Optimal Policy for Bernoulli Bandits: Computation and Algorithm Gauge

Sebastian Pilarski, *Student Member, IEEE*, Slawomir Pilarski, *Member, IEEE*, and Dániel Varró

***Abstract*—Bernoulli multi-armed bandits are a reinforcement learning model used to study a variety of choice optimization problems. Often such optimizations concern a finite-time horizon. In principle, statistically optimal policies can be computed via dynamic programming, but doing so is considered infeasible due to prohibitive computational requirements and implementation complexity. Hence, suboptimal algorithms are applied in practice, despite their unknown level of suboptimality. In this article, we demonstrate that optimal policies can be efficiently computed for large time horizons or number of arms thanks to a novel memory organization and indexing scheme. We use optimal policies to gauge the suboptimality of several well-known finite- and infinite-time horizon algorithms including Whittle and Gittins indices, epsilon-greedy, Thompson sampling, and upper-confidence bound (UCB) algorithms. Our simulation study shows that all but one evaluated algorithm perform significantly worse than the optimal policy. The Whittle index offers a nearly optimal strategy for multi-armed Bernoulli bandits despite its suboptimal decisions—up to 10%—compared to an optimal policy table. Lastly, we discuss optimizations of known algorithms. We derive a novel solution from UCB1-tuned. It outperforms other infinite-time horizon algorithms when dealing with many arms.**

***Impact statement*—Bernoulli bandits are a reinforcement learning model used to improve decisions with binary outcomes. They have various applications ranging from headline news selection to clinical trials. Existing bandit algorithms are suboptimal. This article provides the first practical computation method, which determines the optimal decisions in Bernoulli bandits. It provides the lowest achievable decision regret (maximum expected benefit). In clinical trials, where an algorithm selects treatments for subsequent patients, our method can substantially reduce the number of unsuccessfully treated patients—by up to 5×. The optimal strategy is also used for new comprehensive evaluations of well-known suboptimal algorithms. This can significantly improve decision effectiveness in various applications.**

***Index Terms*—Clinical trials, epsilon-greedy, Gittins index (GI), multi-armed Bernoulli bandits, optimal policy (OPT), POKER, Thompson sampling (TS), upper-confidence bound (UCB), Whittle index (WI).**

Manuscript received December 12, 2020; revised February 9, 2021 and April 8, 2021; accepted April 10, 2021. Date of publication April 19, 2021; date of current version June 18, 2021. This work was supported in part by the Versyn Inc. and in part by the Discovery Grant NSERC RGPIN-2016-04573. This article was recommended for publication by Associate Editor C. Wagner upon evaluation of the reviewers' comments. *(Corresponding author: Sebastian Pilarski.)*

Sebastian Pilarski and Dániel Varró are with the Department of Electrical and Computer Engineering, McGill University, Montreal, QC H3A 0G4, Canada (e-mail: sebastian.pilarski@mail.mcgill.ca; daniel.varro@mcgill.ca).

Slawomir Pilarski is with the Versyn Inc., Vancouver, WA 98660 USA (e-mail: slawomir@versyn.com).

Digital Object Identifier 10.1109/TAI.2021.3074122

I. Introduction

CHOICES must be made when playing games, buying products, and when treating medical patients. The greater the complexity of outcome, the more difficult finding an optimal strategy must be. Traditionally, determining a superior selection choice was accomplished by allocating an equal number of samples into each option and seeing which option performed better on average. However, in marketing or medicine, such a methodology can greatly decrease the number of purchasing consumers or treated patients during testing. Likewise, with small testing populations, this may not be the best strategy for maximizing information gain or optimizing the outcome.

At the beginning of testing (e.g., news headline testing or clinical trials), the efficacy of each option is unknown. During testing, each option is "measured" against others. Some test subjects will receive a more click-inducing headline or better medical treatment than others. This is an inevitable price of knowledge acquisition. Achieving the greatest number of article reads or treating the largest number of patients is of the utmost priority.

In many situations the process of selecting options is sequential and often the sequence length is limited. Such situations are studied using a reinforcement learning model referred to as a multi-armed bandit [1].

Multi-armed Bandits: A multi-armed bandit is the problem of a *player* (or a gambler), who facing a row of different one-arm slot machines attempts to maximize the total gain, or *cumulative reward*, when he or she is allowed us to pull only one arm at a time. The player does not know expected payoffs, so a successful strategy, or *algorithm*, has to effectively balance exploration and exploitation [1]. Some algorithms assume that the game continues forever, while others consider finite-time horizons, i.e., situations where the player has a fixed total number of arm pulls.

Bernoulli Multi-armed Bandits: A large number of real-life choices can be modeled using binary outcomes: true or false, yes or no, success or failure. Examples include A/B testing (e.g., news headline selection, click feedback) [2]–[4], and clinical trials [5]–[11]. Such situations are studied using Bernoulli multi-armed bandits, where rewards take the binary outcomes 1 or 0, and each arm is characterized by its own probability of success, which is initially drawn from a predefined distribution, referred to as a *prior*. Conceptually, given a prior and a time horizon, it is possible to find the best strategy or a deterministic *optimal policy (OPT)* [7]. However, computing such policies for all but small examples is regarded as infeasible in existing research. Hence, in practice, suboptimal algorithms are used.

Example—Clinical Trials: The Bernoulli bandit model is advocated as an alternative to *traditional randomized clinical trails* where patients are divided into similar size cohorts and each cohort receives a different drug or treatment. *Bernoulli bandit algorithms optimize the expected number of patients successfully treated during a trial.* A detailed discussion of Bernoulli bandits in clinical trials and treating rare diseases can be found in [5] and [6]. In [11], a Bernoulli bandit model is extracted from clinical records. In brief, each arm corresponds to a unique treatment and the arm probability is the likelihood that a patient would achieve a success criterion, e.g., reaching a given level of antibodies. An algorithm selects a treatment for each patient based on observed successes and failures in previous patients. A trial vaccine could be compared to a placebo, another vaccine, or various levels of dosage. Such trials may range from a couple hundred to thousands participants and generally compare only two or three treatment options.

The importance of developing methodologies for clinical trials in small populations is recognized by the European Union, which funds multidisciplinary projects on the topic [10].

Note that optimizing the well-being of clinical trial patients may conflict with long-term success objectives for a general population. This topic is discussed in Section VIII.

A. Related Work and Motivation

There exists a significant amount of the literature dedicated to studying multi-armed bandit problem variations and their formulations. Considerable work has been done to survey the most common algorithms [12]–[14] and to compare their relative effectiveness [11], [15]. Some of such work concerns Bernoulli bandits [16] including those that focus on clinical trials, where time horizon is finite [5], [6], [8], [9]. Original work on Gittins index (GI) [17], Whittle index (WI) [18], and UCB1-Tuned (UCBT) [19] is fundamentally relevant (see Section II).

As pointed out in [11], theoretical bounds on effectiveness, if available [12], are too loose for practical evaluation. Empirical studies (e.g., [11]) compare relative effectiveness of various suboptimal algorithms without relating the results to the OPT, which by definition sets an upper bound on cumulative reward of any algorithm. To the best of authors' knowledge, only Villar *et al.* [6] present a comparative table with several OPT results for two-arm and three-arm bandits: up to time horizon 100 and 30, respectively.

Problem Statement: Computing OPT is commonly considered practically infeasible [5]–[7], [20], and [21]. As a consequence, suboptimal algorithms are used in practice. Existing empirical data regarding *the level of suboptimality of such algorithms* is very limited in scope from a practical point of view. Finally, most papers assume only uniform priors, which are usually not satisfied in real-world scenarios.

B. Objectives, Contributions, and Significance

Objectives: The main aim of this article is to explore new computation methods to produce OPTs for previously infeasible time horizons, number of arms, and various priors. It also aims to assess the level of suboptimalities for a wide range of existing bandit algorithms.

Contributions:

1) *Methods to compute Bernoulli bandit OPTs* based on a compact indexing scheme and exploitation of symmetries. (see Sections III and IV).
2) *Computation of OPTs.* We studied optimal performance for two arms and three arms for time horizons of 3000 and 400, respectively. Computations for up to ten arms are also presented for the first time. (see Section V)
3) *Most comprehensive empirical evaluation* of suboptimal algorithms with the OPT as a gauge including nonuniform priors (underrepresented in publications) for more realistic evaluation. (see Section VI)
4) *New algorithm for infinite-time horizons*, which is a variation of the well-known UCBT algorithm. It often performs substantially better than popular suboptimal algorithms (most notably GI). (see Section VII)

Significance: As a main benefit of these contributions, OPTs can now displace suboptimal algorithms in many practical problems. WI was empirically determined to be near-optimal and by far the best suboptimal algorithm for finite-time horizons regardless of the number of arms or priors. Our modification of UCBT significantly improves upon the original algorithm.

II. Preliminaries

A. Definitions and Notation

In this article, we use well-known notions of probability theory such as *expected value*, *(arithmetic) mean*, *standard deviation*, and *variance* of a random variable [22]. We also use the standard notion of a *cumulative mean*, which is the arithmetic mean of means over a period of time. *Cumulative* also applies to other random variables, not just the mean. We use the following common notation.

$\mathbb{P}[A]$	is the probability of an event A.
$\mathbb{E}[X]$	denotes the expected value of a random variable X.
σ	denotes standard deviation of a random variable.
σ^2	denotes variance of a random variable.
$\hat{\sigma}^2$	denotes variance of observed values.
$\mathcal{N}$	is the normal distribution.
$\text{argmax}_{i=1,\ldots,k}(g(i))$	selects i with max value of $g(i)$.
$\text{argmin}_{i=1,\ldots,k}(g(i))$	selects i with min value of $g(i)$.

The Bernoulli multi-armed bandit, $\text{BMAB}(k, H)$, has k arms and time horizon H. In this single-player game, the player (or algorithm) chooses one of k arms at each time (step), $0 \leq t < H$, and receives binary reward 1 (success) or 0 (failure) immediately. Each arm has success probability drawn from some *prior* distribution at the start of each game.

When presenting or discussing bandit algorithms, we use the following symbols.

k	is the number of arms.
H	is the time horizon; number of pulls in a single game.
t	is time, $0 \leq t \leq H$.
n_i	is the number of times arm i was pulled.
s_i	is the number of times arm i produced a success.
f_i	is the number of times arm i produced a failure.
μ_i	is the expected value of reward for arm i.
$\hat{\mu}_i$	is the observed mean reward for arm i.

Note that μ_i, $\hat{\mu}_i$, and n_i apply to a single experiment, i.e., drawing arm's probabilities and playing one *game* that consists of a sequence of single arm pulls.

Note also that for a single game, at any time t, $\Sigma_{i=1}^{k} n_i = t$. In a single game, $n_i = s_i + f_i$ is a function of t and $n_i(t)$ denotes the number of times arm i has been pulled by time t.

When analyzing the performance of a player's algorithm one may look at the number of times the arm with the highest probability of success was selected. We call it the *best arm*.

best_arm $= \text{argmax}_{i=1,\ldots,k}(\mu_i)$.

Symbols associated with best arm are labeled with $*$.

$\mu^* = \max_{i=1,\ldots,k}(\mu_i)$ is best expected reward.

$\hat{\mu}^* = \max_{i=1,\ldots,k}(\hat{\mu}_i)$ is best observed mean reward.

Note that in a single game μ^* is a constant, while $\hat{\mu}_i^*$ is a random variable.

One may also look at the mean of best arm success probabilities over a number of simulation experiments. Expected value of best arm success probability constitutes an upper bound on the mean reward of any player's algorithm at any time t, which is a function of the predefined distribution of arm success probabilities and the number of arms k.

Regret: A common way to compare algorithms is to examine their regret (loss of opportunity [7]), which experimentally is the mean of the difference between the best arm's success rate and the selected arm's success rate, computed over a number of simulation runs [12]. Formally, regret$(t) = \mathbb{E}[\mu^* - \mu(t)]$ where $\mu(t)$ is the expected reward of the arm selected by the player at time t in a single game. μ^* is a constant in a single game, but a random variable when multiple games are run. Minimization of cumulative regret is equivalent to maximizing cumulative reward.

In our simulation experiments, each simulation run starts with assigning each arm its success probability. This probability is a random variable drawn from a probability distribution, a prior. In most of our simulations we use the Beta distribution [23]. The probability density function of the Beta distribution is given by the following equation: $\frac{x^{\alpha-1}(1-x)^{\beta-1}}{\int_0^1 y^{\alpha-1}(1-y)^{\beta-1}dy}$. Fig. 5 presents plots of the Beta distribution for selected values of parameters α and β.

Property 1 (Conjugate Prior): Pulling an arm changes its distribution from Beta(α, β) to Beta$(\alpha+1, \beta)$ if the outcome is a success, or to Beta$(\alpha, \beta+1)$ if the outcome is a failure [23].

This property, together with (1), plays a vital role in computations used in some algorithms discussed in this article.

$$\mathbb{E}\left(\text{Beta}(\alpha, \beta)\right) = \frac{\alpha}{\alpha+\beta}. \tag{1}$$

The Beta distribution's standard deviation [23] is

$$\sigma\left(\text{Beta}(\alpha, \beta)\right) = \sqrt{\frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}}. \tag{2}$$

B. Infinite-Time Horizon Algorithms

Infinite-time horizon bandit algorithms assume the game continues forever and state how the player chooses one of the k arms at any time t.

1) ϵ-Greedy (EPS) [1], [11], and [12]:
This very simple algorithm, at any time, randomly selects exploration or exploitation of best observed arm

$$\textit{choose}(t) = \begin{cases} \text{any arm with probability } \epsilon \\ \underset{i=1,\ldots,k}{\text{argmax}}\left(\hat{\mu}_i(t)\right), \text{ otherwise.} \end{cases}$$

2) Thompson Sampling (TS) [24]–[26]:
This is a randomized algorithm presented here in the context of Bernoulli bandits. In essence, it generates random numbers according to each arms' Beta distribution, and picks the arm with the largest random number

$$\textit{choose}(t) = \underset{i=1,\ldots,k}{\text{argmax}}\left(\textit{random}\Big(\text{Beta}\left(\alpha_i(t), \beta_i(t)\right)\Big)\right).$$

3) UCB1 [1], [12], [19]:
The UCB1 algorithm selects an arm via the evaluation of a weighted sum of average of observed reward and a term which bounds the regret to logarithmic growth

$$\textit{choose}(t) = \underset{i=1,\ldots,k}{\text{argmax}}\left(\hat{\mu}_i(t) + c\sqrt{\frac{2\ln t}{n_i(t)}}\right) \tag{3}$$

where c is a constant, which is usually omitted, i.e., tacitly assumed to be 1. Constant c is explicitly discussed in [1] as a mechanism to control degree of exploration.

4) UCBT [12], [19]:
The UCBT algorithm is similar to UCB1 but changes the second term

$$\textit{choose}(t) = \underset{i=1,\ldots,k}{\text{argmax}}\left(\hat{\mu}_i(t) + c\sqrt{\frac{\ln t}{n_i(t)}\min\left(\frac{1}{4}, V_i(t)\right)}\right) \tag{4}$$

$$V_i(t) = \hat{\sigma}_i^2(t) + \sqrt{\frac{2\ln t}{n_i(t)}} \tag{5}$$

where $\hat{\sigma}_i^2$ is the variance of the success rate and c is a constant. The constant c has been introduced in this article to control degree of exploration as in the UCB1 algorithm mentioned above.

5) Gittins index (GI) [5], [6], [17], [27] (see Appendix A):
GI was a milestone in developing Bernoulli bandit algorithms. It simplifies comparing multiple arms by using precomputed lookup tables

$$\textit{choose}(t) = = \underset{i=1,\ldots,k}{\text{argmax}}\left(G_I\left(s_i(t)+\alpha_i, f_i(t)+\beta_i\right)\right)$$

where G_I is the GI, s_i and f_i are the numbers of successes and failures observed for arm i, and Beta(α_i, β_i) is the distribution from which success probability of arm i was drawn.

C. Finite-Time Horizon Algorithms

Finite-time horizon algorithms assume that a single game ends after exactly H arm pulls, which information is used to optimize cumulative reward.

1) Whittle index (WI) [5], [6], [18] (see Appendix A):

WI is a modification of GI for finite-time horizon

$$\textit{choose}(t) = \underset{i=1,\ldots,k}{\operatorname{argmax}} \left(W_I\left(s_i(t)+\alpha_i, f_i(t)+\beta_i, H-t\right) \right)$$

where W_I is the WI, H is the time horizon, s_i and f_i are the numbers of successes and failures observed for arm i, and Beta(α_i, β_i) is the distribution from which success probability of arm i was drawn.

2) POKER [12], [28] (see Appendix A):
It stands for *price of knowledge and expected reward*

$$\textit{choose}(t) = \underset{i=1,\ldots,k}{\operatorname{argmax}} \left(p_i(t) \right)$$

where $p_i(t)$ is the price of knowledge for arm i.

3) Optimal policy (OPT) [6], [7], and [13]:
For any configuration of successes and failures on all arms, the OPT identifies the arm with best expected reward until the end of the game

$$\textit{choose}(t) = O_{P_H}\left((s_1(t), f_1(t)), \ldots, (s_k(t), f_k(t)) \right)$$

where O_{P_H} is the OPT for time horizon H and a set of Beta priors—one for each arm.

III. Computing the Optimal Policy

For any configuration of successes and failures on all arms, the OPT selects the arm leading to the best expected reward till the end of the game. This principle is captured by the following equations, which compute best expected value O_{V_H} and the OPT O_{P_H} for time horizon H by dynamic programming iterating backward from $t = H$ down to $t = 0$.

1) The game finishes at $t = H$, and there are no more rewards

$$O_{V_H}\left((s_1, f_1), \ldots, (s_k, f_k)\right) = 0, \text{ if } \Sigma_{i=1}^{k}(s_i + f_i) = H.$$

2) For a discount factor d (assumed later to be $d = 1$), the *expected value* V_{H_i} of pulling an arm i is given by

$$\begin{aligned} &V_{H_i}\left((s_1, f_1), \ldots, (s_k, f_k)\right) \\ &= \frac{\alpha_i + s_i}{\alpha_i + s_i + \beta_i + f_i}\left(1 + d \cdot O_{V_H}\left(\ldots, (s_i+1, f_i), \ldots\right)\right) \\ &+ \frac{\beta_i + f_i}{\alpha_i + s_i + \beta_i + f_i}\left(0 + d \cdot O_{V_H}\left(\ldots, (s_i, f_i+1), \ldots\right)\right). \end{aligned}$$

3) For $0 \le t < H$ (where $t = \Sigma_{i=1}^{k}(s_i + f_i)$), the optimal strategy selects the *best expected value* according to the following rule

$$\begin{aligned} &O_{V_H}\left((s_1, f_1), \ldots, (s_k, f_k)\right) \\ &= \max_{i=1,\ldots,k}\left(V_{H_i}\left((s_1, f_1), \ldots, (s_k, f_k)\right) \right) \\ &\qquad \text{if } 0 \le \Sigma_{i=1}^{k}(s_i + f_i) < H. \end{aligned}$$

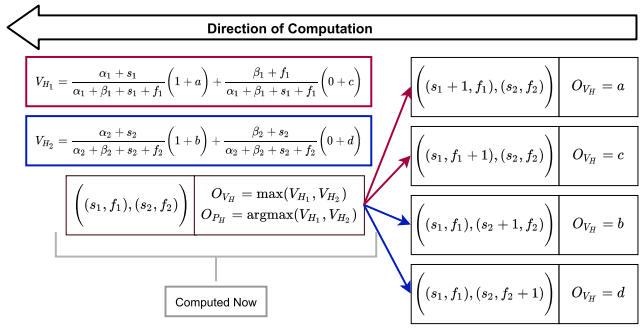


Fig. 1. Single step in computing the optimal policy (O_{P_H}) for two arms.

4) OPT selects the arm with maximal V_{H_i}

$$\begin{aligned} &O_{P_H}\left((s_1, f_1), \ldots, (s_k, f_k)\right) \\ &\quad = \underset{i=1,\ldots,k}{\operatorname{argmax}}\left(V_{H_i}\left((s_1, f_1), \ldots, (s_k, f_k)\right) \right) \\ &\qquad \text{if } 0 \le \Sigma_{i=1}^{k}(s_i + f_i) < H. \end{aligned}$$

Conceptually, an OPT is determined by computation of expected rewards backward from the final arm pull. First, all possible configurations such that $\sum_{i=1}^{k}(s_i + f_i) = H - 1$ (i.e., one pull remaining) are considered. For each such configuration, the arm which has the highest expected reward, V_{H_i}, is selected. This arm is the configuration's optimal O_{P_H} and its expected reward is the configuration's O_{V_H}.

When the optimal policy and expected rewards are computed for pull j, the optimal policy and expected rewards for configurations of pull $j-1$ can be computed next. For any configuration of pull $j-1$, an arm pull leads to two configurations of pull j: one via a success and one via a failure. Computing expected reward for each arm, V_{H_i}, takes into account both possible transitions and the expected rewards associated with the two reached configurations. The optimal decision is the one that selects the arm with the highest expected reward. This is a typical dynamic programming computation [29], [30]. It finds a deterministic optimal policy. An example of a single step in such computation is illustrated in Fig. 1.

The main challenge in optimal policy computation lies in excessive memory and computation requirements [20], [21]. Some publications, e.g., [6], mention high implementation complexity, which probably means issues with iterating through all possible configurations of successes and failures such that $\sum_{i=1}^{k}(s_i + f_i) = j$. To achieve dense memory structure, we propose a new indexing scheme, which exploits order and symmetry of success/failure configurations.

A. Naive Memory Organization

A single configuration of successes and failures for all arms $((s_1, f_1), \ldots, (s_k, f_k))$ consists of k pairs of numbers. Thus, storing O_{P_H} requires storing data indexed by $2k$ variables. Taking into account that for each arm, i, $0 \le s_i < H$ and $0 \le f_i < H$, the naive memory organization is a $2k$-dimensional

array with H^{2k} entries. Naturally, since the total number of pulls is limited by H, i.e., $\sum_{i=1}^{k}(s_i+f_i)<H$, a large portion of the array would be unused. Hence, we may pose the following question.

Question 1: What is the minimum number of entries required to represent O_{P_H}?

Although at this time we cannot provide an exact answer, in the remainder of this section and Section IV, we reduce the number of required entries by orders of magnitude as compared to the naive memory organization.

B. Minimizing Storage for Two-Arm Optimal Policy

If an arm is pulled n times, exactly $n+1$ entries are needed to represent all of its (s,f) configurations. At time t, the first arm can have been pulled $n_1=0\ldots t$ times, and second arm $n_2=t-n_1$ times. Thus, $\Sigma_{n_1=0}^{t}((n_1+1)(t-n_1+1))$ is the number of possible success and failure configurations on both arms. It is easy to show that

$$\sum_{i=0}^{t}\left((i+1)(t-i+1)\right)=\frac{1}{6}(t+1)(t+2)(t+3) \quad (6)$$

where n_1 is replaced by i to simplify the formula. Consequently, since the total storage for O_{P_H} does not need entries for $t=H$ (only up to $t=H-1$) it can be expressed as

$$\sum_{t=0}^{H-1}\sum_{i=0}^{t}\left((i+1)(t-i+1)\right) =\frac{1}{24}H(H+1)(H+2)(H+3). \quad (7)$$

Equation (7) shows that for large values of H storage requirements can be reduced by a factor of 24 compared to the naive memory organization. Equations (6) and (7) provide not only closed form expressions for minimum required storage, but also suggest a memory organization and an indexing scheme.

1) Memory is organized as a 1-D array of size given by (7).
2) Memory is indexed from 0, with each index corresponding to a configuration $((s_1,f_1),(s_2,f_2))$.
3) Memory locations for configurations at time t start after all configurations for time $t-1$. Thus, memory locations for time t start at index equal to total storage for times 0 to $t-1$, denoted as O_{P_t}.
4) Configuration entries for time t are ordered first by s_1+f_1, then by s_1, next by s_2+f_2, and finally by s_2.

Hence, to find the memory location for data associated with $((s_1,f_1),(s_2,f_2))$, where (by definition) $s_1+f_1+s_2+f_2=t$, the following indexing can be used.[1]

$$\begin{aligned}\text{Index}\left((s_1,f_1),(s_2,f_2)\right)&=\text{Storage}(O_{P_t})\\&+\sum_{i=0}^{s_1+f_1-1}\left((i+1)(t-i+1)\right)\\&+\sum_{i=0}^{s_1-1}(t-s_1-f_1+1)+s_2.\end{aligned} \quad (8)$$

[1]Note that the term "index" is used with a different meaning than in GI and WI.

Using (6) and (7), after simplifications, we get

$$\begin{aligned}\text{Index}\left((s_1,f_1),(s_2,f_2)\right)&=\frac{1}{24}t\,(t+1)(t+2)(t+3)\\&+\frac{1}{6}(s_1+f_1)(s_1+f_1+1)(2s_1+2f_1-3t-5)\\&+s_1\,(t-s_1-f_1+1)+s_2.\end{aligned} \quad (9)$$

Note that our OPT storage size and indexing are independent of prior Beta distributions.

Since for $k=2$ single optimal decision storage requires just one bit, 5.24 GB is sufficient to store the entire OPT for $H=1000$, and can be computed in seconds. To accomplish this, by (6), two temporary arrays of size $\frac{1}{6}H(H+1)(H+2)$ of floating point numbers (to store parts of O_{V_H}) are sufficient.

Observation 1: A two-arm Bernoulli bandit problem can be played optimally for time horizons up to 3000 using computing resources available in modern desktop computers. It applies, e.g., to clinical trials, where suboptimal algorithms are used.

C. Exploring Symmetry

In the memory layout and indexing scheme presented in the previous section, each arm could have its unique prior distribution. If both prior distributions are the same, however, the OPT table is symmetrical, i.e., $O_{P_H}((s_1,f_1),(s_2,f_2))$ chooses the opposite arm than $O_{P_H}((s_2,f_2),(s_1,f_1))$. Such a symmetry can be exploited to reduce OPT storage. If we assume that the second arm was pulled at most as many times as the first arm, the number of all configurations of successes and failures on both arms at time t [see (6)] is given by $\sum_{i=\lceil t/2\rceil}^{t}((i+1)(t-i+1))$, which can be simplified

$$\begin{aligned}&\sum_{i=\lceil t/2\rceil}^{t}\left((i+1)(t-i+1)\right)\\&=\frac{1}{6}(t-\lceil t/2\rceil+1)(t^2-2\lceil t/2\rceil^2+5t+(t+1)\lceil t/2\rceil+6).\end{aligned}$$

Consequently, the total storage for O_{P_H} can be expressed as

$$\sum_{t=0}^{H-1}\sum_{i=\lceil t/2\rceil}^{t}\left((i+1)(t-i+1)\right). \quad (10)$$

A pragmatic solution that accommodates our principles of memory organization stated in the previous section is to precompute (10) for subsequent values of H, and use these values as a lookup table. The index can be calculated in a similar way as before [see (8)]

$$\begin{aligned}\text{Index}\left((s_1,f_1),(s_2,f_2)\right)&=\text{Storage}(O_{P_t})\\&+\sum_{i=\lceil t/2\rceil}^{s_1+f_1-1}\left((i+1)(t-i+1)\right)\\&+s_1\,(t-s_1-f_1+1)+s_2.\end{aligned} \quad (11)$$

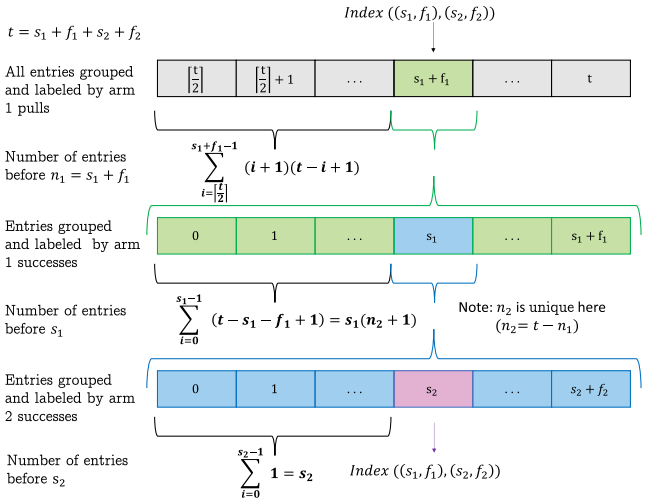


Fig. 2. Derivation of two-arm index.

Derivation of this equation is explained in Fig. 2. The sum in (11) can be expressed in a closed form

$$\sum_{i=\lceil t/2 \rceil}^{s_1+f_1-1} ((i+1)(t-i+1))$$

$$= \frac{1}{6}\Bigg(-\Big(-2\lceil t/2 \rceil + 3t + 5\Big)\lceil t/2 \rceil\Big(\lceil t/2 \rceil + 1\Big) \quad (12)$$

$$-(s_1+f_1)(s_1+f_1+1)(2s_1+2f_1-3t-5)\Bigg).$$

Our index can now be computed as follows:

$$\begin{aligned}\text{Index}\,((s_1,f_1),(s_2,f_2)) &= \text{LookupTable}(t)\\ &+\frac{1}{6}n_1(n_1+1)(3t-2n_1+5)\\ &+s_1(n_2+1)+s_2.\end{aligned} \quad (13)$$

Exploitation of symmetry reduces memory requirements by another factor of 2, which gives approximately 48 times smaller memory compared to the naive memory organization.

It may be worth pointing it out that symmetry can be exploited to reduce storage even if each arm has its unique prior beta distribution. It is a natural extension of the discussion mentioned above. It requires a small memory overhead.

Observation 2: A two-arm Bernoulli bandit problem can be played optimally for time horizons up to 5000 using computing resources available in modern desktop computers. In total, 80 GB of RAM is sufficient to compute OPT for $H = 5000$ without storing the policy table in RAM.

D. Three Arms

Memory layout for three arms is an extension of what we proposed for two arm OPTs. All we need is to add $(s_3 + f_3)$ and s_3 to the order rules. Consequently, without exploring symmetry, at time t, the number of all possible $((s_1,f_1),(s_2,f_2),(s_3,f_3))$ configurations is given by

$$\begin{aligned}&\sum_{i=0}^{t}\sum_{j=0}^{t-i}((i+1)(j+1)(t-i-j+1)) \quad (14)\\ &=\frac{(t+1)(t+2)(t+3)(t+4)(t+5)}{120}\end{aligned}$$

$$\begin{aligned}&\sum_{t=0}^{H-1}\sum_{i=0}^{t}\sum_{j=0}^{t-i}((i+1)(j+1)(t-i-j+1)) \quad (15)\\ &=\frac{H^6+15H^5+85H^4+225H^3+274^2+120H}{720}.\end{aligned}$$

Exploitation of symmetry by ordering the arms by their number of pulls, reduces the number of all possible $((s_1,f_1),(s_2,f_2),(s_3,f_3))$ configurations at time t to

$$\sum_{i=\lceil t/3 \rceil}^{t}\sum_{j=\lceil (t-i)/2 \rceil}^{\min(i,t-i)}((i+1)(j+1)(t-i-j+1)). \quad (16)$$

Let us simplify the inner sum in the last equation

$$\begin{aligned}S_I(x,y) &= \sum_{j=x}^{y}((i+1)(j+1)(t-i-j+1))\\ &= (i+1)\Bigg(-\frac{1}{2}(t-i)(x-y-1)(x+y+2) \quad (17)\\ &+\frac{1}{6}(x-y-1)(2x^2+x(2y-1)+2y^2+y-6)\Bigg).\end{aligned}$$

Taking into account (17), index can be computed as follows:

$$\begin{aligned}\text{Index}\,((s_1,f_1),(s_2,f_2),(s_3,f_3)) &= \text{Storage}(O_{P_t})\\ &+\sum_{i=\lceil t/3 \rceil}^{s_1+f_1-1} S_I\left(\lceil (t-i)/2 \rceil, \min(i,t-i)\right)\\ &+s_1\sum_{i=\lceil (t-s_1-f_1)/2 \rceil}^{\min(s_1+f_1,t-s_1-f_1)}((i+1)(t-s_1-f_1-i+1)) \quad (18)\\ &+\sum_{i=\lceil (t-s_1-f_1)/2 \rceil}^{s_2+f_2-1}((i+1)(t-s_1-f_1-i+1))\\ &+s_2(s_3+f_3+1)+s_3.\end{aligned}$$

The computation of this index is derived in Fig. 3.

Although we have not found a constant time expression as in the case of the two-arm bandit, it is possible to use two precomputed lookup tables of size $(H+1)\times(H+1)$ to guarantee constant time index computation. For $H = 300$ the entire OPT table can be computed in a few minutes, and the single game index computation overhead is negligible.

Note that (16) easily translates into a code loop structure, which efficiently iterates all possible ordered configurations $((s_1,f_1),(s_2,f_2),(s_3,f_3))$. Since for larger values of H such

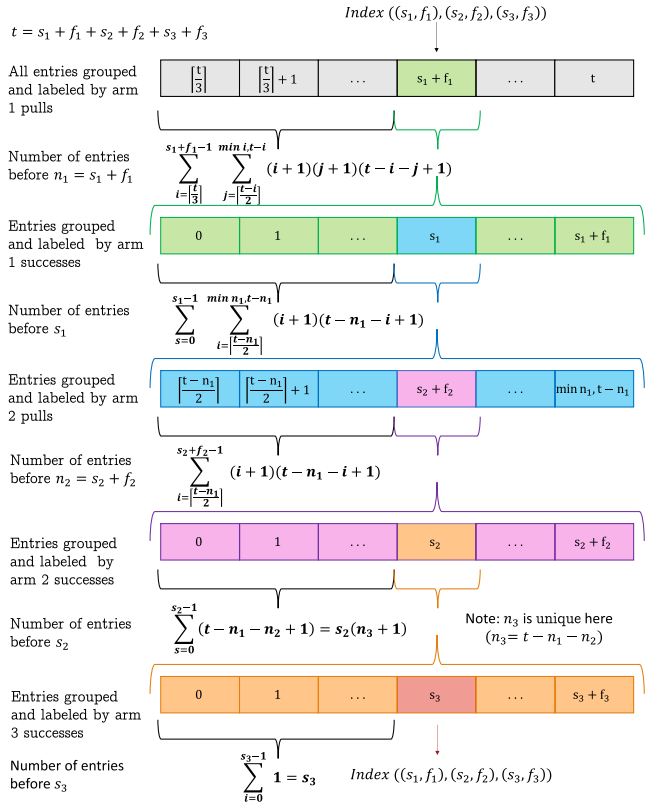


Fig. 3. Derivation of three-arm index.

an iteration takes only a small fraction of OPT computations,[2] the process can easily be accelerated by using multithreading.

Observation 3: Three-arm Bernoulli bandit problems can be played optimally for time horizons exceeding 400 using modern desktop computers. It applies, e.g., to clinical trials, where suboptimal algorithms are used.

E. k-Arms

Our analysis of storage for OPTs for two-arm and three-arm bandits can be generalized. By extending the reasoning presented in Fig. 3, one can notice that the number of possible $((s_1,f_1),\ldots,(s_k,f_k))$ configurations at time t is $g(k,t,t)$, where

$$g(k,n,\text{lim}) = \begin{cases} n+1 & \text{if } k=1 \\ \lim\limits_{\lceil n/k \rceil} \left((i+1)\ g(k-1, n-i, \min(i,n-i)) \right) \end{cases}$$

is the outer sum in expressions in Fig. 3. Consequently,

$$\text{Storage}(O_{P_{k,H}}) = \sum_{i=0}^{H-1} g(k,i,i)$$

$$\begin{aligned} \text{Index}\left((s_1,f_1),\ldots,(s_k,f_k)\right) &= \text{Storage}(O_{P_{k,t}}) \\ &+ \mathit{Offset}\left((s_1,f_1),\ldots,(s_k,f_k)\right) \end{aligned} \tag{19}$$

[2]For $H = 300$, iteration takes 0.45 s out of 32 min; 27 GB RAM is sufficient for floating point computations.

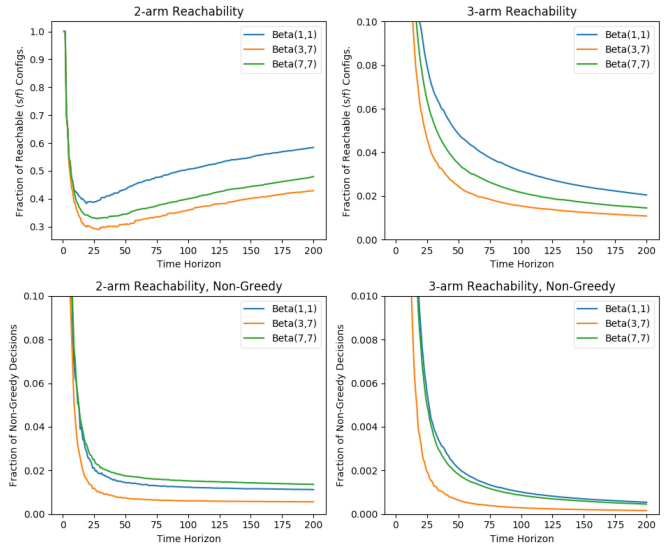


Fig. 4. OPT table: fraction of essential entries. Top plots show entries reachable from the first pull. Bottom plots show reachable entries that do not choose highest empirical mean arms.

where

$$\begin{aligned} \text{Offset}\left((s_1,f_1),\ldots,(s_k,f_k)\right) &= g(k, s_1+f_1-1, t) \\ &+ s_1\, g(k-1, t-s_1-f_1, \min(s_1+f_1, t-s_1-f_1)) \\ &+ \begin{cases} s_2 & \text{if } k=2 \\ \text{Offset}\left((s_2,f_2),\ldots,(s_k,f_k)\right) & \text{otherwise.} \end{cases} \end{aligned} \tag{20}$$

Observation 4: Expression for $g(k,n,lim)$ easily translates into a code-loop structure, which efficiently iterates all possible ordered configurations. Moreover, such an iteration takes only a tiny fraction of OPT computation, which can thus easily be accelerated by multithreading.

IV. Compression of Optimal Policy Table

While the size of OPT tables has been greatly reduced as compared to the naive memory organization, they can still grow very large. Thus, for practical reasons, before we start numerical computations or simulations using OPT tables, we may pose the following question.

Question 2: How much can the storage of OPT tables be compressed?

To compress the OPT storage, we investigate *reachable configurations* and *greedy decisions*.

Reachable configurations: Clearly, when following the optimal strategy from the beginning of the game, some configurations are *unreachable* (e.g., with uniform priors, one arm pulled 500 times with 500 failures and no pulls on any other arm). Hence, they do not need to be stored. The top two plots in Fig. 4 show the fraction of reachable configurations for various priors and time horizons 1–200 for two arms and three arms.

Greedy decisions: A close inspection of an OPT table reveals that most policy decisions are *greedy* and select the arm with the largest recorded reward mean. This regularity can be

TABLE I
OPT TABLES FRACTION OF ESSENTIAL ENTRIES, BETA(3,7)

	Time Horizon		Time Horizon	
	40	50	40	50
2-Arms	2.77e-02	3.08e-01	8.33e-03	7.39e-03
3-Arms	2.98e-02	2.44e-02	8.85e-04	6.36e-04
4-Arms	4.13e-03	2.72e-03	1.33e-04	7.69e-05
5-Arms	7.88e-04	4.20e-04	2.75e-05	1.30e-05
6-Arms	1.97e-04	8.30e-05	7.02e-06	2.77e-06
	Reachable		Reachable Non-greedy	

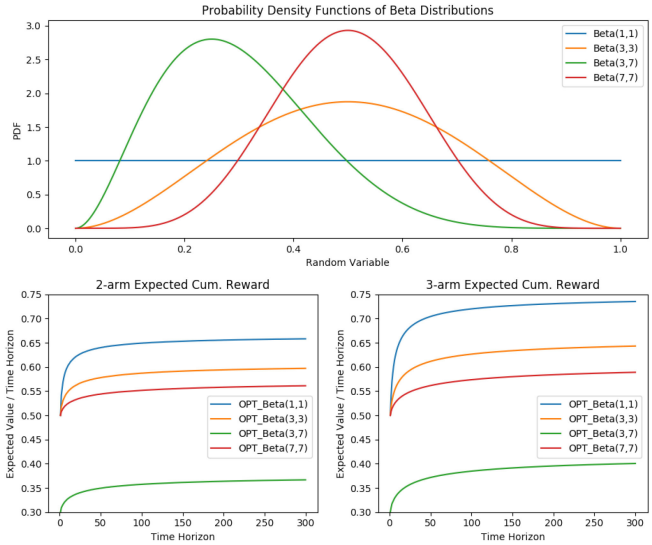


Fig. 5. Normalized expected cumulative rewards for two-arm and three-arm bandits.

used for compression: we only need to store exceptions. The bottom plots of Fig. 4 show the fraction of OPT entries that are reachable and their decisions are nongreedy, i.e., disagree with $\text{argmax}_{i=1,\ldots,k} \frac{s_i+\alpha_i}{s_i+f_i+\alpha_i+\beta_i}$. It identifies the fraction of all entries that need to be stored to represent the entire OPT table without relevant information loss. Table I presents examples of such data for several values of k.

Observation 5: OPT tables can be compressed in size by orders of magnitude (Table I demonstrates 3 to 6 orders of magnitude).[3] *The magnitude of compression appears to increase generally with the number of arms and time horizon.*

V. OPTIMAL POLICY RESULTS

The Bernoulli multi-armed bandit problem requires balancing exploration with exploitation to maximize cumulative reward across a game.

Question 3: What expected cumulative rewards are achievable with perfect play in the Bernoulli multi-armed bandit?

Computing OPT and indexing schemes discussed in Section III can be used to determine and plot expected cumulative rewards for various prior Beta distributions and a range of time horizons. Fig. 5 shows such plots for two-arm and three-arm Bernoulli bandits. By definition, they are the best possible

[3]Note that Observation 5 also relates to Question 1.

TABLE II
NORMALIZED EXPECTED CUM. REWARDS, BETA(3,7)

	Time Horizon				
	40	50	90	150	400
2-Arms	0.346654	0.349509	0.356356	0.361379	0.368515
3-Arms	0.366966	0.371559	0.38281	0.391286	0.403651
4-Arms	0.378354	0.384126	0.398496	0.409534	
5-Arms	0.385578	0.392214	0.408939		
6-Arms	0.390506	0.397815			
7-Arms	0.394028	0.401881			
8-Arms	0.396627				
9-Arms	0.398585				
10-Arms	0.400086				

TABLE III
NORMALIZED EXPECTED CUM. REWARDS, TWO ARMS

	Time Horizon				
	1000	1500	2000	2500	3000
Beta(1,1)	0.662938	0.663874	0.664398	0.664738	0.66498
Beta(3,7)	0.372664	0.373915	0.374635	0.375111	0.375454
Beta(7,7)	0.567415	0.568782	0.569575	0.570102	0.570483

cumulative rewards that can be achieved by any algorithm. They illustrate the perfect achievable balance between exploration and exploitation.

Expected cumulative rewards presented in Fig. 5 are *normalized*, i.e., divided by their time horizons, to better visualize differences between lines for smaller values of time horizon and average received reward per pull.

Table II contains more examples of expected cumulative rewards including for ten arms.

Table III contains examples of expected cumulative rewards for two-arm bandits for various priors and time horizons.

VI. EVALUATING ALGORITHMS WITH THE OPTIMAL STRATEGY AS A GAUGE

Given a method of determining the OPT, next we evaluate the effectiveness of existing bandit algorithms. We assume all arm priors are the same in a single experiment.

Question 4: How far are commonly used bandit algorithms from optimality?

Tool:[4]. To evaluate bandit algorithms, we developed a comprehensive multithreaded simulator using C++ object-oriented programming paradigm. It can simulate various arm priors and various algorithms (players). A player chooses an arm, pulls it, and receives a reward, logged by a monitor. Monitors report various statistics, e.g., reward means and their standard deviations or regrets as functions of time. Implementation of a player requires overloading the choose function. This simulator architecture is easy to test and maintain, which translates into confidence in the correctness of results.

Experiments:[4] We performed a large series of simulation studies for algorithms listed in Sections II-B and II-C, and compared their results with the OPT. We examined them in 2-, 3-, 10-, and 15-arm contexts with various priors. Simulation results were averaged across 1 million runs for each algorithm. This

[4]Data and tools at https://github.com/SebastianPilarski/Bernoulli_bandits and at https://codeocean.com/capsule/5291971/tree

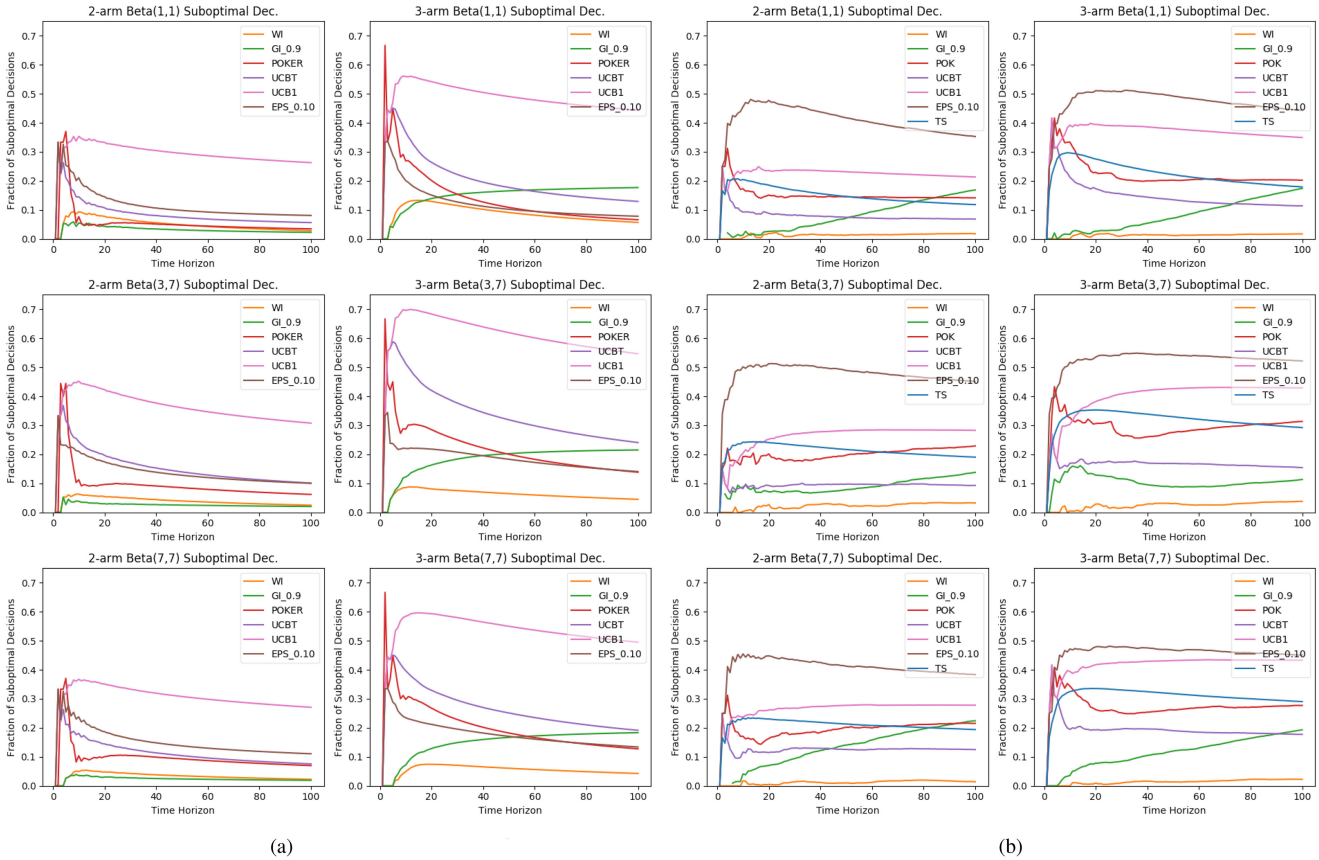


Fig. 6. Decision suboptimality. (a) Fraction of suboptimal decisions for all configurations. (b) Rate of suboptimal decisions per game.

means each point (each time horizon) in a plot for a finite-time horizon algorithm represents average over one million games played specifically for the point's time horizon. Infinite-time horizon algorithms (no dependence on time horizon) require only experiments for the largest time horizon shown. A sample graph for a specific two vaccine trial is presented in Appendix B.

Labels. Note that EPS_0.10 symbolizes EPS with $\epsilon = 0.10$. Labels for other algorithms follow the same convention.

A. Suboptimality of Decisions

We begin our gauging of different algorithms by comparing the number of suboptimal (not optimal) decisions an algorithm makes. We determine the *fraction of suboptimal decisions* across the entire possible configuration space and the *rate of suboptimal decisions*, i.e., the fraction of suboptimal decisions each algorithm makes in a given game.

Fraction of Suboptimal Decisions: There exists a unique OPT table for each time horizon and prior Beta distribution. When computing an OPT one can compare the optimal decision at each configuration with the decisions other algorithms would make. Fig. 6(a) plots the fraction of suboptimal decisions across all configurations for various algorithms for different combinations of Beta priors and time horizons.

Rate of Suboptimal Decisions: While playing, configurations may have various probabilities of being reached; many configurations may never be visited. We determine the rate of suboptimal decisions by having an algorithm play an entire game. At each time step the decision made by the algorithm is compared with the decision the OPT would have made. Fig. 6(b) presents results.

Observation 6: None of the examined algorithms matches exactly the OPT for all time horizons. GI and WI appear to match OPT for several small time horizons H.[5]

B. Performance

Not all suboptimal decisions have equal consequences. Even if making certain suboptimal decisions, an algorithm can still effectively balance exploitation versus exploration. Fig. 7 gauges the performance of various algorithms versus OPTs in terms of *cumulative rewards* and *cumulative regret*.

The relative performance of the algorithms compared in Fig. 7 appears correlated but not predicted by the fraction of suboptimal decisions plots in Fig. 6(a). For example, UCB1 is by far the worst by fraction of suboptimal decisions, yet sometimes it outperforms EPS_0.10 in terms of cumulative mean and regret. This is, in part, because success/failure paths leading to some decisions are much less likely than others. Rates of suboptimal decisions per game [see Fig. 6(b)] are much more correlated to

[5]Gittins Theorem on optimality [17] does not apply under our assumptions.

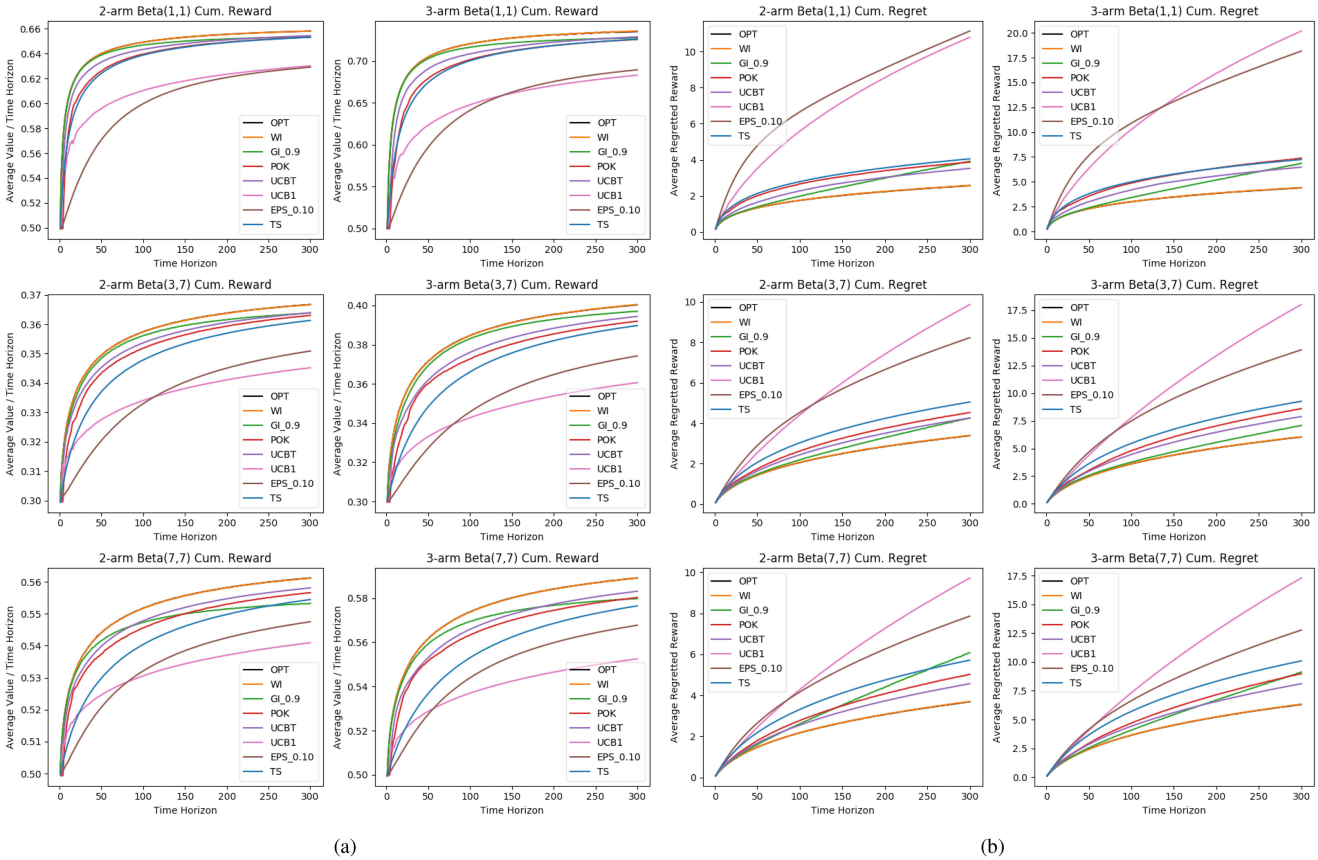


Fig. 7. Gauging algorithms: two-arms and three-arms. WI plots overlap those for the OPT. (a) Normalized cum. mean reward versus time horizon. (b) Cumulative regret versus time horizon.

TABLE IV
NORMALIZED CUM. REWARDS, WI, BETA(3,7) LOSSES TO OPT AVERAGED OVER 100 MILLION GAMES

	Time Horizon				
	40	50	90	150	400
2-Arms	4.7e-05	4.1e-05	5.9e-05	4.7e-05	3.9e-05
3-Arms	3.1e-05	6.3e-05	5.9e-05	6.0e-05	4.6e-05
4-Arms	5.5e-05	6.2e-05	7.9e-05	7.7e-05	
5-Arms	4.4e-05	6.1e-05	6.6e-05		
6-Arms	6.4e-05	8.0e-05			
7-Arms	4.3e-05	4.8e-05			
8-Arms	6.1e-05				
9-Arms	9.0e-05				
10-Arms	7.1e-05				

results in Fig. 7, but the degree of decision suboptimality affects the cumulative reward.

Observation 7: Among suboptimal algorithms, WI performed best by far. In both Fig. 7(a) and (b), the plot covers the OPT plot.

In all our simulation experiments, WI results are always very close to those for the OPT. Table IV presents WI simulation data for various numbers of arms and time horizons. It shows cumulative reward loss with respect to expected OPT results. These data relate to that in Table II. Differences computed for other prior Beta distributions look similar.

Observation 8: The losses of WI wrt. OPT in normalized cumulative rewards were always small—less than 10^{-4}*—regardless of prior Beta distribution.*

We also investigated the relative cumulative regret performance of algorithms for 10 and 15 arms with results presented in Fig. 8. WI performed best.

VII. OPTIMIZING FOR BERNOULLI

Not all suboptimal algorithms presented within this article were specifically optimized for Bernoulli bandits or nonuniform priors. Various plots in Section VI presented performance of selected algorithms with their commonly used parameters, e.g., $\epsilon = 0.1$ in ϵ-greedy and constant $c = 1.0$ for UCB1 and UCBT [see (3) and (4)]. In this section, we modify suboptimal algorithms and parameters to improve their performance in Bernoulli bandits with Beta priors.

Clearly ϵ-greedy and UCB1 are the weakest algorithms, but they are the most "blind" algorithms. Both collect elementary (s_i, f_i) statistics for each arm i; both use $\hat{\mu}_i$ to make decisions; UCB1 takes advantage of $n_i = s_i + f_i$ and is often preferred over ϵ-greedy in practice. UCBT, which performs far better in general, looks also at $\hat{\sigma}_i$. Clearly, using more information about reward distribution pays off.

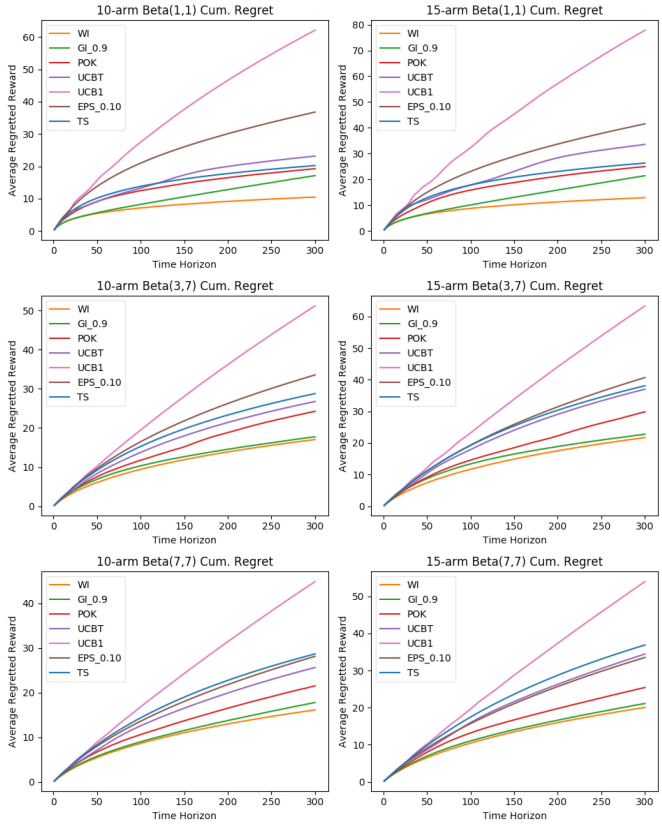


Fig. 8. Cumulative regret versus time horizon: 10-arm and 15-arm. WI is always best.

WI performs better than other algorithms. In particular WI performs better than GI as it is aware of the time horizon and uses this information to optimize decisions, but as we will see changing the discount factor in GI can turn it into a very competitive performer. Both algorithms use very detailed information about Beta reward distribution, namely Property 1 and (1), for each decision. Hence, we may pose the following question.

Question 5: How much can the performance of default EPS, UCB1, UCBT, POKER, and GI algorithms be improved by using Beta(α, β)-specific properties?

1) ϵ-Greedy (EPS): The obvious modification is to apply (1) and add priors to the observed mean, i.e., use $\hat{\mu}_i(t) = \frac{s_i(t)+\alpha_i}{n_i(t)+\alpha_i+\beta_i}$, this helps somewhat if arm probabilities are drawn from distributions with different expected values.
2) UCB1: The same modification to $\hat{\mu}_i(t)$ as in EPS can be applied to UCB1. While mostly ignored in the literature, UCB1's parameter c can be tuned. Fig. 9 illustrates the sensitivity of cumulative reward to the value of constant c for a two arm bandit with selected arm probabilities. Note that tacitly assumed $c = 1$ is not the best choice.
3) UCBT: As in UCB1, we modify $\hat{\mu}_i(t)$ to be $\hat{\mu}_i(t) = \frac{s_i(t)+\alpha_i}{n_i(t)+\alpha_i+\beta_i}$. Additionally, we substitute $\hat{\sigma}_i^2(t)$ with the expression of (2). As these changes are significant, we refer to this *optimized for Bernoulli* variant as UCBT_ob. To explore more optimization options, we generalize constant $\frac{1}{4}$ in the original choose*(t)* function to a parameter d.

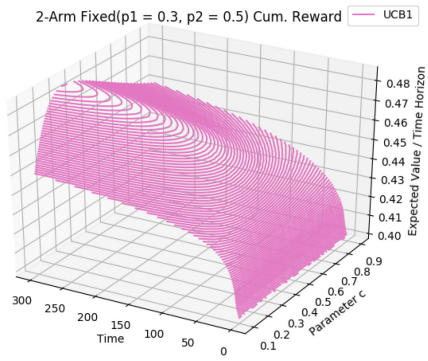


Fig. 9. Norm. cum. reward for UCB1 as a function of c, fixed arm probs. Commonly assumed $c = 1$ is not the best choice.

The following formally define UCBT_ob:

$$choose(t) = \underset{i=1,\ldots,k}{\operatorname{argmax}} \left(\frac{s'_i(t)}{n'_i(t)} + c\sqrt{\frac{\ln T}{n'_i(t)} \min\left(d, V_i(t)\right)} \right)$$

$$V_i(t) = \frac{s'_i(t)\, f'_i(t)}{(n'_i(t))^2(n'_i(t)+1)} + \sqrt{\frac{2\ln T}{n'_i(t)}}$$

$$s'_i(t) = s_i(t) + \alpha_i$$

$$f'_i(t) = f_i(t) + \beta_i$$

$$n'_i(t) = s'_i(t) + f'_i(t)$$

$$T = t + \sum_{i=1}^{k} (\alpha_i + \beta_i).$$

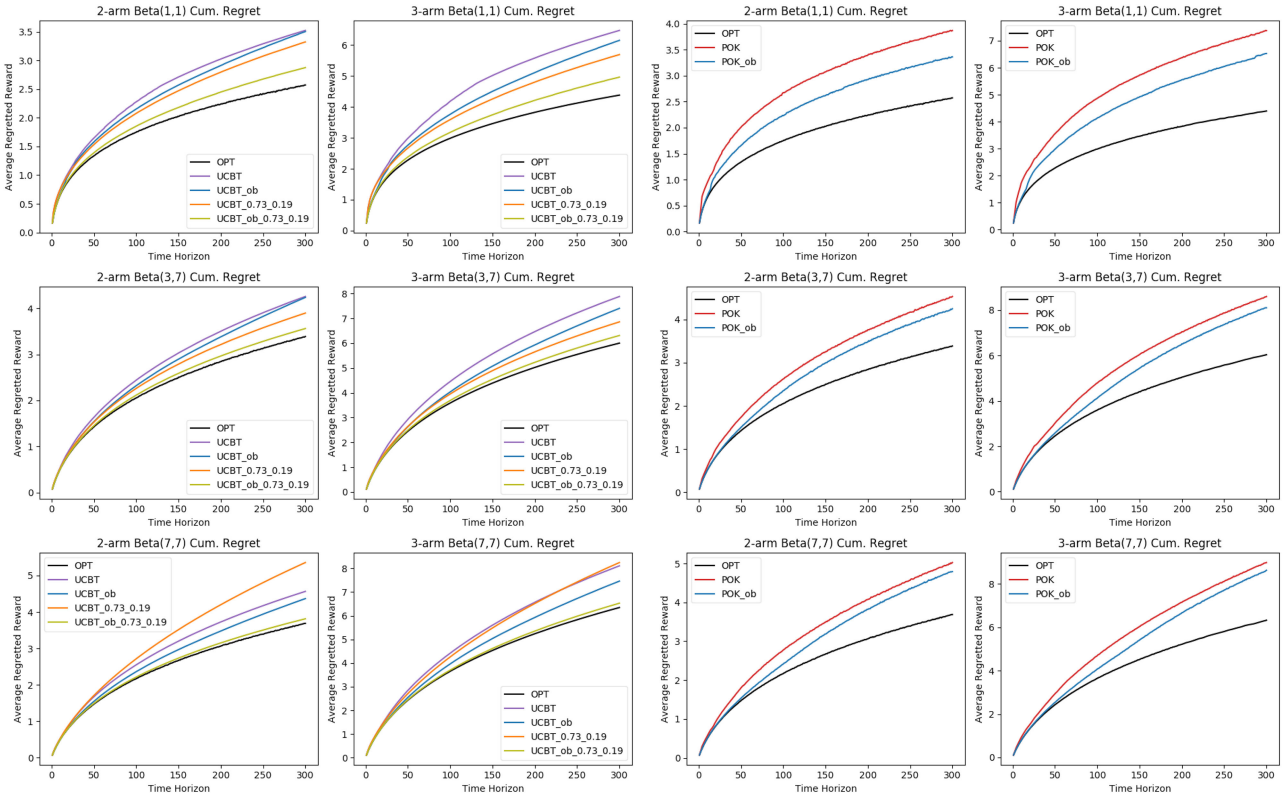
By investigating various combinations of c and d for UCBT_ob with extensive simulation, we found that using $c = 0.73$ and $d = 0.19$ is a good choice regardless of the number of arms and prior Beta distributions, although not necessarily optimal.

Fig. 10(a) showcases significant improvements achieved by our modifications along this parameterization. Note that the same parameter values applied to the traditional UCBT variant do not exhibit the same improvement over default values [especially for two arm, Beta(7,7)].

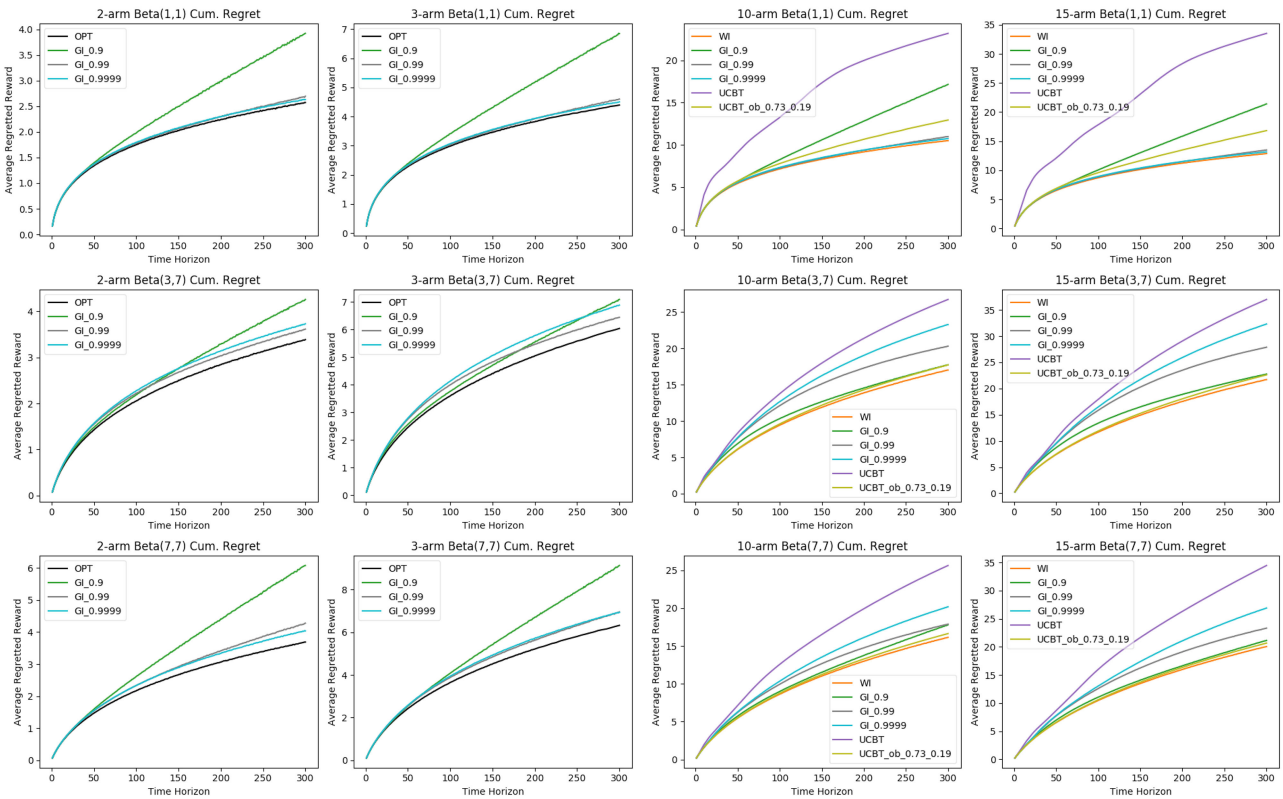
Fig. 10(d) demonstrates that the improvements are even more significant for bandits with more arms.

Observation 9: For nonuniform priors UCBT_ob *with $c = 0.73, d = 0.19$ (*see UCBT_ob_0.73_0.19*) performs better than any other infinite-time horizon algorithm in the simulations performed and presented.*

4) POKER: Variant POKER_ob modifies $\hat{\mu}_i(t)$ and $\hat{\sigma}_i^2(t)$ as in UCBT_ob. Fig. 10(b) shows noticeable improvements attributed to our algorithm modification.
5) GI: GI optimizations are limited to selecting a discount factor. Fig. 10(c) shows how this parameter affects cumulative regret when two or three arms are concerned. Plots in Fig. 8 present GI_0.9 as the second best algorithm after WI. Fig. 10(d) compares GI with three



(a) (b)



(c) (d)

Fig. 10. Optimization for Bernoulli. UCBT_ob_0.73_0.19 offers significant improvements to UCBT. (a) Cumulative regret versus time horizon: two arm and three arm, UCBT. (b) Cumulative regret versus time horizon: two arm and three arm, POKER. (c) Cumulative regret versus time horizon: two arm and three arm, GI. (d) Cumulative regret versus time horizon: 10 arm and 15 arm, ob.

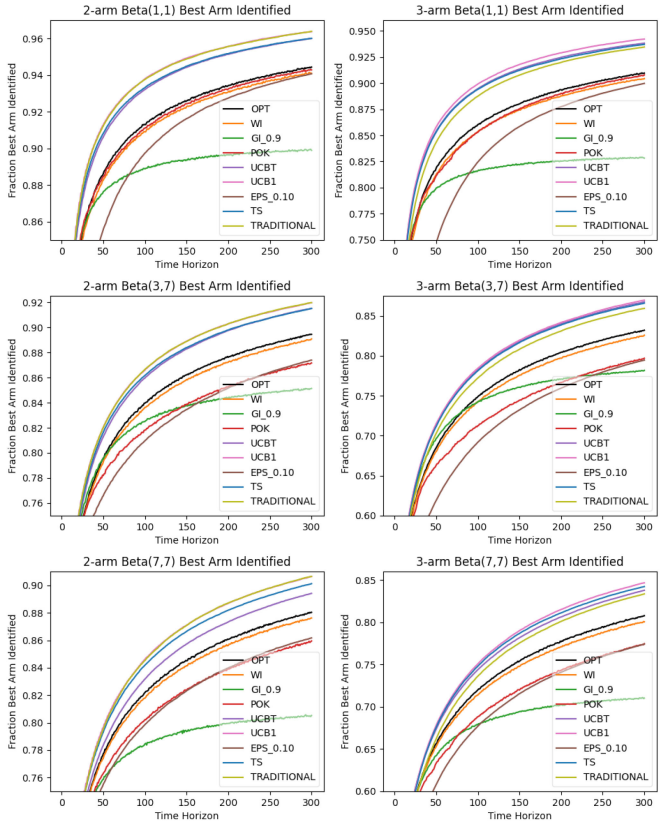


Fig. 11. Fraction of games where best arm is correctly identified. On the left, TRADITIONAL overlaps UCB1. Overall UCB1 produced best results.

discount factors, WI, and UCBT_ob_0.73_0.19. Surprisingly, UCBT_ob_0.73_0.19 outperforms GI regardless of its discount factor if prior Beta distribution is not uniform. Also surprisingly, GI discount factor of 0.9 is the best choice for nonuniform Beta distributions, while it is by far the worst choice for Beta(1,1).

VIII. Best Arm Identification

Unlike regret, best arm identification is not a common goal of multi-armed bandit research. In some applications, however, it may be of vital importance. In the context of clinical trials, the best arm is synonymous to the best drug or treatment. Thus, it may be imperative to answer the following question.

Question 6: Which algorithm is most likely to identify the best arm?

While some publications propose algorithms that seek the best arm within the fewest arm pulls possible [31], in this article, we attempt to identify the best algorithms for a given time horizon in the context of Bernoulli bandits. To answer Question 6, we examine how often the empirical best arm and the true best arm are the same at $t = H$. This is different than the best arm sampling frequency discussed in [12]. Fig. 11 presents simulation results for several common algorithms. The results may be surprising. In two-arm simulations, the traditional randomized clinical trial and UCB1 are most effective and practically equivalent (plots overlap); the OPT is less effective. In three-arm simulations, UCB1 is clearly the best, while the traditional trial algorithm is noticeably less effective. As the number of arms increases, UCBT surpasses UCB1 and the relative effectiveness of the traditional clinical trial algorithm decreases. Since the traditional clinical trial algorithm is inferior in terms of regret, we observe the following.

Observation 10: If the main objective is best arm identification, UCB1 or UCBT is at least as effective as traditional randomized clinical trials. Both offer significantly smaller regret. TS is only marginally less effective than UCB1, but its regret is substantially smaller.

IX. Conclusion

This article presents a novel method to compute the OPT for a Bernoulli bandit. Its core idea is to use a 1-D array with entries indexed by an easy to compute function that optimizes required storage by orders of magnitude when compared to a naive straightforward implementation. *This method makes it possible to compute OPT tables for time horizons and number of arms well beyond what was considered feasible in other publications.* For example, the feasible time horizon for a two-arm Bernoulli bandit has been increased by a factor of 30–50×. The proposed indexing scheme already significantly reduces computation time and also lends itself to multithreading.

Using this method, we computed the reachability and regularity of entries in various OPT tables. Our computations show that the already reduced-in-size OPT tables can be compressed by orders of magnitude. *This article demonstrates that the OPT can be deployed in many situations where suboptimal algorithms are currently used.*

Expected cumulative reward computed for an OPT constitutes an upper bound on the mean cumulative reward of any Bernoulli bandit algorithm. We used it to gauge several well-known algorithms. *Our comprehensive simulations show that WI performs at nearly optimal level.* GI can be tuned to perform very well for uniform priors, but the UCBT_ob algorithm proposed in this article performs better for nonuniform Beta priors. It should be noted that GI discount factors that perform very well for uniform priors often are not the best choices for nonuniform priors and vice versa. The already well performing UCBT_ob can probably be improved even further by employing time-varying parameters, or parameters that depend on the number of arms and targeted time horizon.

Nonuniform priors are scarce in theory and simulations. Yet it is unrealistic to assume that most applications are dealing with processes where success rates are uniformly distributed. This article attempts to narrow the gap by presenting the most comprehensive simulation study of nonuniform priors that we are aware of and the only comprehensive comparison of various common algorithms versus the OPT.

Finally, when investigating a seldom addressed topic in multi-armed bandit research, we demonstrated that UCB1, UCBT, and TS identify the best arm with better probabilities than the OPT. This is a noteworthy observation for clinical trials if the best treatment needs to be identified with the utmost importance. In this context, UCB1 matches the traditional randomized clinical

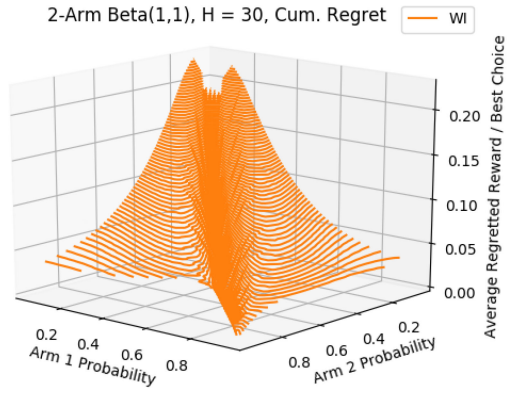


Fig. 12. Relative cumulative regret for W_I, $H = 30$, $\alpha = \beta = 1$. The plot shows relative price of nearly optimal exploration.

trial algorithm when two treatments are considered, and is a better choice with increasing number of evaluated treatments. Taking into account its much smaller regret, UCB1 can be considered superior to the traditional clinical trial algorithm. TS is almost as good as UCB1 at best arm identification and its regret is smaller. *Regardless of the objectives, Bernoulli bandits are always the better option than the traditional randomized clinical trial algorithm.*

Appendix A
Algorithm Specifics

GI: The main idea is as follows. Consider playing just one arm with an unknown probability of success. For any state of the game, it is possible to replace the arm with an arm of known probability if it improves expected reward. The goal is to find the smallest known probability p for any state of the game, which is the value of GI.

As a necessary assumption, we assume that expected reward of an infinite game is finite. A constant discount factor for future rewards ($d < 1$) can be used for this purpose. Consequently, GI can be expressed using the following equations:

$$V_d^*(s,f,p) = \max\left\{\frac{p}{1-d}, \frac{s}{s+f}\left(1 + d\cdot V_d^*(s+1,f,p)\right) + \frac{f}{s+f}\left(0 + d\cdot V_d^*(s,f+1,p)\right)\right\}$$

$$G_{I_d}(s,f) = \min p : \frac{p}{1-d} \geq \frac{s}{s+f}\left(1 + d\cdot V_d^*(s+1,f,p)\right) + \frac{f}{s+f}\left(0 + d\cdot V_d^*(s,f+1,p)\right).$$

Computing GI has been analyzed in [27]. Villar *et al.* [5] contain a link to MATLAB code that computes GI using dynamic programming. It assumes that for sufficiently large $s+f$, $V_d^*(s,f,p) \approx \frac{s}{s+f}$ and gradually increases p. Note that $\frac{s}{s+f}$ in equations mentioned above relates to (1).

WI: The WI replaces infinite geometric series in GI equations with its finite version and considers that reward is 0 after the last pull results in the following equations

$$V_{d,H}^*(s,f,p,0) = 0$$

$$V_{d,H}^*(s,f,p,j) = \max\left\{p\sum_{i=0}^{j-1} d^i, \frac{s}{s+f}\left(1 + d\cdot V_{d,H}^*(s+1,f,p,j-1)\right) + \frac{f}{s+f}\left(0 + d\cdot V_{d,H}^*(s,f+1,p,j-1)\right)\right\}, \quad j = 1,\ldots,H$$

$$W_I(s,f,j) = \min p : p\sum_{i=0}^{j-1} d^i \geq \frac{s}{s+f}\left(1 + d\cdot V_{d,H}^*(s+1,f,p,j-1)\right) + \frac{f}{s+f}\left(0 + d\cdot V_{d,H}^*(s,f+1,p,j-1)\right)$$

where j represents the remaining playing time, and we assume $d = 1$ in this article.

POKER: Before an arm is chosen, expected reward improvement $\delta_\mu(t)$ and price of knowledge $p_i(t)$ for each arm are defined as follows. $\delta_\mu(t) = \mathbb{E}[\mu^* - \mu_m]$, $m = \text{argmax}_{i=1,\ldots,k}(\hat{\mu}_i(t))$
$p_i(t) = \hat{\mu}_i(t) + \mathbb{P}[\mu_i > \mu_m + \delta_\mu(t)]\delta_\mu(t)(H-t)$
The second term in the definition of $p_i(t)$ is sometimes referred to as exploration bonus [32].

Since the value of μ^* is unknown to the player, before an arm is pulled, an approximation of $\delta_\mu(t)$, and an approximation of $p_i(t)$ for each arm are computed. Let $Q(t) = \{i : 1 \leq i \leq k,\ n_i(t) > 0\}$
$\delta_\mu(t) \approx \frac{\text{argmax}_{i\in Q}(\hat{\mu}_i(t)) - \text{argmin}_{i\in Q}(\hat{\mu}_i(t))}{\sqrt{|Q|}}$.

Assuming that $\hat{\mu}_i(t)$ follows the normal distribution and $\mu_m \approx \hat{\mu}^*(t)$ we get
$\mathbb{P}[\mu_i > \mu_m + \delta_\mu(t)] \approx \{\int_{\hat{\mu}^*(t)+\delta_\mu(t)}^{\infty}\}\mathcal{N}(x, \hat{\mu}_i(t), \frac{\hat{\sigma}_i(t)}{\sqrt{n_i(t)}})dx$.

If $n_i(t) = 0$, the algorithm assumes that $\hat{\mu}_i(t)$ and $\hat{\sigma}_i(t)$ can be approximated by means of such values computed for arms that have already been pulled.

Appendix B
Contextualizing Performance

Exploration Price: Comparing cumulative regret plots for a prior Beta distribution does not reveal how much is lost relative to always choosing the best arm. Fig. 12 shows a 3-D plot, which illustrates the fraction of the best possible expected reward lost due to exploration. Small, but different, arm probabilities translate into relatively high costs of exploration. Of course, equal arm probabilities mean zero exploration cost. WI results in this plot are practically optimal.

Two Vaccines: Comparing cumulative mean rewards for a prior Beta distribution does not reveal the scope of expected

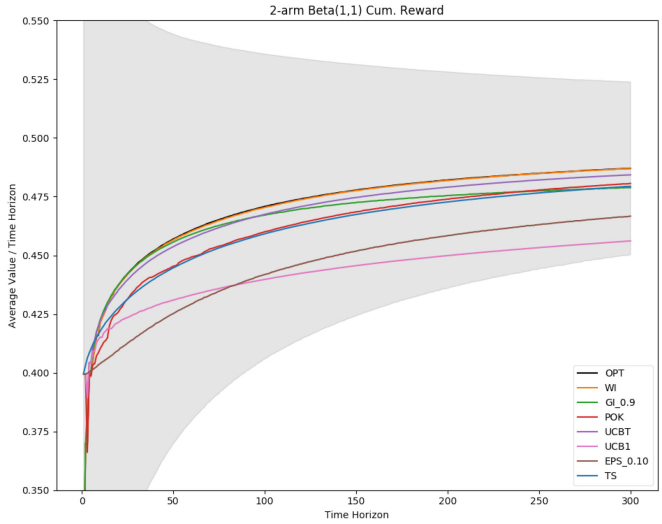


Fig. 13. Clinical trial example. Cumulative rewards, two arms, treatment success probabilities 0.3 and 0.5. Grey area shows standard deviation of the OPT results.

results for a particular set of drawn arm probabilities. Fig. 13 shows an example of expected cumulative means for two arms with success probabilities 0.3 and 0.5. *This would correspond to two vaccines with probabilities of successful treatment of 0.3 and 0.5.* The black line and shaded area show the OPT's cumulative mean and its standard deviation. The figure illustrates how likely it is for a single OPT game to produce cumulative reward below mean cumulative rewards of other algorithms. All algorithms' cumulative means fall within the OPT's standard deviation.

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.

[2] E. Kaufmann, O. Cappé, and A. Garivier, "On the complexity of a/b testing," in *Proc. Conf. Learn. Theory*, 2014, pp. 461–481.

[3] Y. Mao, M. Chen, A. Wagle, J. Pan, M. Natkovich, and D. Matheson, "A batched multi-armed bandit approach to news headline testing," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, 2018, pp. 1966–1973.

[4] S. Katariya, B. Kveton, C. Szepesvári, C. Vernade, and Z. Wen, "Bernoulli rank-1 bandits for click feedback," in *Proc. 26th Int. Joint Conf. Artif. Intell.*, 2017, pp. 2001–2007.

[5] S. S. Villar, J. Bowden, and J. Wason, "Multi-armed bandit models for the optimal design of clinical trials: Benefits and challenges," *Statist. Sci.*, vol. 30, no. 2, pp. 199–215, May 2015.

[6] S. S. Villar, "Bandit strategies evaluated in the context of clinical trials in rare life-threatening diseases," *Probability Eng. Informational Sci.*, vol. 32, no. 2, pp. 229–245, 2018.

[7] D. A. Berry and B. Fristedt, *Bandit Problems: Sequential Allocation of Experiments (Monographs on Statistics and Applied Probability)*, London, U.K.: Chapman and Hall, 1985.

[8] D. A. Berry, "Modified two-armed bandit strategies for certain clinical trials," *J. Amer. Statist. Assoc.*, vol. 73, no. 362, pp. 339–345, 1978.

[9] Y. Cheng, F. Su, and D. A. Berry, "Choosing sample size for a clinical trial using decision analysis," *Biometrika*, vol. 90, no. 4, pp. 923–936, 2003.

[10] T. Friede *et al.* "Recent advances in methodology for clinical trials in small populations: The inspire project," *Orphanet J. Rare Diseases*, vol. 13, no. 1, pp. 186–186, Oct. 2018.

[11] V. Kuleshov and D. Precup, "Algorithms for multi-armed bandit problems," *J. Mach. Learn. Res.*, vol. 1, Feb. 2014. [Online]. Available: https://www.researchgate.net/publication/260367055_Algorithms_for_multi-armed_bandit_problems

[12] G. Burtini, J. Loeppky, and R. Lawrence, "A survey of online experiment design with the stochastic multi-armed bandit," 2015, *arXiv:1510.00757*.

[13] T. Lattimore and C. Szepesvári, *Bandit Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 2020.

[14] L. Zhou, "A survey on contextual multi-armed bandits," *CoRR*, vol. abs/1508.03326, 2015.

[15] O. Chapelle and L. Li, "An empirical evaluation of Thompson sampling," in *Proc. 24th Int. Conf. Adv. Neural Inf. Process. Syst.*, 2011, pp. 2249–2257.

[16] K. Ronoh, R. Oyamo, E. Milgo, M. Drugan, and B. Manderick, "Bernoulli bandits an empirical comparison," in *Proc. 23rd Eur. Symp. Artif. Neural Netw., Comput. Intell. Mach. Learn.*, 2015, pp. 59–64.

[17] J. C. Gittins, "Bandit processes and dynamic allocation indices," *J. Roy. Statist. Society: Ser. B. (Methodological)*, vol. 41, no. 2, pp. 148–164, 1979.

[18] P. Whittle, "Restless bandits: Activity allocation in a changing world," *J. Appl. Probability*, vol. 25, pp. 287–298, 1988.

[19] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multi-armed bandit problem," *Mach. Learn.*, vol. 47, no. 2/3, pp. 235–256, 2002.

[20] S. Bhulai and G. Koole, "On the value of learning for Bernoulli bandits with unknown parameters," *IEEE Trans. Autom. Control*, vol. 45, no. 11, pp. 2135–2140, Nov. 2000.

[21] T. L. Lai, "Adaptive treatment allocation and the multi-armed bandit problem," *Ann. Statist.*, vol. 15, pp. 1091–1114, 1987.

[22] J. Doob, *Stochastic Processes, (Probability and Statistics)*. Hoboken, NJ, USA: Wiley, 1953.

[23] J. M. Bernardo and A. F. Smith, *Bayesian Theory*, vol. 405. Hoboken, NJ, USA: Wiley, 2009.

[24] W. R. Thompson, "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples," *Biometrika*, vol. 25, no. 3/4, pp. 285–294, 1933.

[25] W. R. Thompson, "On the theory of apportionment," *Amer. J. Math.*, vol. 57, no. 2, pp. 450–456, 1935.

[26] E. Kaufmann, N. Korda, and R. Munos, "Thompson sampling: An asymptotically optimal finite-time analysis," in *Proc. Int. Conf. Algorithmic Learn. Theory*, 2012, pp. 199–213.

[27] J. Chakravorty and A. Mahajan, "Multi-armed bandits, Gittins index, and its calculation," in *Methods and Applications of Statistics in Clinical Trials: Planning, Analysis, and Inferential Methods*, vol. 2, Hoboken, NJ, USA: Wiley, 2014, pp. 416–435.

[28] J. Vermorel and M. Mohri, "Multi-armed bandit algorithms and empirical evaluation," in *Proc. 16th Eur. Conf. Mach. Learn.*, 2005, pp. 437–448.

[29] R. Bellman, *The Theory of Dynamic Programming*. Santa Monica, CA, USA: Rand Corp, 1954.

[30] J. Kleinberg and E. Tardos, *Algorithm Design*. Reading, MA, USA: Addison-Wesley, 2005.

[31] K. Jamieson and R. Nowak, "Best-arm identification algorithms for multi-armed bandits in the fixed confidence setting," in *Proc. 48th Annu. Conf. Inf. Sci. Syst.*, 2014, pp. 1–6.

[32] N. Meuleau and P. Bourgine, "Exploration of multi-state environments: Local measures and back-propagation of uncertainty," *Mach. Learn.*, vol. 35, no. 2, pp. 117–154, 1999.

[33] S. Pilarski *et al.* "Bernoulli Bandit Data and Tools Repository." [Online]. Available: https://github.com/SebastianPilarski/Bernoulli_bandits

[34] S. Pilarski *et al.*, "Bernoulli Bandit Code Capsule," doi: 10.24433/CO.1300732.v1.

Sebastian Pilarski (Student Member, IEEE) is currently working toward the Ph.D. degree with the Department of Electrical and Computer Engineering, McGill University, Montreal, QC, Canada.

His research interests include software engineering, machine learning, reinforcement learning, and their applications to systems engineering. He has been investigating applications of artificial intelligence in gas turbine design and control with Siemens Energy.

Slawomir Pilarski (Member, IEEE) received the M.Sc. degree from the Warsaw University of Technology, Poland, and the Ph.D. degree from the Institute of Electron Technology, Warsaw, Poland.

He was a Tenured Professor with Simon Fraser University, Burnaby, BC, Canada. He held Director-level positions with Synopsys, Mountain View, CA, USA, and Magma DA (Magma), San Jose, CA, USA, where he led advanced formal verification R&D teams. He founded two startups—one acquired by Magma. His work on built-in self-test of VLSI circuits was recognized by IEEE Design and Test of Computers as a milestone in test technology and was deployed by several major chip manufacturing companies. He developed core algorithms and architected two industry-leading formal verification tools including second-generation Formality—an equivalence checker. He authored or coauthored more than 40 research papers and an IEEE monograph. His research publication topics range from computer architecture, various aspects of VLSI design, distributed databases, formal verification, and logic synthesis to theoretical computer science and reinforcement learning. He coauthored three patents in three different research domains.

Dániel Varró received the Ph.D. degree from the Budapest University of Technology and Economics.

He is a Full Professor with McGill University, Montreal, QC, Canada. He is a co-author of more than 170 scientific papers with seven Distinguished Paper Awards, and three Most Influential Paper Awards. He serves on the Editorial Board of Software and Systems Modeling and Journal of Object Technology periodicals, and served as a Program Co-Chair of MODELS 2021, SLE 2016, ICMT 2014, FASE 2013 conferences. He delivered keynote talks at numerous conferences (include CSMR, SOFSEM, and SAM) and international summer schools. He is a Co-Founder of the VIATRA open source model query and transformation framework, and IncQuery Labs, a technology-intensive Hungarian company.