Budgeted Learning, Part I: The Multi-Armed Bandit Case

Omid Madani **Daniel J. Lizotte** **Russell Greiner**
Dept. of Computing Science
University of Alberta
Edmonton, T6J 2E8
{madani|dlizotte|greiner}@cs.ualberta.ca

Abstract

We introduce and motivate the task of learning under a budget. We focus on a basic problem in this space: selecting the optimal bandit after a period of experimentation in a multi-armed bandit setting, where each experiment is costly, our total costs cannot exceed a fixed pre-specified budget, and there is no reward collection during the learning period. We address the computational complexity of the problem, propose a number of algorithms, and report on the performance of the algorithms, including their (worst-case) approximation properties, as well as their empirical performance on various different problem instances. Our results show that several obvious algorithms, such as round robin and random, can perform poorly; we also propose new types of algorithms that often work significantly better.

1 Introduction

Learning tasks typically begin with a data sample — *e.g.*, symptoms and test results for a set of patients, together with their clinical outcomes. By contrast, many real-world studies begin with *no* actual data, but instead with a budget — funds that can be used to collect the relevant information. For example, one study has allocated \$2 million to develop a system to diagnose cancer, based on a battery of patient tests, each with its own (known) costs and (unknown) discriminative powers. Given our goal of identifying the most accurate classifier, what is the best way to spend the \$2 million? Should we indiscriminately run every test on every patient, until exhausting the budget? ... or selectively, and dynamically, determining which tests to run on which patients? We call this problem *budgeted learning*.

An initial step in any learning task is identifying the most discriminative features. Our present work studies the budgeted learning problem in the following "coins problems", closely related to feature selection: We are given n (distinguishable) coins with unknown head probabilities. We are allowed to sequentially specify a coin to toss, then observe the outcome of this toss, but only for a known, fixed number of tosses. After this trial period, we have to declare a winner coin. Our goal is to pick the coin with the highest head probability from among the coins. However, considering the limits on our trial period, we seek a strategy for coin tossing that, on average, leads to picking a coin that is as close to the best coin as possible.

There is a tight relation between identifying the best coin and the most discriminative feature: the head probability of a coin is a measure of quality, and corresponds to the discrimination power in the feature selection problem. Our companion paper [LMG03] develops the theory and problem definitions for the more general learning problems; here we remark that the hardness results and the algorithmic issues that we identify in this work also apply to these more general budgeted learning problems, while the latter introduce extra challenges as well.

The first challenge in defining the budgeted problem is to formulate the objective to obtain a well-defined and satisfactory notion of *optimality* for the complete range of budgets. We do this by assigning priors over coin quality, and by defining a measure of regret for choosing a coin as a winner. We describe *strategies* (for determining which coin to toss in each situation), and extend the definition of regret to strategies. The computational task is then reduced to identifying a strategy with minimum regret, among all strategies that respect the budget.

We address the computational complexity of the problem, showing that it is in PSPACE, but also NP-hard under different coin costs. We establish a few properties of optimal strategies, and also explore where some of the difficulties may lie in computing optimal strategies, *e.g.*, the need for contingency in the strategy, even when all coins have the same cost (the unit-cost case). We investigate the performance of a number of algorithms empirically and theoretically, by defining and motivating constant-ratio approximability. The algorithms include the obvious ones, such as round-robin and random, as well as novel ones that we propose based on our knowledge of problem structure. One such algorithm, "biased-robin", works well, especially for the special case of identical priors and unit costs. The paper also raises a number of intriguing open problems.

The main contributions of this paper are:

- Introducing the budgeted learning task and precisely defining a basic problem instance in this space (the "coins problem") as a problem of sequential decision making under uncertainty.
- Addressing the computational complexity of the problem, highlighting important issues both for optimality and approximability. Empirically comparing a number of obvious, and not so obvious, algorithms, towards determining which work most effectively.
- Providing in closed-form the expected regret of using one of the most obvious algorithms: round-robin.

The paper is organized as follows. A discussion of related work and notational overview ends this section. Section 2 describes and precisely defines the coins problem, and Section 3 presents its computational complexity. Section 4 begins by showing that the objective function has an equivalent and simplified form. This simplification allows us to explore aspects of the problem that are significant in designing optimal and approximation algorithms. This section also defines the constant-ratio approximation property, and describes the algorithms we study, and addresses whether they are approximation algorithms. Section 5 empirically investigate the performance of the algorithms over a range of inputs. For a complete listing of the data, together with additional empirical and analytic results, please see [Gre].

1.1 Related work

There is a vast literature on sequential decision making, sample complexity of learning, active learning, and experiment design, all somewhat related to our work; of which we can only cite a few here. Our "coins problem" is an instance of the general class of multi-armed bandit problems [BF85], which typically involve a trade-off between exploration (learning) and exploitation. In our budgeted problem, there is a pure learning phase (determined by a fixed budget) followed by a pure exploitation phase; This difference in the objective changes the nature of the task significantly, and sets it apart from typical finite or infinite-horizon bandit problems and their analyses (e.g., [KL00]). To the best of our knowledge (and to our surprise) our budgeted problem has not been studied in the bandit literature before[1]. Our coins problem is also a special Markov decision problem (MDP) [Put94], but the state space in the direct formulation is too large to allow us to use standard MDP solution techniques. While the research on techniques for solving large MDPs with various forms of structures show promise (e.g., [Duf02]), we believe that our problem has special structure that allows for simpler, more efficient, and more effective algorithms for our special case.

Many on-line learners try to minimize the number of training examples; cf., [MCR93, SG95, EDMM02]. These approaches, however, allow the learner to acquire as many examples as are required to match some requirements — e.g. for some statistical test, or some specified ϵ and δ values in the case of PAC-learners [Val84]. We, however, have a firm total budget, specified before the learning begins.

Budgeted learning falls under the framework of bounded rationality (*e.g.*, [RS95]), and is an instance of active learning and cost-sensitive learning (*e.g.*, [Ang92, CAL94, Tur00, GGR02]). Feature costs in [Tur00, GGR02] refer to costs occuring at *classification* time, while we are concerned with cost during the learning phase. In typical pool-based active learning, the learner knows the feature values but not the label of the instances in the pool. Our problem is closer to unknown feature values. Similar to active learning results [LMRar, TK00, RM01], we show that *selective querying* can be much more efficient than standard method such as round-robin. These previous results suggest that greedy methods are effective; deeper look-aheads are not used due to a combination of inefficiency and no significant gains [LMRar]. However, we observe here that the greedy method has poor performance both in theory and in experiments, while looking deeper pays significant dividends.

This paper considers a relatively simple framework to allow us to obtain crisp theoretical results and many useful technical insights. These results set the stage for our companion paper [LMG03], which extends many of the ideas, algorithms and analyses to handle learning a Naive Bayes model under a budget. In particular, it shows that the basic algorithmic ideas presented here extend to yield effective selective querying algorithms in that context as well, and that similar observations, such as poor performance of greedy algorithms, also hold there.

1.2 Notation

Random variables are denoted by capital letters, such as Θ, and $E(\Theta)$ denotes the expectation of Θ. The probability of an event f is denoted by $p(f)$. We use the conditioning bar for conditional probabilities, probability density functions and expectations, *e.g.*, $E(\Theta_i|h_i)$. Probability distributions may replace probability density functions and summations may replace integrations, or vice versa, whenever appropriate in the exposition.

[1] Personal communication with D. Berry of [BF85].

2 The Coins Problem

Perhaps the simplest budgeted learning problem is the following "coins" problem: You are given a collection of n coins with different and unknown head probabilities. Assume the tail and the head outcomes correspond to receiving no reward and a fixed reward (1 unit) respectively. You are given a trial/learning period for experimenting with the coins only, *i.e.*, you cannot collect rewards in this period. At the end of the period, you are allowed to pick only a single coin for all your future tosses (reward collection). The trial period is defined as follows: Each toss incurs a cost (*e.g.*, monetary or time cost), and your total experimental costs may not exceed a budget m. Therefore the problem is to find the best coin possible subject to the budget b. Let us now define the problem precisely. The input is:

- A collection of n coins, indexed by the set $\mathcal{I}$, where each coin is specified by a query (toss) cost and a probability density function over its head probability. We assume that the priors of the different coins are independent. (Note that these distributions can be different for different coins.)
- A budget b on the total cost of querying.

We let the random variable Θ_i. denote the head probability of a coin c_i, and let $w_i(\Theta_i)$ be the density over Θ_i. Both discrete and continuous densities are useful in our exposition and for our results; Figure 1 illustrates examples with both types of priors. The family of *Beta densities* are particularly convenient for representing and updating priors [Dev95]. A Beta density is a two-parameter function, *Beta*$(\,\alpha_1,\,\alpha_2\,)$, whose pdf has the form $C\Theta^{\alpha_1-1}(1-\Theta)^{\alpha_2-1}$, where α_1 and α_1 are positive integers in this paper, and C is a normalizing constant, so that the density integrates to 1. *Beta*$(\,\alpha_1,\,\alpha_2\,)$ has expectation $\alpha_1/(\alpha_1+\alpha_2)$. Note that the expectation is also the probability that the outcome of tossing such a coin is heads. If the outcome is heads (resp. tails), the density is updated (using Bayes rule) to *Beta*$(\,\alpha_1+1,\,\alpha_2\,)$ (resp. *Beta*$(\,\alpha_1,\,\alpha_2+1\,)$). Thus a coin with the uniform prior *Beta*$(\,1,\,1\,)$, that was tossed 5 times, and gave 4 heads and 1 tail, now has the density *Beta*$(\,5,\,2\,)$, and the probability that its next toss gives heads is $5/7$.

Let us first address the question of which coin to pick at the end of the learning period, *i.e.*, when the budget allows no more tosses. The expected head probability of coin i, aka the *mean* of coin i, is: $E(\Theta_i) = \int_0^1 \Theta_i w_i(\Theta_i) d\Theta_i$. The coin to pick is the one with the highest mean $\mu_{max} = \max_{i\in\mathcal{I}} E(\Theta_i)$, which we denote by i^*. Note that coin i^* may be different from the coin whose density has the highest mode or the coin whose density has the highest probability of having head probability that is no less than any other(*e.g.*, see Figure 1a). The motivation for picking coin i^* is that tossing such a coin gives an expected reward no less than the expected reward obtained from tossing any other coin.

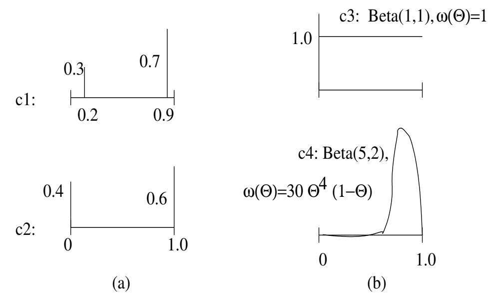


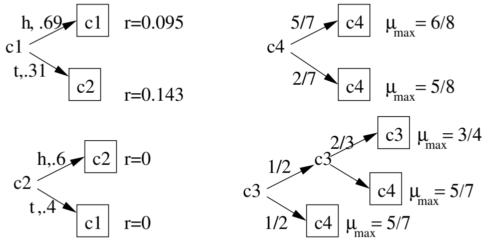
Figure 1: (a) Coins with discrete distributions. Coin c_1, with 0.3 probability has head probability of 0.2, and otherwise (with probability 0.7) has head probability 0.9. The mean (expected head probability) of coin c_1 is $0.69 = 0.3\times 0.2 + 0.7\times 0.9$, while for coin c_2 it is 0.6, thus coin c_1 is the coin to pick with 0 budget. $E(\Theta_{max}) = 0.6\times 1 + 0.4(0.7\times .9 + 0.3\times 0.2) = 0.866$. Therefore the regret from picking coin c_1 is $0.876 - 0.69 = 0.186$. (b) Coins with Beta priors. Coin c_4 has the higher mean at $5/7$. Here $E(\Theta_{max}) \approx 0.77$ (computed by numerical integration).

We will now define a measure of error which we aim to minimize. Let i_{max} be the actual coin with the highest head probability, and let $\Theta_{max} = \max_{i\in\mathcal{I}} \Theta_i$, the *max random variable*, be the random variable corresponding to the head probability of i_{max}. (Note $E(\Theta_{max}) = \int_{\vec{\Theta}} (\max_{i\in\mathcal{I}} \Theta_i) \prod_i w_i(\Theta_i) d\vec{\Theta}$.) The (expected) *regret* from picking coin i, $E(r(i))$, is then $E(r(i)) = E(\Theta_{max} - \Theta_i) = E(\Theta_{max}) - E(\Theta_i)$, *i.e.*, the average amount by which we would have done better had we chosen coin i_{max} instead of coin i. Observe that to minimize regret,[2] we also should pick coin i^*. Thus the (expected) minimum regret is $E(\Theta_{max}) - \mu_{max}$. Note that the (minimum) regret is the difference between two quantities that differ in the order of taking maximum and expectation: $E(\Theta_{max}) = E(\max_{i\in\mathcal{I}} \Theta_i)$, and $\mu_{max} = \max_{i\in\mathcal{I}} E(\Theta_i)$.

2.1 Strategies

Assume now that we are allowed to experiment with the coins before having to choose. Informally, a strategy is a prescription of which coin to query at a given time point. In general, such a prescription depends on the objective (minimizing regret in our case), the outcomes of the previous tosses or equivalently the current densities over head probabilities (*i.e.*, the current *belief state*), and the remaining budget. Note that after each toss of a coin i, the density over its head probability is updated using Bayes formula based on the toss outcome. In the most general form, a strategy may be viewed as a finite rooted-directed tree, (or actually, DAG) where each leaf node is a special leaf (stop) node, and each internal node corresponds to tossing a par-

[2]We could define alternative measures of error: For example the 0/1 error may be defined as the probability that the chosen coin is not the best coin i_{max}. However, a strategy minimizing such error in general leads to choosing a coin with mean lower than the best possible.



(a) (b)

Figure 2: Example strategy trees for the coins of Figure 1a and 1b respectively. Upper edges correspond to the head outcome of the coin tossed, and the leaf nodes box the coin to pick. (a) The regret from tossing coin c_2 (resp. coin c_1) is 0 under either node (resp. ~0.95 and ~0.143), and therefore the expected regret is 0 (resp. $0.69 \times 0.095 + 0.31 \times .143 \approx 0.11$). Tossing coin c_2 is optimal. (b) The highest mean μ_{max} is shown by each leaf (by Proposition 4.1, minimizing regret is equivalent to maximizing expected highest mean). In this case, tossing either coin once does not change the (expected) regret. Hence the optimal strategy when two tosses are allowed is to toss coin c_3 twice.

ticular coin, whose two children are also strategy trees, one for each outcome of the toss (see Figure 2). We will only consider strategies respecting the budget, *i.e.*, the total cost of coin tosses along any branch may not exceed the budget. Thus the set S of strategies to consider is finite though huge (*e.g.*, n^{2^b-1} assuming unit costs). Associated with each leaf node j of a strategy is the regret r_j, computed using the belief state at that node, and the probability of reaching that leaf p_j, where p_j is the product of the transition probabilities along the path from root to that leaf. We therefore define the *regret* of a strategy to be the expected regret over the different outcomes, or equivalently the expectation of regret conditioned on execution of s, or $E(r|s)$:

$$Regret(s) \quad = \quad E(r|s) \quad = \sum_{j \in \text{Tree Leafs}} p_j r_j$$

An *optimal strategy* s^* is then one with minimum regret: $s^* = \arg\min_{s \in S} Regret(s)$. Figure 3 shows the optimal strategy for the case of $n \geq 4$ coins with uniform priors, and a budget of $b = 3$. We have observed that optimal strategies for identical priors enjoy a similar pattern (with some exceptions): their top branch (*i.e.*, as long as the outcomes are all heads) consists of tossing the same coin, and the bottom branch (*i.e.*, as long as the outcomes are all tails) consists of tossing the coins in a round-robin fashion; see biased-robin (Section 4.2.3 below).

2.2 The Computational Problem

Our overall goal is to execute tosses according to some *optimal strategy* s^*. Three relevant computational problems are outputting (1) an optimal strategy, (2) the best coin to toss now (the first action of the optimal strategy), or (3) the minimum regret. As the optimal strategy may be exponential in the input size, whenever we talk about the coins

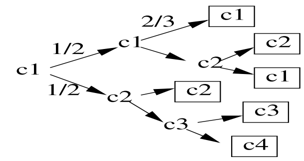


Figure 3: The optimal strategy tree for budget of 3 where all the coins have uniform priors, and there are at least 4 coins. Note that some branches terminal after only 2 tosses; here the outcome is already determined.

problem in a formal sense (*e.g.*, Theorem 3.1), we shall mean the problem of computing the first action of an optimal strategy.

3 Computational Complexity

The coins problem is a problem of sequential decision making under uncertainty, and similar to many other such problems [Pap85], one can verify that it is in PSPACE, as long as we make the assumption that the budget is bounded by a polynomial function of the number of coins[3]. The problem is also NP-hard:

Theorem 3.1 *The coins problem is in PSPACE and NP-hard.*

Proof (sketch). In PSPACE: It takes space polynomial in the budget to compute the value of an optimal policy or the first action of such a policy: fix an action and compute the regret given the outcome is positive (recursively), and reuse the space to do the same for the negative outcome. Fix another action and reuse the space. The space used is no more than a constant multiple of the budget.

NP-Hardness: Assume the priors have non-zero probability at head probability values of 0 or 1 only: $p(0 < \Theta_i < 1) = 0$. In this case, strategies are different only with respect to the collection of coins they query, and not the order: as soon as a coin returns heads, the regret is 0, and otherwise the coin is discarded. We reduce the KNAPSACK problem to this coins problem. In the KNAPSACK problem [GJ79], given a knapsack with capacity b, and a collection of objects o_i with cost c_i and reward r_i, the problem is to choose a subsets S of the objects that fit in the knapsack and maximize the total reward among all subsets: $\max_S \left\{ \sum_{o_i \in S} r_i \text{ subject to } \sum_{i \in S} c_i \leq b \right\}$

Object o_i will be coin i with cost c_i and probability $p_i = p(\Theta_i = 1) = 1 - exp(-r_i)$, so that $-\log(1 - p(\Theta_i = 1)) = r_i$, and $-\log \prod_i (1 - p_i) = \sum_i r_i$, *i.e.*, maximizing the total reward would be equivalent to minimizing the failure probability. Therefore success probability of such a coin collection is maximized iff the total reward of the corresponding object set is maximized. Maximizing the success probability is in turn equivalent to minimizing the (expected) regret. $\square$

Our reduction reveals the packing aspect of the budgeted problem. It remains open whether the problem is NP-hard when coins have unit costs and/or uni-modal distributions. The next section discusses the related issue of approxima-

[3]We assume that the various base-level expectations used in the computations, such as $E(\Theta_i)$, can be computed or approximated in PSPACE as well.

bility.

4 Problem Structure and Algorithm Design

It is fairly intuitive that the expectation over the mean of a coin given the coin is tossed should not be different from the current mean of the coin, as we are taking a sum over the possible outcomes. Here, we observe that the expectation $E(\Theta_{max})$ enjoys the same property: $E(\Theta_{max}|s) = E(\Theta_{max})$. This allows us to observe the following simplifying and perhaps intuitive properties:

Proposition 4.1 *We have,*

1. $E(\Theta_{\max}|s) = E(\Theta_{\max})$*, therefore, the regret of a strategy* s *is:*

$$regret(s) = E(\Theta_{\max}) - E(\mu_{max}|s) \quad (1)$$

where $E(\mu_{max}|s)$ *denotes the conditional expectation of maximum mean* μ_{max}*, conditioned on the execution of strategy* s *(i.e., the expectation of the highest mean over all the outcomes of executing strategy* s*).*

2. *Strategies are harmless: For any strategy* s*,* $E(\mu_{max}|s) \geq E(\mu_{max})$*, therefore regret from using any strategy* s *is not greater than the current regret.*

3. *(no need to query useless coins) Assume that under any outcome (i.e., the execution of any strategy respecting the budget), there is some coin whose mean is at least as high as coin* i*. Then there exists an optimal strategy tree that never queries coin* i*.*

Proof (sketch). Part 1: We consider tossing a single coin j (event j), and we can establish the identity for $E(\Theta_{\max}|j)$ by summing over the different outcomes. The result generalizes to strategies by induction on tree height.

Parts 2 and 3 are also established by induction on strategy tree height, by first showing that querying a single coin can only increase the highest mean, using the fact that the expectation over the mean of a coin if the coin is queried remains the same. Furthermore, tossing a coin is useless if the coin cannot affect the highest mean within a single toss. For the induction step for part 3, we observe that no optimal strategy of smaller height may query the coin (by induction), and no leaf reports the coin (by the stated assumption). Therefore the relative merit of strategies $E(\mu_{max}|s)$ remains the same whether or not the coin is queried at first. Consequently, the optimal strategy is identical whether or not the coin is queried, implying that the regret also remains the same whether or not the coin is queried. □

We conclude that the optimization problem boils down to computing a strategy that maximizes the expectation over the highest mean; $s^* = \text{argmax}_{s \in S}\{E(\mu_{max}|s)\}$. In selecting the coin to query, it is fairly intuitive that two significant properties of a coin are the magnitude of its current mean, and the spread of its density, that is how changeable its density is if it is queried: if a coin's mean is too low, it can be ignored by the above result, and if its density is too peaked (imagine no uncertainty), then querying may yield little or no information (the expectation $E(\mu_{max}|s)$ may not be significantly higher than $E(\mu_{max})$). However, it is fairly surprising that, for the purpose of computing an optimal policy, we cannot make further immediate use of these properties. For example, assume coin c_1 has *Beta*$(1, 2)$ prior, and coin c_2 has *Beta*$(1, 3)$, thus c_1 has a higher mean and a lower spread than c_2. But the optimal strategy for a budget of one (and two) starts by querying c_2. The main reason in this case remains that querying c_1 does not change our decision under either of its two outcomes (c_1 will be the winner), and thus the $E(\mu_{max})$ equals the current highest mean value of $1/3$, while querying c_2 affects the decision, and the expectation of the μ_{max} given c_2 is queried is slightly higher ($1/4 \times 2/5 + 3/4 \times 1/3$). The optimal strategy may also be contingent. For example, in Figure 2b, the policy may simply toss coin 1 twice. However, if we add a third coin with *Beta*$(21, 11)$, then if the outcome of the first toss is tails, the next optimal action is to toss coin 2. These observations suggest that optimization may be hard even in the unit-cost case.

4.1 Approximability

Consider an algorithm $\mathcal{A}$ that given the input, outputs the next action to execute. We call algorithm $\mathcal{A}$ a *constant-ratio approximation algorithm* if there is a constant ℓ (independent of problem size), such that given any problem instance, if r^* is the optimal regret for the problem, the regret $r(\mathcal{A})$, from executing actions prescribed by $\mathcal{A}$, is bounded by ℓr^*. Of course we seek an approximation algorithm (preferably with low ℓ) that is also efficient (polynomial time in input size). A constant-ratio approximation is especially desirable, as the quality of the approximation does not degrade with problem size.

4.2 Algorithms

We describe a number of plausible strategies and algorithms, and explore whether or not they are approximation algorithms for the unit-cost case. In the process, we also gain insight into the types of considerations that are significant for designing approximation algorithms.

4.2.1 Round-Robin, Random, and Greedy Algorithms

The *round-robin* algorithm simply tosses coins in a round-robin fashion, and the *random* algorithm simply queries the coins uniformly at random. These algorithms are plausible algorithms, at least initially in the trial period, and they are typically used in practice. The third algorithm we consider is the *constant-budget* algorithm: For a small constant k (independent of n and b), it computes the optimal strategy for that *smaller* budget k, and tosses the first coin of such a strategy. (Given the outcome, it then computes the optimal strategy from this new state, etc.) We shall refer to the algorithm as simply *greedy* in the case of $k = 1$. Perhaps it is not hard to see these algorithms are suboptimal, but we can say more:

Proposition 4.2 *For each algorithm $\mathcal{A}$ that is round-robin, random-strategy, or constant-budget: For any constant ℓ, there is a problem with minimum regret r^*, on which $r(\mathcal{A}) > \ell r^*$.*

Proof (sketch). On problems with uniform priors, large n, and relatively smaller budget m, round-robin and random strategies are unlikely to query a coin more than once, and are therefore expected to get a highest mean of not much more than $2/3$, from a coin with *Beta*$(2, 1)$. On the other hand, $E(\Theta_{max}) \approx 1$ with large n, and more targeted algorithms (such as envision-single, defined below) find a coin with mean close to this $E(\Theta_{max})$.

For the case of the constant-budget algorithm, assume $k = 1$ without loss of generality. Assume all coins have uniform priors except for two that have almost completely peaked priors and that their mean is high enough that even if querying either one gives all tails, the rest of the coins cannot catch them within one toss. Then the greedy algorithm may waste tosses on these two coins to identify a winner, but the optimal would explore the rest of the coins, given enough budget, and would have a good chance of discovering a significantly better coin. □

4.2.2 Look-ahead and Allocational Strategies

The single-coin look-ahead, or simply the *look-ahead* algorithm, takes the budget into account: at each time point, for each coin i, it considers allocating *all* of the remaining tosses to coin i, computes the expected regret from such allocation, and tosses a coin that gives the lowest such regret. As there are $k + 1$ distinguishable outcomes for tossing a single coin k times (where k is the number of remaining tosses), this computation can be done in polynomial time ($O(nk)$ at every time point). The algorithm tosses a coin that is expected to reduce the regret more than others. As this algorithm considers each coin alone, it fails in querying the right coins, at least initially, in the following type of scenarios: there are two kindes of coins (*i.e.*, two different priors are used): Type 1 coins have probability $1/k$ of having head probability at 1.0 (*e.g.*, $k = n/2$) and otherwise their head probability is 0. With a budget of n, the optimal algorithm would toss n such coins and would obtain a regret close to 0. Coins of type 2 are used to "distract" the look-ahead: in each step the increase in maximum mean from tossing any one of them is higher than tossing type 1 coins, but their highest mean is bounded away from 1. We have seen empirically that in such scenarios the regret ratio of look-ahead to optimal can exceed a factor of 6 for example, but the priors are carefully designed. In the next section, we will see that look-ahead is one of the top two algorithms, in the case of identical priors.

The argument suggesting that the look-ahead algorithm is not an approximation algorithm, sheds another insight into problem structure. It tells us that in designing an approximation (or an optimal) algorithm, it is important to know not only the current budget and the characteristics of the individual coin i, but also whether we have sufficiently many other coins similar to i: while the (expected) reduction in regret from tossing a single coin may be low, there may be sufficiently many such coins and the budget may be large enough, that spending and spreading tosses over such coins is justified. This observation motivates the following generalization of the look-ahead algorithm: compute the (approximately) optimal *allocational* strategy: An allocational strategy is specified by the number of tosses assigned to each coin. For example, given a budget of 5, an allocation may specify that coin 1 should be tossed twice, coin 2 tossed once, and coin 3 twice (and all other coins 0 times). Notice this allocation does not specify which coin to query first (any coin with positive allocation may be queried), and it is not contingent, *i.e.*, it does not change the allocation given the previous outcomes, and therefore it is suboptimal. However, two interesting questions are: whether the expected regret of an optimal allocational strategy is a constant approximation to the minimum regret (of the unrestricted optimal strategy), and whether an (approximately) optimal allocational strategy can be computed efficiently. Section 5 addresses the first question. The attraction of allocational strategies is that they are compactly represented and efficiently evaluated: the expected highest mean of an allocational strategy can be computed in time polynomial in the number of coins and the budget. In the special case of round-robin, with n coins, and an equal allocation of a tosses to every coin (when the budget is a multiple of n), the expression for the expected highest mean, $E(\mu_{\max})$, further simplifies to:

$$\sum_{h=0}^{a-1} [\mathrm{cdf}(h)^n - \mathrm{cdf}(h-1)^n] \frac{h+1}{(h+1)+(a-h+1)}$$
$$= \sum_{h=0}^{a-1} \left[\frac{(h+1)^n - h^n}{a^n}\right] \frac{h+1}{(h+1)+(a-h+1)}$$

where $\mathrm{cdf}(z)$ is $P(Z \leq z)$, and z is the number of heads; by convention, $\mathrm{cdf}(z < 0) = 0$. The next section will use this closed form when reporting on the performance of (static) allocational policies.

4.2.3 Biased-Robin

We have observed on small problem instances that the structure of the optimal strategy for the case of identical priors, unit-costs and when $n \geq b$, has a simple overall pattern (with some exceptions): It first chooses coin 1 and tosses it. While the chosen coin produces heads, the optimal strategy continues to toss that coin, until the budget is exhausted, or the coin gives a tail. In the latter case, it moves to the next (untouched) coin (see Figure 3). As long as $n \geq b$, there always exists an untouched coin. Note that similar algorithmic ideas arise when we are searching for a perfect coin (or model, *e.g.*, [SG95]). An exception to this pattern is that if the remaining budget is too low; the optimal action may be to toss a previous coin: Suppose after a few tosses, three coins have the distributions *Beta*$(\,5,\,2\,)$, *Beta*$(\,4,\,2\,)$, *Beta*$(\,1,\,1\,)$, and we have one more toss. The optimal action is to toss the coin with *Beta*$(\,5,\,2\,)$ rather than the untouched *Beta*$(\,1,\,1\,)$. We have implemented this basic algorithm, and for the cases when $n < b$, the algorithm simply wraps around when there are no more untouched coins. We call it *biased-robin* as it can be viewed as a variant of round-robin, and similar to round-robin, it is budget independent and any-time.

5 Empirical Performance

We report on the performance of the algorithms on the important special case of identical priors and unit costs. Figure 4 shows the performance of round-robin, random strategy, (single-coin) look-ahead, allocational, and the biased-robin algorithms, when $n = 10$, $b = 40$, and the priors are respectively uniform, skewed to the right, *Beta*$(\,10,\,1\,)$, and skewed to the left *Beta*$(\,1,\,10\,)$. Each time point is the average of 1000 trials, where at the beginning of each trial every coin's head probability is drawn from the prior. We computed the regrets at every intermediate time point to illustrate the performance of the algorithms as the budget is reached. The expectation of the actual maximum valued coin, $E(\Theta_{max})$, is about 0.9 in the case of uniform priors, and as a randomly picked coin has expected head probability of 0.5, the initial regret for each algorithm is around 0.4 in this case, as expected.

The biased-robin and envision-single strategies consistently outperform the rest of the strategies tested, while the envision-single is the most costly algorithm. The biased-robin algorithm has performed the best over the many other parameter settings that we have tested. The reason for the poor performance of greedy is simple: it leads to startvation, i.e., due to its myopic property, as soon as some coin gets ahead, a single toss of any coin does not change the leader, and yields to equal expected regret. In this case, ties are broken by tossing the first coin in the list, which may not change the status. If we add a randomization component to greedy, its performance improves, and becomes slightly better than random (not shown), and still inferior to biased-robin. As observed, looking ahead improves the performance significantly.

The allocational strategy computes the optimal allocation at the outset, and sticks to that allocation. For example, for a budget of 40, uniform priors and 10 coins, 8 of the 10 coins are allocated 5 tosses each. The allocational strategy tosses them in a round-robin fashion. Here, as the priors are identical, we need only search the space of equal allocations (b many) to find the optimal (initial) allocation.

The relative performance of the round-robin (and random) strategy compared to biased-robin, *i.e.*, the ratio of the regret of round-robin to the regret of biased-robin, gets worse

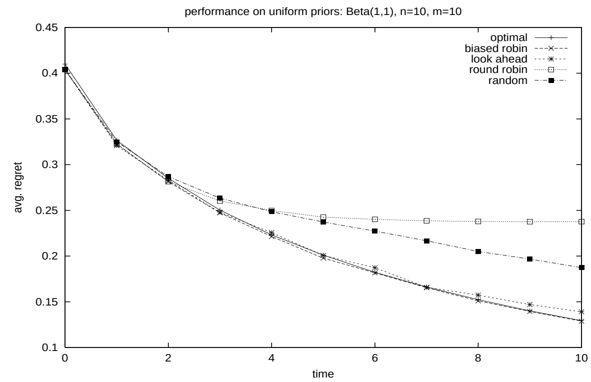


Figure 5: The performance of the algorithms versus optimal with $n = 10, 0 \leq b \leq 10$, and unifrom priors, over 4000 trials. The regret of biased-robin appears lower than that of optimal at several time points due to inaccuracies of averaging.

initially with increasing budget and fixed n, but eventually begins to improve (not shown). The worst ratio is a function of n however, and the relative performance of round-robin gets worse with increasing n (as suggested by the proof of Proposition 4.1). For example, with $n = 10$, the worst ratio is below 3, while with $n = 20$, it surpasses 4. With fixed n and b, the relative performance worsens as we increase the first Beta parameter, skewing the prior to the right; and improves by increasing the second Beta parameter (Figure 4). The behavior of the allocational strategy approaches the round-robin strategy with increasing budget (and with identical priors), eventually allocating equal tosses to all the coins. We have observed empirically that the ratio of the allocational regret to that of biased-robin also degrades with increasing n, *e.g.*, at $n = b = 20$, the ratio is about 1.5, at $n = b = 100$, it is over 2, and at $n = b = 200$, it is over 3. Therefore, periodic reallocation, *i.e.*, dynamic versus static allocation, appears to be necessary if such a technique is to yield an approximation algorithm.

We computed the optimal regret for $n \leq 10$ and $b \leq 10$, on different types of identical priors. Figure 5 shows the performance of the optimal strategy against the other algorithms on uniform priors. On these problems, the performances of look-ahead and biased-robin algorithms are very close to that of optimal, suggesting that these algorithms may be approximation algorithms for the special case of identical priors.

Due to its performance, efficiency, and simplicity, the biased-robin algorithm is the algorithm of choice among the algorithms we have tested, for the case of identical priors and costs.

6 Summary and Future Work

There are a number of directions for future work, in addition to the open problems we have raised. An immediate extension of the problem is to selecting classifiers, which

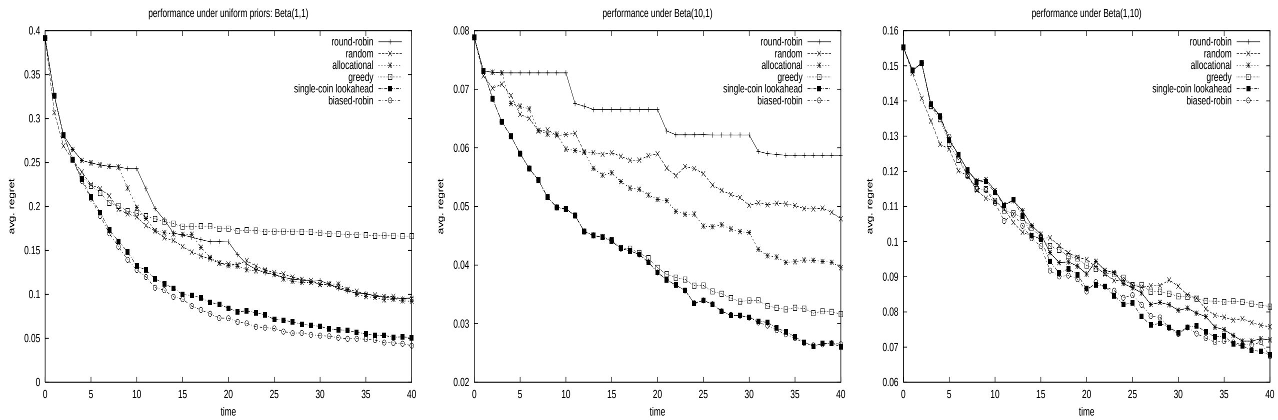


Figure 4: The performance of the algorithms with 10 coins, budget b of 40 tosses, and (a) uniform (*Beta*$(1, 1)$) priors, (b) skewed *Beta*$(10, 1)$, and (c) *Beta*$(1, 10)$.

may be defined in a similar way: we are presented with n candidate classifiers (imagine decision trees) with priors over a measure of their performance, and our task is to declare a winner, one that minimizing the regret of not selecting the actual best. A significant additional dimension in this case is that querying a single feature affects not one but multiple classifiers in general. There are however "flatter" versions of the classifier problem, such as learning a Naive Bayes classifier, in which these dependencies are limited. Our solution techniques more readily extend to the latter problems [LMG03].

Contributions: We introduced and motivated the budgeted multi-armed bandit task and precisely defined a basic problem in this space, the coins problem, as a problem of sequential decision making under uncertainty. We investigated the computational complexity of the coins problem, and the performance of a number of algorithms on the problem, both from a worst-case perspective and empirically. Our analyses demonstrate significant problems with the standard algorithms, round-robin and random, as well as greedy. We presented two different selective querying techniques that significantly outperform the standard algorithms.

References

[Ang92] D. Angluin. Computational learning theory: survey and selected bibliography. In *Proc. 24th Annu. ACM Sympos. Theory Comput.*, pages 351–369. ACM Press, New York, NY, 1992.

[BF85] D. A. Berry and B. Fristedt. *Bandit Problems: Sequential Allocation of Experiments*. Chapman and Hall, New York, NY, 1985.

[CAL94] D. A. Cohn, L. Atlas, and R. E. Ladner. Improving generalization with active learning. *Machine Learning*, 15(2):201–221, 1994.

[Dev95] J. Devore. *Probability and Statistics for Engineering and the Sciences*. Duxbury Press, New York, NY, 1995.

[Duf02] M. Duff. *Optimal Learning: Computational procedures for Bayes-adaptive Markov decision processes*. PhD thesis, University of Massachusets, Amherst, 2002.

[EDMM02] E. Even-Dar, S. Mannor, and Y. Mansour. Pac bounds for multi-armed bandit and Markov decision processes. In *COLT 2002*, 2002.

[GGR02] R. Greiner, A. Grove, and D. Roth. Learning cost-sensitive active classifiers. *Artificial Intelligence*, 139(2), 2002.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability*. W. H. Freeman and Company, 1979.

[Gre] http://www.cs.ualberta.ca/∼greiner/budget.html.

[KL00] S.R. Kulkarni and G. Lugosi. Finite time lower bounds for the two-armed bandit problem. *IEEE Transactions on Automatic Control*, 45(4):711–714, 2000.

[LMG03] D. J. Lizotte, O. Madani, and R. Greiner. The budgeted-learning problem, part II: the Naive Bayes case. submitted, 2003.

[LMRar] M. Lindenbaum, S. Markovitch, and D. Rusakov. Selective sampling for nearest neighbor classifiers. *Machine Learning*, To appear.

[MCR93] R. Musick, J. Catlett, and S. Russell. Decision theoretic subsampling for induction on large databases. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 212 – 219, Amherst, MA, 1993.

[Pap85] C. H. Papadimitriou. Games against nature. *Journal of Computer and Systems Science*, 31:288–301, 1985.

[Put94] M. L. Puterman. *Markov Decision Processes*. Wiley Inter-science, 1994.

[RM01] Nicholas Roy and Andrew McCallum. Toward optimal active learning through sampling estimation of error reduction. In *Proc. 18th International Conf. on Machine Learning*, pages 441–448. Morgan Kaufmann, San Francisco, CA, 2001.

[RS95] S. Russell and D. Subramanian. Provably bounded-optimal agents. *Journal of Artificial Intelligence Research*, 2:575–609, 1995.

[SG95] D. Schuurmans and R. Greiner. Sequential PAC learning. In *Proceedigs of COLT-95*, 1995.

[TK00] S. Tong and D. Koller. Active learning for parameter estimation in bayesian networks. In *NIPS*, pages 647–653, 2000.

[Tur00] P. Turney. Types of cost in inductive concept learning. In *Workshop on Cost-Sensitive Learning at the Seventeenth International Conference on Machine Learning*, pages 15–21, 2000.

[Val84] Leslie G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.