Yale University
Department of Computer Science

Exact Learning of μ-DNF Formulas with Malicious Membership Queries

Dana Angluin

YALEU/DCS/TR-1020
March 1994

This research was supported by the National Science Foundation, grant CCR-9213881.

Exact Learning of μ-DNF Formulas with Malicious Membership Queries

Dana Angluin

Abstract

We consider exact learning of concepts using two types of query: extended equivalence queries, and malicious membership queries, that is, membership queries that are permitted to make errors on some arbitrarily chosen set of examples of a bounded cardinality. We present a randomized algorithm to learn μ-DNF formulas using these queries. The expected running time of the algorithm is polynomial in the number of variables and the maximum number of strings on which the membership oracle is allowed to make errors.

1 Introduction

We continue the investigation begun by Angluin and Krikis [2] concerning the effects of errors in the answers to membership queries in the model of equivalence queries and malicious membership queries. We are interested in the question of whether polynomial-time learnability of a concept class using equivalence queries and (error-free) membership queries implies polynomial-time learnability in the error model of equivalence queries and malicious membership queries, perhaps under some natural restrictions on the concept class. Angluin and Krikis present some encouraging positive evidence: any concept class that is polynomially closed under finite exceptions and learnable in polynomial time using equivalence queries and (error-free) membership queries can be learned in polynomial time using equivalence queries and malicious membership queries. Corollaries of this include polynomial-time learning algorithms for DFAs and boolean decision trees using equivalence queries and malicious membership queries. They also present a polynomial-time learning algorithm to

This research was supported by the National Science Foundation, grant CCR-9213881.

learn monotone DNF formulas using equivalence queries and malicious membership queries.

There have been a number of ingenious algorithms to learn read-once or μ-formulas over various bases using equivalence queries and membership queries (or substitutions) [3, 4, 5]. These concept classes are typically not closed under finite exceptions, and therefore are not covered by the general transformation found by Angluin and Krikis. Thus, classes of concepts defined by read-once formulas are a particularly interesting case for the model of polynomial-time learnability using equivalence queries and malicious membership queries.

In this paper, we present a randomized algorithm that learns in expected polynomial time the class of μ-DNF formulas using equivalence and malicious membership queries. The intuitive idea behind the algorithm is simple. Define a boundary point of a concept to be a point in the concept that is adjacent to a point not in the concept. Then the local neighborhood of any boundary point of a concept represented by a μ-DNF formula provides direct information about a term of the formula. Moreover, either there are "very few" points in the concept or its complement, or the set of its boundary points is "fairly large." In the first two cases, we can afford to learn the concept counterexample by counterexample, and in the last case, the malicious membership queries cannot hope to corrupt the answers in the neighborhoods of enough points of the boundary to fool the algorithm. We expect to be able to generalize this algorithm to learn the class of μ-DNF formulas with exceptions (see [2] for definitions.)

2 Preliminaries

Let n be any natural number. We denote by B_n the set $\{0,1\}^n$ of boolean n-vectors. The i-th coordinate of a vector $y \in B_n$ is denoted $y[i]$. The Hamming distance $d(y,z)$ of two elements $y, z \in B_n$ is the number of coordinates i such that $y[i] \neq z[i]$. If $y, z \in B_n$, then z is a *neighbor* of y if and only if $d(y,z) = 1$. Clearly, every element of B_n has n neighbors.

The set of all boolean formulas over the variables X_i for $i = 1, \ldots, n$ is denoted by F_n. For each formula $f \in F_n$ and vector $y \in B_n$, we denote by $f(y)$ the value of the formula f under the truth assignment $X_i = y[i]$ for $i = 1, \ldots, n$. For each formula $f \in F_n$, we define the *concept represented by* f as the set of boolean n-vectors $y \in B_n$ such that $f(y) = 1$.

A μ-DNF formula is a boolean formula in disjunctive normal form in which each

variable occurs at most once. For example, the following formula is a μ-DNF formula.

$$(X_2 \wedge \neg X_4 \wedge X_6) \vee \neg(X_5 \wedge \neg X_1).$$

Note that the number of terms of a μ-DNF formula cannot exceed the number of variables that occur in it.

The subset of F_n consisting of μ-DNF formulas is denoted by T_n; it is the set of target formulas of n variables. We assume that some $f \in T_n$ is chosen, the *target formula*, and the task of the learning algorithm is to find a formula $g \in F_n$ that represents the same concept as f, in other words, such that g is logically equivalent to f. Note that we do not require the result to be in the form of a μ-DNF formula.

The information available to the learning algorithm about the target function f is the value of n, the value of another parameter ℓ, (the bound on the number of lies permitted) and the answers to two types of queries: extended equivalence queries and malicious membership queries.

In an *extended equivalence query*, the learning algorithm specifies as input some boolean formula $g \in F_n$, and the response depends on whether g is logically equivalent to f. If they are equivalent, then the answer is "yes." Otherwise, the answer is a *counterexample*, that is, a vector $y \in B_n$ such that $f(y) \neq g(y)$. Note that we permit equivalence queries with general boolean formulas; in fact, equivalence queries with general DNF formulas suffice for our algorithm.

In a *malicious membership query*, the learning algorithm specifies as input a boolean vector $y \in B_n$, and the response is either 1 or 0. If the response is not equal to $f(y)$, then we say that there has been an *error* on the vector y. We abbreviate malicious membership query by MMQ.

Note that for extended equivalence queries, there is a choice of which counterexample to return in the case of inequivalence, and for malicious membership queries, there is a choice of which vectors are answered with errors and which are answered correctly. We assume that these choices are made by an adversary, subject to the following constraints.

The adversary is *on-line*, that is, when a query is made, the choice of the answer may depend on knowledge of the learning algorithm and the queries it has made and the responses it has received so far. Moreover, the parameter ℓ is an upper bound on the number of distinct vectors that may be answered with errors in the malicious membership queries. We assume also that the adversary answers malicious membership queries consistently, that is, the replies to subsequent queries made to a given vector are the same as the first one. Errors with this property are called *persistent*. (If the adversary does not have this property, the learning algorithm

could enforce it by keeping a table of "first answers" and consulting this table instead of the adversary for any vector queried more than once.)

A randomized learning algorithm learns the class of μ-DNF formulas in expected polynomial time using extended equivalence and malicious membership queries provided that there is a polynomial $p(n, \ell)$ such that for any positive integers n and ℓ, for any target function $f \in T_n$ and any adversary as described above, the algorithm with input n and ℓ and access to extended equivalence queries and malicious membership queries about f answered by the adversary, has an expected running time of at most $p(n, \ell)$ steps (counting each query as one step) and, when it halts, outputs a boolean formula g equivalent to f.

Remark: It was useful during the development of this algorithm to consider a somewhat more limited adversary, replacing errors in malicious membership queries by omissions. That is, the adversary is permitted to answer "I don't know" on up to ℓ arbitrarily chosen different vectors – the rest of the membership queries must be answered correctly. This model might be termed "malicious don't knows." This is a more limited adversary because the learning algorithm knows that it can depend on the correctness of the answers that it does get, whereas with malicious membership queries, each answer is possibly suspect. Clearly, a learning algorithm that can cope with malicious membership queries can cope as well with malicious don't knows by assigning an arbitrary answer to the vectors answered with "I don't know", but the converse is not necessarily true.

3 Warm-up

There is a polynomial-time algorithm for learning general boolean μ-formulas using equivalence queries and (error-free) membership queries [1]. In the special case of μ-DNF formulas the algorithm can be considerably simplified. We review the simpler version here, since it is the basis of the algorithm we present.

Let f be the target μ-DNF formula. The basic idea is that if we can find a pair of assignments y and y' that differ on a single variable such that $f(y) \neq f(y')$, then we can use membership queries to determine a term of f.

Suppose y and y' are such that $f(y) = 1$ and $f(y') = 0$, and y and y' differ only in the value assigned to the variable X_j. Then because of the μ restriction, X_j occurs in a unique term T of f and the assignment y makes just the term T true. We can find all the literals that occur in T by making membership queries $f(y'')$ for all n neighbors y'' of y. A variable X_k occurs in T if and only if $f(y'') = 0$ for y'' obtained

from y by changing just the value assigned to X_k. Moreover, if X_k occurs in T, it occurs positively if y assigns 1 to X_k and negatively otherwise.

We can get such a pair y and y' by interpolating between a positive and a negative counterexample and asking membership queries, as follows. We first make equivalence queries with the constant functions TRUE and FALSE. If either equivalence query is answered "yes", we are done. Otherwise, $f(x)$ is a non-constant function and the equivalence queries are answered with counterexamples x_0 and x_1 such that $f(x_0) = 0$ and $f(x_1) = 1$.

We now interpolate between x_1 and x_0 any sequence of assignments $y_1, y_2, \ldots, y_m$ such that $y_1 = x_1$, $y_m = x_0$ and $d(y_i, y_{i+1}) = 1$ for each $i = 1, \ldots, m-1$. Since $f(y_1) = 1$ and $f(y_m) = 0$, there must be a value of i such that $f(y_i) = 1$ and $f(y_{i+1}) = 0$. We can find such a pair by asking a sequence of membership queries with the vectors $y_1, y_2, \ldots$.

Thus, we can find one term of f. To continue, we would like to "remove" the variables that occur in the term(s) we have found so far and find a new term of f, if any remain. This is also not difficult, as follows.

Suppose we have found terms $T_{r+1} \ldots, T_s$ of f, and terms $T_1, \ldots, T_r$ remain to be found. By renumbering the variables, if necessary, we may assume that the terms that we have found contain the variables $X_{m+1}, \ldots, X_n$. Then we can simulate membership and extended equivalence oracles for the formula $f' = T_1 \vee \ldots \vee T_r$ over the variables $X_1, \ldots, X_m$ as follows.

In a membership query to f' the input is an assignment to the variables $X_1, \ldots, X_m$, which we extend to an assignment to $X_1, \ldots, X_n$ in such a way that the terms $T_{r+1}, \ldots, T_s$ are all rendered false. Then we make a membership query to f with this assignment. The result will be true if and only if the original assignment to $X_1, \ldots, X_m$ makes one of the remaining terms of f true, that is, makes f' true.

For an extended equivalence query to f' we are given a boolean formula h over the variables $X_1, \ldots, X_m$. We then make an extended equivalence query with the formula

$$h \vee T_{r+1} \vee \ldots \vee T_s.$$

Clearly, if h is equivalent to f', this formula will be equivalent to f, and the reply to the query will be "yes". If, conversely, this formula is equivalent to f, then by considering assignments that make all of $T_{r+1}, \ldots, T_s$ false, we can establish that h must be equivalent to f'. (In the original algorithm, the equivalence queries can be taken to be proper, that is, asked only about μ-DNF formulas, but we need the generality of extended equivalence queries for our new algorithm.)

Thus, putting it all together, we use two equivalence queries and $O(n)$ membership queries to find a term of f. We can then "remove" the variables in that term, and find another term of f, and so on, until we have found all the terms of f. Because we can re-use the negative counterexample from the first iteration, we can find all the terms of f using at most $n+1$ equivalence queries and $O(n^2)$ membership queries.

It is clear that this procedure relies strongly on the correctness of the membership queries it makes. To adapt this method to the setting of malicious membership queries, we attempt to repeat the basic procedure to construct a term in enough different places in the space of examples that a vote of the results will give us at least one correct term of the target function.

4 The Term-Finding Algorithm

Assume the target concept is represented by the μ-DNF formula f over n variables. Assume that an upper bound ℓ is given on the number of points on which the MMQ oracle may make errors. Our algorithm is randomized; it is open whether there exists a deterministic polynomial-time algorithm for this problem.

The idea of the term-finding algorithm is to determine a value b such that $f(x) = b$ for at least $1/4$ of all points x, and then to use equivalence queries to collect a large set of points x_i such that $f(x_i) = 1 - b$. From each of these points x_i we take a random walk of length $O(n \ln n)$, making queries to the MMQ oracle along the walk. With high probability, a large fraction of the endpoints of these walks will be points where f has value b. For each walk, if we find a change in the replies of MMQ along the walk, we use the neighborhood procedure with queries to MMQ to determine a candidate term T_i for the walk. If a candidate term T_i receives at least $\ell + 1$ "votes" with non-intersecting support sets, then it is a correct term of the target function. We show that this happens with reasonable probability. The details of the algorithm and its analysis follow.

4.1 The Neighborhood Procedure

We first generalize the neighborhood procedure of the preceding section for finding a candidate term using an oracle Q that returns 0 or 1 for each point x.

The input is a point y such that $Q(y) = 1$ and for at least one neighbor y' of y, $Q(y') = 0$. We construct a candidate term T as follows. For each neighbor y'' of y such that $Q(y'') = 0$, let X_j be the variable on which y and y'' differ, and if X_j is

assigned 1 by y then add the literal X_j to T, otherwise add the complement of X_j to T. Return the candidate term T.

The *support set* for the candidate term T is the point y and its n neighbors; these are the points where queries are made to Q to construct the candidate term. Note that if the oracle Q agrees with f on the support set of T, then T is in fact a term of f.

4.2 Finding a Popular Value

We wish to choose a value b such that $f(x) = b$ for at least $1/4$ of all points x. To do so, we choose a large odd number of n-bit vectors x uniformly at random and query the MMQ oracle on these points; b is chosen as the majority reply. We bound the probability of the event that we choose b incorrectly, that is, $f(x) = b$ for fewer than $1/4$ of all the points.

If we choose R samples, then with probability at least

$$1 - \frac{R^2}{2^n}$$

the sample points are all distinct. Suppose the sample points are all distinct and MMQ replies b to at least $R/2$ of them. Then for at least $R/2 - \ell$ samples x, we really do have $f(x) = b$. Suppose $f(x) = b$ for fewer than $1/4$ of all vectors x. Then the probability that in R samples we draw at least $R/2 - \ell$ with $f(x) = b$ is bounded above by

$$e^{-\frac{R}{8}+\ell}.$$

Thus, the probability that we choose a value b such that $f(x) = b$ for fewer than $1/4$ of all vectors is bounded above by

$$\frac{R^2}{2^n} + e^{-\frac{R}{8}+\ell}.$$

4.3 Finding the Starting Points

Having chosen b, we use equivalence queries to elicit M distinct counterexamples x with the property that $1 - b = f(x) = MMQ(x)$. These may not be easy to acquire by sampling if there are relatively few of them. Let these counterexamples be denoted

$$S = \{x_1, \ldots, x_M\}.$$

We briefly describe the procedure to generate them.

If $1-b=1$, then the successive conjectures are the constant FALSE function, the function true on just the first counterexample, the function true on just the first two counterexamples, and so on. It is clear how to construct a general DNF formula of m terms true on just m arbitrary given vectors.

If $1-b=0$, then the successive conjectures are the constant TRUE function, the function false on just the first counterexample, the function false on just the first two counterexamples, and so on. (Given m arbitrary points, we can construct a decision tree with $O(mn)$ leaves that classifies just these m points as false, and then convert this to a general DNF formula with $O(mn)$ terms.)

In either case, if an equivalence query returns "yes" before M points with the specified properties have been collected, the target function has been exactly identified, and the procedure returns. When a counterexample is returned by an equivalence query, we make a query to the MMQ oracle to check whether the answer is $1-b$. It is clear that this procedure will either identify the target function with at most $M+\ell$ equivalence queries, or it will return the required set of M points.

4.4 Constructing the Random Walks

Assume that the procedure above generates M points with the required properties. Then, for each $x_i \in S$, we construct a random walk starting at x_i of length $L = \lceil 2n \ln n \rceil$ as follows. Let $w(i,0) = x_i$. Assuming $w(i,0), \ldots, w(i,t)$ have been defined for $0 \leq t < L$, we select uniformly and independently one of the n dimensions and then flip an unbiased coin to decide whether to stay at the current point ($w(i,t+1) = w(i,t)$) or to move to the neighbor of the current point along the selected dimension ($w(i,t+1)$ is the neighbor of $w(i,t)$ along the selected dimension.) Also denote the final endpoint of the walk by x_i', that is, $x_i' = w(i,L)$. Note that two successive points along the walk are at distance 0 or 1 from each other.

Let W_1 be the set of walks in which each dimension is selected at least once, regardless of whether the walk has moved along that dimension. The endpoints of walks in W_1 are independently and uniformly distributed over the set of all n-bit vectors. For each walk i, the probability that it is in W_1 is at least $1-1/n$. The probability that the endpoints of all the walks in W_1 are distinct is bounded below by $1-M^2/2^n$.

Let W_2 consist of all those walks i from W_1 such that $f(x_i') = b$. That is, W_2 is the set of walks in which every dimension is chosen at least once, and the value of f

at the endpoint of the walk differs from the value of f at the start of the walk.

Assume that b has been chosen so that $f(x) = b$ for at least $1/4$ of all assignments x. For each walk i independently the probability that walk i is in W_2 is at least $(1/4)(1 - 1/n)$. Thus the expected number of walks i in W_2 is at least

$$\frac{M}{4}(1 - \frac{1}{n}),$$

and the probability that fewer than half this many walks are in W_2 is bounded above by

$$e^{-\frac{M}{32}+\frac{M}{16n}}.$$

Hence, the probability that W_2 does not contain at least

$$\frac{M}{8}(1 - \frac{1}{n})$$

walks with distinct endpoints is bounded above by

$$\frac{M^2}{2^n} + e^{-\frac{M}{32}+\frac{M}{16n}}.$$

4.5 Constructing Candidate Terms

For each walk i, for $i = 1, \ldots, M$, we use the MMQ oracle to query successive points $w(i,0), w(i,1), \ldots$ along the walk until either we run out of points or we find a pair of points $w(i,t)$ and $w(i,t+1)$ such that the MMQ oracle returns different values. If we find such a pair, we use the neighborhood procedure (defined above) with the MMQ oracle to construct a candidate term, which we denote by T_i. The support set of the candidate term T_i is denoted by S_i.

By our choice of S, the MMQ oracle returns $1 - b$ on every point $x_i \in S$. Thus, a walk will be assigned a candidate term T_i if and only if the MMQ oracle returns a value of b somewhere along the walk. In particular, if the MMQ oracle returns a value of b for the endpoint of a walk, the walk will be assigned a candidate term. We can give a lower bound on the number of walks assigned candidate terms as follows.

Each of the walks in the set W_2 has an endpoint x_i' with $f(x_i') = b$. If all of these endpoints are distinct, then the MMQ oracle can make errors on at most ℓ of them. Thus, from the bound above on the number of elements of W_2 with distinct endpoints, at least

$$\frac{M}{8}(1 - \frac{1}{n}) - \ell$$

walks will be assigned candidate terms, except with probability bounded above by

$$\frac{M^2}{2^n} + e^{-\frac{M}{32}+\frac{M}{16n}}.$$

4.6 Counting the Votes

What we want to find is a term T such that some set of $\ell + 1$ walks return T as a candidate term, and the corresponding $\ell + 1$ support sets are all pairwise disjoint. A direct approach would lead to the clique problem, so we instead use a greedy approach.

We process in some order all the random walks that are assigned candidate terms. Each such walk i has a candidate term T_i and associated support set S_i. We keep a list of all the terms encountered so far, and for each term on the list, an associated list of support sets for the term. To process walk i, we check to see if T_i is already on the list. If not, we add it, with an associated list of support sets containing just S_i. Otherwise, we add S_i to the list of support sets associated with T_i, provided that S_i is disjoint from each of the support sets already on the list for T_i. If this operation causes the list of support sets for T_i to contain at least $\ell+1$ elements, the term-finding algorithm returns with output T_i.

It is clear from the construction that if the term-finding procedure returns a term T, then T has at least $\ell + 1$ pairwise disjoint support sets on its associated list. Hence, the MMQ oracle must have answered all the queries in at least one support set correctly (that is, in agreement with f), and the term T is a correct term of f.

We now show that there is a high probability that the term-finding algorithm will find a term with at least $\ell + 1$ disjoint support sets. Intuitively, we take M large enough that there will be a large set of "well separated" walks, whose candidate terms must have disjoint support sets.

4.7 Separating the Walks

Let the Hamming distance between two walks be the minimum Hamming distance between any pair of points, one from each walk. If the Hamming distance of two walks is at least 3, then no point is within distance 1 of both walks, which means that if they are both assigned candidate terms, the corresponding support sets of the terms are necessarily disjoint.

At this point we need an upper bound on the number of walks that come within

distance 2 of a specified walk. A simplified version of a proof of Valiant and Brebner [6] suffices. We analyze the situation in which a random walk i is constructed as described above from every n-bit vector, so there are 2^n walks. The upper bounds we give hold also in the case that walks are constructed only from the points in S.

Let x be any fixed n-bit vector. $H_x(i,t)$ is an indicator variable for the event that $w(i,t) = x$, that is, walk i is at x at time t, for $0 \leq t \leq L$. $H_x(i)$ is an indicator variable for the event that $w(i,t) = x$ for some t, that is, walk i visits x at least once. N_x is the number of different walks that ever visit x, that is,

$$N_x = \sum_{i=1}^{2^n} H_x(i).$$

We bound the value of N_x as follows:

$$N_x \leq \sum_{i=1}^{2^n} \sum_{t=0}^{L} H_x(i,t),$$

and therefore

$$\sum_{x \in B^n} N_x \leq \sum_{x \in B_n} \sum_{i=1}^{2^n} \sum_{t=0}^{L} H_x(i,t).$$

Summing over all walks, times, and vertices, there are exactly $2^n(L+1)$ events $H_x(i,t)$ that are 1; therefore we have

$$\sum_{x \in B_n} N_x \leq 2^n(L+1).$$

This is true also of expected values, so

$$\sum_{x \in B_n} E(N_x) \leq 2^n(L+1).$$

By symmetry (independent walks from every n-bit vector), the value of $E(N_x)$ is the same for all 2^n values of x, so for each x we have

$$E(N_x) \leq (L+1).$$

That is, in the situation where we construct a random walk from every n-bit vector, the expected number of walks that ever visit x is bounded above by $(L+1)$.

Now N_x is the sum of 2^n independent random variables $H_x(i)$ (which of course will have different means) and we may bound the probability that it falls very far

from its expected value. The probability that more than $2(L+1)$ different walks visit x is bounded above by

$$e^{-(L+1)/3} \leq n^{-2n/3}.$$

This is clearly also an upper bound in the case that we consider only those random walks made from vertices in the set S.

If we fix any random walk i and consider the set D of vertices within distance 2 of walk i,

$$|D| \leq (L+1)n^2.$$

Let

$$p(n) = 2(L+1)^2 n^2.$$

If more than $p(n)$ different random walks visit nodes of D, then there is some node of D visited by more than $2(L+1)$ different random walks. Thus, the probability that more than $p(n)$ different walks visit nodes of D is bounded above by

$$|D| n^{-2n/3} \leq (L+1)n^{2-2n/3}.$$

In other words, for each random walk i, this is an upper bound on the probability that more than $p(n)$ walks come within distance 2 of walk i. Finally, the probability that there exists a walk i (for $i = 1, \ldots, M$) such that more than $p(n)$ random walks come within within distance 2 of walk i is bounded above by

$$M(L+1)n^{2-2n/3}.$$

If for each $i = 1, \ldots, M$, there are at most $p(n)$ walks that come within distance 2 of walk i, then considering all the support sets for all the walks assigned candidate terms, each support set can intersect at most $p(n) - 1$ other support sets.

4.8 The Probability of Finding a Term

Let us suppose the following.

1. M is chosen sufficiently large that

$$\frac{M}{8}(1 - \frac{1}{n}) - \ell \geq p(n)((n+1)\ell + 1).$$

2. We choose b correctly, i.e., so that $f(x) = b$ for at least $1/4$ of all n-bit vectors.

3. We find the required set S of M points.

4. At least $(M/8)(1 - 1/n) - \ell$ random walks are assigned candidate terms.

5. For each $i = 1, \ldots M$, at most $p(n)$ walks come within distance 2 of walk i.

Then the term-finding algorithm must return a correct term of f.

To see this, we note that under the assumptions given, the number of walks assigned candidate terms is at least $p(n)((n+1)\ell + 1)$, and each disjoint support set found can eliminate at most $p(n) - 1$ other support sets, so at least $((n+1)\ell + 1)$ disjoint support sets are found. At most ℓ of them can contain errors, so there must be at least $n\ell + 1$ support sets for correct terms of f. Since f has at most n terms, at least one term will be assigned at least $\ell + 1$ disjoint support sets. We have already argued that any term that is assigned at least $\ell + 1$ disjoint support sets is a correct term of f.

Therefore, provided M is chosen to satisfy (1) above, the probability that the term-finding algorithm will neither correctly identify f nor find a correct term of f is bounded above by the sum of the probabilities of the following events.

1. b is incorrectly chosen.

2. Fewer than $(M/8)(1 - 1/n) - \ell$ walks are assigned candidate terms, given that b is correctly chosen.

3. More than $p(n)$ walks come within distance 2 of some walk i.

The upper bounds we have derived on these probabilities are as follows.

1. $\frac{R^2}{2^n} + e^{-\frac{R}{8}+\ell}$.

2. $\frac{M^2}{2^n} + e^{-\frac{M}{32}+\frac{M}{16n}}$.

3. $M(L+1)n^{2-2n/3}$.

We in turn bound these as follows. There exists a function $b(n) = \Omega(2^{n/2})$ with the property that if $\ell \leq b(n)$ and we choose $R = \lceil 8\ell + 8\ln 10 \rceil$, then

$$\frac{R^2}{2^n} + e^{-\frac{R}{8}+\ell} \leq \frac{1}{5}.$$