J. SYSTEMS SOFTWARE
1990; 11: 205–214

Local Correction of Mod(k) Lists

I. J. Davis and D. J. Taylor
Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada

A mod(k) list is a robust double-linked list in which each back pointer has been modified so that it addresses the kth 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 both pointers addressing a specific node have been damaged, causing this node to become disconnected. 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 [1, 2, 8] 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 kth previous node. A particular *instance* of a mod(k) structure consists of k consecutive *header* nodes, whose addresses are known, and all nodes reachable by following pointers from these header nodes. These header nodes are contained within the double-linked list of nodes, and are the only nodes in the instance when the instance is *empty*. Each node within an instance contains an *identifier* whose value uniquely identifies the instance to which the node belongs. A *count* of the number of non-header nodes within an instance is stored in one of the header nodes of the instance. An *error* is an incorrect value in a single pointer, identifier, or count *component* [9].

Although a mod(k) structure contains considerable redundancy, a small number of well-chosen errors can produce an instance which is not correctable [10]. However, if we assume that erroneous components are distributed fairly evenly throughout the instance being corrected, a large number of errors can potentially be corrected. It is this assumption which is exploited by a local correction procedure [5, 6, 13, 14].

A local correction procedure visits all of the components of a storage structure instance in some deterministic order, by following pointers from the headers of the instance, and corrects errors when these are first encountered. Having ensured that a component is correct, this component becomes *trusted*. Errors are identified and corrected by examining previously trusted components, and at most some constant number of potentially erroneous *untrusted* components. This bounded set of untrusted components forms a *locality* which is assumed to contain at most some constant number of errors. Informally, these are the constraints that are imposed on a local correction procedure. More precise characterizations of such procedures [3] are too complex to be attempted here.

The local correction procedure described in this paper operates under the assumption that at most two errors occur in any locality. When presented with a set of header nodes, it proceeds backwards from these header nodes through the mod(k) instance, iteratively attempting to identify the correct address of the previous node. This previous node is called the *target.*

Having established the location of the target, the back pointer that should address this target can be corrected, as can the forward pointer and identifier in this target. Having performed any necessary corrections, these components become trusted, and the target node becomes the *last* trusted node. Alternatively, having established that no correct pointer addresses the target, it can be reported that the target node is *disconnected.*

2. TERMINOLOGY

Nodes will be labeled N and subscripted by the correct forward distance from them to the last trusted node. The last trusted node is therfore N_0, while earlier trusted nodes have negative subscripts. The target node is always N_1.

Back pointers will be labeled b and forward pointers f with subscripts indicating the correct distance spanned

Address correspondence to I. J. Davis, Department of Computer Science, University of Waterloo, Waterloo, Ontario, N2L 3G1 Canada.

0164-1212/90/$3.50

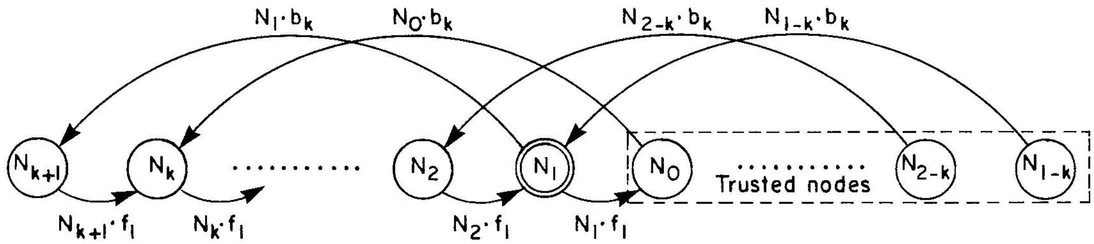


Figure 1. A correct mod(k) locality.

by these pointers. Pointers will be prefixed by the node in which they reside, or by extension a path that addresses them. When appropriate, superscripts will indicate the number of consecutive occurrences of a pointer type within a path. $N_x \cdot b_k / N_{x+k} \cdot f_1$ represents exactly one of $N_x \cdot b_k$ and $N_{x+k} \cdot f_1$. Figure 1 illustrates this notation, by showing a locality in a mod(k) list.

When explicitly discussing the k header nodes these will be labeled H. In a correct instance, $H_i \cdot f_1 = H_{i-1}$ for $k > i > 0$. If the instance is not empty, then $H_0 \cdot f_1$ addresses the first non-header node, and $H_0 \cdot b_k$ the last non-header node. Otherwise, in an empty instance, $H_0 \cdot f_1 = H_{k-1}$ and $H_i \cdot b_k = H_i$ for $k > i \geq 0$. This is illustrated in Figure 2.

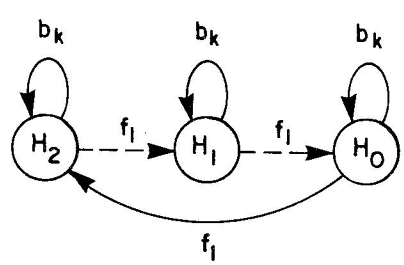
One method of attempting to identify the target is to use *votes* [3]. Each *constructive* vote is a function which follows a path from a trusted node and returns a *candidate* node N_n for consideration as the target. Constructive votes are labeled C and distinguished by subscripts. Each *diagnostic* vote is a predicate which when presented with a candidate node N_n, assumes that this candidate is the target node N_1, examines a path proceeding from this candidate, and returns true if this path appears correct. Diagnostic votes are labeled D, and also distinguished by subscripts.

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* is a vote which has associated with it a nonnegative constant called its *weight*. The weight assigned to a vote X will be labeled $\bar{X}$. Each candidate *receives a vote* equal to the sum of the weights of all votes which support 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	$N_{1-k} \cdot b_k$	
C_i, $2 \leq i \leq k$	$N_{i-k} \cdot b_k \cdot f_1^{i-1}$	
D_1	$N_n \cdot f_1$	N_0
D_i, $2 \leq i \leq k$	$N_n \cdot b_k \cdot f_1^{k-i+1}$	$N_{i-k} \cdot b_k$

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

Figure 2. An empty instance of a mod(k = 3) structure.



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 [10] 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 behavior of any local correction algorithm.

Theorem 1 shows how an algorithm can detect and correct up to two errors in the empty instance. Subsequently, it is assumed that the instance being corrected is not empty. Theorem 2 establishes some necessary constraints that weights must satisfy if the target is to received 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 arbitrarity 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 received a vote of at most one-half, if distinct from the last k trusted nodes. Theorem 4 identifies voting weights that minimize the

J. SYSTEMS SOFTWARE
1990; 11: 205-214

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, presented in Appendix A, that performs local correction of mod($k \geq 3$) linked lists 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 header node H_0 addresses the last header node H_{k-1}, and the count is zero. For the mod(3) structure, this is shown in Figure 2. Given at most two errors, the instance is therefore empty if and only if at least three of these components indicate that the instance is 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.

$$\bar{C}_1 = \bar{D}_1 = \bar{C}_0 = \bar{D}_0 = 1/4 \tag{1}$$

$$\bar{C}_i + \bar{D}_i \leq 1/4, \qquad \text{for } 2 \leq i \leq k \tag{2}$$

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

Proof. Damaging any two of $\{N_{1-k} \cdot b_k, N_1 \cdot f_1, N_2 \cdot f_1, N_1 \cdot b_k\}$ 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 $\{N_{1-k} \cdot b_k, N_n \cdot f_1, N_2 \cdot f_1, N_n \cdot b_k\}$ 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 $\bar{C}_1 = \bar{C}_0 = \bar{D}_1 = \bar{D}_0 = 1/4$.

Suppose that $N_{i-k} \cdot b_k$ is damaged, for some $2 \leq i \leq k$. 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 $N_1 \cdot f_1$ was also damaged. Since it is required that the target receive a vote of at least one-half, it is therefore necessary that $\bar{C}_i + \bar{D}_i \leq 1/4$, for $2 \leq i \leq k$.

Now suppose that $N_i \cdot f_1$ is damaged, for some $3 \leq i \leq k$. 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 $N_1 \cdot f_1$ was also damaged. Thus, it it neccessary that

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

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 $\bar{C}_1 = \bar{D}_1 = \bar{C}_0 = \bar{D}_0 = 1/4$, damaging any of $\{N_{1-k} \cdot b_k, N_1 \cdot f_1, N_2 \cdot f_1, N_1 \cdot b_k/N_{k+1} \cdot f_1\}$ removes a vote of one-quarter from the target. Since $\bar{C}_i + \bar{D}_i \leq 1/4$ 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 $\Sigma_{j=1}^{k} \bar{C}_j + \Sigma_{j=2}^{i-1} \bar{D}_j \leq 1/4$ 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 $N_{1-k} \cdot b_k$ contains an error. $N_{1-k} \cdot b_k$ is distinct from $N_n \cdot b_k$ since N_n is not a trusted node, and inductively $N_{1-k} \cdot b_k$ is distinct from $N_i \cdot b_k$ for $0 \geq i \geq 2 - k$. Thus, an error in $N_{1-k} \cdot b_k$ 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 $N_n \cdot f_1$. Then $N_n \cdot f_1$ 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 $N_n \cdot f_1$, are therefore correct. This implies that C_i supports one of the last k trusted nodes—a contradiction. Thus, D_1 is distinct from C_0.

Now suppose that D_1 and some $D_{2 \le i \le k}$ support N_n, as a result of both using $N_n \cdot f_1$. 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 $N_n \cdot f_1$. After using $N_n \cdot f_1$, 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, $N_{i-k} \cdot b_k$ must also address this node. No error can exist in $N_{i-k} \cdot b_k$ since two distinct errors exist in pointers followed by D_i, and $N_{i-k} \cdot b_k$ is distinct from both of these pointers. Since the instance is not empty, $N_{i-k} \cdot b_k$ therefore points back between 1 and $(k - 2)$ nodes. But $N_{i-k} \cdot b_k$ correctly points back k nodes—a 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 $\bar{C}_1 = \bar{D}_1 = 1/4$. If neither C_1 nor D_1 support N_n, then N_n receives a vote of at most one-half, since $\bar{C}_0 = \bar{D}_0 = 1/4$. 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 $N_{i-k} \cdot b_k$, for some $2 \le i \le k$, then C_i and D_i may support N_n, but no other vote can, since back pointers within the locality are distinct. Such an error cannot cause N_n to receive a vote of more than one-quarter, since we require that $\bar{C}_i + \bar{D}_i \le 1/4$, for $2 \le i \le k$.

So suppose that a single error in a forward pointer $N_x \cdot f_1$ causes votes supporting N_n to sum to more than one-quarter. Then it must cause some $C_{2 \le i \le k}$, and some $D_{2 \le j \le k}$ to support N_n, since $\bar{C}_0 = \bar{D}_0 = 1/4$. 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 $N_x \cdot f_1$, N_x is not one of the last $(k - 1)$ trusted nodes. Since D_j correctly passes through $N_x \cdot f_1$ 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 supports N_n but follows only forward pointers after using the erroneous $N_x \cdot f_1$ pointer, N_n lies between N_0 and N_x—a 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($k \ge 3$) structure damaging two of $\{N_{1-k} \cdot b_k, N_1 \cdot f_1, N_2 \cdot f_1, N_1 \cdot b_k / N_{k+1} \cdot f_1\}$ causes the target to receive a vote of one-half. In a mod(3) structure damaging two of $\{N_{-2} \cdot b_3, N_{-1} \cdot b_3, N_0 \cdot b_3, N_1 \cdot f_1\}$ also causes the target to receive a vote of one-half. The weights $\bar{C}_1 = \bar{D}_1 = 1/4$, $\bar{C}_2 = \bar{D}_k = 3/16$; and $\bar{C}_3 = \bar{D}_{k-1} = 1/16$, 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 nonzero 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 $\bar{C}_1 = \bar{D}_1 = \bar{C}_0 = \bar{D}_0 = 1/4$, damaging any two of $\{N_{1-k} \cdot b_k, N_1 \cdot f_1, N_2 \cdot f_1, N_1 \cdot b_k / N_{k+1} \cdot f_1\}$ therefore removes a vote of one-half from the target.

In a mod(3) structure, we require that $\bar{C}_2 + \bar{D}_2 \le 1/4$, $\bar{C}_3 + \bar{D}_3 \le 1/4$, $\bar{C}_2 + \bar{C}_3 \le 1/4$, and $\bar{D}_2 + \bar{D}_3 \le 1/4$. Collectively, these inequalities imply that $\bar{C}_2 + \bar{D}_2 = 1/4$, and $\bar{C}_3 + \bar{D}_3 = 1/4$. Thus, in a mod(3) structure damaging any two of $\{N_{-2} \cdot b_3, N_{-1} \cdot b_3, N_0 \cdot b_3, N_1 \cdot f_1\}$ 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 $\bar{C}_2$, $\bar{C}_3$, $\bar{D}_{k-1}$, and $\bar{D}_k$ are each nonzero, the single errors that cause the target to lose a vote of one-quarter in a mod($k \ge 4$) structure occur only in $\{N_{1-k} \cdot b_k, N_1 \cdot f_1, N_2 \cdot f_1, N_1 \cdot b_k / N_{k+1} \cdot f_1\}$.

In a mod(3) structure, the single errors that cause the target to lose a vote of one-quarter occur in $\{N_{-2} \cdot b_3, N_{-1} \cdot b_3, N_0 \cdot b_3, N_1 \cdot f_1, N_2 \cdot f_1, N_1 \cdot b_3 / N_4 \cdot f_1\}$. The target receives a vote of more than one-half when one of $\{N_2 \cdot f_1, N_1 \cdot b_3 / N_4 \cdot f_1\}$ and one of $\{N_{-1} \cdot b_3, N_0 \cdot b_3\}$ 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 $N_{-1} \cdot b_3$ to address N_1, and $N_0 \cdot b_3$ to address N_2, is indistinguishable from damage that causes $N_{-2} \cdot b_3$ to address N_2, and $N_2 \cdot f_1$ 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

J. SYSTEMS SOFTWARE
1990; 11: 205–214

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 $N_1 \cdot f_1$ and $N_1 \cdot b_k/N_{k+1} \cdot f_1$ have been damaged, or $N_{1-k} \cdot b_k$ and $N_2 \cdot f_1$ 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, then this can be immediately detected, and disconnection reported. Suppose instead that N_n lies within the instance. Consider following $N_n \cdot b_k \cdot f_1^k$. If N_n is the target, then since $N_1 \cdot f_1$ and $N_1 \cdot b_k/N_{k+1} \cdot f_1$ 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 $N_{1-k} \cdot b_k$ and $N_2 \cdot f_1$ 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 $N_n \cdot b_k/N_{n+k} \cdot f_1$ or both $N_0 \cdot b_k$ and $N_{-1} \cdot b_k/N_{n+k-1} \cdot f_1$ must be damaged. But these pointers are distinct from $N_{1-k} \cdot b_k$ and $N_2 \cdot f_1$, 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, $N_n \cdot f_1$ must contain an error that causes it to address N_0. Since $N_2 \cdot f_1$ is the only erroneous forward pointer in the locality, N_n must be N_2. Since $N_2 \cdot f_1$ 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, $N_{4-k} \cdot b_k$ is consistent with pointers $N_{2-k} \cdot b_k$ and $N_{3-k} \cdot b_k$ 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 $\{N_{1-k} \cdot b_k, N_1 \cdot f_1, N_2 \cdot f_1, N_1 \cdot b_k/N_{k+1} \cdot f_1\}$ are damaged, or in a mod(3) structure if two of $\{N_{-2} \cdot b_3, N_{-1} \cdot b_3, N_0 \cdot b_3, N_1 \cdot f_1\}$ 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 $N_{1-k} \cdot b_k$ and $N_2 \cdot f_1$ are damaged, or in a mod(3) structure two of $\{N_{-2} \cdot b_3, N_{-1} \cdot b_3, N_0 \cdot b_3\}$ 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 an incorrect candidate it must be supported by at least one constructive vote. Thus, one of $\{N_{1-k} \cdot b_k, N_{-1} \cdot b_k, N_0 \cdot b_k, N_2 \cdot f_1\}$ 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 $N_1 \cdot f_1$. Then N_n receives a vote of at most one-quarter from constructive votes, since $N_1 \cdot f_1$ is not used by correct constructive votes. D_1 cannot support any candidate, since neither $N_1 \cdot f_1$ nor $N_2 \cdot f_1$ address N_0. Since N_n receives a vote of one-half, all nonzero votes in D_0 must therefore support N_n. For this to occur, either $N_n \cdot b_k/N_{n+k} \cdot f_1$, or both $N_0 \cdot b_k$ and $N_{-1} \cdot b_k/N_{n+k-1} \cdot f_1$ must be damaged. $N_n \cdot b_k$ is correct since N_n is not one of the last k trusted nodes, and only two errors occur in the locality. $N_0 \cdot b_k$ and $N_{-1} \cdot b_k$ cannot both be damaged since it is assumed that an error occurs in $N_1 \cdot f_1$. One of $\{N_{n+k} \cdot f_1, N_{n+k-1} \cdot f_1\}$ therefore contains an error and is thus one of $\{N_1 \cdot f_1, N_2 \cdot f_1\}$. However, in this case $N_n \cdot b_k$ 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, $N_1 \cdot f_1$ must be correct.

If $N_n \cdot f_1$ does not address N_0, then since $N_1 \cdot f_1$ must, the target can be immediately identified. So suppose that both $N_1 \cdot f_1$ and $N_n \cdot f_1$ address N_0. Since $N_n \cdot f_1$ is distinct from $N_1 \cdot f_1$, it contains an error. Since only two errors exist in the locality, $N_n \cdot f_1$ must therefore be either $N_2 \cdot f_1$ or $N_{k+1} \cdot f_1$. $N_n \cdot f_1$ cannot be $N_2 \cdot f_1$ since an erroneous $N_n \cdot f_1$ addresses N_0 while an erroneous $N_2 \cdot f_1$ addresses N_n, which is distinct from N_0. Thus, $N_n \cdot f_1$ is $N_{k+1} \cdot f_1$, implying that N_n is N_{k+1}. The two errors in the locality thus occur in $N_{k+1} \cdot f_1$ and one of $\{N_{1-k} \cdot b_k, N_{-1} \cdot b_k, N_0 \cdot b_k, N_2 \cdot f_1\}$. $N_1 \cdot b_k$ and $N_{k+1} \cdot b_k$ are therefore correct, since N_{k+1} is not a trusted node. $N_1 \cdot b_k$ therefore addresses the incorrect candidate N_{k+1}. $N_{k+1} \cdot b_k$, however, does not address the target, since N_n is not the trusted node N_{1-k}. Thus, if $N_n \cdot f_1$ and $N_1 \cdot f_1$ address N_0, the candidate whose back pointer addresses the other candidate must be the target.

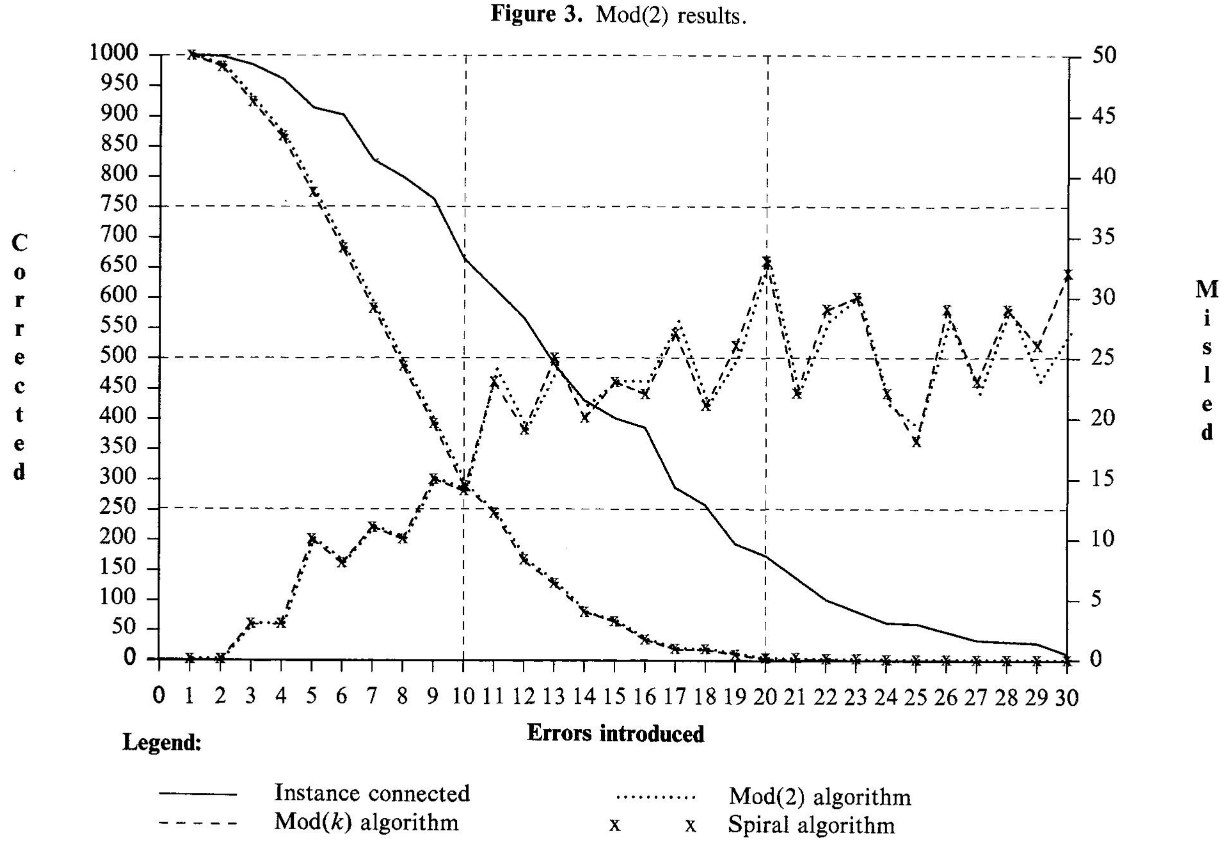
4. COMPARISONS

A standard double-linked list is not locally correctable [9] although single errors can be corrected within it [10]. A mod(2) structure is locally correctable, if the structure is traversed backwards. However, if a single locality contains two errors, then it may only be possible to correct these errors by traversing the instance in the opposite direction. For these reasons, this paper concentrated on mod($k \geq 3$) structures.

The method presented here for improving the robustness of a double-linked list requires the presence of one additional identifier component per node, the presence of ($k - 1$) additional header nodes, and a count component. This storage overhead is typically smaller than that required if error correcting codes are used, since at least two checksum components are needed to protect two data components against single errors [4].

The modification to the distance spanned by back pointers will increase the cost of performing updates in the proposed structure, and an alternative structure having two header nodes, an identifier, a forward pointer, and a virtual back pointer has therefore been proposed [7]. The virtual backpointer in node N_i contains the exclusive *OR* of the addresses of N_{i+1} and N_{i-1}. The true back pointer can therefore be determined by performing an exclusive *OR* of the virtual back pointer with $N_i \cdot f_1$. Similarly, the forward pointer $N_i \cdot f_1$ can be verified by performing an exclusive *OR* of the virtual back pointer with the address of the previous node. This clever modification to the back pointer produces a locally correctable structure which is as strongly connected as a mod(3) structure, and a correction algorithm which empirically appears to be competitive with historical methods of correcting mod(k) structures.

Figure 3. Mod(2) results.



J. SYSTEMS SOFTWARE
1990; 11: 205–214

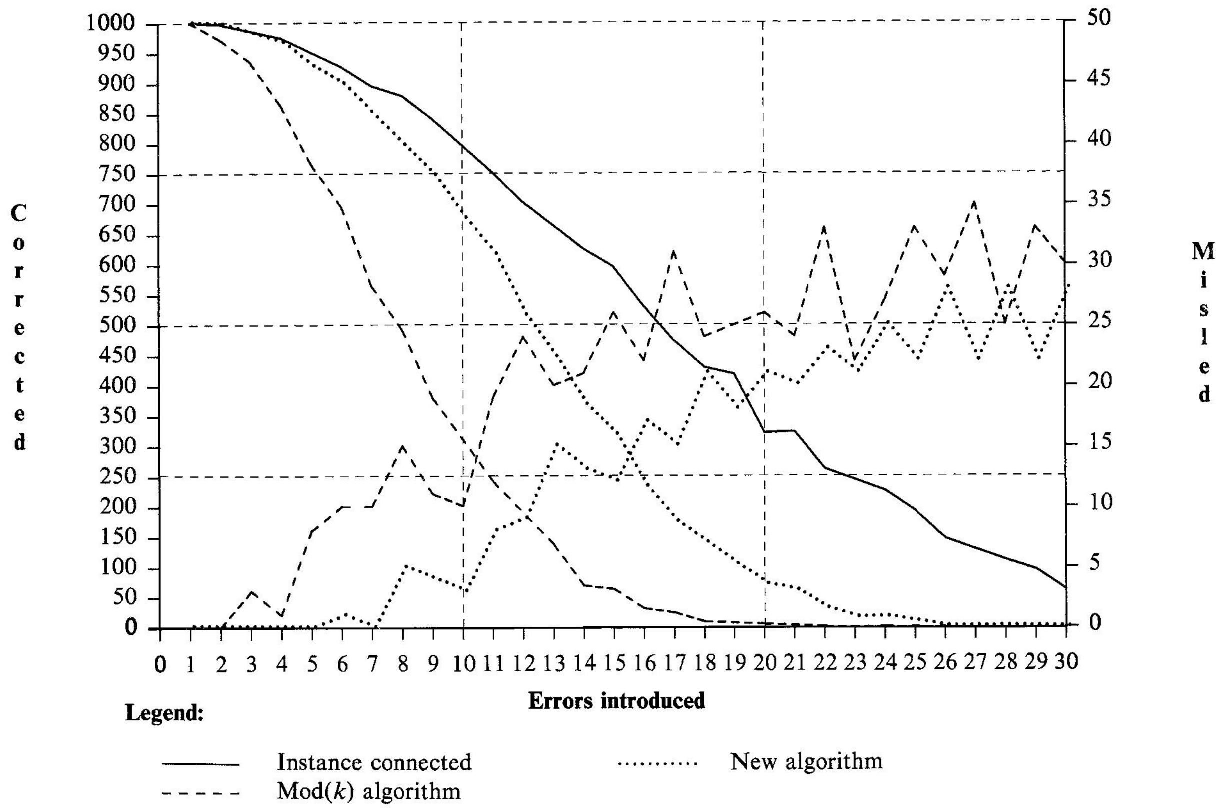


Figure 4. Mod(3) results.

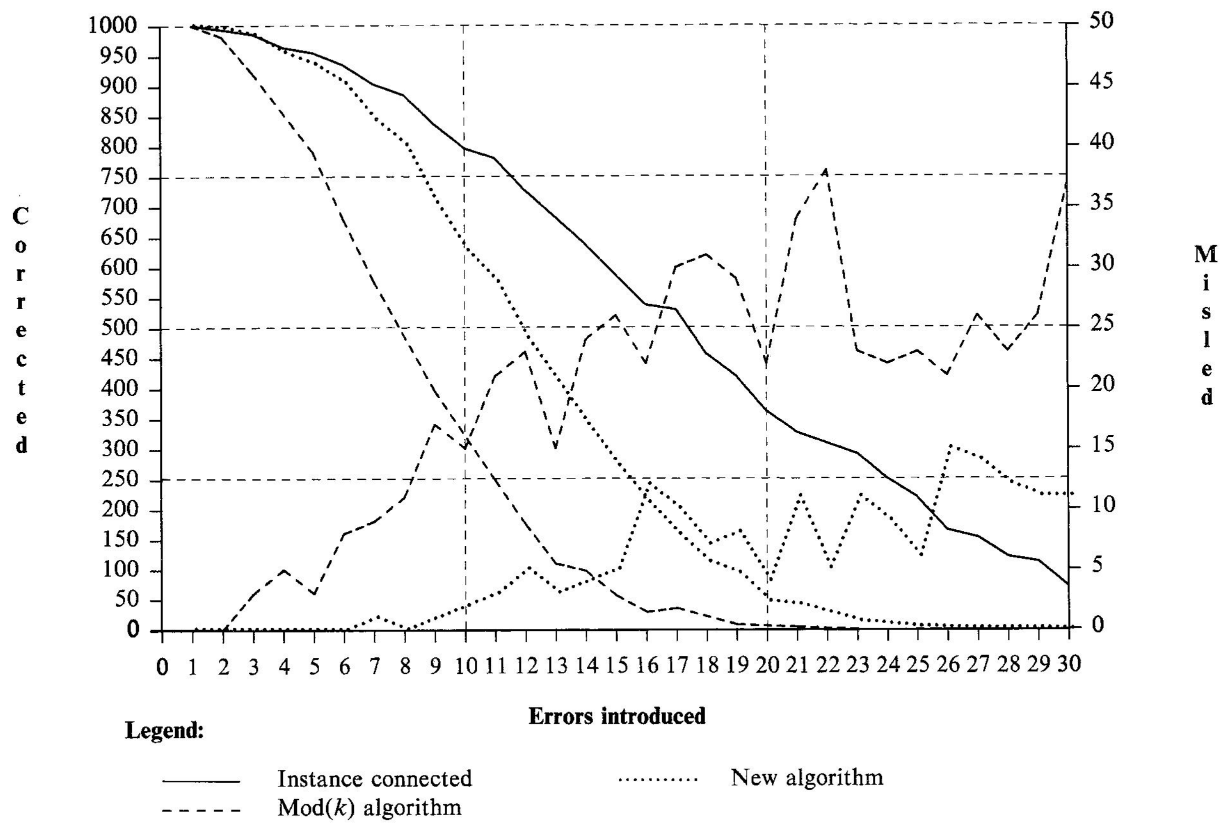


Figure 5. Mod(4) results.

Empirical results presented in Appendix B 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 this algorithm cannot however correct mod(2) structures, these other algorithms are still valuable.

5. CONCLUSIONS

The above material provides some constructive foundations for investigating the local correctability of an arbitrary structure [5]. Locally correctable structures are desirable for a number of reasons. The expected number of errors that can be corrected in an instance of such a structure increases as the instance grows. Local correction algorithms typically examine a small, bounded, number of components at each correction step, and therefore operate in time proportional to the size of the instance. When this is the case, locks can be used to allow local error correction to proceed concurrently with other operations that manipulate the instance being corrected.

Much remains to be explored. It is currently very unclear what types of errors occur frequently in data structures, or how these errors can best be corrected. It is hard to decide what structural modifications will facilitate error correction, or to visualize how correction of these structures can best be undertaken [12]. Having developed correction algorithms, it is difficult to predict how effective these algorithms will be, without resorting to empirical studies. All of these issues are interesting areas for further research.

ACKNOWLEDGMENT

The authors are indepted to Dr. A. R. Crowe, Dr. J. P. Black, and those individuals who refereed earlier versions of this paper, 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, D. J. Taylor, and D. E. Morgan, An Introduction to Robust Data Structures, *Digest of Papers: 10th Annual Int. Symp. on Fault-Tolerant Computing,* pp. 110–112, Oct. 1980.
2. J. P. Black, D. J. Taylor, and D. E. Morgan, A Compendium of Robust Data Structures, *Digest of Papers: 11th Annual Int. Symp. on Fault-Tolerant Computing,* pp. 129–131, June 1981.
3. J. P. Black and D. J. Taylor, Local Correctability in Robust Storage Structures, CS-84-44, Dept. of Computer Science, University of Waterloo, Canada, Dec. 1984.
4. R. E. Blahut, *Theory and Practice of Error Control Codes,* Addison-Wesley, Reading, Massachusetts (1983).
5. I. J. Davis, Local Correction of Helix(k) Lists, *IEEE Trans. Computers* 38(5), 718–724 (1989).
6. I. J. Davis, A Locally Correctable AVL Tree, *Digest of Papers: 17th Int. Symp. of Fault-Tolerant Computing,* pp. 85–88, July 1987.
7. C. C. Li, P. P. Chen, and W. K. Fuchs, Local Concurrent Error Detection and Correction in Data Structures Using Virtual Backpointers, *IEEE Trans. Computers* 38(11), 1481–1492 (1989).
8. S. C. Seth and R. Muralidhar, Analysis and Design of Robust Data Structures, *Digest of Papers: 15th Annual Int. Symp on Fault-Tolerant Computing,* pp. 14–19, June 1985.
9. D. J. Taylor, D. E. Morgan, and J. P. Black, Redundancy in Data Structures: Improving Software Fault Tolerance, *IEEE Trans. Software Eng.* SE-6(6), 585–594 (1980).
10. D. J. Taylor, D. E. Morgan, and J. P. Black, Redundancy in Data Structures: Some Theoretical Results, *IEEE Trans. Software Eng.* SE-6(6), 595–602 (1980).
11. D. J. Taylor and J. P. Black, Principles of Data Structure Error Correction, *IEEE Trans. Computers* C-31(7), 602–608 (1982).
12. D. J. Taylor and J. P. Black, Guidelines for Storage Structure Error Correction, *Digest of Papers: 15th Annual Int. Symp. on Fault-Tolerant Computing,* pp. 20–22, June 1985.
13. D. J. Taylor and J. P. Black, A Locally Correctable B-Tree Implementation, *Computer J.* 29(3), 269–276 (1986).
14. D. J. Taylor and C. J. Seger, Robust Storage Structures for Crash Recovery, *IEEE Trans. Computers* C-35(4), 288–295 (1986).

APPENDIX A. PSEUDOCODE FOR CORRECTION ALGORITHM

```
correct_headers():                                  /*Terminate if null instance     */
for (count = 0; count < max_possible; count = count + 1){

  candidates = 0;
  for (i = 0; i < 3; i = i + 1){                    /*Apply constructive votes       */
    N_x = N_{1-k+i}·b_k·f_1^{i-1};                  /*For simplicity assume N_x exists */
    if (N_x ≠ any candidate[j]){
      j = candidates; candidate[j] = N_x; vote[j] = 0;
```

```
        candidates = candidates + 1;
      }                                          /*N_x = candidate[j]                    */
      vote[j] = vote[j] + weight[i];             /*weight[i] = {1/4, 3/16, 1/16}         */
    }
    for (i = 0; i < candidates; i = i + 1){      /*Apply diagnostic votes                */
      N_x = candidate[i];
      if (N_x·f_1 = N_0)              vote[i] = vote[i] + 1/4;
      if (N_x·b_k·f_1 = N_0·b_k)      vote[i] = vote[i] + 3/16;
      if (N_x·b_k·f_1^2 = N_-1·b_k)   vote[i] = vote[i] + 1/16;
      if (N_x = any N_0≥i>-k)         vote[i] = 0;
    }

    case 'Only N_a got vote > 1/2': break;

    case 'Only N_a got vote of 1/2':
      if (candidates = 1){
        if (N_a·id bad or N_a·b_k·f_1^k ok) abort(Target disconnected);
        break;
      }
      if (N_a·f_1 = N_0 and N_a = N_2-k·b_k and N_a = N_3-k·b_k·f_1){
        if (k = 3) abort(Target may be disconnected);
        if (n_4-k·b_k·f_1 = N_3-k·b_k) abort(Target disconnected);
      }

    case 'Only N_a and N_b got vote of 1/2':
      if (N_a·f_1 ≠ N_b·f_1){
        if (N_b·f_1 = N_0) N_a = N_b;
      } else if (N_b·b_k = N_a) N_a = N_b;

    case 'Otherwise': abort(Target disconnected);
                                                 /*N_a is target node N_1                */
    N_1-k·b_k = N_a; N_a·id = id; N_a·f_1 = N_0; /*Assignments may be unnecessary        */
    if (N_a = last header) correct_count();      /*Terminate successfully                */
  }
  abort(Algorithm looping);
```

APPENDIX B. EMPIRICAL RESULTS

B.1. Explanation

This appendix presents empirical results obtained when "random" errors were introduced into 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 historical mod(k) local correction algorithm*, the mod(2) local correction algorithm presented in Taylor and Black [11], and the spiral local correction algorithm presented in Black and Taylor [3]. 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.

The mod(2) results, shown in Figure 3, are of some general interest, but do not directly pertain to the algorithm presented in this paper. They do, however, pertain to the mod(k) algorithm against which our algorithm is compared, and provide evidence that this comparison is appropriate. Mod(3) results are shown in Figure 4 and mod(4) results in Figure 5.

Each algorithm was executed on exactly the same "randomly" damaged instances. Each test was performed 100 times before the number of pointers being damaged was increased. Statistics were collected on the

* This algorithm uses 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.

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.

Because the instances being considered were small, the probability that errors caused disconnection was high. Because pointers were modified by a small amount, the probability that votes supported common incorrect candidates was high. This appendix therefore presents pessimistic estimates of the expected behavior of the mod(k) correction algorithms described in this paper.

B.2. Comments

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 historical 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.