Combining Initial Segments of Lists

Manfred K. Warmuth[⋆1], Wouter M. Koolen[⋆⋆23], and David P. Helmbold[1]

[1] Department of Computer Science, UC Santa Cruz manfred@cse.ucsc.edu
[2] Department of Computer Science, Royal Holloway, University of London
[3] Centrum Wiskunde en Informatica, Amsterdam wmkoolen@cwi.nl

Abstract. We propose a new way to build a combined list from K base lists, each containing N items. A combined list consists of top segments of various sizes from each base list so that the total size of all top segments equals N. A sequence of item requests is processed and the goal is to minimize the total number of misses. That is, we seek to build a combined list that contains all the frequently requested items. We first consider the special case of disjoint base lists. There, we design an efficient algorithm that computes the best combined list for a given sequence of requests. In addition, we develop a randomized online algorithm whose expected number of misses is close to that of the best combined list chosen in hindsight. We prove lower bounds that show that the expected number of misses of our randomized algorithm is close to the optimum. In the presence of duplicate items, we show that computing the best combined list is NP-hard. We show that our algorithms still apply to a linearized notion of loss in this case. We expect that this new way of aggregating lists will find many ranking applications.

1 Introduction

We propose a new approach for aggregating ranked lists. Assume we have K lists of N items each, as illustrated in Figure 1. Our comparator is the best combined list of size N that is composed of the tops of the K lists. A combined list might take the top 20% of list 1, the top 0% of list 2, the top 60% of list 3, and so forth. Note that the contents of the base lists might change over time and there are exponentially many (roughly N^K) combined lists altogether.

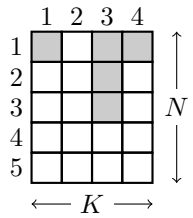

Fig. 1: We depict one combined list formed by taking the tops from $K = 4$ lists. Note that all lists have size $N = 5$ and the combined list has the same size. The list shown is $\boldsymbol{c} = (1, 0, 3, 1)$.

We seek efficient online algorithms that construct such combined lists on the fly. In each trial the following happens: First, the current contents of all base lists are provided to the algorithm. Then the algorithm assembles (either deterministically or randomly) its combined list. After that some item is requested. If it is not in the chosen combined list, then the

⋆ Supported by NSF grant IIS-0917397 and a Google gift grant.
⋆⋆ Supported by a Rubicon grant from the Netherlands Organisation for Scientific Research (NWO).

algorithm incurs a miss (one unit of loss). Some or all of the base lists might also miss the requested item and might update themselves accordingly for the next trial.

The goal of the algorithm is to endure small additional loss (regret) over the best combined list chosen in hindsight once the entire request sequence and the time-varying contents of the base lists are known. We seek efficient online algorithms that implicitly maintain some information about all roughly N^K combined lists and can efficiently choose or sample a combined list from this information. As we shall see, probabilistic algorithms will have the advantage.

We claim that this setup has many applications. For example we can use our method to combine the listings produced by different search engines. Here a first goal is to combine the tops of the base lists for the purpose of maximizing hits. However in the case of search engines, we are also concerned with accumulating hits at the top of our chosen combined list and this aspect is not yet modeled by our approach. We briefly discuss such extensions in the conclusion Section 7.

Another important application is caching. In this case each list is the ranked list of items selected by a different caching strategy. We assume that all items have the same size and exactly N fit into the fast memory. The algorithm can simulate K caching strategies as "virtual caches" and then combines these virtual caches to form one "real cache". The virtual caches only record a few bytes of meta-data about each item in their cache: ID, link to the data, and calculated priority. Object data is only kept for the N items of the real combined cache. The memory cost for maintaining the virtual caches is negligible. The real combined cache can be updated as the virtual caching strategies change their item lists and the algorithm observes the performance of its combined cache.

Previous work Methods for building a real cache by combining a number of virtually maintained caching strategies using exponential weights were explored in [GWBA02]. The employed heuristics performed well experimentally. However no performance bounds were proven. The idea for building a real cache by combining the tops of two virtual lists was first developed in [MM03, MM04]. In this case the first list contained the most recently requested items and the second contained those that were requested more than once. The goal for building a combined list was to optimally balance recency and frequency information. A deterministic heuristic was developed for adjusting the top portion taken from each list. In this paper we prove a lower bound for any deterministic algorithm that shows that any such algorithm can be forced to have loss at least $K\,B$ where B is the loss of the optimal combined list chosen in hindsight. We also give a randomized online algorithm whose expected loss is at most B plus an additional sublinear regret and show that the regret of this algorithm is optimal within a factor of $\min(\sqrt{K}, \sqrt{\ln(N/K)})$.

This paper was motivated by our previous work on combining caching heuristics [GWBA02]. In general, there are always two approaches to a rich problem setting. Either we can restrict the setting enough so that theoretical bounds are obtainable, or we can explore heuristics without provable guarantees that per-

form well in practice. Ideally the two approaches will meet eventually. In this paper we focus on algorithms for which we can prove regret bounds and we believe we made significant progress towards addressing practically interesting problems.

Aggregating experts One of the main metaphors of online learning has been the aggregation of many simple "experts" [LW94, Vov98, FS97]. In our application, each combined list serves as an expert. The online algorithm maintains its uncertainty over all experts as a mixture distribution and predicts based on this mixture. In more detail, the probability of a combined list $\boldsymbol{c}$ is proportional to $\beta^{\mathbf{M}(\boldsymbol{c})}$, where $\mathbf{M}(\boldsymbol{c})$ is the number of misses of list $\boldsymbol{c}$ and the update factor β lies in $[0, 1)$.

There are exponentially many mixture weights (one per expert). However, as we shall see later, we still manage to obtain very efficient algorithms by manipulating the weights implicitly. We first discuss how to obtain a combined list from a mixture. One could consider thresholding the mean, using the median (this is new but only works for 2 lists) or sampling.

The weighted average (mean) does not correspond to a combined list A deterministic algorithm would naturally predict with the weighted majority (mean) of the mixture. That is, an item is in the chosen combined list if the total weight of all combined lists in the mixture that contain this item is at least 50%. Unfortunately, the weighted majority does not always correspond to a list of size N: For $N = 4$ and $K = 3$, consider the case displayed in Figure 2, where we mix the 3 combined lists that contain the top halves of 2 out of the 3 list; if the mixture is uniform on these 3 combined lists, then all items in the top halves of each of the 3 lists have total weight 2/3, which is 6 elements altogether and this is more than $N = 4$.

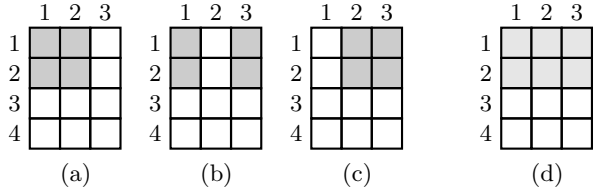

Fig. 2: Counterexample against obtaining a combined list by mean thresholding. Consider the combined lists (a), (b) and (c), which each use 4 items with weight 1. The uniform mixture over (a), (b) and (c) (displayed as (d)), sports 6 elements with weight 2/3 each. By including the items whose mean weight exceeds some threshold, we end up with either 0 or 6 elements, but not the desired 4.

Deterministic algorithm for the case of two lists based on the median In the case of $K = 2$ (i.e. two lists of size N), there are $N + 1$ combined lists $\boldsymbol{c}_0, \ldots, \boldsymbol{c}_N$, where $\boldsymbol{c}_i = (i, N-i)$ contains the i top elements from the first list and the $N - i$ top elements from the second list. These combined lists are displayed in Figure 3.

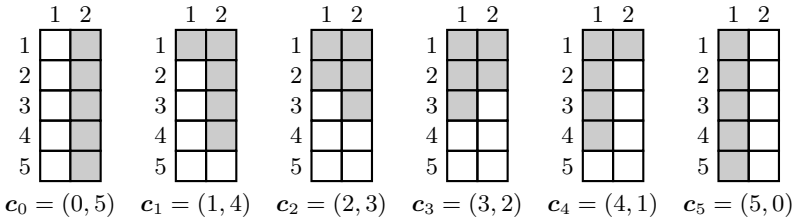

Fig. 3: The 6 combined lists of size $N = 5$ from $K = 2$ base lists

For $K = 2$ disjoint lists there is a simple algorithm producing a single combined list of the right size N based on the median that circumvents the problem with the mean discussed in the previous section. It maintains the same exponential weight discussed before but chooses a combined list $\boldsymbol{c}_i$ s.t. the total weights of the lists $\boldsymbol{c}_0, \ldots, \boldsymbol{c}_{i-1}$ and the lists $\boldsymbol{c}_{i+1}, \ldots, \boldsymbol{c}_N$ are both at most half. In other words the algorithm picks the combined list from $\langle \boldsymbol{c}_0, \boldsymbol{c}_1, \ldots, \boldsymbol{c}_N \rangle$ with the *median* weight and we call this deterministic algorithm the *Weighted Median algorithm*. Whenever a miss occurs, then at least half of the total weight is multiplied by β: If the item was on the first list and was missed by the median list $\boldsymbol{c}_i$, then at least $\boldsymbol{c}_0, \ldots, \boldsymbol{c}_i$ are multiplied by β, which contain at least half of the total weight. Similarly, if the item was on the second list and missed by $\boldsymbol{c}_i$, then at least $\boldsymbol{c}_i, \ldots, \boldsymbol{c}_N$ are multiplied by β, which is again at least half of the total. The case when the item did not appear on either base list is trivial.

Since at least half of the total weight is multiplied by β, an analysis paralleling the analysis of the deterministic Weighted Majority algorithm [LW94] gives the following loss bound after tuning β based on N and the *budget* B, i.e. the loss of the best combined list in hindsight:

$$2B + O\left(\sqrt{B \ln(N+1)} + \ln(N+1)\right).$$

There are two problems with this deterministic algorithm: It has the factor 2 in front of the budget and we don't know how to generalize it to more than 2 lists. This is not surprising because we show in this paper that any deterministic algorithm can be forced to incur loss $K\,B$, where B is the loss of the best. Nevertheless, algorithms based on this approach perform well experimentally [Sca07] and beat the previous deterministic heuristics developed in [MM03, MM04].

Probabilistic algorithms based on sampling Our approach is to apply the Hedge algorithm [FS97] to the exponentially many combined lists. It uses the same exponential weights that are proportional to $\beta^{\mathbf{M}(\boldsymbol{c})}$ for list $\boldsymbol{c}$ but now we simply pick a combined list at random according to the mixture coefficients on all the combined lists. This ensures that the randomly chosen combined list is of the right size and hence circumvents the fact that the mixture does not represent a single combined list. The expected loss of the mixture is the mixture of the losses of the individual combined lists. When β is properly tuned as a function of the

budget B and the number of combined lists, then the probabilistic algorithms achieve expected loss $B + O(\sqrt{BK \ln N})$. Note that now the factor in front of the budget B is one (whereas any deterministic algorithm can be forced have loss at least $K\,B$, see Section 5). The caveat of the probabilistic algorithms is that the list they predict with may change significantly between trials and in applications where there is a cost for changing the prediction, such algorithms are not useful. We discuss this issue again in the conclusion Section 7.

Hedge vs Follow the Perturbed Leader The two main flavors of efficient online algorithms for dealing with a linear loss are Hedge [FS97] and Follow the Perturbed Leader [KV05]. Hedge based algorithms usually have slightly better loss bounds, whereas FPL-type algorithms typically have computational advantages. In this paper, completely contrary to those general rules, we present a Hedge-based algorithm that is fundamentally *faster* for our problem than the best-known FPL algorithm. The reason for this anomaly is that the computations for Hedge can be accelerated using the Fast Fourier Transform, whereas only a slower acceleration is known for finding the best combined list.

Outline of the paper After setting the stage formally in Section 2, we begin by sketching the batch algorithm in Section 3. We then give our main randomized algorithm in Section 4 whose (expected) regret can be bounded by $O(\sqrt{BK \log N})$, where B is the loss budget of the best combined list. It simulates the Hedge algorithm on the exponentially many combined lists. We first assume that all base lists are disjoint, i.e. the requested item appears on at most one base list. This assumption lets us express the 0-1 loss/miss of a combined list as a sum over the initial segments of the base lists. The straightforward implementation requires $O(KN^2)$ time per trial. However we found a way to speed up the algorithm using Fast Fourier Transform for an improved time of $O(KN \ln N)$ per trial.

A number of lower bounds are given in Section 5. Any deterministic algorithm can be made to suffer loss at least $K\,B$, i.e. such algorithms cannot achieve sublinear regret. We also prove that any probabilistic algorithm can be forced to have regret at least $\Omega(\max(\sqrt{B \ln N}, \sqrt{B\,K}))$. Thus when either K is constant or $N = \Theta(K)$, the regret of our probabilistic algorithm is optimal within a constant factor.

In Section 6 we then address the case where duplicates are allowed between base lists. In this case we show that the question of whether there is a list with no misses is NP-hard. We then address the same problem with a surrogate loss. We "linearize" the loss much in the same way NP-completeness was avoided when learning disjunctions (by switching from 0-1 loss to attribute loss). In short, if the requested item occurs 5 times on the lists, then a combined list that hits this item 2 times now has loss 3. We can show that our algorithm now has a regret bound of $O(\sqrt{BK^2 \log N})$ for the linearized loss. Note the additional factor of K which is due to the fact the range of the loss per trial is now $[0, K]$. We also discuss an alternate method for solving the problem with duplicates based on the online shortest path problem and using Component Hedge [KWK10]. This

algorithm achieves regret $O(\sqrt{BK \log N})$ for the problem with duplicates (i.e. no additional range factor). However we do not know how to bound the running time for the iterative projection computation employed by the algorithm. We conclude with a number of open problems in the final section.

2 Setting

Recall that we have K base lists of N slots each. Until Section 6 we assume that each item appears at most once on all list. For $k \leq K$, we identify a (partial) *combined list* with a vector of counts $\boldsymbol{c} = (c_1, \ldots, c_k)$, where c_i denotes the number of items taken from the top of base list i. We denote the set of combined lists using n elements from k base lists by

$$\mathbb{C}_{n,k} \;\coloneqq\; \left\{\boldsymbol{c} \in \{0, \ldots, n\}^k \;\middle|\; \textstyle\sum_{i=1}^k c_i = n\right\}.$$

We also think about $\mathbb{C}_{n,k}$ as the set of paths from $(0,0)$ to (n,k) through the graph shown in Figure 4. Representational issues aside, the number of combined lists is rather large:

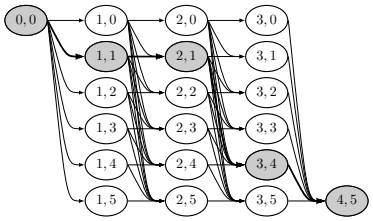

Fig. 4: Dynamic programming diagram for $K = 4, N = 5$. There is a 1–1 correspondence between $(0,0) - (4,5)$ paths in this graph and combined lists. The marked path corresponds to the combined lists shown in Figure 1. In general, combined list $\boldsymbol{c} = (c_1, \ldots, c_K)$ corresponds to path $(0,0) - (1,c_1) - (2, c_1 + c_2) - \ldots - (k, \sum_{i=1}^k c_i) - \ldots - (K, N)$.

$$\ln|\mathbb{C}_{N,K}| = \ln \binom{N+K-1}{K-1} \leq (K-1) \ln \frac{(N+K-1)\,\mathrm{e}}{K-1} = O\left(K \ln \frac{N}{K}\right) \quad (1)$$

The right equality holds under our assumption that $N \geq K$. To appreciate the sheer number of lists, consider for example that $|\mathbb{C}_{10,20}| \approx 10^7$, while $|\mathbb{C}_{100,20}| \approx 4.9 \cdot 10^{21}$.

Different trials can have different lists and requested items. What is important is the list and position containing the requested item. A single trial is summarized by a $(N+1) \times K$ dimensional *miss matrix* $\mathbf{M}$, where for $0 \leq c \leq N$ and $1 \leq k \leq K$, the entry $M_{c,k}$ is one if the requested item appears on base list k in a position strictly greater than c and zero otherwise. The sum $\mathbf{M}(\boldsymbol{c}) \coloneqq \sum_{i=1}^k M_{c_i,i}$ is zero when combined list $\boldsymbol{c}$ contains the request item, and one when $\boldsymbol{c}$ misses the requested item on the disjoint lists. We often sum miss matrices over trials: $\mathbf{M}^t$ denotes the miss matrix for trial t and $\mathbf{M}^{<t}$ is the cumulative miss matrix before trial t, i.e. $M^{<t}_{c,k} \coloneqq \sum_{s<t} M^s_{c,k}$. Therefore $\mathbf{M}^{<t}(\boldsymbol{c}) = \sum_{i=1}^k M^{<t}_{c_i,i}$ is the total number of misses made by the combined list $\boldsymbol{c}$ in the first $t-1$ trials.

We allow items to occur on multiple lists in Section 6. There, when $\mathbf{M}$ is a single trial miss matrix, $\mathbf{M}(\boldsymbol{c})$ can be greater than one. For example, if the requested item occurs on 5 different base lists and $\boldsymbol{c}$ has it twice then $\mathbf{M}(\boldsymbol{c}) = 3$.

3 Batch algorithm

Let $\mathbf{M} = \sum_{t=1}^{T} \mathbf{M}^t$ denote the cumulative miss matrix after T trials. The hindsight-optimal list $\boldsymbol{c}^*$ and its loss B are given by

$$\boldsymbol{c}^* \;\coloneqq\; \operatorname*{argmin}_{\boldsymbol{c}\in\mathbb{C}_{N,K}} \mathbf{M}(\boldsymbol{c}) \qquad\qquad B \;\coloneqq\; \mathbf{M}(\boldsymbol{c}^*) \;=\; \min_{\boldsymbol{c}\in\mathbb{C}_{N,K}} \mathbf{M}(\boldsymbol{c}).$$

Brute-force evaluation of the minimum is intractable, viz (1). But we can exploit the structure of $\mathbb{C}_{N,K}$ and $\mathbf{M}(\boldsymbol{c})$ to compute $\boldsymbol{c}^*$ and B efficiently. Let $B_{n,k}$ denote the loss of the best combined list with n elements from the first k base lists:

$$B_{n,k} \;\coloneqq\; \min_{\boldsymbol{c}\in\mathbb{C}_{n,k}} \mathbf{M}(\boldsymbol{c}).$$

Hence $B = B_{N,K}$. Now observe that $B_{\star,\star}$ satisfies the recurrence for $0 \le n \le N$:

$$B_{n,1} \;=\; M_{n,1} \quad \text{and} \quad B_{n,k} \;=\; \min_{0\le c\le n} B_{n-c,k-1} + M_{c,k}, \quad \text{for } 1 < k \le K.$$

By straightforward tabulation, the loss $B = B_{N,K}$ of the best combined list can be computed in time $O(KN^2)$. Interestingly, we can tabulate even faster. The column $B_{\star,k}$ is the infimal convolution[4] of $B_{\star,k-1}$ and $\mathbf{M}_{\star,k}$. The best-known algorithm for general infimal convolution is $O(\frac{N^2}{\ln N})$ due to [BCD+06]. In our setting, both $B_{\star,k-1}$ and $M_{\star,k}$ are non-increasing. However it is an open problem whether these special properties lead to an improved time bound. Once $B_{\star,\star}$ is known, it is easy to recover the optimal combined list $\boldsymbol{c}^\star$ in $O(NK)$ time.

The above dynamic programming algorithm immediately leads to an online algorithm, called Follow the Perturbed Leader (FPL)[KV05], which has small regret compared to the best list chosen in hindsight. At trial t, FPL adds a random perturbation matrix to $\mathbf{M}^{<t}$ and chooses the best combined list with respect to that perturbed miss matrix. FPL has slightly weaker regret bounds [HP05] than Hedge, but is usually faster. Surprisingly we show in the next section that for combined list the Hedge algorithm is actually faster: it requires $O(KN \ln N)$ time instead of $O(\frac{KN^2}{\ln N})$ for FPL.

4 The randomized online algorithm based on sampling

In this section we develop an efficient randomized online algorithm and prove that its loss is not much larger than B, the loss of the best combined list chosen in hindsight. The algorithm is the well-known Hedge algorithm [FS97] where the combined list function as the experts. The algorithm (implicitly) maintains weights proportional to $\beta^{\mathbf{M}(\boldsymbol{c})}$ for each combined list $\boldsymbol{c}$. There are two versions of the Hedge algorithm. The first one predicts with the mixture vector over the experts and incurs loss equal to dot product between the mixture vector times the loss vector. Since the mixture vector is exponential in size we have to use

[4] Aka $(\min, +)$ convolution or min-convolution or inf-convolution or epigraphical sum.

the second version. Like the Randomized Weighted Majority algorithm [LW94], this version outputs an expert drawn at random from the mixture. The loss is the loss of the drawn combined list and the goal is to bound the expected loss of the algorithm in relation to the loss of the best list chosen in hindsight.

Our contribution lies in the efficient implementation of the sampling version of the Hedge algorithm for combined lists. Following [TW03], we crucially use the fact that in our application, the loss $\mathbf{M}(\boldsymbol{c})$ of a combined list $\boldsymbol{c}$ decomposes into a sum: $\mathbf{M}(\boldsymbol{c}) = \sum_{k=1}^{K} M_{c_k,k}$. Thus the weight of $\boldsymbol{c}$ is proportional to the product $\prod_{k=1}^{K} \beta^{M_{c_k,k}}$. This property of the weights allows us to efficiently sample a combined list according to the mixture prescribed by the exponential weights of Hedge. The computation is based on dynamic programming, similar to the batch algorithm, but it is faster: $O(KN \ln N)$ instead of $O(\frac{KN^2}{\ln N})$ per trial.

Before showing this speedup, recall that the expected regret of the Hedge algorithm after tuning the learning rate η [FS97] is bounded by $B + \sqrt{2B \ln E} + \ln E$ where E is the number of experts with loss range $[0, 1]$ per trial and B is the loss budget of the best expert. In our application, $\ln E$ is $O(K \ln \frac{N}{K})$ by (1), giving us the following regret bound for our algorithm:

$$B + O\left(\sqrt{BK \ln \frac{N}{K}} + K \ln \frac{N}{K}\right). \tag{2}$$

Note that the loss of a combined list lies in $\{0, 1\}$ since we assumed that the base lists are disjoint.

We now give the efficient implementation of the sampling. Let $\mathbf{M} = \mathbf{M}^{<t}$ denote the cumulative miss matrix before trial t. In trial t, we need to efficiently sample combined list $\boldsymbol{c} \in \mathbb{C}_{N,K}$ with probability proportional to $\prod_{k=1}^{K} \beta^{M_{c_k,k}}$. We first compute the *normalization* $Z_{N,K}$, which is defined for all (partial) combined lists as follows

$$Z_{n,k} \;\coloneqq\; \sum_{\boldsymbol{c} \in \mathbb{C}_{n,k}} \beta^{\mathbf{M}(\boldsymbol{c})}.$$

To efficiently compute $Z_{\star,\star}$, we again derive a recurrence. For all $0 \le n \le N$,

$$Z_{n,1} \;=\; \beta^{M_{n,1}} \quad \text{and} \quad Z_{n,k} \;=\; \sum_{c_k=0}^{N} Z_{n-c_k,k-1} \beta^{M_{c_k,k}}, \quad \text{for } 1 < k \le K.$$

Even more compactly, we have

$$Z_{\star,k} \;=\; Z_{\star,k-1} * \beta^{\mathbf{M}_{\star,k}} \quad \text{for } 1 < k \le K$$

where $*$ denotes real discrete convolution, i.e. $(x * y)_n \;=\; \sum_i x_{n-i} y_i$. Since the convolution of two vectors of length N each can be performed in time $O(N \ln N)$ using the Fast Fourier Transform [CT65], we can tabulate $Z_{\star,\star}$ in time $O(KN \ln N)$.[5]

[5] A similar approach was used in [KM05] to obtain a similar speedup in the completely different context of computing the stochastic complexity of the Multinomial model.

Sampling a list from $\boldsymbol{c} \in \mathbb{C}_{n,k}$ with probability proportional to $\beta^{\mathbf{M}(\boldsymbol{c})}$ is now done as follows: First draw the last element $c_k = j$ with probability equal to $Z_{n-j,k-1}\beta^{M_{j,k}}/Z_{n,k}$. Then recursively sample the initial part of the list from $\mathbb{C}_{n-j,k-1}$. The probability of drawing the full $\boldsymbol{c} \in \mathbb{C}_{N,K}$ telescopes to $\beta^{\mathbf{M}(\boldsymbol{c})}/Z_{N,K}$ as desired. Once we have $Z_{\star,\star}$ tabulated, the sampling itself takes $O(KN)$ time.

An analogous approach can be used to compute the expected loss of Hedge if that is of interest, e.g. for external model selection.

We already mentioned in Section 2 that it is possible to identify combined lists with paths through the specific graph shown in Figure 4. Therefore any algorithm that has small regret compared the best path chosen in hindsight is a competitor to our algorithm. The online shortest path problem has received considerable attention both in the full-information and in the bandit setting [TW03, KV05, AHR08, CBL09, KWK10]. However, for that problem it is assumed that the adversary can control the loss of each individual edge and as a result any algorithm requires an update time that is at least on the order of the number of edges. This is not the case in our setting. We have $O(KN^2)$ edges in our graph, and yet we achieve update time $O(KN \ln N)$. How is this possible? First note that a miss matrix $\mathbf{M}$ has only KN parameters. Put another way, the losses of the edges are partitioned into equivalence classes of size $O(N)$ and all edges of the same class always have the same loss:

$$\forall 0 \le n, n' \le N-i, 1 \le k < K : \ (n,k) \to (n+i,k+1) \ \sim \ (n',k) \to (n'+i,k+1).$$

This feature of our problem allows us to use convolutions and obtain update time that is lower than the number of edges in the shortest path formulation.

5 Lower bounds

In the noise-free case (there is a combined list with no misses), the minimax regret for deterministic algorithms was proven to be $\Omega(K \ln N)$ with the first author's students in a class project [SS07]. Here we focus on the noisy case. We begin with a simple adversary argument against any deterministic algorithm and then turn to lower bounds for randomized algorithms.

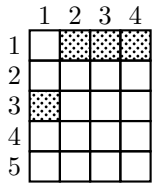

Fig. 5: For $N = 5$, $K = 4$ the special items are put in the marked positions. One must be missed by any combined list of size N.

We first show a simple lower bound of KB against any deterministic algorithm, where $B > 0$ is the number of misses of the best combined list in hindsight. We assume that B is known to both the algorithm and adversary and $N \ge K$. As illustrated by Figure 5, the adversary tags the following K items as "special": item $N - K + 2$ from the 1st list and the top items from the remaining $K - 1$ lists. Any combined list of size N must miss at least one of these K special items because to contain them all would require a list of size $N-K+2+K-1 = N+1$. In each trial the adversary simply hits one of the special items missed by the combined list produced by the deterministic algorithm. Since combined lists have size N, at least one special item will be missed in each trial.

In T trials, the algorithm incurs T misses, but the number of misses of the best combined list is at most $\frac{T}{K}$. The reason is that the best combined list leaves out the special item with the lowest number of hits and the lowest number is at most $\frac{T}{K}$. So if the best has loss B then any deterministic algorithm can be forced to have loss at least KB, and thus regret at least $B(K-1)$.

We now build techniques for proving our lower bound against any randomized algorithm in stages. The basic component of our lower bound constructions will be the standard 2-expert game. The game is parametrized by a loss budget B. There are two players, the Algorithm and the Environment, and the game proceeds in rounds. At the beginning of each round, the Algorithm chooses its prediction, which is a pair (w_1, w_2) with $w_1, w_2 \geq 0$ and $w_1 + w_2 = 1$. The Environment then chooses one of two possible outcomes: expert 1 makes a mistake, or expert 2 makes a mistake. If expert 1 makes a mistake, the Algorithm incurs loss w_1; if expert 2 makes a mistake, the Algorithm incurs loss w_2. The game ends when one expert has made at least $B+1$ mistakes. Let $g_2(B)$ be the largest total loss the Environment can force the Algorithm to incur in this game. It can be shown based on results in [AWY08] that for any integer budget B

$$g_2(B) \geq B + \sqrt{\frac{B}{\pi}}.$$

We now prove a lower bound on the regret in the $K = 2$ list case of

$$B + \sqrt{\frac{B \log_2(N+1)}{\pi}}$$

against any randomized algorithm A. We do this by constructing a trial sequence using the above 2-expert case as a building block on which A incurs at least this much loss while there is at least one combined list with loss at most B. For the sake of simplicity of the presentation we assume that $N = 2^S - 1$ for some integer S that divides the budget B.

The game consists of $S = \log_2(N+1)$ stages, each stage using up a proportion B/S of the loss budget and causing a loss of at least $g_2(B/S)$ to the algorithm. Therefore, the total loss of the algorithm will be at least the desired result

$$S\left(\frac{B}{S} + \sqrt{\frac{B}{S\pi}}\right) = B + \sqrt{\frac{BS}{\pi}}.$$

We have two lists, each with N elements. During stage 1, we access only two elements, the middle elements (those in position $(N+1)/2$ on the first and position $(N+1)/2$ on the second list). Notice that a deterministic prediction can include at most one of the middle elements. Intuitively, stage 1 will decide whether it is better to include the middle element from list 1 or from list 2. This is done using the 2-expert game in a manner described below.

Suppose we decided to include the middle element from list 1. This means we have removed from consideration the first half of elements of list 1 (they are included) and the last half of elements of list 2 (they are not included). During

stage 2 we similarly decide between the middle element of the second half of list 1, and the middle element of the first half of list 2. Suppose we now decide in favor of list 2. This means that we have decided to include at least $(N+1)/2$ elements from list 1, at least $(N+1)/4$ elements from list 2, and $(N-3)/4$ elements are yet to be decided. We continue this process, halving the remaining range at each stage, until only one good combined list remains. This argument proves the following lower bound:

Theorem 1. *Let $N+1$ be a power of 2 and B be a budget that is divisible by $\log_2(N+1)$. Then for any randomized online algorithm A for the $K=2$ list problem there is a sequence of request on which A incurs at least loss*

$$B + \sqrt{\frac{B \log_2(N+1)}{\pi}}$$

but there is at least one combined list with at most B misses.

Note that the lower bound construction uses the same base lists in each trial and the base lists are disjoint.

It is fairly easy to see that the above $\Omega(\sqrt{B \ln N})$ expected regret bound for any randomized algorithm also holds for $K > 2$ lists. When $N \geq K/2$, we can also prove a second lower bound of $\Omega(\sqrt{BK})$ by simulating $K/2$ many 2-expert games using pairs of base lists. However the details of this second bound are complex and omitted due to space constraints. We conjecture that the lower bound on the expected regret is $\Omega(\sqrt{BK \ln(N/K)})$, i.e. that the expected regret bound of the algorithm of Section 4 is optimal within a constant factor.

6 Combining Lists with Duplicates

We now turn to the scenario where the base lists are not disjoint any more, i.e. items can occur on multiple lists. This minor difference in the setup has major implications. For one, finding the list that minimizes the number of misses is NP-hard, even if the base lists contain the same items in every trial. So we cannot hope for efficient algorithms with low regret unless RP = NP. We sketch a reduction from Set Cover in Section 6.1 and then work around this hardness result in Section 6.2 by linearizing the loss.

6.1 NP-hardness by Reduction from Set Cover

An instance of the Set Cover problem consists of a collection $\mathcal{C}$ of subsets of some universe U, and a number m. The question is whether a subcollection $\mathcal{D} \subseteq \mathcal{C}$ of size m exists such that the union of the subcollection is U. We transform this instance into a zero miss combined list existence problem with $K = |\mathcal{C}|$ base lists of size $N = m(p + |U|)$ and a sequence of $|U|$ requests. For each subset $S \in \mathcal{C}$, we introduce a base list that starts with p many padding items, then includes item i_x for each $x \in S$, and ends with padding to length N. By design, N is

such that a combined list can contain all non-padding items from m base lists. To ensure that it cannot contain non-padding items from $m+1$ base lists, we must have that $(m+1)(p+1) > N$. The least such p equals $m(|U|-1)$. The request sequence hits i_x for each $x \in U$.

If an m-cover exists, there is a combined list with zero loss. If not, then any combined list must miss at least one item. So an m-cover exists iff there is a combined list without any misses. Also unless RP=P, there cannot exist a polynomial time randomized algorithm with polynomial expected loss on request sequences for which there exists a combined list with no misses. (By polynomial we mean polynomial in N and K.)

If such an algorithm existed and achieved regret $p(N,k)$ then one could present it with random requests chosen with replacement from the given request sequence. If there does not exist a combined list with zero misses, then for such random request the expected loss must be at least 1 over the length of the request sequence. By presenting the algorithm with polynomially many random requests, the expected loss is either bounded by $p(N,k)$ or the expected loss must grow linearly in the number of requests. This clearly can be used to design a polynomial-time randomized decision procedure for the combined list problem we just showed to be NP-complete.

6.2 Working around the hardness

When the lists are disjoint then in any trial the miss count $\mathbf{M}(\boldsymbol{c})$ is 0 either 1 for any combined list $\boldsymbol{c}$ and the 0/1 loss is identical to $\mathbf{M}(\boldsymbol{c})$. When items can appear on multiple list, then $\mathbf{M}(\boldsymbol{c})$ can be larger than 1, whereas the 0/1 loss is $I(d-\mathbf{M}(\boldsymbol{c}))$, where d is the number of duplicates of the requested item, and $I(h) = 1$ if $h = 0$ and 0 if $h \geq 1$. The above reductions show that we cannot hope for an efficient algorithm with small expected regret measured by the 0/1 loss when the items can appear on multiple lists. Observe that $I(d-\mathbf{M}(\boldsymbol{c})) \leq \mathbf{M}(\boldsymbol{c})$. Therefore, in this section we propose to replace the 0/1 loss $I(d-\mathbf{M}(\boldsymbol{c}))$ by its upper bound $\mathbf{M}(\boldsymbol{c})$. This amounts to linearizing the loss because $\mathbf{M}(\boldsymbol{c}) = \sum_{k=1}^{K} M_{c_k,k}$. Note that the 0/1 loss $I(d-\mathbf{M}(\boldsymbol{c}))$ decidedly does not decompose into a sum over the component of $\boldsymbol{c}$.

This approach parallels how NP-hardness was avoided when learning disjunctions. There the 0/1 loss was replaced by the attribute loss which decomposes into a sum of the literals in the disjunction [Lit88]. This linearization of the loss is extensively discussed in [TW03].

Our efficient implementation of Hedge is based on the decomposition of the miss count and remains unchanged in the setting when items can appear on multiple lists. However, $\mathbf{M}(\boldsymbol{c})$ now lies in $[0,K]$ and the straightforward regret bounds have an additional factor of K due to the extended range. This means that in the case of combined lists the resulting regret bound has leading term

$$\sqrt{2BK \ln|\mathbb{C}_{N,K}|} \;\approx\; \sqrt{2BK^2 \ln(N+1)}.$$

The additional range factor also appears in the regret of the FPL algorithm. Several method have been developed for eliminating the range factor. First, the

range factor can be eliminated if the so called *unit rule* hold for the given loss and algorithm [KWK10]. However, there are simple counter examples for combined lists. Second, we can change the algorithm to the Component Hedge algorithm which has a range factor free regret bound with a leading term of

$$\sqrt{4BK \ln \frac{(N+K)\mathrm{e}}{K}}.$$

This algorithm maintains its uncertainty as a *usage vector*, i.e. an element of the convex hull of the $(N+1) \times K$ indicator matrices corresponding to the combined lists. The weight update involves a relative entropy projection onto this convex hull, which can be expressed using few linear equality constraints as a linear image of the flow polytope. This projection can be computed using an iterative algorithm, but it's convergence properties have not yet been analyzed [KWK10].

7 Open problems

In summary, we hope that we opened up a new approach for combining lists (of the same size N). We invented a new notion of mixture of such base lists in which the sizes of the top segments that were chosen from each list sum to N. We developed an efficient algorithm that implicitly maintains a mixture over exponentially many combined lists and proved regret bounds that are tight within a factor of $\min(\sqrt{K}, \sqrt{\ln(N/K)})$. Besides proving a tight lower bound for the $K > 2$ list case (notoriously hard), our approach leaves many open questions:

- We have ignored the question of how to rank the items within the combined list. Our notion of loss simply charges one unit depending on whether the requested item is in the combined list or not. Ideally, the combined list should be ranked and we would like to develop online algorithms for the case when the cost is proportional to how close each request is to the top of the list. One idea is to overlay N algorithms optimized for combined list sizes 1,2,... N. Intuitively the item i on the list would miss the combined lists of size 1,2,..., $i-1$ and incur loss proportional to $i-1$. However this approach will only work when the the combined lists of size less than i are contained in the list of size i. So far we were not able to achieve good regret bounds with this approach.
- In caching applications it costs to change the "real cache" because items need to be reloaded. Our online algorithms currently ignore the reloading costs and this is particularly egregious for the probabilistic algorithms. The Shrinking Dartboard Algorithm, a method for lazily updating the expert followed by the Hedge algorithm was developed in [GVW10], and seems quite readily applicable to our setting. The Follow-the-Lazy-Leader algorithm [KV05] is another method requiring fewer updates. Also some good practical heuristics for reloading were given in [Gra03, GWBA02]. However no performance guarantees were provided for these heuristics.

– At this point our algorithm is designed to achieve small regret compared to the best fixed combined list chosen in hindsight. In the expert setting there is a long history of algorithms that can handle the case when the best expert is allowed to shift over time. This is achieved by first doing an exponential update and then mixing in a bit of either the uniform distribution over all experts or the average of all past distributions over the experts [HW98, BW02, GLL05]. In all experimental evaluations that the authors are aware of, it was crucial to extend the online algorithms to the shifting expert case [HLSS00, GWBA02]. It is a tedious but straightforward exercise to mix in a bit of the uniform distribution into the dynamic programming algorithm of Section 4, thus implementing the Fixed Share algorithm from [HW98]. The method is again based on recurrences, and maintains all weights implicitly. However, it adds an $O(T^2)$ factor to the running time of our current algorithm. We don't know how to do the fancier method efficiently, which mixes in a bit of the past average distribution. The reason is that the exponential weight updates on the N^K many combined lists seem to be at loggerheads with mixing in the past average weight. The resulting update is neither multiplicative nor additive and makes it difficult to implicitly maintain the weights.
  In this respect Component Hedge [KWK10] may have the advantage, as mixing in the past average usage vector can be done in constant time per component per trial. However, for this approach to be viable, an efficient implementation of the required relative entropy projections must be found.
– This paper is theoretical in that we focus on models for which we can prove regret bounds. However it would be useful to do practical experiments of the type done in [Gra03, GWBA02]. This would require us to blend the methods developed here with heuristics for handling for example the reloading issue.

Acknowledgments Thanks to Anindya Sen and Corrie Scalisi for valuable ground work on this problem as part of a class and a Master's project [SS07, Sca07]. Thanks to Jyrki Kivinen for generous brainstorming.

References

[AHR08] Jacob Abernethy, Elad Hazan, and Alexander Rakhlin. Competing in the dark: An efficient algorithm for bandit linear optimization. In *Proceedings of the 21st Annual Conference on Learning Theory*, July 2008.

[AWY08] J. Abernethy, M. K. Warmuth, and J. Yellin. Optimal strategies for random walks. In *Proceedings of the 21st Annual Conference on Learning Theory*, July 2008.

[BCD+06] David Bremner, Timothy M. Chan, Erik D. Demaine, Jeff Erickson, Ferran Hurtado, John Iacono, Stefan Langerman, and Perouz Taslakian. Necklaces, convolutions, and $x + y$. In *Algorithms ESA 2006*, volume 4168 of *Lecture Notes in Computer Science*, pages 160–171. Springer, 2006.

[BW02] O. Bousquet and M. K. Warmuth. Tracking a small set of experts by mixing past posteriors. *Journal of Machine Learning Research*, 3:363–396, 2002.

[CBL09] Nicolò Cesa-Bianchi and Gábor Lugosi. Combinatorial bandits. In *Proceedings of the 22nd Annual Conference on Learning Theory*, June 2009.

[CT65] James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, 19(90):297–301, 1965.

[FS97] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55:119–139, 1997.

[GLL05] Andrs Gyrgy, Tams Linder, and Gbor Lugosi. Tracking the best of many experts. In Peter Auer and Ron Meir, editors, *Proceedings of the 18th Annual Conference on Learning Theory*, pages 51–74. Springer, 2005.

[Gra03] R. B. Gramacy. *Adaptive caching by experts.* PhD thesis, University of California at Santa Cruz, 2003.

[GVW10] Sascha Geulen, Berthold Vöcking, and Melanie Winkler. Regret minimization for online buffering problems using the Weighted Majority algorithm. In *Proceedings of the 23rd Annual Conference on Learning Theory*, 2010.

[GWBA02] Robert B. Gramacy, Manfred K. Warmuth, Scott A. Brandt, and Ismail Ari. Adaptive caching by refetching. In Suzanna Becker, Sebastian Thrun, and Klaus Obermayer, editors, *NIPS*, pages 1465–1472. MIT Press, 2002.

[HLSS00] David P. Helmbold, Darrell D. E. Long, Tracey L. Sconyers, and Bruce Sherrod. Adaptive disk spin-down for mobile computers. *ACM/Baltzer Mobile Networks and Applications (MONET)*, pages 285–297, 2000.

[HP05] Marcus Hutter and Jan Poland. Adaptive online prediction by following the perturbed leader. *Journal of Machine Learning Research*, 6:639–660, April 2005.

[HW98] Mark Herbster and Manfred K. Warmuth. Tracking the best expert. *Machine Learning*, 32:151–178, 1998.

[KM05] Petri Kontkanen and Petri Myllymäki. A fast normalized maximum likelihood algorithm for multinomial data. In *IJCAI*, pages 1613–1615, 2005.

[KV05] A. Kalai and S. Vempala. Efficient algorithms for online decision problems. *J. Comput. Syst. Sci.*, 71(3):291–307, 2005.

[KWK10] Wouter M. Koolen, Manfred K. Warmuth, and Jyrki Kivinen. Hedging structured concepts. In *Proceedings of the 23rd Annual Conference on Learning Theory*, pages 93–105, June 2010.

[Lit88] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2(4):285–318, 1988.

[LW94] N. Littlestone and M. K. Warmuth. The Weighted Majority algorithm. *Information and Computation*, 108(2):212–261, 1994.

[MM03] Nimrod Megiddo and Dharmendra S. Modha. One up on LRU. *;login: The Magazine of USENIX and SAGE*, 28(4):6–11, August 2003.

[MM04] Nimrod Megiddo and Dharmendra S. Modha. Outperforming LRU with an adaptive replacement cache algorithm. *IEEE Computer*, 37(4):58–65, April 2004.

[Sca07] Corrie Ann Scalisi. An adaptive caching algorithm. Master's thesis, University of California Santa Cruz, June 2007.

[SS07] Anindya Sen and Corrie Scalisi. Making online predictions from k-lists. Project report for CMPS 290C, Advanced Topics in Machine Learning, June 2007.

[TW03] Eiji Takimoto and Manfred K. Warmuth. Path kernels and multiplicative updates. *Journal of Machine Learning Research*, 4:773–818, 2003.

[Vov98] V. Vovk. A game of prediction with expert advice. *J. of Comput. Syst. Sci.*, 56(2):153–173, 1998.