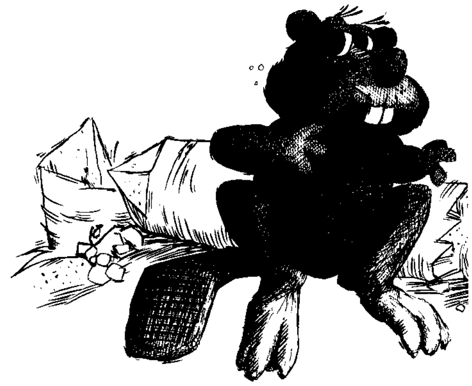
UNIVERSITY OF WATERLOO COMPUTER SCIENCE DEPARTMENT



Local Correction of Mod(k) Lists

I.J. Davis
D.J. Taylor

Data Structuring Group
CS-85-55

December, 1985

Local correction of mod(k) lists

Ian J. Davis
David J. Taylor

Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada

ABSTRACT

A mod(k) list, is a robust double-linked list in which each back pointer has been modified so that it addresses the *k'th* previous node. This paper presents a new algorithm for performing local correction in a mod($k \geq 3$) list. Given the assumption that at most two errors are encountered during any single correction step, this algorithm performs correction whenever possible, and otherwise reports failure. The algorithm generally reports failure only if the instance being corrected has a disconnected node. However, in a mod(3) structure one specific type of damage that causes disconnection is indistinguishable from alternative damage that does not. This also causes the algorithm to report failure.

1. Introduction:

A modified(k), or mod(k), storage structure is a circular double-linked list of nodes, in which each node contains a forward pointer that links it to the next node, and a back pointer that links it to the *k'th* previous node. Each node also contains an identifier field that identifies it as belonging to a specific instance of a mod(k) structure. A count of the number of nodes in the instance is also present. A mod(k) structure has k consecutive headers that allow access to the instance [2].

While a mod(1) structure is not locally correctable [1], single errors can be corrected within it [2,3]. A mod(2) structure is locally correctable, if errors are sufficiently distant from each other. However, if two errors occur close together it may be impossible to distinguish them from a different error. In addition two errors may cause some nodes to be reachable only via forward pointers. For these reasons this paper concentrates on mod($k \geq 3$) structures.

The local correction algorithm described in this paper for a mod($k \geq 3$) structure starts at headers whose addresses are assumed to be correct, and proceeds backwards through the entire instance iteratively identifying the previous node. At each step, paths expected to lead to this previous node contribute a weighted vote to the nodes they address. The addressed nodes receive additional weighted votes when paths proceeding from them appear correct. These votes, and occasionally additional information obtained from the structure being corrected, allow the previous node to be identified whenever possible.

2. Definitions

Each node in a mod(k) structure contains identifier, back pointer, and forward pointer *components*. A count component exists in one of the header nodes. An *error* is an erroneous value in one such component.

Initially it is assumed that the addresses of the header nodes within the *instance* being considered are known and can therefore be *trusted*. As correction proceeds, components of the instance become trusted. Any node addressed by a trusted component is trusted. However, trusted nodes may initially contain untrusted components.

At any correction step, the node that should immediately precede the trusted nodes will be called the *target*. The target is *disconnected* if no correct pointer in the instance addresses it. Components examined in attempting either to identify the location of the target, or to detect that it is disconnected, define the current *locality*. Local correction requires that the number of untrusted components in any locality be bounded by a constant. When the target has been identified, the identifier and forward pointer in the target, and the back pointer that should address this target, are components of trusted nodes. Since the correct values of these components are known, the values of these components can be corrected if erroneous. Once correct, these components and the target become trusted.

Within the locality, nodes will be labelled N and subscripted by the correct forward distance from them to the *last* trusted node. The last trusted node is therefore N_0, while earlier trusted nodes have negative subscripts. Forward pointers will be labelled F and back pointers will be labelled B. Pointers will be subscripted by the node number that they reside in. The notation B_x/F_{x+k} refers to either pointer B_x or pointer F_{x+k} but not both.

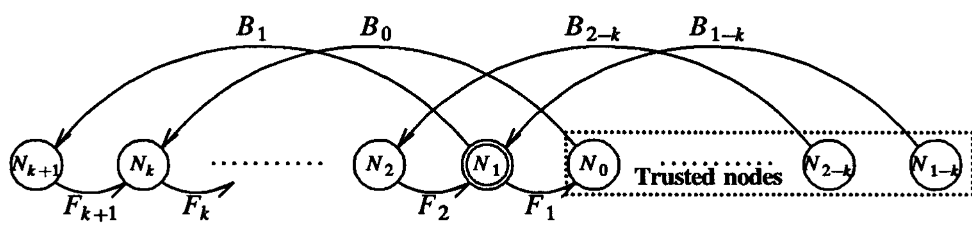


Fig 1. A correct mod(k) locality

One method of attempting to identify the target is to use *votes* [1]. Each *constructive* vote is a function which follows a path from a trusted node and returns a *candidate* for consideration as the target. Constructive votes are labelled C. Each *diagnostic* vote is a predicate which when presented with a candidate, assumes that this candidate is the target, examines a path proceeding from this candidate, and returns true if the path appears correct. Diagnostic votes are labelled D. A candidate receives the *support* of each constructive vote that returns it, and each diagnostic vote which returns true when presented with it. A *weighted vote* associates a constant non-negative *weight* with all candidates that it supports. The weight assigned to a vote X will be labelled $\bar{X}$. Each candidate *receives a vote* equal to the sum of all weights associated with it. If the candidate is not the target then it is an *incorrect* candidate. Votes are *distinct* if they cannot support the same candidate as a result of using a common component. The following weighted votes are used in this paper:

Vote	Pointers followed	Compared against
C_1	B_{1-k}	
$C_{2\leq i\leq k}$	$B_{i-k}F_i \ldots F_2$	
D_1	F_1	N_0
$D_{2\leq i\leq k}$	$B_1F_{k+1} \ldots F_{i+1}$	B_{i-k}

For notational convenience the set of votes $C_{2\leq i\leq k}$, will be referred to as C_0. Similarly, the set of votes $D_{2\leq i\leq k}$, will be referred to as D_0.

3. Theoretical results

It is assumed throughout this section that at most two errors occur in any single locality, and that the Valid State Hypothesis [3] holds. This asserts that in the absence of errors, identifiers and pointers within the instance being corrected contain information that differs from information occurring at the same offset in other nodes within the node space. Without some assumption about the number of errors occurring in a locality, and the number of errors seen when invalid components are examined, little can be said about the behaviour of any local correction algorithm.

Theorem 1 shows how an algorithm can detect and correct up to two errors in the empty instance. Theorem 2 establishes some necessary constraints that weights must satisfy if the target is to receive a vote of at least some constant value, and incorrect candidates are to receive a vote of at most this constant value. Throughout this paper this constant is arbitrarily assumed to be one-half. Theorem 3 shows that these constraints are sufficient to ensure that the target does receive a vote of at least one-half, and to ensure that incorrect candidates receive a vote of at most one-half, if distinct from the last k trusted nodes. Theorem 4 identifies voting weights that minimise the occasions when the target receives a vote of exactly one-half. Theorem 5 specifies when disconnection of the target can be suspected, and in all but one case determined. Theorem 6 demonstrates how the target can be identified in all other cases. Collectively, these results can be used to construct a simple, efficient algorithm, that performs local correction whenever possible.

Theorem 1

If an instance of a mod($k\geq 3$) structure contains at most two errors, it can be determined if this instance is empty. Having determined that an instance is empty, any errors in the instance can be trivially corrected.

Proof

In a mod($k\geq 3$) instance $k+2\geq 5$ components indicate when the instance is empty. Specifically, the back pointer in each of the k header nodes points back zero nodes, the forward pointer in the earliest header addresses the last header, and the count is zero. Given at most two errors, the instance is therefore empty iff at least three of these components indicate that the instance is empty. □

Comment: Since the empty instance is correctable, subsequent theorems may assume that the instance being corrected is not empty.

Theorem 2

If a connected target is always to receive a vote of at least one-half, and any incorrect candidate is always to receive a vote of at most one-half, whenever at most two errors occurs in any locality within a mod($k \geq 3$) structure, it is necessary that the voting weights satisfy the following inequalities.

$$1)\ \overline{C}_1 = \overline{D}_1 = \overline{C}_0 = \overline{D}_0 = 1/4$$

$$2)\ \overline{C}_i + \overline{D}_i \leq 1/4\text{, for } 2 \leq i \leq k$$

$$3)\ \sum_{j=i}^{k} \overline{C}_j + \sum_{j=2}^{i-1} \overline{D}_j \leq 1/4\text{, for } 3 \leq i \leq k$$

Proof

Damaging any two of $\{B_{1-k}, F_1, F_2, B_1\}$ causes the corresponding two votes in the set $\{C_1, D_1, C_0, D_0\}$ to fail to support the target. This leaves only the other two votes supporting the target. Damaging two of $\{B_{1-k}, F_n, F_2, B_n\}$ appropriately causes the corresponding two votes in the set $\{C_1, D_1, C_0, D_0\}$ to support an incorrect candidate N_n. Since the target is required to receive a vote of at least one-half, and incorrect candidates are required to receive a vote of at most one-half, it follows that any pair of the above votes must necessarily have weights that sum to one-half. Solving gives $\overline{C}_1 = \overline{C}_0 = \overline{D}_1 = \overline{D}_0 = 1/4$.

Suppose that one of $B_{2-k \leq i \leq 0}$ is damaged. Then the target loses the support of votes C_i and D_i. If C_i and D_i had weights that summed to more than one-quarter, the target would be left receiving a vote of less than one-half when F_1 was also damaged. Since it is required that the target receives a vote of at least one-half, it is therefore necessary that $\overline{C}_i + \overline{D}_i \leq 1/4$, for $2 \leq i \leq k$.

Now suppose that one of $F_{3 \leq i \leq k}$ is damaged. Then the target loses the support of all votes $C_{i \leq j \leq k}$ and $D_{2 \leq j \leq i-1}$. If these votes had weights that summed to more than one-quarter, the target would again receive a vote of less than one-half when F_1 was also damaged. Thus it is necessary that $\sum_{j=i}^{k} \overline{C}_j + \sum_{j=2}^{i-1} \overline{D}_j \leq 1/4$, for $3 \leq i \leq k$. □

Theorem 3

If no more than two errors occur in any locality within a mod($k \geq 3$) structure; the instance being corrected is not empty; forward pointers are corrected when this first becomes possible; and votes are modified so that they do not support any of the last k trusted nodes, then the constraints imposed on voting weights in Theorem 2 ensure that (1) the target receives a vote of at least one-half, and (2) incorrect candidates receive a vote of at most one-half.

Proof of (1)

Since the instance is not empty, the target is distinct from the last k trusted nodes. Thus, modifying votes so that they cannot support any of the last k trusted nodes leaves the vote for the target unchanged. Since $\overline{C}_1 = \overline{D}_1 = \overline{C}_0 = \overline{D}_0 = ¼$, damaging any of $\{B_{1-k}, F_1, F_2, B_1/F_{k+1}\}$ removes a vote of one-quarter from the target. Since $\overline{C}_i + \overline{D}_i \leq ¼$ for $2 \leq i \leq k$, damaging any other back pointer in the locality removes a vote of at most one-quarter from the target. Since $\sum_{j=i}^{k} \overline{C}_j + \sum_{j=2}^{i-1} \overline{D}_j \leq ¼$ for $3 \leq i \leq k$, damaging any other forward pointer in the locality removes a vote of at most one-quarter from the target. When multiple errors occur in the locality the target loses the support of at most those votes containing errors. Thus if two errors occur in the locality the target loses the support of at most two sets of votes each having weights that sum to at most one-quarter. Since all weights sum to one, the target therefore receives a vote of at least one-half. □

Proof of (2)

Suppose that C_1 supports an incorrect candidate N_n, which is therefore distinct from the last k trusted nodes. Then there is an error in B_{1-k}. B_{1-k} is distinct from B_n since N_n is not a trusted node, and inductively B_{1-k} is distinct from $B_{2-k \leq i \leq 0}$. Thus an error in B_{1-k} causes only C_1 to support N_n. Thus C_1 is distinct from all other votes.

Suppose that D_1 and some $C_{2 \leq i \leq k}$ support N_n, as a result of both using F_n. Then F_n addresses the last trusted node. If forward pointers have been repaired as early as possible, at least the last $k-1$ forward pointers in the trusted set are correct, since $k-1$ forward pointers can be corrected in the headers during initialisation. All pointers followed by C_i, after C_i uses F_n, are therefore correct. This implies that C_i supports one of the last k trusted nodes, contradiction. Thus D_1 is distinct from C_0.

Now suppose that D_1 and some $D_{2 \leq i \leq k}$ support N_n, as a result of both using F_n. Since the instance being examined is not empty, some other distinct error must exist in components used by D_i in supporting N_n, for D_i to use F_n. After using F_n, D_i can follow at most $k-i$ forward pointers. Thus D_i addresses one of the trusted nodes N_0 through N_{i-k}. Since D_i supports N_n, B_{i-k} must also address this node. No error can exist in B_{i-k} since two distinct errors exist in pointers followed by D_i, and B_{i-k} is distinct from both of these pointers. Since the instance is not empty B_{i-k} therefore points back between 1 and $k-2$ nodes. But B_{i-k} correctly points back k nodes, contradiction. Thus D_1 is distinct from D_0.

The above demonstrates that C_1 and D_1 are distinct from all other votes. If C_1 and D_1 support N_n, they contain two distinct errors, and these errors cause no other vote to support N_n. In this case N_n receives a vote of one-half, since $\overline{C}_1 = \overline{D}_1 = ¼$. If neither C_1 nor D_1 support N_n, then N_n receives a vote of at most one-half, since $\overline{C}_0 = \overline{D}_0 = ¼$. Thus if N_n is to receive a vote of more than one-half, it must receive the support of one of C_1 or D_1, and a single independent error must cause N_n to receive the support of votes that sum to more than one-quarter.

If a single error occurs in a back pointer $B_{2-k\leq i\leq 0}$ then C_{i+k} and D_{i+k} may support N_n, but no other vote can, since back pointers within the locality are distinct. Theorem 2 has established that $\overline{C}_i+\overline{D}_i\leq ¼$, for $2\leq i\leq k$. Thus such an error cannot cause N_n to receive a vote of more than one-quarter.

So suppose that a single error in a forward pointer F_x causes votes supporting N_n to sum to more than one-quarter. Then it must cause some $C_{2\leq i\leq k}$, and some $D_{2\leq j\leq k}$ to support N_n, since $\overline{C}_0=\overline{D}_0=¼$. Since N_x is correctly addressed by the path used by C_i, N_x lies within the instance. If N_n lies outside the instance, and the Valid State Hypothesis holds, then inductively no correct path from N_n addresses a node within the instance. But the path used by D_j in supporting N_n correctly passes through N_x which lies within the instance. Thus N_n lies within the instance.

Since an error occurs in F_x, N_x is not one of the last $k-1$ trusted nodes. Since D_j correctly passes through F_x in supporting N_n, and N_n is not one of the last k trusted nodes, N_n lies strictly between N_x and N_0. Since C_i uses F_x and supports N_n, N_n also lies between N_0 and N_x, contradiction. Thus no single error can cause N_n to receive a vote of more than one-quarter. □

Theorem 4

If weights satisfying the requirements of Theorem 2 are used, then in a mod($k\geq 3$) structure damaging two of $\{B_{1-k}, F_1, F_2, B_1/F_{k+1}\}$ causes the target to receive a vote of one-half. In a mod(3) structure damaging two of $\{B_{1-k=-2}, B_{-1}, B_0, F_1\}$, also causes the target to receive a vote of one-half. The weights $\overline{C}_1=\overline{D}_1=¼$; $\overline{C}_2=\overline{D}_k=3/16$; and $\overline{C}_3=\overline{D}_{k-1}=1/16$, satisfy the requirements of Theorem 2, and ensure that the target receives a vote of more than one-half in all other cases.

Proof

For an error to remove a vote of one-quarter from the target, it must damage all non-zero votes in one of the expressions in Theorem 2 that sum to one-quarter. The target receives a vote of exactly one-half when two errors are introduced into the locality, and each independently removes a vote of one-quarter from the target. Because $\overline{C}_1=\overline{D}_1=\overline{C}_0=\overline{D}_0=¼$, damaging any two of $\{B_{1-k}, F_1, F_2, B_1/F_{k+1}\}$ therefore removes a vote of one-half from the target.

In a mod(3) structure, Theorem 2 has established that $\overline{C}_2+\overline{D}_2\leq ¼$; $\overline{C}_3+\overline{D}_3\leq ¼$; $\overline{C}_2+\overline{C}_3\leq ¼$; and $\overline{D}_2+\overline{D}_3\leq ¼$. Collectively these inequalities imply that $\overline{C}_2+\overline{D}_2= ¼$, and $\overline{C}_3+\overline{D}_3= ¼$. Thus in a mod(3) structure damaging any two of $\{B_{-2}, B_{-1}, B_0, F_1\}$ also removes a vote of one-half from the target.

Assume that the weights proposed are used. Then the only equations that sum to one-quarter in Theorem 2 are those identified above as necessarily summing to one-quarter. Since $\overline{C}_2$, $\overline{C}_3$, $\overline{D}_{k-1}$, and $\overline{D}_k$ are each non-zero, the single errors that cause the target to lose a vote of one-quarter in a mod($k\geq 4$) structure occur only in $\{B_{1-k}, F_1, F_2, B_1/F_{k+1}\}$.

In a mod(3) structure the single errors that cause the target to lose a vote of one-quarter occur in $\{B_{-2}, B_{-1}, B_0, F_1, F_2, B_1/F_4\}$. The target receives a vote of more than one-half when one of $\{F_2, B_1/F_4\}$ and one of $\{B_{-1}, B_0\}$ are damaged.

Thus if the proposed weights are used, then the target receives a vote of one-half only under the types of damage suggested. □

Theorem 5

In a mod(3) structure, damage that causes B_{-1} to address N_1, and B_0 to address N_2, is indistinguishable from damage that causes B_{-2} to address N_2, and F_2 to address N_0. Thus it cannot alway be determined if the target is connected.

However, if the weights proposed in Theorem 4 are used, nodes contain identifier components, and at most two errors occur in any locality, then in all other cases it can be determined if the target is connected.

Proof

If all candidates receive a vote of less than one-half then the target must be disconnected, since Theorem 3 ensures that the target receives a vote of at least one-half. Conversely, if any candidate receives a vote of more than one-half this must be the target, since Theorem 3 ensures that no incorrect candidate receives such a vote. So assume that no candidate receives a vote of more than one-half, but some candidate receives a vote of exactly one-half. Then either this is the only candidate or multiple candidates exist. These cases are addressed separately.

Single candidate: If all constructive votes agree on a common candidate N_n, and N_n receives a vote of one-half, then N_n receives no diagnostic votes. Thus either N_n is the target and both F_1 and B_1/F_{k+1} have been damaged, or B_{1-k} and F_2 address an incorrect candidate. In either case the identifier field in the candidate addressed must be unchanged, since at most two errors exist in the locality. Thus if the node addressed lies outside the instance this can be immediately detected, and disconnection reported.

Suppose instead that N_n lies within the instance. Consider following B_n, and then k forward pointers. If N_n is the target, then since F_1 and B_1/F_{k+1} are damaged and represent the only damage in the locality, this path must either arrive at some node other than N_n, or arrive back at N_n prematurely. Conversely, if N_n is an incorrect candidate, but clearly not a trusted node since it receives a vote of one-half, then all pointers used in the above path are correct. Since N_n lies within the instance, this path must address N_n without passing through N_n. These tests can therefore be used to detect disconnection when all constructive votes agree on a common candidate.

Multiple candidates: If the target is disconnected and constructive votes do not all agree on a common candidate, then B_{1-k} and F_2 must address distinct incorrect candidates or address no node. Since it is assumed that some candidate N_n receives a vote of one-half, N_n must receive a vote of one-quarter from diagnostic votes. For N_n to receive a vote of one-quarter from D_0, either B_n/F_{n+k} or both B_0 and B_{-1}/F_{n+k-1} must be damaged. But these pointers are distinct from B_{1-k} and F_2, since N_n is not a trusted node. This implies that three errors exist in the locality contradicting the assumption that at most two errors occur in any locality. Thus the diagnostic vote must come from D_1.

For D_1 to support an incorrect candidate N_n, F_n must contain an error that causes it to address N_0. Since F_2 is the only erroneous forward pointer in the locality, N_n must be N_2. Since F_2 addresses N_0, C_0 does not support N_2. Thus C_1 does. The statement of the theorem has acknowledged that if this occurs in a mod(3) structure, then it cannot be determined if the target is connected. However, for a mod($k \geq 4$) structure in this case B_{4-k} is consistent with pointers B_{2-k} and B_{3-k} if and only if disconnection occurs. □

Theorem 6

If the conditions of Theorem 5 are satisfied, and it has been determined that the target is connected as described in Theorem 5, then the target can always be identified.

Proof

If the target is the only candidate, or receives a vote greater than any other candidate, then the target is trivially identifiable. For an incorrect candidate N_n to receive the same vote as the target, both must receive a vote of one-half. Theorem 4 has established that the target receives a vote of one-half only if two of $\{B_{1-k}, F_1, F_2, B_1/F_{k+1}\}$ are damaged, or in a mod(3) structure if two of $\{B_{1-k=-2}, B_{-1}, B_0, F_1\}$ are damaged.

Suppose that constructive votes not supporting the target disagree. Then two distinct pointers used by correct constructive votes must be damaged. Thus either B_{1-k} and F_2 are damaged, or in a mod(3) structure two of $\{B_{-2}, B_{-1}, B_0\}$ are damaged. In the first case the target is disconnected, while in the second each invalid candidate receives a vote of less than one-half. Thus an incorrect candidate N_n receives a vote of one-half only if all constructive votes not supporting the target support this candidate.

Since N_n is a incorrect candidate it must be supported by at least one constructive vote. Thus one of $\{B_{1-k}, B_{-1}, B_0, F_2\}$ must be damaged. If no other error exists in the locality then N_n receives a vote of one-quarter. Thus a second error in the locality must cause additional votes to support N_n whose weights sum to one-quarter.

Suppose that a second error occurs in F_1. Then N_n receives a vote of at most one-quarter from constructive votes, since F_1 is not used by correct constructive votes. D_1 cannot support any candidate, since neither F_1 nor F_2 address N_0. Since N_n receives a vote of one-half, all non-zero votes in D_0 must therefore support N_n. For this to occur either B_n/F_{n+k}, or both B_0 and B_{-1}/F_{n+k-1} must be damaged. B_n is correct since N_n is not one of the last k trusted nodes, and only two errors occur in the locality. B_0 and B_{-1} cannot both be damaged since it is assumed that an error occurs in F_1. One of $\{F_{n+k}, F_{n+k-1}\}$ therefore contains an error and is thus one of $\{F_1, F_2\}$. However, in this case B_n correctly addresses one of $\{N_1, N_2, N_3\}$. This implies that N_n is one of the last k trusted nodes, which it is not. Thus if any incorrect candidate receives the same vote as the target, F_1 must be correct.

If F_n does not address N_0, then since F_1 must, the target can be immediately identified. So suppose that both F_1 and F_n address N_0. Since F_n is distinct from F_1 it contains an error. Since only two errors exist in the locality,

F_n must therefore be either F_2 or F_{k+1}. F_n cannot be F_2 since an erroneous F_n addresses N_0 while an erroneous F_2 address N_n, which is distinct from N_0. Thus F_n is F_{k+1}, implying that N_n is N_{k+1}. The two errors in the locality thus occur in F_{k+1} and one of $\{B_{1-k}, B_{-1}, B_0, F_2\}$. B_1 and B_{k+1} are therefore correct, since N_{k+1} is not a trusted node. B_1 therefore addresses the incorrect candidate N_{k+1}. B_{k+1} however does not address the target, since N_n is not the trusted node N_{1-k}. Thus if F_n and F_1 address N_0, the candidate whose back pointer addresses the other candidate must be the target. □

4. Conclusions

The above material provides some constructive foundations for investigating the local correctability of an arbitrary structure. Much remains to be explored. For structures that are at all complex it is very difficult to visualise how correction can be safely undertaken [5]. Empirical results in the appendix suggest that the algorithm presented in this paper is superior to previous mod(k) local correction algorithms, when applied to mod($k \geq 3$) structures. Since the algorithm presented here cannot correct all structures corrected by these other algorithms, these other algorithms are still valuable.

APPENDIX

Empirical results

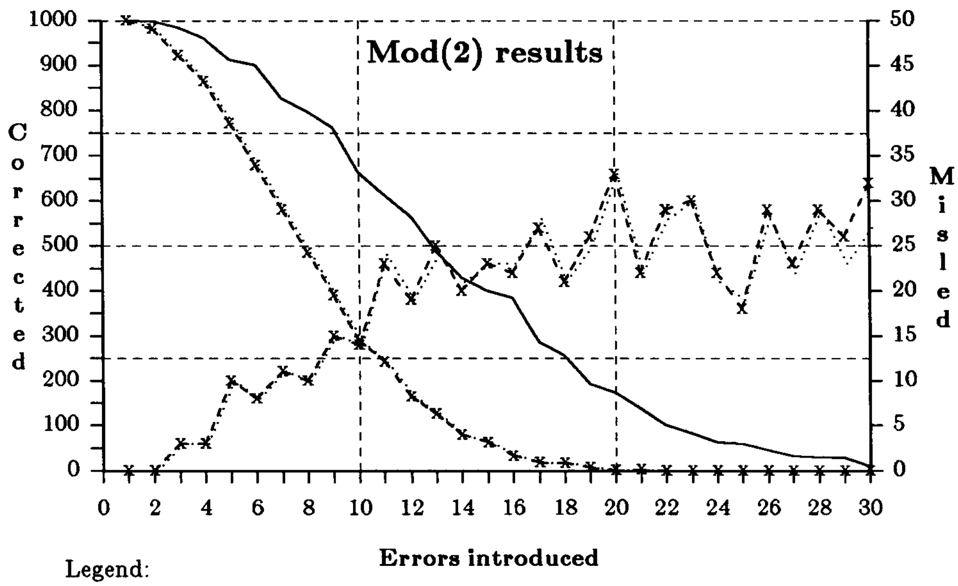
1. Explanation

This appendix presents empirical results obtained when "random" errors were introduced into an instance of a mod(2) structure, a mod(3) structure, and a mod(4) structure. Each instance contained 100 consecutively located nodes plus headers. Increasing numbers of pointers were randomly selected from within this instance, and modified by adding or subtracting a random number between 1 and 10.

For the mod(2) instance correction was attempted using a previous mod(k) local correction algorithm[1], the mod(2) local correction algorithm presented in [4], and the spiral local correction algorithm presented in [1]. For the mod(3) and mod(4) instances correction was attempted using the mod(k) local correction algorithm, and the local correction algorithm presented in this paper.

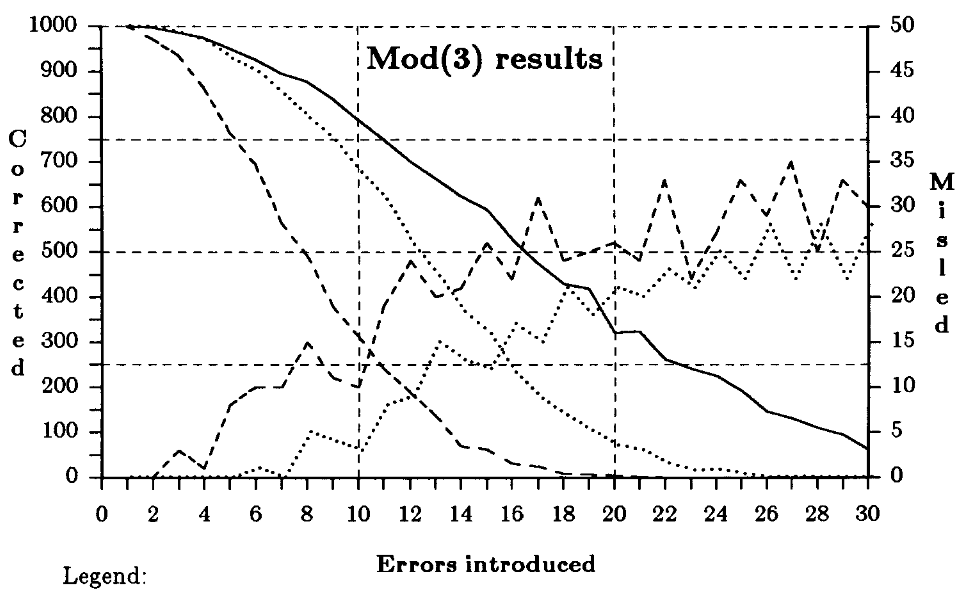
Each algorithm was executed on exactly the same "randomly" damaged instances. Each test was performed 1000 times before the number of pointers being damaged was increased. Statistics were collected on the number of times that the damaged instance remained connected, and was thus potentially correctable. Statistics were also collected on the number of times each algorithm was able to correct the structure, and the number of times that each algorithm was misled into attempting to apply an incorrect change.

[1] This algorithm used the voting scheme $\bar{C}_1 = \bar{C}_2 = \bar{D}_1 = 1/3$, and always corrects one error in this smaller locality, within any mod($k \geq 2$) structure.



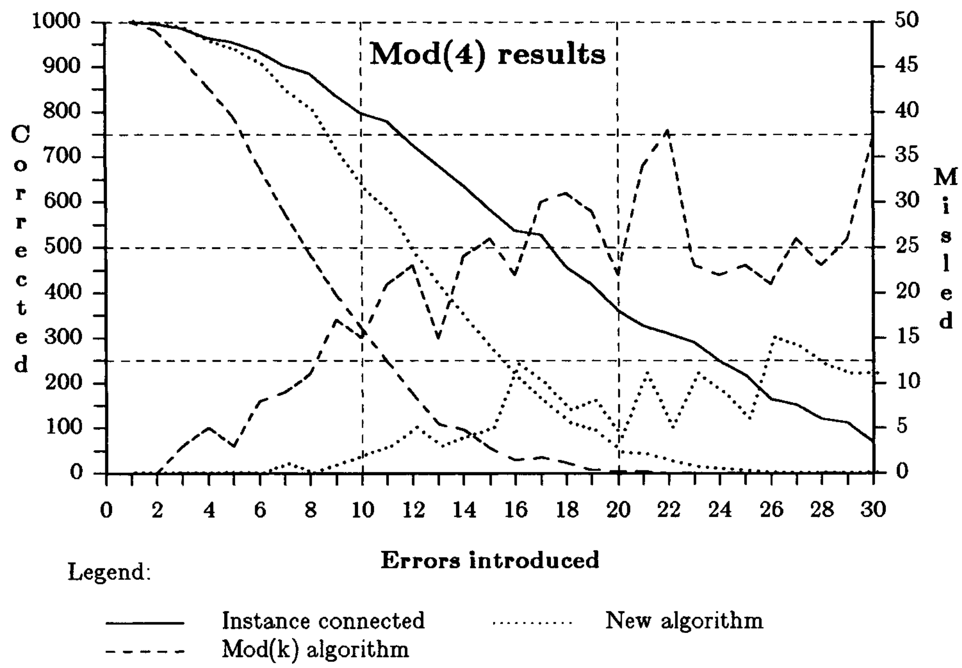
Legend:

———	Instance connected	··········	Mod(2) algorithm
- - - - -	Mod(k) algorithm	x x	Spiral algorithm



Legend:

———	Instance connected	··········	New algorithm
- - - - -	Mod(k) algorithm		



2. Comments

While the behaviour of the mod(k) correction algorithm is similar to the spiral correction algorithm, and to a lesser extent the correction algorithm presented in this paper, the mod(2) correction algorithm is quite different, since it uses two parallel traversals of the instance, and an elaborate fault dictionary to assist in correction. It is therefore surprising that the results of attempting to correct a mod(2) instance are almost identical, regardless of the algorithm used.

Under the various errors introduced, the mod(2) structure remained connected 44% of the time, the mod(3) structure 55% of the time, and the mod(4) structure 60% of the time. The mod(k) correction algorithm corrected 26% of errors regardless of the structure presented to it.

Superficially it appears that the local correction algorithm presented in this paper should correct more errors in a mod($k \geq 4$) structure than in a mod(3) structure. However the locality, in which it is assumed that at most two errors occur, is smaller in a mod(3) structure than in a mod($k \geq 4$) structure, and this becomes significant when many errors are introduced into the instance being corrected. It is therefore not surprising that this algorithm corrected 40% of errors in mod(3) instances, and 38% of errors in mod(4) instances.

The statistics presented above are very dependent on the number of errors introduced into the instance, the type of error introduced, and the size of the instance being damaged. However, these statistics provided some assurance that the algorithm presented in this paper is indeed superior to algorithms previously presented, when applied to a mod($k \geq 3$) structure.

Acknowledgements

The authors are indebted to Dr. A. R. Crowe, and Dr. J. P. Black for the interest that they showed in this research, their encouragement, and the contribution that they made to the final format of this paper. This research was supported, in part, by the Natural Sciences and Engineering Research Council of Canada under grant A3078.

References

1. J. P. Black and D. J. Taylor, Local correction in robust storage structures, CS-84-44, Dept. of Computer Science, University of Waterloo (December 1984). Accepted for publication in *IEEE Transactions on Software Engineering*.
2. D. J. Taylor, D. E. Morgan, and J. P. Black, Redundancy in data structures: Improving software fault tolerance, *IEEE Transactions on Software Engineering* **SE-6**(6) pp. 585-594 (November 1980).
3. D. J. Taylor, D. E. Morgan and J. P. Black, Redundancy in data structures: Some theoretical results, *IEEE Transactions on Software Engineering* **SE-6**(6) pp. 595-602 (November 1980).
4. D. J. Taylor and J. P. Black, Principles of data structure error correction. *IEEE Transactions on Computers* **C-31**(7) pp. 602-608 (July 1982).
5. D. J. Taylor and J. P. Black, Guidelines for storage structure error correction, *Digest of papers: Fifteenth Annual International Symposium on Fault Tolerant Computing*, pp. 20-22 (19-21 June 1985).