, , 1–42 ()
© Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

On-line Prediction and Conversion Strategies

NICOLÒ CESA-BIANCHI cesabian@dsi.unimi.it
DSI, Università di Milano, Via Comelico 39,
20135 Milano, Italy.

YOAV FREUND yoav@research.att.com
AT&T Bell Laboratories, 600 Mountain Avenue, Room 2B-428,
Murray Hill, NJ 07974-0636, USA.

DAVID P. HELMBOLD dph@cse.ucsc.edu
MANFRED K. WARMUTH manfred@cse.ucsc.edu
Computer Science Department, University of California,
Santa Cruz, CA 95064, USA.

Received May 1, 1991

Editor:

Abstract. We study the problem of deterministically predicting boolean values by combining the boolean predictions of several experts. Previous on-line algorithms for this problem predict with the weighted majority of the experts' predictions. These algorithms give each expert an exponential weight β^m where β is a constant in $[0, 1)$ and m is the number of mistakes made by the expert in the past. We show that it is better to use sums of binomials as weights. In particular, we present a deterministic algorithm using binomial weights that has a better worst case mistake bound than the best deterministic algorithm using exponential weights. The binomial weights naturally arise from a version space argument. We also show how both exponential and binomial weighting schemes can be used to make prediction algorithms robust against noise.

Keywords: On-line learning, conversion strategies, noise robustness, binomial weights, exponential weights, weighted majority algorithm, expert advice, mistake bounds, Ulam's game.

1. Introduction

This paper studies a simple on-line model where predictions are made in a series of trials. At each trial t the prediction algorithm receives the tth observation x_t and produces a boolean prediction $\hat{y}_t$. It then receives the correct outcome y_t as feedback. A mistake occurs if prediction $\hat{y}_t$ and outcome y_t disagree. Following Littlestone [14], we seek prediction algorithms that minimize the number of mistakes over a worst case sequence of x_t and y_t. Of course in the unconstrained worst case a mistake can occur in every trial. In order to make good predictions the predictor needs to have some prior knowledge that enables it to makes predictions about the future based on the past. In a Bayesian regression framework, one can encode this knowledge using a prior distribution over the set of sequences or over a set of sequence models. In this work we are interested in performance bounds that make no probabilistic assumptions, and so we define the prior knowledge somewhat differently.

We assume that there are N experts each of which is a prediction strategy. Our goal is to design an algorithm, which we shall call the "master algorithm", that combines the predictions of the experts in the following way. At the beginning of trial t, the master algorithm feeds the given observation, x_t, to all experts. The master then uses some function of the N predictions produced by the experts to form its own prediction, $\hat{y}_t$. At the end of the trial the feedback, y_t, is shared with all experts. We prove worst-case bounds on the number of mistakes made by the master when the number of mistakes made by the best expert is bounded.

Generalizations of the above model, where the predictions of the experts and/or of the master algorithm may be in the continuous range $[0, 1]$, have been studied by Vovk [21], Littlestone and Warmuth [17], Cesa-Bianchi *et al.* [9], and Kivinen and Warmuth [13]. In this paper we return to the simplest setting where all predictions and outcomes are boolean. This is the problem solved by the basic Weighted Majority (WM) algorithm [17]. Here we study the boolean case in more depth and devise a better algorithm that we call the "Binomial Weighting" algorithm or BW. The worst case number of mistakes that BW makes is smaller than the number of mistakes made by previously known algorithms. In fact, if the number of experts is large enough and all predictions are deterministic and boolean, then we show that BW has the smallest possible worst-case mistake bound among all master algorithms. In our analysis of BW we explore some elegant combinatorial structures that might be applicable elsewhere.

The Weighted Majority algorithms cited above attempt to minimize the number of mistakes made as a function of the number of mistakes made by the best expert. They assign to each expert a weight of the form β^m, where β is a constant in $[0, 1)$ and m is the total number of mistakes (or more generally the total loss) incurred by the expert so far[1]. The essential property is that the experts making many mistakes get their weights rapidly slashed. The WM algorithm uses the weighted average of the experts' predictions to form its own prediction: It simply predicts 1 if the weighted average is greater than $1/2$, and 0 otherwise.

The new master algorithm BW uses its weights in a similar way to WM for predicting, however, these weights are not in exponential form. Instead, they are tails of a binomial sum. A further difference between WM and BW is the following. On each trial WM predicts 1 if and only if the total current weight of the experts predicting 1 is larger than the total current weight of the experts predicting 0. BW, instead, predicts 1 if and only if the total updated weight resulting from the outcome being 1 is larger than the total updated weight resulting from the outcome being 0.

This binomial weighting scheme is motivated by a version space[2] argument. The mistake bound of the Weighted Majority algorithm approximates the mistake bound of the BW algorithm in the same way that Chernoff bounds approximate sums of binomial tails. We show that the gap between the mistake bounds of the Weighted Majority algorithm and our new algorithm can be arbitrarily large.

Finally, a perhaps subtler difference between exponential weights and our new scheme is that each expert's weight in the latter scheme depends not only on the current mistake count of the expert, but also on the current mistake count of the master.

We show that our algorithm has the best possible worst-case mistake bound when the number of experts is very large compared to the loss of the best expert. This lower bound analysis is based on a relation between our prediction problem and Ulam's searching game with a fixed number of lies [19], [20]. We also present a second lower bound argument for our prediction model. This second argument uses a probabilistic construction to prove that both the BW and the tuned Weighted Majority algorithm are asymptotically optimal. That is, the ratio between the mistake bound of either algorithm and the best possible worst case mistake bound goes to 1 as the number N of experts or the loss k of the best expert go to infinity. An equivalent lower bound has been previously obtained by Vovk [21] using arguments from coding theory.

We use the ideas behind the BW master algorithm to devise a method (which we call a *conversion strategy*) to make prediction algorithms robust against noise. The conversion strategy feeds different feedbacks to several copies of the same prediction algorithm. If the noise level is low then one copy will get noiseless data, enabling the conversion strategy to make good predictions. Our upper bound has slightly better constants than the one independently obtained by Auer and Long [5], and is close to the lower bound given by Littlestone and Warmuth [17].

It remains open whether binomial weights also lead to improved master prediction algorithms for the case when the prediction of the master is allowed to be in the continuous interval $[0, 1]$. In this more general setting mistake bounds are replaced by bounds on the total absolute loss. There are master prediction algorithms for this problem [21], [9] using exponential weights, whose mistake bounds are exactly half of the corresponding mistake bounds in the boolean case. However, our attempts to construct a continuous prediction algorithm that achieves half (plus possibly a constant) the loss of the BW algorithm have so far been unsuccessful.

The paper is organized as follows. In Section 2 we present the new algorithm BW, compare it against WM, and prove general lower bounds. In Section 3 we introduce two conversion strategies: one based on binomial weights and one based on exponential weights. Section 4 is devoted to conclusions.

Notation.
The set X represents the set of possible observations. We use $(X \times \{0,1\})^+$ for the set of all finite sequences over $(X \times \{0,1\})$ of nonzero length and $\boldsymbol{s}$ for a sequence $\langle (x_t, y_t) \rangle_t$ (of unspecified length) in $(X \times \{0,1\})^+$ of observations and outcomes. Let $\mathbb{N}$ denote the natural numbers including 0. The notation $\boldsymbol{s}^n$, for any $n \in \mathbb{N}$, represents either a sequence of length n or the length n prefix of a longer sequence $\boldsymbol{s}$. The correct interpretation will be clear from the context.

An *expert* is any function mapping $(X \times \{0,1\})^* \times X$ to $\{0,1\}$. In this paper we treat experts in an on-line fashion. On the tth trial, each expert E makes the prediction $E(\boldsymbol{s}^{t-1}, x_t)$ where $x_t \in X$ is the current observation and $\boldsymbol{s}^{t-1}$ is the sequence of observation/outcome pairs from the previous $t-1$ trials. At the end of the trial the expert is given the feedback $y_t \in \{0,1\}$ for the current trial (and $\boldsymbol{s}^t$ for the next trial is created by appending (x_t, y_t) to $\boldsymbol{s}^{t-1}$). We say that expert E either is wrong, makes a mistake, or is incorrect when its prediction at trial t, $E(\boldsymbol{s}^{t-1}, x_t)$, is different from y_t.

Also, we use $d_H(\boldsymbol{y}, \boldsymbol{z})$ to denote the Hamming distance between any two boolean sequences $\boldsymbol{y}$ and $\boldsymbol{z}$ of equal length. For the sum of binomials, we use the notation $\binom{m}{\leq k} \stackrel{\text{def}}{=} \sum_{i=0}^{k} \binom{m}{i}$ for all integers m and k, using the convention $\binom{m}{\leq k} = 0$ when m or k negative. We conventionally set $\binom{m}{i} = 0$ when $i > m$ or when either m or i is negative. We will often make use of the well-known combinatorial identity

$$\binom{q}{\leq i} = \binom{q-1}{\leq i} + \binom{q-1}{\leq i-1} \tag{1}$$

that holds for all nonzero integers q and all integers i. We denote the binary logarithm by "log" and the natural logarithm by "ln". Furthermore, let $H(\cdot)$ denote the binary entropy function, $H(x) = -x \log x - (1-x)\log(1-x)$, defined for all $0 \leq x \leq 1$ (note that $H(0) = H(1) = 0$ and $H(\frac{1}{2}) = 1$).

2. Master Algorithms for Combining the Predictions of Experts

In this section we introduce a master algorithm that sequentially predicts boolean sequences by combining the predictions of a set of experts. Throughout the section, we assume that a bound k on the number of mistakes made on the sequence by the best expert in the set is available and known to the master algorithm.

For any expert E and for any sequence $\boldsymbol{s} \in (X \times \{0,1\})^+$ of instances and outcomes we denote the number of mistakes (i.e. total loss) of expert E on sequence $\boldsymbol{s}$ by $L_E(\boldsymbol{s})$. Also, if $\mathcal{E}$ is a set of experts, we use $L_{\mathcal{E}}(\boldsymbol{s})$ for the minimum $L_E(\boldsymbol{s})$ over the experts $E \in \mathcal{E}$. We usually make the assumption that $L_{\mathcal{E}}(\boldsymbol{s}) \leq k$ for some constant k known to the master algorithm. We point out that our master algorithms are domain independent, using the information provided by the sequence of instances $\langle x_t \rangle_t$ only to obtain the predictions of the experts.

Our goal is to solve the following problem:

> Suppose a set $\mathcal{E}$ of N experts is available and the task is to predict in an on-line fashion the bits $y_1, y_2, \ldots, y_\ell$ of some sequence $\boldsymbol{s} = (x_1, y_1), (x_2, y_2), \ldots, (x_\ell, y_\ell)$ in a set of sequences $\Sigma \subseteq (X \times \{0,1\})^\ell$. Suppose also that an upper bound k on the loss of the best expert in $\mathcal{E}$ is known, i.e. for each $\boldsymbol{s} \in \Sigma$, $L_{\mathcal{E}}(\boldsymbol{s}) \leq k$. How can a master algorithm combine the experts' predictions so that its worst case number of mistakes is minimized?

If the master algorithm knew which expert $E \in \mathcal{E}$ made only k mistakes, then it could simply predict the same way that expert E does. However, the "good" expert (or experts) is not known in advance.

In the fortunate case where $k = 0$, the master algorithm knows that one of the experts predicts perfectly on $\boldsymbol{s}$. In this case the well-known Halving algorithm [3], [7] can be used. On each trial the Halving algorithm predicts the same way as the majority of those experts that have never made a mistake (the consistent experts). The number of consistent experts is reduced by at least a factor of two each time the Halving algorithm makes a mistake, so the master makes at most $\log N$ mistakes on any $\boldsymbol{s}$ where one of the N experts always predicts correctly.

We now present a simple master algorithm called the Version Space algorithm that will be used to motivate the Binomial Weighting (BW) algorithm. To do this we make the simplifying assumption that the length of the sequence of instances, ℓ, is known as well. This assumption will be removed shortly.

Since the master algorithm knows that the best expert makes at most $k > 0$ mistakes, it can use the following trick. The master algorithm expands each expert into a set of variants so that some variant of some expert predicts perfectly, and then uses the Halving algorithm on the variants. If expert E makes *exactly* j mistakes on some sequence $\boldsymbol{s}$ of length ℓ then expert E can be expanded into a collection of $\binom{\ell}{j}$ variants containing a perfect variant. Each variant in the collection predicts as E on $\ell - j$ of the trials and predicts with the opposite of E's predictions on the other j trials. Thus expert E is expanded into a collection of $\binom{\ell}{j}$ variants, including one that changes E's predictions on exactly those trials where E predicts incorrectly.

For our problem, the master algorithm knows that at least one of the N experts makes at most k incorrect predictions, but the master algorithm knows neither which expert is the best nor the exact number of mistakes made by the best expert. However, the master algorithm can expand each expert into a collection of $\binom{\ell}{\leq k}$ variants. The union of these collections contains at most $N\binom{\ell}{\leq k}$ variants and is guaranteed to contain at least one variant that predicts correctly on all ℓ trials. Our Version Space algorithm runs the Halving algorithm on the union of these collections, and has a worst case mistake bound of $\log N + \log\binom{\ell}{\leq k}$ (when the bounds ℓ on the number of trials and k on the number of mistakes made by the best expert are known in advance).

Intuitively, the Version Space algorithm uses all the knowledge it has about the experts and the sequences, which is that there is one expert that makes at most k mistakes on the sequence. It does not know which expert will be best, in what trials the best expert will make its mistakes, or even how many mistakes the best expert will make (other than the upper bound k). Since the goal of the algorithm is to minimize the number of mistakes that it makes in the worst case, it has to treat all of the scenarios that are possible under the assumptions equally.

Observe that the version space at the beginning of trial t can be represented by one weight per expert. The weight of an expert is simply the number of its $\binom{\ell}{\leq k}$ variants that are consistent with the sequence so far[3]. If expert E makes at most k mistakes on the ℓ trials and has made j mistakes in trials 1 through t, then expert E can make at most $k - j$ more mistakes in the remaining $\ell - t$ trials. Thus the weight of E on the $t + 1$st trial should be $\binom{\ell - t}{\leq k - j}$, which is exactly the number of variants created from E that are consistent. (The initial weight of each expert is $\binom{\ell}{\leq k}$).

Thus the Version Space algorithm can be implemented by manipulating binomials representing the weights (number of consistent variants) of the experts. If expert E has made j mistakes in the first t trials, then during trial $t+1$ expert E votes with weight $\binom{\ell-t}{\leq k-j}$ for its own prediction and with weight $\binom{\ell-t}{\leq k-(j+1)}$ for the opposite prediction. Note that these votes correspond to the number of E's variants that are consistent with all t previous trials and agree (or do not agree, respectively) with the prediction of E. Also, expert E's total weight is split between the two choices since $\binom{\ell-t}{\leq k-j} + \binom{\ell-t}{\leq k-j-1} = \binom{\ell-t+1}{\leq k-j}$.

This implementation of the Version Space algorithm totals the votes for outcome 0 and outcome 1 and predicts with the majority. At the end of each trial t, the Version Space algorithm updates the weights of the experts to reflect the outcome on that trial, y_t. In addition, the value y_t is given to all the experts since their future predictions might depend on the past sequence. The Version Space algorithm, which runs the Halving algorithm directly on the $N\binom{\ell}{\leq k}$ variants, and the implementation which manipulates binomial weights for each expert, clearly make the same predictions.

The Binomial Weighting (BW) algorithm is similar to the Version Space algorithm using weights, but the BW algorithm uses another trick that removes the requirement that the algorithm knows ℓ, the length of the sequence. This trick also makes the upper bound on the number of mistakes made by the BW algorithm independent of ℓ. There are two versions of the Halving algorithm: one that discards all inconsistent experts in each trial and one that does this only in trials when the Halving algorithm makes a mistake (such algorithms are called "conservative" by Littlestone [15]). Both versions of the Halving algorithm have the same worst case mistake bound ($\log N$), so nothing is lost by making the Version Space algorithm conservative. The Binomial Weighting algorithm is the implementation of the conservative Version Space algorithm with binomial weights and is described in Figure 1.

Because the BW algorithm is conservative, we do not need a variant that perfectly predicts the outcome. It suffices to have only those variants whose mistakes occur when the BW master algorithm predicts incorrectly. Since the BW algorithm discards variants only when the master makes a mistake, such a variant will never be discarded. Thus the BW algorithm considers only $\binom{m+1}{\leq k}$ variants[4] of each expert, where $m = \max\left\{q \in \mathbb{N} : q \leq \log N + \log\binom{q}{\leq k}\right\}$ as in Figure 1. It is easy to show that BW makes at most m mistakes. Assume to the contrary that it makes $m+1$ mistakes. Since at least one of the N experts makes at most k mistakes, at least one of the $N\binom{m+1}{\leq k}$ variants is consistent with the $m+1$ outcomes where BW made mistakes. On the other hand, the number of consistent variants drops by a factor of at least two each time BW makes an incorrect prediction. Thus the number of consistent variants after BW makes $m+1$ mistake is at least one

```
Master Algorithm BW
Input: A set of $N$ experts $\mathcal{E}$ and a nonnegative integer $k$.

1. Let $m := \max\left\{q \in \mathbb{N} : q \le \log N + \log \binom{q}{\le k}\right\}$.

2. Set the initial weight of each expert to $\binom{m+1}{\le k}$, and set $m'$, the number of mistakes
   made by the master, to 0.

3. For each trial $t = 1, 2, \ldots$

   (A) For each expert $E \in \mathcal{E}$:
       Let $j$ be the number of previous trials where both $E$ and the master made
       incorrect predictions. Then expert $E$ has current weight $\binom{m+1-m'}{\le k-j}$ and
       votes for its own prediction with weight $\binom{m-m'}{\le k-j}$ and with weight $\binom{m-m'}{\le k-j-1}$
       for the opposite prediction.

   (B) Sum the votes for bit 0 and for bit 1 and predict with the majority (arbi-
       trary in case of a tie).

   (C) Get the correct prediction $y_t$.

   (D) If a mistake occurred, then increment $m'$ and update the weight of each
       expert to the weight with which it voted for correct bit $y_t$.
```

Figure 1. The Binomial Weighting algorithm.

and at most $N\binom{m+1}{\le k}/2^{m+1}$. It follows that $1 \le N\binom{m+1}{\le k}/2^{m+1}$ and equivalently $m+1 \le \log N + \log\binom{m+1}{\le k}$, contradicting the definition of m in Figure 1.

This analysis gives us the following theorem:

Theorem 1 *For all $k \in \mathbb{N}$, all nonempty sets $\mathcal{E}$ of experts, and all sequences $\boldsymbol{s} \in (X \times \{0,1\})^+$; if $L_{\mathcal{E}}(\boldsymbol{s}) \le k$, then the total number of mistakes of $BW(k)$ on $\boldsymbol{s}$ is at most*

$$\max\left\{q \in \mathbb{N} : q \le \log N + \log\binom{q}{\le k}\right\} \quad , \tag{2}$$

where $N > 0$ is the number of experts in $\mathcal{E}$.

We now describe a variant of algorithm BW, called BW′ (see Figure 2), that has the same worst-case mistake bound proven in Theorem 1. However, for many sequences of examples the new algorithm BW′ makes fewer mistakes than the original algorithm. The current weight of an expert E is now $\binom{m+1}{\le k-j}$, where j is the number of mistakes of E in *all* previous trials and not just in the trials in which the master made mistakes as well. The value of m is recomputed at the beginning of each trial. This value will decrease by at least one after all trials in which the master made a mistake, because the total weight after such a trial is at most half of what

```
Master Algorithm BW'
Input: A set of $N$ experts $\mathcal{E}$ and a nonnegative integer $k$.

1. For each expert $E \in \mathcal{E}$ set the mistake budget $k_E$ equal to $k$.

2. For each trial $t = 1, 2, \ldots$
    (A) Let $m := \max\left\{q \in \mathbb{N} : q \le \log\left(\sum_{E\in\mathcal{E}} \binom{q}{\le k_E}\right)\right\}$.
    (B) For each expert $E \in \mathcal{E}$:
        Expert $E$ has current weight $\binom{m+1}{\le k_E}$ and votes for its own prediction with
        weight $\binom{m}{\le k_E}$ and with weight $\binom{m}{\le k_E - 1}$ for the opposite prediction.
    (C) Sum the votes for bit 0 and for bit 1 and predict with the majority (arbi-
        trary in case of a tie).
    (D) Get the correct prediction $y_t$.
    (E) Decrease the mistake budget, $k_E$, of all experts that predicted incorrectly
        in this trial by 1.
```

Figure 2. The Modified Binomial Weighting algorithm.

it was before the trial (decreasing m by at least one corresponds to increasing m' in BW). The value of m can never increase but it might also decrease after trials in which the master made no mistakes. Again it can be shown by induction that the number of mistakes from any trial onward is at most the value of m computed at the beginning of that trial.

2.1. Comparison with Weighted Majority

In this section we compare the performances of the BW and Weighted Majority (WM) algorithms. The WM algorithm has a parameter $\beta \in [0, 1)$. An expert E votes for its own prediction with weight β^j, where j is the number of mistakes made by expert E in the past, and for the opposite prediction[5] with weight β^{j+1}.

Both master algorithms predict 1 if and only if the experts predicting 1 outweigh[6] the experts predicting 0. The weights used by the BW algorithm are binomial tails whereas the WM algorithm uses exponential weights of the form β^j. We often refer to β as the "update factor" of the WM algorithm because an expert's weight gets multiplied by β when the expert predicts incorrectly. As one would expect, the choice of β greatly affects how the WM algorithm performs.

In our setting the master algorithms are given two parameters: N, the number of experts and a bound k on the number of mistakes made by the best expert. We are interested in worst case bounds on the algorithm's performance as functions of N and k.

For any master algorithm A, define the worst case number of mistakes $\mathrm{WC}_A(N,k)$ as:

$$\mathrm{WC}_A(N,k) \stackrel{\mathrm{def}}{=} \max_{\mathcal{E} \text{ of } N \text{ experts}} \quad \max_{\boldsymbol{s}: L_{\mathcal{E}}(\boldsymbol{s}) \leq k} [\text{number of mistakes of } A(\mathcal{E},k) \text{ on } \boldsymbol{s}] \,.$$

Furthermore, denote the performance of the best master algorithm by $\mathrm{WC}(N,k)$, so

$$\mathrm{WC}(N,k) \stackrel{\mathrm{def}}{=} \min_{\text{algorithms } A} \mathrm{WC}_A(N,k).$$

We will show in Subsection 2.3 that if the number of experts is large enough then the BW algorithm is (essentially) optimal. That is, for any $k \geq 0$, there exists N_k such that for all $N > N_k$

$$\mathrm{WC}_{\mathrm{BW}}(N,k) \leq \mathrm{WC}(N,k) + 1.$$

We can only prove the above for $N_k = \Omega(2^{2^k})$. However we show in Subsection 2.2 that BW is asymptotically optimal, i.e. the ratio $\mathrm{WC}_{\mathrm{BW}}(N,k)/\mathrm{WC}(N,k)$ goes to 1 when N or k goes to infinity (see Theorem 3).

Comparing the BW and WM algorithms is complicated by the fact that WM's mistake bound depends on how the update factor β is chosen (as a function of N and k). For $\beta \in [0,1)$, let WM^β denote the WM algorithm that chooses the update factor β. From Littlestone and Warmuth [17] we have the following mistake bound for the WM algorithm

$$\mathrm{WC}_{\mathrm{WM}^\beta}(N,k) \leq \frac{\log N + k \log \frac{1}{\beta}}{\log \frac{2}{1+\beta}}. \tag{3}$$

Since we will be frequently using this upper bound on $\mathrm{WC}_{\mathrm{WM}^\beta}(N,k)$, we define

$$\mathrm{up}(N,k,\beta) \stackrel{\mathrm{def}}{=} \frac{\log N + k \log \frac{1}{\beta}}{\log \frac{2}{1+\beta}}. \tag{4}$$

Let β^* be the value of β (as a function of N and k) that minimizes $\mathrm{up}(N,k,\beta)$. Vovk [21] gives an implicit formula for β^*. An explicit approximation to β^* is given in Cesa-Bianchi *et al.* [9]. With β set to this approximation, they show that $\mathrm{up}(N,k,\beta) \leq 2k + 2\sqrt{k \ln N} + \log N$. We show that $\mathrm{up}(N,k,\beta^*) \sim \mathrm{WC}(N,k)$ whenever N or k goes to infinity (see Theorem 3).

Although both $\mathrm{up}(N,k,\beta^*)$ and $\mathrm{WC}_{\mathrm{BW}}(N,k)$ have the same leading term when N and/or k is large, there can be significant differences between them. We show below that our bound on the BW algorithm is always at least as good as the known bounds on the WM algorithm, i.e. that $\mathrm{WC}_{\mathrm{BW}}(N,k) \leq \mathrm{up}(N,k,\beta^*)$ for all choices of N and k (see Theorem 2). However, as we shall discuss below, at least for small values of N, the upper bound on the WM algorithm, $\mathrm{up}(N,k,\beta^*)$, is weak and misleading.

Let WM* be the WM algorithm that uses update factor β^* and WM$^+$ be the WM algorithm that chooses β as a function of N and k so that $\mathrm{WC}_{\mathrm{WM}^{\beta}(N,k)}$ is minimized. Unfortunately, we don't know how to efficiently compute the value of β used by WM$^+$. The value of $\mathrm{WC}_{\mathrm{WM}^+}(N,k)$ is much smaller than $\mathrm{WC}_{\mathrm{WM}^*}(N,k)$ for some choices of N and k. It is even conceivable that $\mathrm{WC}_{\mathrm{WM}^+}(N,k)$ is smaller than $\mathrm{WC}_{\mathrm{BW}}(N,k)$ for some N,k pairs, although this disagrees with our intuition.

To make the weakness of inequality (3) concrete, consider the case when there are three experts ($N = 3$). It is easy to see that $\mathrm{BW}(3,k) = 2k+1$, which is the best possible. Also $\mathrm{WC}_{\mathrm{WM}^{\beta}(N,k)} = 2k+1$ whenever $0 < \beta < 1/2$. However, the value of β that minimizes $\mathrm{up}(3,k,\beta)$ approaches 1 when $N = 3$ and k becomes large. In fact, $\mathrm{up}(3,k,\beta^*)$ grows as $2k + \Omega(\sqrt{k})$. Thus the bound $\mathrm{up}(3,k,\beta^*)$ overestimates the number of mistakes made by WM$^+$ by an (additive) $\Omega(\sqrt{k})$ term. Intuitively, a reason for this is that when β is large then two poorly performing experts can outweigh the good expert and cause the master to make unnecessary mistakes.

The main difference between the WM and BW algorithms is how the weights are updated. The WM algorithm uses a fixed update factor throughout the entire learning process. The update factor β can be written as $e^{-\eta}$, where $\eta > 0$ has the natural interpretation as a learning rate. When η is small, β is large, and the WM algorithm learns slowly. When η is large, β is small and the WM algorithm rapidly slashes the weights of poorly performing experts. The disadvantage of a high learning rate is that the algorithm might discount experts too quickly, causing its predictions to be dominated by only a few experts.

When the BW algorithm changes an expert's weight from $\binom{m-m'+1}{\le k-j}$ to $\binom{m-m'}{\le k-j-1}$ then this can be seen as multiplying the expert's weight by an update factor that depends on m', the number of mistakes made so far by the master algorithm (as well as j, the number of mistakes made by the expert, N, and k). These update factors used by BW become less drastic as the number of mistakes made by the master increases (and the upper index of the binomial coefficients decreases). This represents a kind of annealing schedule performed on the learning rate (see e.g. [1] for examples of annealing): when the master knows nothing the learning rate is relatively high and as the master learns the learning rate decreases in order to preserve the previously acquired knowledge. Although one could use any of a number of *ad hoc* heuristics for "cooling down" the learning rate, we have seen that the binomial weights are theoretically justified by the version space argument.

Our belief is that the single update factor used by WM$^*(N,k)$ attempts to approximate the sequence of update factors used by $\mathrm{BW}(N,k)$. In addition to the update relationships between the two algorithms, our proof techniques provide further evidence for this belief. Both the optimization of WM's update factor β as a function of N and k (Lemma 1) and the proof that the bound for WM* is always worse than the BW bound (Theorem 2) use techniques similar to those used to prove Chernoff bounds for binomial tails [11].

We now proceed to compare the bounds on the WM and BW algorithms, beginning with an examination of the β^* minimizing $\mathrm{up}(N,k,\beta)$. Here we re-derive the implicit form of β^* given by Vovk [21]. Recall that H denotes the binary entropy.

Lemma 1 (See also [21]) *For all $N \geq 2$, for all $k \geq 0$, and for all $\beta \in [0,1)$; if $m = k(1+\beta)/\beta$ (so that $m > 2k$ and $\beta = \frac{k}{m-k}$), then the following are equivalent:*

a. $\dfrac{\partial \mathrm{up}(N,k,\beta)}{\partial \beta} \leq 0$,

b. $\beta \leq \dfrac{k}{\mathrm{up}(N,k,\beta) - k}$,

c. $m \geq \mathrm{up}\left(N, k, \dfrac{k}{m-k}\right)$, *and*

d. $m \geq \log N + mH\left(\dfrac{k}{m}\right)$,

where the function up *is defined in (4). Also, there is exactly one $m^* > 2k$ for which the last inequality is an equality and the corresponding β^* is the unique minimum of* $\mathrm{up}(N,k,\beta)$.

The proof of this Lemma is shown in Appendix B.

Lemma 1 shows that, when N and k are fixed, the unique solution m^* to $m = \log N + mH(\frac{k}{m})$ is the minimum value of $\mathrm{up}(N,k,\beta)$. Although m^* (and $\beta^* = \frac{k}{m^*-k}$) is a function of N and k, we suppress this dependence to simplify our notation. Also if $m \geq m^*$ and $\beta = \frac{k}{m-k}$ then m is an upper bound on $\mathrm{up}(N,k,\beta) \geq \mathrm{WC}_{\mathrm{WM}^\beta}(N,k)$. Since we are computing integer-valued mistake bounds, it suffices to find any $m' \in \mathbb{R}$ such that $\lfloor m' \rfloor = \lfloor m^* \rfloor$. Note that $m > \log N + mH(\frac{k}{m})$ when $m > m^*$ and $m < \log N + mH(\frac{k}{m})$ when $m < m^*$. Therefore we can find an appropriate m' by doing binary search. Since $\mathrm{WC}(N,k) \geq 2k + \lfloor \log N \rfloor$ (as proven by Littlestone and Warmuth [17]) and $m^* \leq 2k + 2\sqrt{k \ln N} + \log N$ as shown by Cesa-Bianchi *et al.* [9], the search can be limited to the range $[2k + \lfloor \log N \rfloor, 2k + 2\sqrt{k \ln N} + \log N]$. Thus the binary search takes at most $O(\log k + \log\log N)$ time.

Our experience indicates that m^* tends to be close to the right edge of this range. For $N = 3$, m^* is within 1 of $2k + 2\sqrt{k \ln N} + \log N$. For arbitrary N the right boundary seems to be at most $\log N$ greater than m^*. However these considerations are based on numerical plots and have not been verified analytically.

We now show that BW beats the bound obtained by minimizing the upper bound for WM^β. We need a preliminary lemma that is easily derived from the Binomial Theorem.

Lemma 2 *For all $m, k \in \mathbb{N}$ and all $0 \leq \beta \leq 1$, if $k \leq m$ then*

$$\binom{m}{\leq k} \leq \frac{(1+\beta)^m}{\beta^k}. \qquad (5)$$

Recall that $m^* = \mathrm{up}(N,k,\beta^*)$ for $\beta^* = \frac{k}{m^*-k}$ is the minimum of $\mathrm{up}(N,k,\beta)$ over all $\beta \in [0,1)$. Similarly, let q^* be the largest integer q such that $q \leq \log N + \log \binom{q}{\leq k}$.

While m^* is the upper bound on Weighted Majority derived from inequality (3), q^* is the upper bound on the Binomial Weighting algorithm in Theorem 1 (q^*, like m^*, implicitly depends on N and k).

THEOREM 2 *For all nonnegative integers k and positive integers N, if q^* is the largest integer q such that $q \leq \log N + \log \binom{q}{\leq k}$, then $\mathrm{WC}_{\mathrm{BW}}(N,k) \leq q^*$ and $q^* \leq \mathrm{up}(N,k,\beta)$, for all $\beta \in [0,1)$.*

Proof: The fact that $\mathrm{WC}_{\mathrm{BW}}(N,k) \leq q^*$ follows from Theorem 1. Let β be any real in $[0,1)$. By Lemma 2 the solution to $q = \log N + \log \binom{q}{\leq k}$ is never larger than the solution m_β to $m = \log N + m \log(1+\beta) - k \log \beta$. Since solving for m_β yields

$$m_\beta = \frac{\log N + k \log \frac{1}{\beta}}{\log \frac{2}{1+\beta}} = \mathrm{up}(N,k,\beta),$$

this proves the theorem. ∎

As mentioned above, when $N = 3$ the worst case performance of WM^+ (which uses the best choice of β, rather than the β^* minimizing the bound) equals q^*. Furthermore, the gap between these two and m^* grows as $\Omega(\sqrt{k})$. If N is large compared to k, we believe that the upper bound m^* is much closer to $\mathrm{WC}_{\mathrm{WM}^+}(N,k)$. However, even when N is large, q^* can be significantly less than m^*.

Pick any $k \geq 1$. If N satisfies[7]

$$\frac{2^{4k}}{\binom{4k}{\leq k}} \leq N < \frac{2^{4k+1}}{\binom{4k+1}{\leq k}}$$

then $q^* = 4k$. With a bit of algebra (and Stirling's approximation) it can be shown that m^* is at least $4k + \frac{\log(3k)-1}{2}$. In other words, when N is about $2^{4k}/\binom{4k}{\leq k}$, the mistake bound on BW of Theorem 1 is at least $\frac{\log(3k)-1}{2}$ better than the best known bound for the Weighted Majority algorithm. Although our bounds on the BW algorithm are better than the $\mathrm{up}(N,k,\beta^*)$ bounds on the WM algorithm, asymptotically the two bounds have the same leading term. This is shown in the following section.

2.2. Asymptotic performance of the algorithms

This subsection shows that both BW and WM^* are asymptotically optimal in the worst case. The proof uses a probabilistic argument to show the existence of "hard" sets of experts. Using these hard sets of experts, an adversary can force any prediction algorithm to make a mistake on each trial proving the desired lower bound. We use the notation $f_i \sim g_i$ when $\lim_{i\to\infty} f_i/g_i = 1$. We define the following functions to serve as lower bounds

$$\mathrm{low}(N,k) \stackrel{\mathrm{def}}{=} \max\left\{ q \in \mathbb{N} \,:\, q \leq \log N + \log \binom{q}{\leq k} - \log\left(1 + \ln \binom{q}{\leq k}\right)\right\},$$

$\mathrm{Low}(N,k) \overset{\mathrm{def}}{=} \max(\mathrm{low}(N,k), 2k + \log N)$.

We now state the two results of this section.

THEOREM 3 *For any integers $N \geq 2$ and $k \geq 0$, there exists a set $\mathcal{E}$ of N experts such that the following holds for any deterministic master algorithm A: there exists a sequence $\boldsymbol{s}$ of trials such that $L_{\mathcal{E}}(\boldsymbol{s}) \leq k$ and A makes at least* $\mathrm{Low}(N,k)$ *mistakes on $\boldsymbol{s}$.*

The above lower bound is then used to show that BW and WM* are both asymptotically optimal.

THEOREM 4 *For any sequence $\{(N_i,k_i)\}_{i\in\mathbb{N}}$ of pairs of positive integers, if $N_i \geq 2$ for all i and $\lim_{i\to\infty} N_i = \infty$ or $\lim_{i\to\infty} k_i = \infty$, then as $i \to \infty$,*

$$\mathrm{Low}(N_i,k_i) \sim \mathrm{WC}_{\mathrm{BW}}(N_i,k_i) \sim \mathrm{WC}_{\mathrm{WM}^*}(N_i,k_i) \sim \mathrm{up}(N_i,k_i,\beta_i^*) \ ,$$

where $\beta_i^* = \dfrac{k_i}{\mathrm{up}(N_i,k_i,\beta_i^*) - k_i}$.

Before proving Theorem 3, we need some definitions and lemmas. The first lemma is from Littlestone and Warmuth.

LEMMA 3 ([17]) *For any integers $N \geq 2$ and $k \geq 0$, there exists a set $\mathcal{E}$ of N experts such that the following holds for any deterministic master algorithm A: there exists a sequence $\boldsymbol{s}$ of trials such that $L_{\mathcal{E}}(\boldsymbol{s}) \leq k$ and A makes at least $2k + \log N$ mistakes.*

The above lemma proves the first lower bound used in the definition of Low. The second lower bound is proven using a covering argument. For any positive integer q and any nonnegative integer $k \leq q$, a k-covering of the q-dimensional boolean hypercube is a subset $\mathcal{B}$ of $\{0,1\}^q$ such that for any $\boldsymbol{v} \in \{0,1\}^q$ there is a $\boldsymbol{p} \in \mathcal{B}$ such that $d_H(\boldsymbol{p},\boldsymbol{v}) \leq k$. If in the on-line prediction setting the experts' predictions are solely a function of the trial number, then each expert can be viewed as a sequence of bits. Furthermore, a set $\mathcal{E}$ of such experts is a k-covering for some subset $\{t_1, t_2, \ldots, t_q\}$ of trials if the set of the sequences of length q representing the predictions of the experts in the trials $t_1, t_2, \ldots, t_q$ is a k-covering of $\{0,1\}^q$.

Now we give a technical lemma showing that some coverings are not too large. We adapt a nonconstructive argument of Alon and Spencer from [2], Theorem 2.2, page 6.

LEMMA 4 *For all $N \geq 1$ and for all $k \geq 0$, if $m = \mathrm{low}(N,k)$, then there is a k-covering of $\{0,1\}^m$ of size at most N.*

Proof: We prove the lemma using a probabilistic argument. Let $R \subseteq \{0,1\}^m$ be chosen randomly so that the event $\boldsymbol{v} \in R$ occurs with probability $p > 0$ (to be specified later) independently for any $\boldsymbol{v} \in \{0,1\}^m$. Let R' be the subset of $\{0,1\}^m$ containing all points not k-covered by R. Clearly $R \cup R'$ is a k-covering of $\{0,1\}^m$.

Observe that any $\boldsymbol{z}$ belongs to R' if and only if for any $\boldsymbol{v} \in R$, $d_H(\boldsymbol{z}, \boldsymbol{v}) > k$. This implies $\Pr(\boldsymbol{z} \in R') = (1-p)^{\binom{m}{\le k}}$, since there are $\binom{m}{\le k}$ corners of the m-dimensional boolean hypercube with Hamming distance at most k from $\boldsymbol{z}$ ($\boldsymbol{z}$ itself included). From the above it is easy to compute the expectation of the random variable $|R| + |R'|$.

$$\mathbf{E}[|R| + |R'|] = 2^m p + 2^m (1-p)^{\binom{m}{\le k}}.$$

Now set $p = \frac{\ln \binom{m}{\le k}}{\binom{m}{\le k}}$. Then

$$\begin{aligned} \mathbf{E}[|R| + |R'|] &= 2^m \left[\frac{\ln \binom{m}{\le k}}{\binom{m}{\le k}} + \left(1 - \frac{\ln \binom{m}{\le k}}{\binom{m}{\le k}} \right)^{\binom{m}{\le k}} \right] \\ &\le 2^m \left[\frac{\ln \binom{m}{\le k}}{\binom{m}{\le k}} + \exp\left(- \ln \binom{m}{\le k} \right) \right] \qquad (6) \\ &= 2^m \frac{1 + \ln \binom{m}{\le k}}{\binom{m}{\le k}} \end{aligned}$$

where inequality (6) holds since $1 - x \le e^{-x}$ for all $x > 0$. Thus, if $N \ge 2^m \frac{1+\ln \binom{m}{\le k}}{\binom{m}{\le k}}$ then the m-dimensional boolean cube is k-covered by a set of size N. Solving this inequality for m yields that $m \le \log N + \log \binom{m}{\le k} - \log(1 + \ln \binom{m}{\le k})$, or equivalently that $m \le \mathrm{low}(N, k)$ ensures that the m-dimensional boolean cube has a k-covering of size N. ■

Proof of Theorem 3: In view of the lower bound stated in Lemma 3 it suffices to prove a second lower bound of $\mathrm{low}(N, k)$ mistakes. We use Lemma 4 to do this. Choose a sequence $\{x_i\}_{i \in \mathbb{N}}$ of distinct observations. Choose integers $N \ge 2$ and $k \ge 0$. Let $m = \mathrm{low}(N, k)$. By Lemma 4, there exists a set $\mathcal{E}$ of N experts, whose predictions depend only on the trial number, such that $\mathcal{E}$ is a k-covering for the first m prediction trials. Now notice that, if $\mathcal{E}$ is a k-covering for the first m trials, an adversary can force m mistakes on any deterministic prediction algorithm. The adversary simply chooses the sequence $\boldsymbol{y}$ of outcomes, of length m, such that y_t is the opposite of the algorithm's prediction on the tth trial. Since $\mathcal{E}$ is a k-covering of $\{0,1\}^m$, for any such sequence $\boldsymbol{y}$ of outcomes there is some expert in $\mathcal{E}$ which makes at most k mistakes on $(x_1, y_1), \ldots, (x_m, y_m)$. ■

Proof of Theorem 4: By Theorem 3 we know that $\mathrm{Low}(N, k)$ is a lower bound on the number of mistakes for any deterministic master algorithm.

Let $\omega = \{(N_i, k_i)\}_{i \in \mathbb{N}}$ be a sequence as in the statement of the theorem. Since by Lemma 1 and Theorem 2

$$\mathrm{Low}(N_i, k_i) \le \mathrm{WC}_{\mathrm{BW}}(N_i, k_i) \le \mathrm{up}(N_i, k_i, \beta_i^*)$$

and

$$\mathrm{Low}(N_i, k_i) \le \mathrm{WC}_{\mathrm{WM}^*}(N_i, k_i) \le \mathrm{up}(N_i, k_i, \beta_i^*)$$

it is sufficient to show that

$$\lim_{i\to\infty} \frac{\mathrm{Low}(N_i, k_i)}{\mathrm{up}(N_i, k_i, \beta_i^*)} = 1 \ . \tag{7}$$

The proof of (7) is shown in Appendix C. ■

2.3. Lower bounds based on Ulam's game

In this section we give lower bounds on the performance of prediction strategies. We show that for any fixed number of mistakes k of the best expert and for any prediction algorithm, there exists a set $\mathcal{E}$ of experts and a sequence $\boldsymbol{s}$ such that $k = L_{\mathcal{E}}(\boldsymbol{s})$ for which the number of mistakes made by the prediction algorithm is at least as large as the number of mistakes made by BW.

We start by introducing some notation that lets us give a precise statement of our lower bound. We then describe Ulam's game with lies and its relation to our prediction problem. Finally, we show how Spencer's results [19] can be used to prove our lower bound.

In all of the following discussion we shall think of k, the upper bound on the number of mistakes made by the best expert, as being fixed. Let $J(k, q)$ be the following sequence of numbers indexed by q:

$$J(k, q) = 2^q \Big/ \binom{q}{\le k}.$$

It is easy to check that $J(k, q+1) \ge (5/4)J(k, q)$, for any $q \ge 3k + 2$, thus the sequence $J(k, q)$ increases (at least) exponentially.

Theorem 5 *For every nonnegative integer k there exists an integer N_k such that for all $N > N_k$ the following holds:*
If q is the integer such that $J(k, q) \le N < J(k, q+1)$, then

1. $\mathrm{WC}_{\mathrm{BW}}(N, k) \le \mathrm{WC}(N, k) + 1$.
2. *If* $J(k, q) + 2^k \le N$, $\mathrm{WC}_{\mathrm{BW}}(N, k) = \mathrm{WC}(N, k)$.

Observe that the upper bound on algorithm BW is always guaranteed to be within 1 mistake of the optimal algorithm when N is large enough. Also, since the size of the segment $[J(k, q), J(k, q+1)]$ increases exponentially with q, as q increases the set of values for N where the second case holds (i.e. the lower bound is off by 1 from BW's upper bound) becomes an insignificantly small fraction of the possible values for N. This shows that BW is very close to optimal for large values of N. The gap of 1 when $N < J(k,q)+2^k$ arises from complicated number-theoretic considerations. In Appendix A we show how algorithm BW can be modified so that it is completely optimal for large N. The weakness of this lower bound construction is that the threshold N_k above which the lower bound holds is rather large, on the order of 2^{2^k}. This double-exponential dependence on k arises from our use of Spencer's results [19].

Before we give the proof of Theorem 5, we briefly describe Ulam's game with a fixed number of lies and show how this game relates to chip games and to the problem of combining the predictions of experts.

In the searching game introduced by Ulam (see [20]) there are two players: a *chooser* (also called Carol) and a *partitioner* (also called Paul). A game is defined by three nonnegative integers N, k, and q that are known to both players. Carol is assumed to select a secret number x from the set $\{1,\ldots,N\}$. Paul's goal is to find out what this number is by asking Carol questions of the form "Is x in S?", where S is any subset of $\{1,\ldots,N\}$. Carol is required to answer either "yes" or "no". However, she is allowed to lie (i.e. give the incorrect answer to Paul's question) up to k times.[8] We say that Paul wins the (N,k,q) game if and only if he can always identify Carol's secret number after at most q questions, regardless of Carol's strategy.

The interesting fact is that there is a common abstraction of Ulam's game with lies and of our problem. The abstraction can be seen as the following chip game (for more work on chip games, see [4]). We think of each number in the set $\{1,\ldots,N\}$ as a "chip" and consider $k+1$ (disjoint) subsets of these chips, which we call "bins", and denote by $B_0,\ldots,B_k$. At each point of the game, the bin B_j contains all the chips that correspond to a number $x \in \{1,\ldots,N\}$ with the property that if x is the number chosen by Carol, then j of the answers that Carol gave so far have been lies. Thus the union of all the bins contain those choices of x that are consistent with the bound k on the number of lies that Carol is allowed to make. Essentially, it is sufficient to describe each configuration reached during the game by the number of chips in each bin. We denote by $I^j = (I_0^j,\ldots,I_k^j)$ the configuration of the chip game after at the jth trial, where I_i^j is a natural number denoting the number of chips in B_i. For example, the initial configuration is always $I^0 = (N,0,\ldots,0)$.

When Paul asks "Is x in S ?", his question partitions the chips into two sets, those in S versus those outside S. If Carol answers "no" her answer constitutes a lie with respect to the numbers in S. This translates to advancing each chip corresponding to a number in S from its current bin to the next bin (e.g. from bin B_j to B_{j+1}). If a chip corresponding to a number in S is already in the last bin B_k, it is discarded as there is no bin B_{k+1}. If Carol answers "yes", then those chips corresponding to numbers not in S are advanced.

Clearly Paul cannot know which number Carol has chosen as long as the union of the bins contains at least two chips. Thus Carol's goal is to keep two chips in the union of the bins for as long as possible. Paul wins the (N,k,q) iff there is a strategy for choosing partitions guaranteeing that after q steps there is at most 1 chip remaining in the union of the bins.

We can think of the prediction problem as a "prediction game" where the predictor is playing against an adversary that picks both the predictions generated by the experts, and the outcomes.[9] We restrict our attention to those adversary strategies that force the prediction algorithm to make a mistake on each and every trial for as long as possible. This means until one expert has made k mistakes and every other expert has made more than k mistakes, the adversary chooses the feedback so that the prediction algorithm makes a mistake on every trial. From this point on, the predictions of the single best expert are guaranteed to be without mistakes, and by copying the predictions of this expert the master algorithm will correctly predict the remainder of the sequence. This restriction is helpful to map to the prediction game into a chip game, and restricting the adversary in this way does not reduce its power since we are able to obtain a lower bound that essentially matches the upper bound of the BW algorithm.

We can easily relate this "prediction game" to a chip game. Each chip corresponds to an expert and the bin B_j, for $0 \leq j \leq k$, contains those chips corresponding to experts that have made exactly j mistakes on previous trials. Each iteration of the game starts with the adversary partitioning the chips to two sets according to the predictions given by the corresponding experts. The prediction algorithm then chooses its prediction, and the adversary forces a mistake by generating an outcome opposite to the prediction. This causes those chips corresponding to experts whose predictions were mistaken to advance one bin. Thus the prediction algorithm (indirectly) chooses which subset of the chips gets advanced, so the prediction algorithm corresponds to Carol and the adversary corresponds to Paul. The game ends when the configuration $(0, 0, \ldots, 1)$ is reached, we shall refer to this configuration as the *terminal* configuration. This is a slight difference from the chip game that corresponds to Ulam's game with k lies. Another, much more significant difference, is that the goals of the opponents have been reversed. In the chip game corresponding to the prediction problem, Carol (the prediction algorithm) wants to *shorten* the game as much as possible since the length of the game measures the number of mistakes that the prediction algorithm is forced to make.

As the goals of Carol and Paul have been reversed, it would seem that their strategies for playing the two games would be very different. Surprisingly, it turns out that the optimal strategy for Paul is the same in the two games when the different ending condition is ignored. If $N \geq N_k$ then this optimal strategy Paul can force both games to have the same length, regardless of the actions taken by Carol. In other words, if Paul uses this strategy then Carol is unable to make the game either longer or shorter.

This strategy for Paul has been developed by Spencer [19], and is the basis of the proof of Theorem 5. We shall briefly describe the strategy, give Spencer's result, and then use it to prove Theorem 5.

Spencer identifies the same binomial weights that are used in the BW algorithm as the central quantities on which the strategies of both Carol and Paul are based.

We shall denote by $W_q(I)$ the weight associated with the configuration I and the integer q, i.e.

$$W_q(I) = \sum_{i=0}^{k} I_i \binom{q}{\leq k-i}.$$

Spencer gives a strategy for Carol. Under this strategy Carol advances those chips that keep the future configurations as heavy as possible. The exact opposite choice is made by the BW algorithm, which advances the heavier chips, resulting in a *lighter* configuration. This makes intuitive sense, because Carol has opposite goals in the two games.

The main result of Spencer's paper [19] identifies a class of "good" configurations. For each configuration in this class there exists a partition such that both future configuration have equal weight, equal to half the weight of the current configuration, and both configurations are either good or consist of a single chip. Thus, starting from a good configuration, Paul can repeatedly partition the chips in such a way that in each step the weight is halved until only a single chip remains. It is clear that, by choosing these partitions, Paul can completely neutralize Carol once one of the good configurations is reached. The definition of the good configurations rests on the observation that the weight associated with the chips in bin B_k is always 1, because $\binom{q}{\leq 0} = 1$. These chips are appropriately referred to as "pennies". It is clear that if a configuration has a sufficient number of pennies, and the total weight is even, then by moving pennies from one set of the partition to the other one can equalize the weight of the two successor configurations. Paul's strategy is to choose a partition whose two successor configurations are almost balanced and then use pennies to balance them completely. The main theorem in Spencer's paper shows that if the initial configuration has a sufficient number of pennies, Paul can use this technique repeatedly, without running out of pennies until a configuration with a single chip is reached.

We now give the main result from Spencer's paper in a form that fits our needs.

Theorem 6 ([19]) *For any number $k > 0$ of bins, there exist finite integers $c(k)$ and $q_0(k)$ such that the following holds for any $q > q_0(k)$: if $I^0 = (I_0^0, \ldots, I_k^0)$ is an initial configuration such that $I_k^0 > c(k)q^k$ and $W_q(I^0) = 2^q$, then there exists a strategy for Paul such that, independent of the choices made by Carol, a configuration I^m is reached such that $\sum_{i=0}^{k} I_i^m = 1$ and $W_{q-m}(I^m) = 2^{q-m}$.*

In other words, Paul can guarantee that the total weight is exactly halved at each step, until only a single chip is left.

Proof of Theorem 5: The proof is divided into two parts, we first show that if N is large enough then from the initial configuration $I^0 = (N, 0, \ldots, 0)$ Paul can reach, in k steps, a configuration that meets the conditions of Theorem 6. In the second part we show that the final configuration reached in Theorem 6 guarantees the bound given in the theorem.

In the proof we make use of the idea that Paul "marks" chips as useless. If a chip is marked on some particular trial, then this chip is placed arbitrarily in the partitions generated by Paul on subsequent trials. We shall prove that Paul can delay reaching a terminal configuration even when only the unmarked chips are considered. It is clear that if the marked chips were also considered, then reaching the terminal configuration would be delayed for at least as long, which proves the lower bound on the number of trials.

Initially, all N chips are in bin B_0. It takes at least k steps to get chips to bin B_k and thus make them into pennies. We shall devise a strategy for the first k trials that is guaranteed to give rise to a sufficient number of pennies at the kth trial. First, Paul marks some chips so as to make the number of unmarked chips divisible by 2^k. Clearly, less than 2^k chips need to be marked. Ignoring the marked chips Paul generates the following partitions. The (unmarked) chips in each bin are divided into two equal parts, one part from each bin is placed in the first set of the partition, and the other part is placed in the second. It is easy to check that, independently of Carol's actions, such partitioning of the unmarked chips is possible for k steps. It is also simple to see that after k trials exactly a fraction of 2^{-k} of the unmarked chips reach bin B_k and become pennies.

Let q be the integer such that $J(k,q) \le N \le J(k,q+1)$. From (1) it is clear that the weight that is associated with the unmarked chips is divided by two at each step. Thus, independently of Carol's choices, the weight of the configuration after k steps satisfies

$$W_{q-k}(I^k) > 2^{-k}(N-2^k)\binom{q}{\le k}. \tag{8}$$

To apply Theorem 6 we need that the remaining weight (after k steps) of the unmarked chips is a power of two. We first find an appropriate $\tilde{q}$ such that $W_{\tilde{q}}(I^k) > 2^{\tilde{q}}$.

By the definition of q, $J(k,q) \le N \le J(k,q+1)$. If N is large enough then $J(k,q) - J(k,q-1) \ge 2^k$ and thus $N \ge J(k,q-1) + 2^k$. This implies that $(N-2^k)\binom{q-1}{\le k} \ge 2^{q-1}$ and thus by inequality (8), $W_{q-k-1}(I^k) > 2^{q-k-1}$. It follows that if N is large enough then we can always choose $\tilde{q} = q-k-1$. However if $N \ge J(k,q) + 2^k$, then by the same derivation we get $W_{q-k}(I^k) > 2^{q-k}$ and we can set $\tilde{q} = q-k$.

We now wish to apply the results of Theorem 6 to the configuration I^k, whose weight satisfies $W_{\tilde{q}} > 2^{\tilde{q}}$. However, in order to obey the conditions of the theorem we have to mark some more chips in order to make the weight of the configuration satisfy $W_{\tilde{q}}(I^k) = 2^{\tilde{q}}$. We do this marking carefully, so that afterwards we still have enough unmarked pennies to apply the theorem. We mark chips using the following simple procedure: we mark nonpenny chips until we cannot mark a nonpenny chip without reducing $W_{\tilde{q}}(I)$ below $2^{\tilde{q}}$. We then mark enough pennies to reduce the weight to $2^{\tilde{q}}$. As the heaviest chips (those in B_0) weigh $\binom{\tilde{q}}{\le k} \le (3\tilde{q})^k$, we need to mark at most $(3\tilde{q})^k$ pennies. Taking into account both the initial marking of less than 2^k chips and this additional marking phase, we get that the number of unmarked pennies is at least $\lfloor 2^{-k}(N - 2^k + 1)\rfloor - (3\tilde{q})^k \geq 2^{-k}N - (3\tilde{q})^k - 2$.

On the other hand, in order to apply Theorem 6 we need at least $c(k)\tilde{q}^k$ unmarked pennies. This is satisfied if $2^{-k}N - (3\tilde{q})^k - 2 \geq c(k)\tilde{q}^k$. As for any fixed value of k, q and thus $\tilde{q}$ is $O(\log N)$, this condition is satisfied for every $N > N_k$ for a large enough N_k.

We can thus apply Theorem 6 with the initial configuration being the unmarked chips in the kth configuration, that we denote by I^k. The weight of this configuration is $W_{\tilde{q}}(I^k) = 2^{\tilde{q}}$. The theorem guarantees that Paul can find partitions so that after some m steps a configuration I^{k+m} is reached such that $\sum_{i=0}^{k} I_i^{k+m} = 1$ and $W_{\tilde{q}-m}(I^m) = 2^{\tilde{q}-m}$. Thus only a single chip will be left. It is easy to verify that as the weight of the chip is $2^{\tilde{q}-m}$ it must be in bin $B_{k-(\tilde{q}-m)}$. After another $\tilde{q} - m$ steps the single chip will be in the last bin and the game is over.

Finally, we sum up the number of trials, or mistakes, that Paul can force on Carol. We have k trials before getting the pennies, m trials using the Spencer's strategy, and $\tilde{q} - m$ mistakes at the end. Summing these terms and using the definition of $\tilde{q}$ we get that Paul can always force at least $q - 1$ mistakes and if $N \geq J(k,q) + 2^k$ then Paul can force at least q mistakes. ∎

3. Conversion strategies

In this section we show how the ideas behind the BW algorithm can be used to modify prediction algorithms so that they can tolerate malicious noise. Assume we are given a prediction algorithm A that makes at most k mistakes on any sequence in some set $\Sigma \subseteq (X \times \{0,1\})^*$. We assume that algorithm A makes at most k mistakes even if it is presented with a *subsequence* of any sequence in Σ. Formally, we require that Σ is subsequence closed. Any deterministic prediction algorithm can be converted to an algorithm that changes its state only when its prediction is incorrect. This is achieved by resetting the state of A after each trial in which A predicts correctly to the state of A before the trial. This conversion does not increase the worst case number of mistakes on the subsequence closed set Σ. The converted algorithm is called *conservative* (see [15]). For the rest of this section we shall always assume that the set of sequences is subsequence closed and that the prediction algorithm is conservative.

Algorithm A is allowed to perform arbitrarily badly if given an instance/outcome sequence that is not in Σ. For example, if $\Sigma = (X \times \{0\})^* \cup (X \times \{1\})^*$ (i.e. all sequences where the outcome is held constant) then the algorithm A which always predicts with the first outcome seen makes at most one mistake when given a sequence in Σ. However, if the first label is corrupted by malicious noise then all subsequent predictions made by algorithm A will be incorrect.

Here we show how to convert A into another algorithm that performs well on sequences in Σ that are corrupted by noise. In particular, for any r we can build an algorithm that performs well on those sequences which can be created from a sequence in Σ by arbitrarily changing up to r examples. We use Σ' to denote this set of noisy sequences. As the above example indicates, algorithm A may make arbitrarily many mistakes on sequences in Σ'. Furthermore, the sequences in Σ' might have different outcomes for the same instance and algorithm A might not even be defined on this larger set of sequences. In that case we extend the definition of A by assigning it the default prediction 0 and restarting it at its initial state. Thus we assume throughout that A's prediction and successor state is always defined. In this section we use the methods developed in Section 2 to construct master algorithms, called *conversion strategies*, whose mistake bounds increase slowly as a function of r.

As in Section 2, we use a version space argument and expand A into a set of variants so that at least one variant will be correct on all trials where the conversion strategy makes a mistake. However, here the elements of the version space are somewhat dynamic as they represent computations of A on sequences in Σ. In addition to discarding irrelevant computations from the version space, the conversion strategy will also need to extend certain computations by simulating A on the current trial. Since the members of the version space managed by the conversion strategy are somewhat dynamic, it may be a slight misnomer to call it a version space. However "version space" does convey the proper intuition.

Since our conversion strategies are conservative we can concentrate on those trials where the conversion strategy itself makes mistakes. Here we use m for a bound on the number of mistakes made by the conversion strategy, k to denote the mistake bound of algorithm A on sequences in Σ, and r as the number of examples corrupted by noise.

We first outline the C_{bin} conversion strategy that is based on binomial weights, and later describe a second conversion strategy, C_{exp}, based on exponential weights. These strategies are described in more detail in Sections 3.1 and 3.2 respectively.

A major difference between the conversion problem discussed here and the one addressed in Section 2 is that with experts there were only two possibilities for each trial — the expert was either correct or incorrect. Here we consider *three* different cases. The first case is when algorithm A correctly predicts the outcome. In the other two cases the prediction is incorrect. In the second case the wrong prediction is due to the fact that the example is corrupted by noise and in the third case the example is unchanged but the algorithm makes a mistake in predicting the label. Therefore, instead of associating a bit string to each member of the version space, the C_{bin} strategy attaches a string of "trits" from the set $\{0, \mathit{noise}, \mathit{mstk}\}$. Each member of the version space is a stored state of algorithm A together with a string $\boldsymbol{\tau} = (\tau_1, \ldots, \tau_m) \in \{0, \mathit{noise}, \mathit{mstk}\}^m$. These strings have an interpretation like the bit strings of Section 2. If a $(\mathit{state}, \boldsymbol{\tau})$ pair is in the version space when the conversion strategy C_{bin} makes its ith mistake, then the value of τ_i represent the following possibilities. The value 0 represents the possibility that A predicted the label of the example correctly. The values *noise* and *mstk* represent the possibility that A predicted incorrectly, where the cause for the incorrect prediction is attributed to noise or to a mistake by A respectively.

Since algorithm A makes at most k mistakes, each string τ contains *mstk* at most k times. Similarly, since we assume that at most r of the trials are corrupted by noise, *noise* appears at most r times in each string. Therefore only some of the 3^m strings in $\{0, \mathit{noise}, \mathit{mstk}\}^m$ are legitimate. In particular, if there are j nonzero elements in a string, j will be between 0 and $r+k$. Furthermore, at most r and at least $j-k$ of the elements in the string will be *noise*. This gives us

$$\textsc{size}(r,k,m) \stackrel{\text{def}}{=} \sum_{j=0}^{r+k} \binom{m}{j} \left[\binom{j}{\leq r} - \binom{j}{\leq j-k-1} \right]$$

strings that must be considered. An examination of the term in brackets shows that $\textsc{size}$ is symmetric in r and k, as expected. Furthermore, $\textsc{size}(r,k,m) = O(m^{r+k}(r+k)^{\min(r,k)})$.

The C_{bin} conversion strategy starts with a version space containing $\textsc{size}$ elements, each with the initial state of algorithm A and a different legitimate string $\boldsymbol{\tau}$. The conversion strategy manages the version space by predicting with the halving algorithm. However, it is no longer quite so clear what this means.

Consider the situation after the conversion strategy C_{bin} has made $i-1$ mistakes and sees instance $x \in X$. In this case each element of the version space, $(\mathit{state}, \boldsymbol{\tau})$ will be using its τ_i to see if its variant of A is correct, has a noisy trial, or makes a mistake. Each variant will see how A (in state *state*) predicts. If its τ_i is 0 then the variant predicts the same way, otherwise the variant predicts with the opposite value. Conversion strategy C_{bin} may update the version space after getting the outcome. If the conversion strategy C_{bin} predicted correctly then all variants are kept unchanged. If C_{bin} predicted incorrectly then those variants also predicting incorrectly are discarded. In addition, when C_{bin} predicts incorrectly those variants predicting correctly may be updated based on their τ_i values. There are three cases, according to the value of τ_i.

1. **Case $\tau_i = 0$:** This means that the variant predicted the outcome correctly. Since A is conservative, C_{bin} leaves the state of the algorithm A for this variant unchanged.

2. **Case $\tau_i = \mathit{noise}$:** This means that the prediction of A is incorrect but would have been correct if the example was not corrupted by noise. As in the previous case, C_{bin} leaves the state of the algorithm A unchanged.

3. **Case $\tau_i = \mathit{mstk}$:** This means that the prediction of A is incorrect because A has made one of its k allowed mistakes and that the example is not corrupted by noise. In this case C_{bin} updates the state of A. This is done by simulating A, starting from the old state, on the example received in the current trial. The resulting state of A replaces the old state in the variant.

We show in Lemma 5 that:

1. On each trial where C_{bin} makes a mistake, the size of the version space drops by a factor of at least 2.

2. For any sequence in Σ' at least one variant is never removed from the version space during the run of the master algorithm.

We need a few definitions before we can precisely state our bounds on the C_{bin} conversion strategy. For all $n \in \mathbb{N}$ and for all pairs $\boldsymbol{s} = (x_1, y_1), \ldots, (x_n, y_n)$ and $\boldsymbol{u} = (x'_1, y'_1), \ldots, (x'_n, y'_n)$ of sequences in $(X \times \{0,1\})^n$, we say that $\boldsymbol{s}$ is an r-*corrupted* version of $\boldsymbol{u}$ if and only if $(x_i, y_i) \neq (x'_i, y'_i)$ for at most r indices i, where $1 \leq i \leq n$. We shall also use the notation $d_C(\boldsymbol{s}, \boldsymbol{u}) = r$ to indicate that $\boldsymbol{s}$ is an r-corrupted, but not an $(r-1)$-corrupted, version of $\boldsymbol{u}$. Thus

$$d_C(\boldsymbol{s}, \boldsymbol{u}) \stackrel{\text{def}}{=} \min\{r \in \mathbb{N} : \boldsymbol{s} \text{ is an } r\text{-corrupted version of } \boldsymbol{u}\ \} \ .$$

We define $d_C(\boldsymbol{s}, \boldsymbol{u}) = \infty$ if the sequences differ in length or if they have an infinite number of disagreements, and say $\boldsymbol{s}$ is a corrupted version of $\boldsymbol{u}$ if $d_C(\boldsymbol{s}, \boldsymbol{u})$ is finite.

We will show in Section 3.1 that the conversion strategy C_{bin} achieves the following bound.

Theorem 7 *For all conservative, deterministic algorithms A, all subsequence-closed sets of sequences $\Sigma \subseteq (X \times \{0,1\})^*$, and all $\boldsymbol{s} \in (X \times \{0,1\})^+$, if*

- $k \geq \max\{L_A(\boldsymbol{u}) : \boldsymbol{u} \in \Sigma\}$ *and*
- $\boldsymbol{s}$ *is an r-corrupted version of some sequence in Σ,*

then the number of mistakes made by $C_{\text{bin}}(r, k, A)$ on the sequence $\boldsymbol{s}$ is at most

$$\max\{q \in \mathbb{N} \,:\, q \leq \log(\text{SIZE}(r, k, q))\}\ . \tag{9}$$

In Theorem 9 we will show that the bound in (9) is $O(r + k)$. Note also that the C_{bin} strategy needs to know the upper bounds k and r.

In Section 3.2 we describe a second conversion strategy that we call the C_{exp} strategy. The C_{exp} strategy uses exponential weights (as used in the Weighted Majority algorithm) and does not require advance knowledge of r and k. However one cannot optimize the mistake bounds of C_{exp} without knowing these parameters. The following theorem gives the mistake bound we prove for the conversion strategy C_{exp}.

Theorem 8 *For all conservative, deterministic algorithms A, all subsequence-closed sets of sequences $\Sigma \subseteq (X \times \{0,1\})^*$, and all $\boldsymbol{s} \in (X \times \{0,1\})^+$, if*

- *α and β are nonnegative real numbers such that $\alpha + \beta < 1$, and*
- $\boldsymbol{s}$ *is a corrupted version of some $\boldsymbol{u} \in \Sigma$,*

then the number of mistakes made by $C_{\text{exp}}(\alpha, \beta, A)$ on sequence $\boldsymbol{s}$ is at most

$$\left\lfloor \min_{\boldsymbol{u} \in \Sigma} \max_{\boldsymbol{u}' \subseteq \boldsymbol{u}} \frac{d_C(\boldsymbol{s}, \boldsymbol{u}) \log \frac{1}{\alpha} + L_A(\boldsymbol{u}') \log \frac{1}{\beta}}{\log \frac{2}{1+\alpha+\beta}} \right\rfloor , \tag{10}$$

where $\boldsymbol{u}' \subseteq \boldsymbol{u}$ means that $\boldsymbol{u}'$ is any subsequence of $\boldsymbol{u}$.

It is easy to verify numerically that by choosing $\alpha = \beta = 0.147$, the upper bound for C_{exp} displayed in (10) is at most

$$\min_{\boldsymbol{u}\in\Sigma} \max_{\boldsymbol{u}'\subseteq\boldsymbol{u}} 4.4035(d_C(\boldsymbol{s},\boldsymbol{u}) + L_A(\boldsymbol{u}')).$$

Thus we get a reasonable bound that holds for all values of $d_C(\boldsymbol{s},\boldsymbol{u})$ and $L_A(\boldsymbol{u}')$.

However, if one wants to set α and β so that the mistake bound of C_{exp} is optimized then one needs to know upper bounds k and r on $d_C(\boldsymbol{s},\boldsymbol{u})$ and $L_A(\boldsymbol{u}')$, respectively. The case when r or k is 0 is degenerate. Thus we assume that $min(r,k) \geq 1$. The following inequality was numerically checked using MAPLE$^{\mathrm{TM}}$, a software package for symbolic computation,

$$\frac{r\log\frac{1}{\alpha} + k\log\frac{1}{\beta}}{\log\frac{2}{1+\alpha+\beta}} \leq f(r,k)$$

for $\alpha = \frac{r}{f(r,k)-r-k}$ and $\beta = \frac{k}{f(r,k)-r-k}$, where

$$f(r,k) \stackrel{\mathrm{def}}{=} 2(r+k) + 2\sqrt{rk\ln(e-1+\max(r,k)/\min(r,k))} + 2.807\sqrt{rk}.$$

If $r \geq k$, then by dividing the inequality by k, we are left with an inequality in r/k, where $r/k \in [1,\infty)$. We plotted the difference between the left-hand side and right-hand side of the latter inequality as a function of r/k and checked the values of the difference and its derivatives with respect to r/k at the end points 1 and ∞.

One can also show that there is no constant c independent of r and k such that the mistake bound of C_{exp} (with α and β optimized) is at most $2(r+k) + c\sqrt{rk}$.

Notice however that C_{exp} has a worst-case mistake bound larger than C_{bin}: In much the same way we proved Theorem 2 in Section 2.1 we can also prove the following (see Section 3.2).

Theorem 9 *For all $k, r \in \mathbb{N}$ and all $\alpha, \beta \in [0,1)$, if $\alpha + \beta < 1$, then*

$$\max\{q \in \mathbb{N} : q \leq \log(\textsc{size}(r,k,q))\} \leq \left\lfloor \frac{r\log\frac{1}{\alpha} + k\log\frac{1}{\beta}}{\log\frac{2}{1+\alpha+\beta}} \right\rfloor. \tag{11}$$

To show an immediate application of Theorems 7 and 8 consider the special case when the set $\Sigma \subseteq (X \times \{0,1\})^*$ of uncorrupted sequences is the set of all sequences consistent with some family $\mathcal{F}$ of $\{0,1\}$-valued functions f on X. That is

$$\Sigma = \Sigma_{\mathcal{F}} = \left\{\langle(x_t, f(x_t))\rangle_t : f \in \mathcal{F} \wedge \langle x_t\rangle_t \in X^+\right\}.$$

This more restricted setting was studied by Littlestone [15] and Littlestone and Warmuth [17] where they define the quantities $\mathrm{Opt}(\mathcal{F},0)$, i.e. the optimal worst-case number of mistakes over all sequences from $\Sigma_{\mathcal{F}}$, and $\mathrm{Opt}(\mathcal{F},r)$, i.e. the optimal worst-case number of mistakes over all r-corrupted sequences from $\Sigma_{\mathcal{F}}$. Littlestone and Warmuth [17] show that $\mathrm{Opt}(\mathcal{F}, r) \geq 2r + \mathrm{Opt}(\mathcal{F}, 0)$, but the problem of finding an equivalent upper bound is left open. By applying Theorem 7 (or the weaker Theorem 8) when $\Sigma = \Sigma_{\mathcal{F}}$ and the sub-algorithm A is optimal, we obtain the upper bound $\mathrm{Opt}(\mathcal{F}, r) = 4.4035(r + \mathrm{Opt}(\mathcal{F}, 0))$, therefore showing $\mathrm{Opt}(\mathcal{F}, r) = \Theta(r + \mathrm{Opt}(\mathcal{F}, 0))$. Auer and Long [5] independently developed an algorithm essentially equivalent to our C_{exp} strategy.[10]

All of our conversion schemes use deterministic prediction algorithms. This means that the algorithm's prediction depends only on its current state and the observation. After making its prediction, the algorithm enters a new state based on the observation and the outcome. We denote the initial state of the prediction algorithm by S_{init} and use A_S to denote prediction algorithm A in state S. When the observation is fixed, the next state entered by algorithm A depends only on the outcome. We use $S^{x,0}$ (and $S^{x,1}$) to denote the (possibly identical) next state entered by A_S after A_S receives observation x and outcome 0 (or outcome 1 respectively). In the rest of this section we state and prove the mistake bounds for C_{bin} and C_{exp}.

3.1. The conversion strategy C_{bin}

In this section we formally describe the C_{bin} strategy and prove its mistake bound.

The C_{bin} strategy uses a concise representation of the version space in much the same way that the BW algorithm keeps a single binomial weight for each expert. In order to avoid confusion with the states of the algorithm being converted, we call the states of the C_{bin} algorithm *configurations*. Each configuration encodes the appropriate version space as well as a value (which we usually denote c') indicating an upper bound on the number of mistakes yet to be made by the conversion strategy. The C_{bin} algorithm changes configurations only when it makes a mistake.

The version space is encoded in a configuration as a (multi-)set of triples representing computations of algorithm A on corrupted versions of subsequences of the past trials. More precisely, the version space is represented by a collection of (S, r', k') triples, where S is a possible state of algorithm A and the other two components are integers. Intuitively, r' represents the maximum number of future examples that can be corrupted by noise and k' represents the maximum number of mistakes made by algorithm A in the remaining trials. Thus if c' is the upper bound on the number of mistakes yet to be made by the conversion strategy, the single triple (S, r', k') represents

$$\sum_{i=0}^{r'+k'} \binom{c'}{i} \left[\binom{i}{\leq r'} - \binom{i}{\leq i - k' - 1} \right]$$

different elements in the version space (or $(S, \boldsymbol{\tau})$ pairs for $\boldsymbol{\tau} \in \{0, noise, mstk\}^{c'}$). It is important to understand that the values r', k', and c' all start at the upper bounds r, k, and m, respectively, and count down.

The initial configuration of the C_{bin} conversion strategy contains the single triple, (S_{init}, r, k) where S_{init} is the initial state of algorithm A, r is the bound on the number of noisy trials, and k is the mistake bound of A on sequences in Σ. The initial configuration of C_{bin} also contains the mistake budget[11] $c' = m+1$, 1 greater than the mistake bound of C_{bin}.

An important concept is the *successors* of a configuration. For any possible state S of algorithm A and any $x \in X$ we use $S^{x,0}$ and $S^{x,1}$ to denote the states entered by A from state S after processing the single observation-outcome pair $(x, 0)$ or $(x, 1)$, respectively. Given a configuration $\mathcal{C}$ with mistake budget c', we define the *successors*, $\mathcal{C}^{x,0}$ and $\mathcal{C}^{x,1}$, of configuration $\mathcal{C}_t$ with respect to observation x in the following way.

> Both successor configurations have mistake budget $c' - 1$. For each triple (S, r', k') in $\mathcal{C}_t$, consider the prediction of A_S on observation x.
>
> If A_S predicts 1, then
>
> - configuration $\mathcal{C}^{x,1}$ contains the single triple (S, r', k'), and
> - configuration $\mathcal{C}^{x,0}$ contains the triples $(S^{x,0}, r', k'-1)$ and $(S, r'-1, k')$ representing the possibilities of a incorrect prediction by A and a noisy trial respectively.
>
> Similarly, if A_S predicts 0 on observation x then
>
> - configuration $\mathcal{C}^{x,0}$ contains the triple (S, r', k'), and
> - configuration $\mathcal{C}^{x,1}$ contains the triples $(S^{x,1}, r', k'-1)$ and $(S, r'-1, k')$.

We define the weight of a configuration to be the size of the version space represented by that configuration. In particular, the weight $W_{c'}(S, r', k')$ of the triple (S, r', k') in a configuration with mistake budget c' is

$$\sum_{i=0}^{r'+k'} \binom{c'}{i} \left[\binom{i}{\leq r'} - \binom{i}{\leq i - k' - 1} \right],$$

and the weight of a configuration $\mathcal{C}$, $W_{c'}(\mathcal{C})$, is the sum of the weights of the triples in $\mathcal{C}$. Triples (S, r', k') where either $r' < 0$ or $k' < 0$ represent sequences disallowed by our assumptions, and these disallowed triples are given weight 0. Deleting disallowed triples from a configuration has no effect on the strategy's predictions.

On each trial the C_{bin} conversion strategy in configuration $\mathcal{C}$ receives the new instance x and computes the weights of the two successor states, $\mathcal{C}^{x,1}$ and $\mathcal{C}^{x,0}$. The C_{bin} conversion strategy predicts 1 if the weight of $\mathcal{C}^{x,1}$ is greater than the weight of $\mathcal{C}^{x,0}$ and 0 otherwise. If the C_{bin} strategy predicted correctly, it keeps the configuration $\mathcal{C}$. If the C_{bin} strategy predicted incorrectly, then it changes its configuration from $\mathcal{C}$ to $\mathcal{C}^{x,b}$ where b is the outcome of the current trial.

A sketch of the conversion strategy C_{bin} is given in Figure 3.1. The algorithm C_{bin} can be further improved in the same way that BW$'$ improved BW (See Section

```
Strategy $C_{\text{bin}}$
Input: Two positive integers $r, k$, and a prediction algorithm $A$ with initial state $S_{init}$.

1. Let $g = m + 1$, where

       $m := \max\{q \in \mathbb{N} : q \leq \log(\text{SIZE}(r, k, q))\}$        (12)

2. Initialize configuration $\mathcal{C}_0$ to have mistake budget $c_0 = g$ and contain the single
   triple $(S_{init}, r, k)$.

3. For each trial $t = 1, 2 \ldots$
   (A) Get the $t$th observation $x_t$.
   (B) Compute the successors $\mathcal{C}_{t-1}^{x_t,0}$ and $\mathcal{C}_{t-1}^{x_t,1}$ of the current configuration $\mathcal{C}_{t-1}$.
   (C) Predict with $p \in \{0, 1\}$ such that

           $W_{c_t-1}(\mathcal{C}_{t-1}^{x_t,p}) = \max\{W_{c_t-1}(\mathcal{C}_{t-1}^{x_t,0}), W_{c_t-1}(\mathcal{C}_{t-1}^{x_t,1})\}$

       (predict arbitrarily in case of a tie.)
   (D) Get the outcome $y_t$.
   (E) If $p \neq y_t$ then decrease the mistake budget and update the current configu-
       ration by setting $\mathcal{C}_t := \mathcal{C}_{t-1}^{x_t,y_t}$; if $p = y_t$, then keep the current configuration
       by setting $\mathcal{C}_t := \mathcal{C}_{t-1}$.
```

Figure 3. Pseudo-code for the conversion strategy C_{bin}.

2). However these changes do not improve the worst-case mistake bounds, and thus we chose not to include them for the sake of the simplicity of the presentation.

The next result shows some useful properties of sequences of configurations.

LEMMA 5 *For all conservative, deterministic prediction algorithms A, all subsequence closed sets $\Sigma \subseteq (X \times \{0,1\})^*$, and all $r \in \mathbb{N}$, if*

- *$k \geq \max\{L_A(\boldsymbol{u}) : \boldsymbol{u} \in \Sigma\}$,*
- *$\boldsymbol{s} = \langle (x_t, y_t) \rangle$ in $(X \times \{0,1\})^+$ is an r-corrupted version of a sequence in Σ, and*
- *$\mathcal{C}_0$ is the configuration with mistake budget $c_0 = g$ containing the single triple (S_{init}, r, k) where S_{init} is the initial state of A, and*
- *$\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_g$ is the sequence of distinct configurations generated by a run of C_{bin} applied to A on the sequences $\boldsymbol{s}$,*

then

1. *for each* $t = 0, 1, \ldots, g-1$,
   $W_{c_t}(\mathcal{C}_t) = W_{c_t-1}(\mathcal{C}_t^{x_{t+1},0}) + W_{c_t-1}(\mathcal{C}_t^{x_{t+1},1}) \geq W_{c_t-1}(\mathcal{C}_{t+1})$
   where c_t *is the mistake budget of* $\mathcal{C}_t$, *and*

2. *for each* $t = 0, 1, \ldots, g$, $W_g(\mathcal{C}_t) \geq 1$;

The proof is shown in Appendix D.

Proof of Theorem 7: Choose $n, k \in \mathbb{N}$ and a sequence $\boldsymbol{s}^n \in (X \times \{0,1\})^n$ that is a r-corrupted version of some $\boldsymbol{u} \in \Sigma$. Let m be the integer defined by formula (9) and, assume to the contrary that $C_{\text{bin}}(r,k,A)$ makes at least $g = m+1$ mistakes on $\boldsymbol{s}$. Let ℓ be the trial on which $C_{\text{bin}}(r,k,A)$ makes its gth mistake and c' the mistake budget after the ℓth trial. We will show that

$$W_{c'}(\mathcal{C}_\ell) \leq \frac{W_g(\mathcal{C}_0)}{2^g} \tag{13}$$
$$< 1. \tag{14}$$

Let $t_1, t_2, \ldots, t_g$ be the trials at which algorithm C_{bin} makes its first g mistakes and $\boldsymbol{u}'$ be the associated subsequence of $\boldsymbol{u}$. Since Σ is closed under subsequences, $\boldsymbol{u}' \in \Sigma$. We apply Lemma 5 to sequence $\boldsymbol{u}'$ and the associated sequence $\mathcal{C}_0, \mathcal{C}_{t_1}, \ldots, \mathcal{C}_{t_g}$ of configurations generated by the algorithm. By construction, the algorithm predicts on each trial t ($1 \leq t \leq n$) according to the heaviest successor of the current configuration $\mathcal{C}_{t-1}$. The current configuration is unchanged if C_{bin} predicts correctly. If the algorithm makes a mistake on trial t, the successor $\mathcal{C}_{t-1}^{x_t,y_t}$ corresponding to the correct prediction y_t becomes the new current configuration. Because algorithm C_{bin} predicts on each trial according to the heaviest successor, it follows from part 1 of Lemma 5 that $W_{g-1}(\mathcal{C}_{t_1}) \leq W_g(\mathcal{C}_0)/2$ and that $W_{c_j-1}(\mathcal{C}_{t_{j-1}}) \leq W_{c_j}(\mathcal{C}_{t_j})/2$, for $2 \leq j \leq g$, where c_j (for $0 \leq j \leq g$) is the mistake budget of $\mathcal{C}_{t_j}$. This implies inequality (13). By definition of m in (9) and the fact that $g = m+1$ we derive inequality (14). Now part 2 of Lemma 5 shows that $W_{m_t}(\mathcal{C}_t) \geq 1$, contradicting (14). Thus C_{bin} makes at most $m = g-1$ mistakes on $\boldsymbol{s}$, concluding the proof. ■

A good outcome of the fact that C_{bin} is conservative is that the number of triplets does not increase on trials where C_{bin} predicts correctly. However, it seems that the number of triples kept by algorithm C_{bin} can potentially double each time C_{bin} makes an incorrect prediction. We now show that this apparent worst-case behavior is not possible, and that the maximum number of triples in any configuration of $C_{\text{bin}}(r,k,A)$ is bounded by $\binom{m}{\leq \min\{r,k\}} = O(m^{\min\{r,k\}})$, where m is the number of mistakes made by C_{bin} before the configuration is reached.

Theorem 10 *For all conservative, deterministic prediction algorithms* A, *and all subsequence closed sets* $\Sigma \subseteq (X \times \{0,1\})^*$, *if*

- *$k \geq \max\{L_A(\boldsymbol{u}) : \boldsymbol{u} \in \Sigma\}$,*
- *$\boldsymbol{s} = \langle (x_t, y_t) \rangle$ is an r-corrupted version of some sequence in Σ,*
- *$\mathcal{C}_0$ is the configuration with some mistake budget m containing the single triple (S_{init}, r, k) where S_{init} is the initial state of A, and*
- *$\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_m$ is the sequence of distinct configurations generated by a run of C_{bin} applied to A on the sequences $\boldsymbol{s}$,*

then for each $1 \leq t \leq m$, configuration $\mathcal{C}_t$ contains at most $\binom{t}{\leq \min\{r,k\}}$ triples with nonzero weight.

Proof: We prove the theorem when $r = \min\{r, k\}$, the other case is similar. For all $t = 0, 1 \ldots, m$ and $0 \leq i \leq r$ let $M_t(i)$ be the number of triples $(S, r', k') \in \mathcal{C}_t$ with $r' = r - i$. Thus $M_0(0) = 1$ (for the initial configuration), and $M_0(i) = 0$ for all $i > 0$. Note that some triples counted in $M_t(r - r')$ might have 0 weight if their $k' < 0$.

From the definition of successors, $M_{t+1}(i) \leq M_t(i) + M_t(i-1)$. The unique function $f = f(t, i)$ satisfying

$$\begin{aligned} f(0,0) &= 1, \\ f(0,i) &= 0, \qquad \text{for } 1 \leq i \leq r, \\ f(t+1,i) &= f(t,i) + f(t,i-1), \qquad \text{for } t > 0 \text{ and } 1 \leq i \leq r, \end{aligned}$$

is the binomial coefficient $\binom{t}{i}$. Therefore $M_t(i) \leq \binom{t}{i}$ yielding that the number of triples (S, r', k') in $\mathcal{C}_t$ with $0 \leq r' \leq r$ is at most

$$\sum_{i=0}^{r} M_t(i) \leq \sum_{i=0}^{r} \binom{t}{i} = \binom{t}{\leq r},$$

as desired. ■

3.2. The conversion strategy C_{exp}

We now move on to the description of the conversion strategy C_{exp}. Where C_{bin} was based on binomial weights, C_{exp} uses exponential weights. The advantage of using exponential weights is that the conversion strategy does not need to know the bounds r and k that C_{bin} requires as inputs. However if one wants to optimize the mistake bound of C_{exp} so that it is in the form $2(r+k)$ plus a square root term, then knowledge of k and r is required for C_{exp} as well. Analogously to C_{bin}, the bound of C_{exp} does not depend on the length of the sequence to predict. The weighting scheme used by C_{exp} has two real parameters, α and β, such that $0 \leq \alpha, \beta < 1$.

Here we define a configuration by a set of triples for different computations of algorithm A. Unlike the description of strategy C_{bin} given before, here a configuration does not have a mistake count or mistake budget. However, as before each

```
Strategy $C_{\rm exp}$[12]
Input: Two real numbers $\alpha, \beta$ such that $0 \le \alpha, \beta < 1$ and a prediction algorithm
$A$ with initial state $S_{init}$.

1. Initialize configuration $\mathcal{C}_0$ to contain the single triple $(S_{init}, r, k)$.

2. On each step $t = 1, 2, \ldots$
    (A) Get the $t$th observation $x_t$.
    (B) Compute the successor configurations $\mathcal{C}_{t-1}^{x_t,0}$ and $\mathcal{C}_{t-1}^{x_t,1}$ of the current con-
        figuration $\mathcal{C}_{t-1}$.
    (C) Predict with $p \in \{0, 1\}$ such that
            $W_{\alpha,\beta}(\mathcal{C}_{t-1}^{x_t,p}) = \max\{W_{\alpha,\beta}(\mathcal{C}_{t-1}^{x_t,0}), W_{\alpha,\beta}(\mathcal{C}_{t-1}^{x_t,1})\}$
        (predict arbitrarily in case of a tie.)
    (D) Get the outcome $y_t$.
    (E) If $p \ne y_t$, then update the current configuration by letting $\mathcal{C}_t := \mathcal{C}_{t-1}^{x_t,y_t}$; or,
        if $p = y_t$, let $\mathcal{C}_t := \mathcal{C}_{t-1}$.
```

Figure 4. Pseudo-code for the conversion strategy $C_{\rm exp}$.

triple is of the form (S, i, j) where S is a possible state of algorithm A and i, j are both nonnegative integers. For any fixed $0 \le \alpha, \beta < 1$, the weight $W_{\alpha,\beta}(S)$ of the triple (S, i, j) is the product $\alpha^i \beta^j$. As before, the weight of a configuration, $W_{\alpha,\beta}(\mathcal{C})$, is the total weight of the triples in $\mathcal{C}$. The role played here by the components i and j in each triple is analogous to the role respectively played by the components r' and k' in the triple (S, r', k') defining algorithm $C_{\rm bin}$.

We use essentially the same definition of successors as the one introduced in Section 3.1 for the strategy $C_{\rm bin}$ with only two differences. Namely, the mistake count is absent and a triple is never removed since its weight never drops to 0.

A sketch of the conversion strategy $C_{\rm exp}$, using the above weighting scheme, is given in Figure 4. The next lemma establishes some properties of such weighting schema.

LEMMA 6 *For all conservative and deterministic prediction algorithms A, and all subsequence closed sets $\Sigma \subseteq (X \times \{0, 1\})^*$, if*

- $\boldsymbol{s} = \langle (x_t, y_t) \rangle$ *is an r-corrupted version of some sequence $\boldsymbol{u} \in \Sigma$,*
- $\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_n$ *is the sequence of distinct configurations generated by a run of $C_{\rm exp}$ applied to A on the sequence $\boldsymbol{s}$, and*
- $\alpha, \beta \in [0, 1)$,

then $W_{\alpha,\beta}(\mathcal{C}_n) \geq \alpha^{d_C(\boldsymbol{u},\boldsymbol{s})}\beta^{L_A(\boldsymbol{u})}$*, and for each* $t = 1, \ldots, n$

$$W_{\alpha,\beta}(\mathcal{C}_t) \leq \left(\frac{1+\alpha+\beta}{2}\right) W_{\alpha,\beta}(\mathcal{C}_{t-1}),$$

The proof of this lemma is an easy generalization of Littlestone and Warmuth's proof of the worst-case bound for the Weighted Majority algorithm [17].

We now turn to the proof of the worst-case mistake bound for the conversion strategy $C_{\exp}$.

Proof of Theorem 8: Choose any sequence $\boldsymbol{s} = \langle(x_t, y_t)\rangle$ and choose $\boldsymbol{u} \in \Sigma$. By construction, $C_{\exp}$ predicts on each step t according to the heaviest successor of the current configuration $\mathcal{C}_t$. If a mistake occurs, then the successor $\mathcal{C}_{t-1}^{x_t,y_t}$, corresponding to the correct prediction y_t, becomes the new current configuration. Moreover, again by construction of $C_{\exp}$, the current configuration is unchanged if the algorithm predicts correctly. We can therefore apply Lemma 6 to the subsequence $\boldsymbol{s}' \subseteq \boldsymbol{s}$ determined by the sequence $t_1, t_2, \ldots, t_m$ of the indices of the prediction trials where $C_{\exp}$ makes a mistake. Since Σ is subsequence-closed, the subsequence $\boldsymbol{u}'$ of $\boldsymbol{u}$ that corresponds to these trials lies in Σ. By applying part 1 of the same lemma, and given that $\alpha + \beta < 1$, we conclude that the total weight of the current configuration decreases by a factor of at least $\frac{1+\alpha+\beta}{2}$ each time $C_{\exp}$ makes a mistake. Also, $d_C(\boldsymbol{s}', \boldsymbol{u}') \leq d_C(\boldsymbol{s}, \boldsymbol{u})$ and hence, if $\mathcal{C}_{fin}$ is the configuration following the last prediction mistake made by $C_{\exp}$ on $\boldsymbol{s}$, part 2 of Lemma 6 implies that

$$W_{\alpha,\beta}(\mathcal{C}_{fin}) \geq \alpha^{d_C(\boldsymbol{s},\boldsymbol{u})}\beta^{L_A(\boldsymbol{u}')}.$$

Hence, assuming $C_{\exp}(\alpha, \beta)$ makes m mistakes on $\boldsymbol{s}$ and recalling that $W_{\alpha,\beta}(\mathcal{C}_0) = 1$,

$$\left(\frac{1+\alpha+\beta}{2}\right)^m \geq W_{\alpha,\beta}(\mathcal{C}_t) \geq \alpha^{d_C(\boldsymbol{s},\boldsymbol{u})}\beta^{L_A(\boldsymbol{u}')}.$$

Solving for m, recalling that m is integer, yields

$$m \leq \left\lfloor \frac{d_C(\boldsymbol{s}, \boldsymbol{u}) \log \frac{1}{\alpha} + L_A(\boldsymbol{u}') \log \frac{1}{\beta}}{\log \frac{2}{1+\alpha+\beta}} \right\rfloor.$$

Since $\boldsymbol{s} \in (X \times \{0,1\})^+$ and $\boldsymbol{u} \in \Sigma$ were chosen arbitrarily, the proof is concluded.

■

We conclude this section by proving the last of the three theorems stated in Section 3. We will need a preliminary lemma.

Lemma 7 *For all* $k, r, m \in \mathbb{N}$ *and for all* $\alpha, \beta \in [0, 1)$*, if* $\alpha + \beta < 1$ *and* $m \geq r + k$ *then*

$$\sum_{i=0}^{r+k} \binom{m}{i}\binom{i}{\leq k} \leq \frac{(1+\alpha+\beta)^m}{\alpha^r \beta^k}.$$

Proof of Lemma 7: By a double application of the Binomial Theorem we show

$$(1+\alpha+\beta)^m = \sum_{i=0}^{m} \binom{m}{i} (\alpha+\beta)^i \geq \alpha^k \beta^r \sum_{i=0}^{r+k} \binom{m}{i} \sum_{j=0}^{k} \binom{i}{j}. \qquad \blacksquare$$

Proof of Theorem 9: We shall upper bound the maximal value of a larger set.

$$\max\left\{ q \in \mathbb{N} \,:\, q \leq \log \sum_{i=0}^{r+k} \binom{q}{i}\binom{i}{\leq k} \right\} \leq \left\lfloor \frac{r \log \frac{1}{\alpha} + k \log \frac{1}{\beta}}{\log \frac{2}{1+\alpha+\beta}} \right\rfloor. \tag{15}$$

It is easy to see that $r+k$ is a lower bound on the number of mistakes of any master algorithm. The left-hand side of (11) is an upper bound on the number of mistakes made by C_{bin}, therefore it is larger than $r+k$. Thus we can apply Lemma 7 to (15) obtaining

$$\begin{aligned} \max\left\{ q \in \mathbb{N} \,:\, q \leq \log \sum_{i=0}^{r+k} \binom{q}{i}\binom{i}{\leq k} \right\} &= \max\left\{ q \in \mathbb{N} \,:\, 2^q \leq \sum_{i=0}^{r+k} \binom{q}{i}\binom{i}{\leq k} \right\} \\ &\leq \max\left\{ q \in \mathbb{N} \,:\, 2^q \leq \frac{(1+\alpha+\beta)^q}{\alpha^r \beta^k} \right\} \\ &= \max\left\{ q \in \mathbb{N} \,:\, q \leq \frac{r \log \frac{1}{\alpha} + k \log \frac{1}{\beta}}{\log \frac{2}{(1+\alpha+\beta)}} \right\} \\ &= \left\lfloor \frac{r \log \frac{1}{\alpha} + k \log \frac{1}{\beta}}{\log \frac{2}{(1+\alpha+\beta)}} \right\rfloor. \qquad \blacksquare \end{aligned}$$

If we give C_{exp} an additional input parameter k such that $k \geq \max_{\boldsymbol{u} \in \Sigma} L_A(\boldsymbol{u}')$, the strategy can exploit this information in order to minimize the number of states in each configuration. In particular, C_{exp} can discard from the current configuration each triple (S, i, j), such that $j \geq k$. By using this trick, we can show, analogously to what we did for C_{bin} in Theorem 10, that the maximum number of triples in each configuration of $C_{\text{exp}}(\alpha, \beta, A, k)$ is bounded by $O(\binom{m}{k})$, where m is the number of mistakes made by C_{exp} up to the current configuration.

Furthermore, as we mentioned above, the knowledge of bounds r or k can be used to optimize the parameters α and β.

Note that both the conversion strategy C_{bin} and C_{exp} are conservative in the sense that they only update their configuration when they make a mistake. At least one copy of algorithm A receives only the subsequence of clean examples on which the conversion strategies makes a mistake. Therefore we require that the mistake bound of algorithm A holds on all subsequences of sequences in Σ. This is the reason we assumed that the set of sequences Σ in Theorems 7 and 8 is subsequence-closed. We would like conversion strategies that do not require this assumption. It seems that this is possible only for a mistake bound that increases with the length of the sequence. If we somehow could give A the "correct" feedback in trials in which the conversion strategy makes no mistake, then we could drop the assumption and update the configuration in all trials. The simple method of using the prediction of the conversion strategy as feedback does not work. This is illustrated by the following example. Assume the original algorithm A predicts 0 in the first trial and afterwards it simply predicts always with the label of the first example. Now let the sequence of examples be labeled as $\langle 0, 1, 1, 1, \cdots \rangle$. The conversion strategy will correctly predict 0 in the first trial and feeding 0 to A will "spoil" A. If we want to update in each trial, then we need to simulate noise and mistakes on all trials and this will lead to increased mistake bounds.

4. Conclusions

We have investigated the problem of on-line boolean prediction from two different viewpoints. We first improved known results about strategies that predict deterministically using the advice from a set of experts. These improvements are obtained using a weighting scheme that uses Binomial coefficients rather than exponential weights of the form β^m. These binomial coefficients can be interpreted as counting the members of an appropriate version space. In the expert setting the mistake bound based on binomial weights is never larger than the mistake bound based on exponential weights. Furthermore, the advantage of the binomial weights can be made arbitrarily large. Nevertheless both bounds can be shown to have the optimum leading term using probabilistic techniques. We also prove that, for an infinite subset of the possible problem parameters, the bound using binomial weights is best possible. The proof of this fact relies on a new translation of our prediction problem to Ulam's game with lies.

Secondly, we introduced a novel approach for making on-line algorithms robust to noise. We show how to convert an on-line prediction algorithm that is guaranteed to make at most k mistakes when given an observation-outcome sequence from its domain into an algorithm that works well when up to r of the outcomes are corrupted by noise. The converted algorithm has a conjectured mistake bound of

$$2(r+k) + 2\sqrt{rk \ln(e - 1 + \max(r,k)/\min(r,k))} + 2.807\sqrt{rk}$$

on any of the corrupted sequences (the conjecture is supported by numerical evidences.) The best lower bound we know of is $2r + k$; tightening the gap between these bounds remains an open problem.

Based on our experience, binomial weights seem to lead to better mistake bounds than exponential weights. They have the advantage of being motivated by a version space argument that leads to a deeper understanding of the on-line learning problem. The exponential weights seem to approximate the binomial weights and are sometimes easier to use, especially when the number of mistakes made by the best expert is unknown (although optimizing their mistake bounds requires knowledge of these parameters as well). Also exponential weights can be used for designing randomized prediction algorithms. In the case of exponential weights the worst-case expected number of mistakes of the randomized algorithm is exactly half of the worst-case number of mistakes of the deterministic algorithm [17], [9]. We were unable to find a randomized binomial weighting algorithm that had an expected mistake bound significantly smaller than the deterministic BW algorithm.

Acknowledgments

David P. Helmbold was supported by NSF grant CCR-9102635. Manfred Warmuth and Yoav Freund were supported by ONR grant N00014-91-j-1162. Part of this research was done while Nicolò Cesa-Bianchi was visiting UC Santa Cruz partially supported by the "Progetto finalizzato sistemi informatici e calcolo parallelo" of CNR under grant 91.00884.69.115.09672, and the Institute for Theoretical Computer Science at the Graz University of Technology (Austria).

Appendix A

A prediction algorithm that is strictly optimal for a large number of experts

As was shown in Section 2.3, the number of mistakes that the BW algorithm makes is within 1 from optimal when N, the number of experts, is large enough. In fact, we have shown that, for most values of N, BW obtains strict optimality. In this section we describe a variant of BW, which we call EBW (Enhanced Binomial Weighting), that achieves optimality in the worst case for *all* sufficiently large values of N. This modification and its analysis is a direct adaptation of a result of Spencer's ([19], Section 3).

As we have seen in the proof of Theorem 5, the only slack which allows for the gap between the upper and the lower bounds is in the way the game is played for the first k trials. In these trials there are no pennies available to Paul and thus he may not be able to split the chips into two sets of equal weight. When the weights do not split evenly, then Carol can choose a next configuration whose weight is less than half of the current one. From some starting configurations Carol can reduce the weight fast enough to "save" a mistake. However, the value of m chosen by the BW algorithm ignores this possibility of saving a mistake. Thus, since it is using the "wrong" weights, BW might play suboptimally and miss the opportunity to save a mistake. The solution is to refine the calculation of m used by the BW algorithm to account for the savings from when Paul is forced to split unevenly. We call the resulting algorithm EBW, and (for large enough N) this strategy is the best possible as there also exists a refined strategy for Paul that can force any algorithm to make the exact same number of mistakes.

The key observations are that the weight of every configuration is a multiple of the greatest common divisor (gcd) of the chip weights, and that after $t < k$ trials all of the chips are in bins 0 through t. Thus, on the first trial, Carol can ensure not only that the weight goes down by at least a factor of 2, but also that it is divisible by the gcd of the (new) weights of the first two bins. After the second trial Carol can again reduce the weight by at least half, in addition to being divisible by the (new) weights of the first three bins, and so forth.

We now describe the EBW algorithm. Recall step 1 in BW (Figure 1), in this step the bound on the number of mistakes, m, is calculated. Algorithm EBW has an additional step 1^*, between steps 1 and 2 of BW. In this step EBW checks if there will be enough unevenness in the partitions to guarantee that at most $m-1$ mistakes will be made. Specifically, it computes a new variable, m^* that is equal to either m or $m-1$. The value of m^* is an improved upper bound on the worst case number of mistakes. The rest of the algorithm stays almost the same, the only difference being that m^* is used instead of m in steps 2 and 3.

We now describe the computation of m^* in step 1^*. First, the algorithm checks if $N - 2^k \geq \lceil 2^m / \binom{m}{\leq k} \rceil$. If the inequality holds, then it is known from Theorem 5 that the bound cannot be improved and m^* is set to be m. Otherwise, a reduction of one error *might* be possible. As observed above, the total weight of any configuration is a multiple of the gcd of the weights of the chips. The algorithm computes these common divisors for each of the first k configurations $1 \leq i \leq k$:

$$A_i \stackrel{\text{def}}{=} \gcd\left(\binom{m-1-i}{k}, \binom{m-1-i}{k-1}, \ldots, \binom{m-1-i}{k-i+1}\right).$$

It then calculates the initial weight that corresponds to $m-1$

$$V_0 \stackrel{\text{def}}{=} N\binom{m-1}{\leq k}.$$

Using these values, the observations given above, and the fact that the algorithm can reduce the total weight at each step by at least a factor of two, the algorithm calculates an upper bound on $W_{m-i}(I^i)$ for $1 \leq i \leq k$:

$$V_i \stackrel{\text{def}}{=} \max\left\{ j \in \mathbb{N} \,:\, j \equiv V_0 \bmod A_i, \text{ and } j \leq \frac{V_{i-1}}{2} \right\}.$$

If $V_k \leq 2^{m-1-k}$ then the algorithm can guarantee at most $m-1$ mistakes, and m^* is set to $m-1$. If the condition does not hold, then m^* is set to m.

It remains to be shown that the number of mistakes made by EBW is at most m^* and that no other algorithm can make a smaller number of mistakes for large enough values of N. The proof of both of these claims is based on showing the the upper bounds V_i are tight, the proof is a direct translation of the proof of the theorem in section 3 of [19].

Appendix B
Proof of Lemma 1

Since $\text{up}(N, k, \beta) = (\log N + k \log \frac{1}{\beta}) / \log \frac{2}{1+\beta}$ we have

$$\frac{\partial \text{up}(N, k, \beta)}{\partial \beta} = -\frac{k}{\beta \ln \frac{2}{1+\beta}} + \frac{\text{up}(N, k, \beta)}{(1+\beta) \ln \frac{2}{1+\beta}}.$$

Note that $\ln \frac{2}{1+\beta} > 0$ since $\beta \in [0, 1)$. So the equivalence between (a.) and (b.) is easily verified by setting the above derivative to 0, multiplying by $\beta(1+\beta) \ln \frac{2}{1+\beta}$, and solving for β. The equivalence between (b.) and (c.) is obtained by substituting $\beta = \frac{k}{m-k}$ into (b.) and solving for m. To show equivalence between (c.) and (d.) we multiply (c.) by the denominator of $\text{up}(N, k, \frac{k}{m-k})$.

Using $\log \frac{2}{1+\frac{k}{m-k}} = 1 + \log(1 - \frac{k}{m})$ we get the inequality

$$m \geq \log N - k \log \frac{k}{m-k} - m \log \left(1 - \frac{k}{m}\right) \tag{B.1}$$

whose right-hand side equals $\log N + mH(\frac{k}{m})$.

Note that $2k < \log N + 2kH(\frac{1}{2})$, so $m \leq \log N + mH(\frac{k}{m})$ for m close to $2k$. Since $H(\frac{k}{m}) < 1$ for $m > 2k$, the left-hand side of (B.1) grows faster than the right-hand side (as a function of m). Thus there will be exactly one m^* where $m^* = \log N + m^* H(\frac{k}{m^*})$. From the equivalences it follows that $\partial \text{up}/\partial \beta$ evaluated at $\beta = \beta^* = \frac{k}{m^*-k}$ is 0, and this β^* is the unique minimizer of $\text{up}(N, k, \beta)$.

Appendix C
Proof of Equation (7)

Suppose for contradiction that the limit in (7) does not hold.
Since $0 \leq \text{Low}(N_i, k_i)/\text{up}(N_i, k_i, \beta_i^*) \leq 1$, there is a subsequence $\omega' = \{(N_i', k_i')\}_{i \in \mathbb{N}}$ of ω such that $\lim_{i\to\infty} \frac{\text{Low}(N_i', k_i')}{\text{up}(N_i', k_i', \beta_i^{*\prime})}$ converges to some constant less than 1.

We now consider two cases based on the limiting behavior of $k_i'/\log N_i'$ as $i \to \infty$.

The first case is when $\{k_i'/\log N_i'\}_{i\in\mathbb{N}}$ has an accumulation point at 0 or infinity. This means that there is an infinite subsequence $\omega'' = \{(N_i'', k_i'')\}_{i\in\mathbb{N}}$ of ω' such that $\lim_{i\to\infty} k_i''/\log N_i'' = 0$ or $\lim_{i\to\infty} k_i''/\log N_i'' = \infty$. In either case we use the upper bound on the function "up" proven in [9],

$$\text{up}(N, k, \beta^*) \leq \log N + 2k + 2\sqrt{k \ln N} \tag{C.1}$$

to get

$$\lim_{i\to\infty} \frac{\text{Low}(N_i'', k_i'')}{\text{up}(N_i'', k_i'', \beta_i^{*\prime\prime})} \geq \lim_{i\to\infty} \frac{\log N_i'' + 2k_i''}{\log N_i'' + 2k_i'' + 2\sqrt{k_i'' \ln N_i''}}$$

$$= \lim_{i\to\infty} \frac{1 + 2k_i''/\log N_i''}{1 + 2k_i''/\log N_i'' + 2\sqrt{k_i''/\log N_i''}} = 1.$$

Since ω'' is a subsequence of ω' this contradicts the assumption that $\frac{\mathrm{Low}(N_i',k_i')}{\mathrm{up}(N_i',k_i',\beta_i^{*\prime})}$ converges to a constant strictly less than 1.

For the other case we assume that there are positive constants a and b such that

$$a \le k_i'/\log N_i' \le b \tag{C.2}$$

for all i. Thus both N_i' and k_i' go to infinity. For the remainder of the proof we only deal with the sequence $\omega' = \{(N_i', k_i')\}_{i\in\mathbb{N}}$ and thus we can simplify our notation by dropping the primes.

Let m_i^* denote $\mathrm{up}(N_i, k_i, \beta_i^*)$. Recall from Lemma 1 that $m_i^* > 2k_i$ and that m_i^* is the largest real solution to the equation

$$x = \log N_i + xH\left(\frac{k_i}{x}\right).$$

Similarly, define $\hat{m}_i$ as the largest real solution of the equation

$$x = \log N_i + \log\binom{x}{\le k_i} - \log\left(1 + \ln\binom{x}{\le k_i}\right). \tag{C.3}$$

We will now show that $\hat{m}_i > (2+\frac{1}{2b})k_i$. Since $\lim_{i\to\infty} k_i = \infty$, for large enough i we have $(2+\frac{1}{2b})k_i < k_i/b + 2k_i - 1 - \log(1 + (2k_i - 1)\ln 2)$. Using $\log N_i \ge k_i/b$ and $\binom{2k_i}{\le k_i} = 2^{2k_i - 1}$ we obtain $(2+\frac{1}{2b})k_i < \log N_i + \log\binom{2k_i}{\le k_i} - \log\left(1 + \ln\binom{2k_i}{\le k_i}\right)$ for sufficiently large i. Next we observe that (a) the right-hand side of equation (C.3) increases with x and (b) when x is very large, x is larger than the right-hand side of equation (C.3). Therefore, if $y < z < \log N_i + \log\binom{y}{\le k_i} - \log\left(1 + \ln\binom{y}{\le k_i}\right)$, then $z < \hat{m}_i$. Applying this with $y = 2k_i$ and $z = (2+\frac{1}{2b})k_i$ proves that

$$\hat{m}_i > \left(2 + \frac{1}{2b}\right)k_i, \tag{C.4}$$

when i is sufficiently large.

Finally, define m_i as the maximum of $2k_i + \log N_i$ and $\hat{m}_i$. Note that m_i is within 1 of $\mathrm{Low}(N_i, k_i)$. As we are interested in asymptotics, we use m_i instead of $\mathrm{Low}(N_i, k_i)$. In addition,

$$\hat{m}_i \le m_i \le m_i^* \tag{C.5}$$

and, by (C.1) and (C.2)

$$\hat{m}_i \le 2k_i + \log N_i + 2\sqrt{k_i \ln N_i} \le k_i\left(2 + \frac{1}{a} + 2\sqrt{\ln 2}\sqrt{\frac{1}{a}}\right) \tag{C.6}$$

Since $k_i \to \infty$ for $i \to \infty$, it follows from (C.4) that $\hat{m}_i \to \infty$ as well. We now examine the asymptotic behavior of $\hat{m}_i$ in more detail.

$$\begin{aligned}
\hat{m}_i &= \log N_i + \log \binom{\hat{m}_i}{\le k_i} - \log\left(1 + \ln \binom{\hat{m}_i}{\le k_i}\right) \\
&= \log N_i + \log \binom{\hat{m}_i}{\le k_i} - \frac{\log\left(1+\ln\binom{\hat{m}_i}{\le k_i}\right)}{\log N_i + \log\binom{\hat{m}_i}{\le k_i}}\left[\log N_i + \log\binom{\hat{m}_i}{\le k_i}\right] \\
&= \log N_i + \log \binom{\hat{m}_i}{\le k_i} - o(1)\left[\log N_i + \log\binom{\hat{m}_i}{\le k_i}\right] \qquad \text{since } \hat{m}_i \to \infty \\
&= (1-o(1))\left[\log N_i + \log\binom{\hat{m}_i}{\le k_i}\right] && \text{(C.7)} \\
&= (1-o(1))\left[\log N_i + \hat{m}_i H\left(\frac{k_i}{\hat{m}_i}\right)\right]. && \text{(C.8)}
\end{aligned}$$

To get (C.8) we use the identity $\log\binom{m}{\le k} = mH(k/m) - \frac{1}{2}\log m + O(1)$, which holds when m goes to infinity and $m/2k$ is bounded away from both 0 and 1/2 (see [12], exercise 9.42). Since $k_i/\hat{m}_i$ is bounded away from both 0 and 1/2 for large i (see (C.4) and (C.6)), we have that $H(k/m)$ is at least some constant depending only on a and $\log\binom{\hat{m}_i}{\le k_i} = (1-o(1))\hat{m}_i H(k_i/\hat{m}_i)$.

Let $f_i(x) = \log N_i + xH(k_i/x)$. From the definition of m_i^* we know that $m_i^* = f_i(m_i^*)$. Equation (C.8) means that for any $\epsilon > 0$ there exists some i_ϵ such that for all $i > i_\epsilon$, $\hat{m}_i(1+\epsilon) \ge f_i(\hat{m}_i)$. Recall that $\hat{m}_i \le m_i \le m_i^*$. We need to show that $m_i \sim m_i^*$.

To do this we first uniformly bound the derivatives of the functions $f_i(x)$ in some ranges. Notice that $f_i'(x) = \log(x/(x-k_i))$. Thus for all $x \ge 2k_i + \log N_i$,

$$f_i'(x) \le \log \frac{2k_i + \log N_i}{k_i + \log N_i} \le \log\left(1 + \frac{1}{1 + k_i/\log N_i}\right).$$

Since $k_i/\log N_i \ge a$ we get that $f_i'(x) \le 1 - c$, for some $c > 0$ independent of i.

Using the mid-point theorem, we can lower bound $f_i(m_i)$ in the following way: $f_i(m_i) = f_i(m_i^*) - f_i'(\theta)(m_i^* - m_i)$ for some $m_i \le \theta \le m_i^*$. Using the bound on the derivative we get that

$$f_i(m_i) \ge f_i(m_i^*) - (1-c)(m_i^* - m_i) = c(m_i^* - m_i) + m_i. \qquad \text{(C.9)}$$

On the other hand, $\hat{m}_i(1+\epsilon) \ge f_i(\hat{m}_i)$, and $f_i'(x) \le 1$ for all $x \ge 2k_i$. As $m_i \ge \hat{m}_i \ge 2k_i$, (see (C.4)) we get that

$$f_i(m_i) \le (1+\epsilon)m_i. \qquad \text{(C.10)}$$

Combining (C.9) and (C.10) we get that $c(m_i^* - m_i) + m_i \le (1+\epsilon)m_i$. This implies that $m_i^*/m_i \le (c+\epsilon)/c$. As we can choose ϵ arbitrarily small, we get that $m_i \sim m_i^*$.

Appendix D

Proof of Lemma 5

To prove part 1 we show, for each triple (S, r', k'), that the sum of the weights of the successor triples equals the weight of the original. That is, if the example is x_t, y_t then

$$
\begin{aligned}
&W_{c_t-1}(S, r', k') + W_{c_t-1}(S^{x_t,y_t}, r'-1, k') + W_{c_t-1}(S, r', k'-1)\\
&= \sum_{j=0}^{r'+k'} \binom{c_t-1}{j} \left[\binom{j}{\le r'} - \binom{j}{\le j-k'-1} \right]\\
&+ \sum_{j=0}^{r'+k'-1} \binom{c_t-1}{j} \left[\binom{j}{\le r'-1} - \binom{j}{\le j-k'-1} \right]\\
&+ \sum_{j=0}^{r'+k'-1} \binom{c_t-1}{j} \left[\binom{j}{\le r'} - \binom{j}{\le j-k'} \right]\\
&= \sum_{j=0}^{r'+k'} \binom{c_t-1}{j} \left[\binom{j}{\le r'} - \binom{j}{\le j-k'-1} \right]\\
&+ \sum_{j=0}^{r'+k'-1} \binom{c_t-1}{j} + \left[\binom{j+1}{\le r'} - \binom{j+1}{\le j-k'} \right]\\
&= \sum_{j=0}^{r'+k'} \binom{c_t-1}{j} \left[\binom{j}{\le r'} - \binom{j}{\le j-k'-1} \right]\\
&+ \sum_{j=1}^{r'+k'} \binom{c_t-1}{j-1} + \left[\binom{j}{\le r'} - \binom{j}{\le j-k'-1} \right]\\
&= \sum_{j=0}^{r'+k'} \binom{c_t}{j} \left[\binom{j}{\le r'} - \binom{j}{\le j-k'-1} \right] = W_{c_t}(S, r', k').
\end{aligned}
$$

To prove part 2 choose a sequence $\boldsymbol{u}$ in Σ and let $\boldsymbol{s} = \langle (x_t, y_t) \rangle$ be a r-corrupted version of $\boldsymbol{u}$. Let $\boldsymbol{v}$ be the subsequence of $\boldsymbol{s}$ containing all the pairs (x_t, y_t) where C_{bin} makes a mistake by predicting $1 - y_t$. Let $\boldsymbol{w}$ be the subsequence of $\boldsymbol{v}$ obtained by deleting the examples corrupted by noise. Finally, for each $t \ge 1$ let $p(t) \le t$ be the number of uncorrupted examples in $\boldsymbol{v}^t$ (recall that $\boldsymbol{v}^t$ is the length t prefix of $\boldsymbol{v}$), so $t - p(t)$ is the number of corrupted examples in $\boldsymbol{v}^t$ and $\boldsymbol{w}^{p(t)}$ is the sequence obtained from $\boldsymbol{v}^t$ by deleting the corrupted examples.

Let $C(\boldsymbol{v}^t)$ be the set of (S, r', k') triples in C_{bin}'s configuration immediately after C_{bin} has seen the sequence $\boldsymbol{v}^t$. Recall that $C(\boldsymbol{v}^0) = \{(S_{init}, r, k)\}$, and a triple (S, r', k') is discarded from the configuration if either $r' < 0$ or $k' < 0$.

To prove the statement in part 2 of the lemma it suffices to prove the following claim.

Claim. For each $0 \leq t \leq |\boldsymbol{v}|$, there is a triple $(S, r', k') \in C(\boldsymbol{v}^t)$ such that:

1. S is the state of $A(\boldsymbol{w}^{p(t)})$,

2. $0 \leq k - k'$ is the number of mistakes made by A on sequence $\boldsymbol{w}^{p(t)}$, and

3. $0 \leq r - r' \leq t - p(t)$, the number of corrupted trials in $\boldsymbol{v}^t$.

Proof: First note that $\boldsymbol{w}$ is a subsequence of $\boldsymbol{u}$, so A makes at most k mistakes on $\boldsymbol{w}$. Furthermore, $\boldsymbol{v}$ is a subsequence of $\boldsymbol{s}$ and $\boldsymbol{s}$ contains at most r noisy examples, so $\boldsymbol{v}$ contains at most r noisy trials. Therefore both $k - k'$ and $r - r'$ are at least 0.

We now prove by induction on t that an appropriate triple is in the configuration $C(\boldsymbol{v}^t)$. For the base case consider $t = 0$, and recall that $p(0) = 0$. There is only one triple, (S_{init}, r, k) in $C(\boldsymbol{v}^0)$. Since $\boldsymbol{w}^0$ is the empty sequence, $A(\boldsymbol{w}^0) = S_{init}$, and A makes no mistakes on sequence $\boldsymbol{w}^0$. Thus all three conditions are satisfied by this triple.

For the inductive step assume some triple $(S, r', k') \in C(\boldsymbol{v}^t)$ satisfies the three conditions of the claim. We now show that either (S, r', k') or one of its successors in $C(\boldsymbol{v}^{t+1})$ also satisfies the claim

Case 1: the $t+1$st trial is a corrupted trial, so $\boldsymbol{w}^{p(t+1)} = \boldsymbol{w}^{p(t)}$. If A_S agrees with the corrupted outcome, then (S, r', k') is also in $C(\boldsymbol{v}^{t+1})$, and the three parts of the claim continue to hold. If A_S disagrees with the corrupted outcome then $(S, r' - 1, k')$ is in $C(\boldsymbol{v}^{t+1})$ and since $\boldsymbol{v}^{t+1}$ has one more corrupted trial than $\boldsymbol{v}^t$, the three parts of the claim also holds for $C(\boldsymbol{v}^{t+1})$.

Case 2: the $t+1$st trial is not a corrupted trial, so $\boldsymbol{v}_{t+1} = \boldsymbol{w}_{p(t)+1} = \boldsymbol{w}_{p(t+1)}$. If A_S predicts correctly on $w_{p(t)+1}$, then the triple (S, r', k') remains in the configuration. Also, since A is conservative, $S = A(\boldsymbol{w}^{p(t)+1}) = A(\boldsymbol{w}^{p(t+1)})$ and the claim holds for $C(\boldsymbol{v}^{t+1})$. If A_S predicts incorrectly then so does $A(\boldsymbol{w}^{p(t)})$. Thus A makes $k - k' + 1$ mistakes on $\boldsymbol{w}^{p(t+1)}$. Let e be the example $w_{p(t+1)}$ and thus S^e is the state $A(\boldsymbol{w}^{p(t+1)})$. In this situation, the triple $(S^e, r', k' + 1)$ is in $C(\boldsymbol{v}^{t+1})$, satisfying the claim. ■

Notes

1. A similar approach can be taken for learning the best combination of experts, although different forms of the weights are used when the loss of the master is to be close to the loss of the best convex [16] or linear [10] combination of experts.
2. The notion of “version space” for learning algorithms was originally introduced by Tom Mitchell in [18].
3. A weighting scheme based on the sum of binomial coefficients was first introduced by Berlekamp [8].

4. Expanding each expert into $\binom{m}{\leq k}$ variants instead of $\binom{m+1}{\leq k}$ variants (where m is defined as in Figure 1) does not lead to the mistake bound of m stated in Theorem 1. For example, consider the case where there is $N = 1$ expert guaranteed to make at most $k = 1$ mistake, so $m = 1$. Assume the expert is expanded into just $\binom{m}{\leq k} = 2$ variants (one predicting as the expert and one predicting the other way), and the expert is correct on the first trial. The master algorithm would see a tie vote and could predict as the variant and make a mistake. Now only the (unmodified) expert is consistent, and the master will predict as the expert does. However, this expert still has a mistake to make, and thus the master might make a total of two mistakes. Although the number of consistent variants has been reduced to one (the original expert), the surviving variant may still have mistakes to make. By considering $\binom{m+1}{\leq k}$ variants of each expert we guarantee that if only one variant is consistent, then the expert producing that variant has already made k mistakes (and thus will be correct on all future trials).
5. In the original algorithm expert E simply votes with weight β^j for its own prediction. The more complicated voting scheme given in the text is more similar to the voting scheme of the BW algorithm. Both variants of the WM algorithm generate the same predictions.
6. The algorithms predict arbitrarily if the weights are tied.
7. These values are chosen to make the algebra tractable, rather than indicating a particular region of interesting behavior.
8. An important point is that Carol does not have to "commit" to a specific number x ahead of time. The requirement is only that her choice of answers be such that at all times there exists $x \in \{1, \ldots, N\}$ that is consistent with all but at most k of her answers.
9. In this section we completely ignore the instances x_t that are given as inputs to the experts. Because we are dealing with worst case lower bounds, we can assume that for any $S \subseteq \mathcal{E}$, there is always an observation $x_S \in X$ that causes the experts in S to predict 1, and the experts not in S to predict 0. Thus the adversary can control the predictions of the experts by choosing the appropriate observation.
10. In a subsequent paper [6] a randomized variant of their conversion strategy is introduced. The worst-case expected number of mistake of their randomized strategy is significantly lower than the worst-case mistake bound of (the deterministic strategy) C_{bin}.
11. Recall from footnote D that using $c' = m$ can lead to more than m mistakes.
12. An alternative way of arriving at the same prediction is the following. Given an instance x each triple (S, r', k') votes with weight $\alpha^{r'}\beta^{k'}$ for the prediction of A_S on the instance x. The master algorithm then predicts with the vote that got the larger total weight. When this method of prediction is used the successor configuration has to be computed only when a mistake occurs.

References

1. E. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines.* John Wiley and Sons, 1989.
2. N. Alon, J.H. Spencer, and P. Erdős. *The Probabilistic Method.* John Wiley and Sons, 1992.
3. D. Angluin. Queries and concept learning. *Machine Learning*, 2:319–342, 1988.
4. J.A. Aslam and A. Dhagat. Searching in the presence of linearly bounded errors. In *Proceedings of the 23rd ACM Symposium on the Theory of Computation*, pages 486–493. ACM Press, 1991.
5. P. Auer and P.M Long. Structural results about on-line learning models with and without queries. *Machine Learning.* To appear.
6. P. Auer and P.M Long. Simulating access to hidden information while learning. In *Proceedings of the 26th ACM Symposium on the Theory of Computation*, pages 263–272. ACM Press, 1994.
7. J.M. Bardzin and R.V. Freivalds. On the prediction of general recursive functions. *Soviet Math. Dokl.*, 13:1224–1228, 1972.

8. E.R. Berlekamp. *Error-Correcting Codes*. John Wiley and Sons, 1968.
9. N. Cesa-Bianchi, Y. Freund, D.P. Helmbold, D. Haussler, R. Schapire, and M.K. Warmuth. How to use expert advice. Technical Report UCSC-CRL-95-19, University of California at Santa Cruz, 1995. An extended abstract appeared in the Proceedings of the 25th ACM Symposium on the Theory of Computation.
10. N. Cesa-Bianchi, P.M. Long, and M.K. Warmuth. Worst-case quadratic loss bounds for a generalization of the Widrow-Hoff rule. In *Proceedings of the 6th Annual ACM Workshop on Computational Learning Theory*, pages 429–438. ACM Press, 1993.
11. H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Mathematical Statistics*, 23:493–507, 1952.
12. R.L. Graham, D.E. Knuth, and O. Patashnik. *Concrete Mathematics*. Addison Wesley, 1989.
13. J. Kivinen and M.K. Warmuth. Using experts for predicting continuous outcomes. In *Proceedings of the First Euro-COLT Workshop*. The Institute of Mathematics and its Applications, 1994.
14. N. Littlestone. Learning quickly when irrelevant attributes abound: a new linear-threshold algorithm. *Machine Learning*, 2(4):285–318, 1988.
15. N. Littlestone. *Mistake Bounds and Logarithmic Linear-threshold Learning Algorithms*. PhD thesis, University of California at Santa Cruz, 1989.
16. N. Littlestone, P.M. Long, and M.K. Warmuth. On-line learning of linear functions. *Computational Complexity*, 5(1):1–23, 1995.
17. N. Littlestone and M.K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108:212–261, 1994.
18. T. M. Mitchell. Version spaces: A candidate elimination approach to rule learning. In *Proceedings International Joint Conference on Artificial Intelligence*, pages 305–310, Cambridge, Mass., 1977.
19. J. Spencer. Ulam's searching game with a fixed number of lies. *Theoretical Computer Science*, 95:307–321, 1992.
20. S. Ulam. *Adventures of a Mathematician*. Scribners, 1977.
21. V.G. Vovk. Aggregating strategies. In *Proceedings of the 3rd Annual Workshop on Computational Learning Theory*, pages 372–383, 1990.