Algorithm Selection as a Bandit Problem with Unbounded Losses

Matteo Gagliolo[1,2] and Jürgen Schmidhuber[1,2,3]

[1] IDSIA, Galleria 2, 6928 Manno (Lugano), Switzerland
[2] University of Lugano, Faculty of Informatics, Via Buffi 13, 6904 Lugano, Switzerland
[3] TU Munich, Boltzmannstr. 3, 85748 Garching, München, Germany
{matteo,juergen}@idsia.ch

Abstract. Algorithm selection is typically based on models of algorithm performance learned during a separate *offline* training sequence, which can be prohibitively expensive. In recent work, we adopted an *online* approach, in which a performance model is iteratively updated and used to guide selection on a sequence of problem instances. The resulting *exploration-exploitation* trade-off was represented as a bandit problem with expert advice, using an existing solver for this game, but this required the setting of an arbitrary bound on algorithm runtimes, thus invalidating the optimal regret of the solver. In this paper, we propose a simpler framework for representing algorithm selection as a bandit problem, with partial information, and an *unknown* bound on losses. We adapt an existing solver to this game, proving a bound on its expected regret, which holds also for the resulting algorithm selection technique. We present experiments with a set of SAT solvers on a mixed SAT-UNSAT benchmark.

1 Introduction

Decades of research in the fields of Machine Learning and Artificial Intelligence brought us a variety of alternative algorithms for solving many kinds of problems. Algorithms often display variability in performance quality, and computational cost, depending on the particular problem instance being solved: in other words, there is no single "best" algorithm. While a "trial and error" approach is still the most popular, attempts to automate algorithm selection are not new [1], and have grown to form a consistent and dynamic field of research in the area of *Meta-Learning* [2]. Many selection methods follow an *offline* learning scheme, in which the availability of a large training set of performance data for the different algorithms is assumed. This data is used to learn a model that maps (*problem*, *algorithm*) pairs to expected performance, or to some probability distribution on performance. The model is later used to select and run, for each new problem instance, only the algorithm that is expected to give the best results. While this approach might sound reasonable, it actually ignores the computational cost of the initial training phase: collecting a representative sample of performance data has to be done via solving a set of training problem instances, and each instance is solved repeatedly, at least once for each of the available algorithms, or more if the algorithms are randomized. Furthermore, these training instances are assumed to be representative of future ones, as the model is not updated after training.

In other words, there is an obvious trade-off between the *exploration* of algorithm performances on different problem instances, aimed at learning the model, and the *exploitation* of the best algorithm/problem combinations, based on the model's predictions. This trade-off is typically ignored in offline algorithm selection, and the size of the training set is chosen heuristically. In our previous work [3,4,5], we have kept an *online* view of algorithm selection, in which the only input available to the meta-learner is a set of algorithms, of unknown performance, and a sequence of problem instances that have to be solved. Rather than artificially subdividing the problem set into a training and a test set, we iteratively update the model each time an instance is solved, and use it to guide algorithm selection on the next instance.

Bandit problems [6] offer a solid theoretical framework for dealing with exploration-exploitation trade-offs in an online setting. One important obstacle to the straightforward application of a bandit problem solver to algorithm selection is that most existing solvers assume a bound on losses to be available beforehand. In [7,5] we dealt with this issue heuristically, fixing the bound in advance. In this paper, we introduce a modification of an existing bandit problem solver [8], which allows it to deal with an unknown bound on losses, while retaining a bound on the expected regret. This allows us to propose a simpler version of the algorithm selection framework GAMBLETA, originally introduced in [5]. The result is a parameterless online algorithm selection method, with a provable upper bound on regret.

The rest of the paper is organized as follows. Section 2 describes a tentative taxonomy of algorithm selection methods, along with a few examples from literature. Section 3 presents our framework for representing algorithm selection as a bandit problem, discussing the introduction of a higher level of selection among different algorithm selection techniques (*time allocators*). Section 4 introduces the modified bandit problem solver for unbounded loss games, along with its bound on regret. Section 5 describes experiments with SAT solvers. Section 6 concludes the paper.

2 Related Work

In general terms, algorithm selection can be defined as the process of allocating computational resources to a set of alternative algorithms[1], in order to improve some measure of performance on a set of problem instances. Algorithm selection techniques can be further described according to different orthogonal features:

Decision vs. optimization problems. A first distinction needs to be made among *decision* problems, where a binary criterion for recognizing a solution is available; and *optimization* problems, where different levels of solution quality can be attained, measured by an *objective* function [9]. Literature on algorithm selection is often focused on one of these two classes of problems. The selection is normally aimed at minimizing solution time for decision problems; and at maximizing performance quality, or improving some speed-quality trade-off, for optimization problems.

[1] The algorithm set may also contain multiple copies of a same algorithm, differing in their parameter settings; or even identical randomized algorithms differing only in their random seeds.

Per set vs. per instance selection. The selection among different algorithms can be performed once for an entire set of problem instances (*per set* selection, following [10]); or repeated for each instance (*per instance* selection).

Static vs. dynamic selection. A further independent distinction [11] can be made among *static* algorithm selection, in which allocation of resources precedes algorithm execution; and *dynamic*, or *reactive*, algorithm selection, in which the allocation can be adapted during algorithm execution.

Oblivious vs. non-oblivious selection. In *oblivious* techniques, algorithm selection is performed from scratch for each problem instance; in *non-oblivious* techniques, there is some knowledge transfer across subsequent problem instances, usually in the form of a *model* of algorithm performance.

Off-line vs. online learning. Non-oblivious techniques can be further distinguished as *offline* or *batch* learning techniques, where a separate training phase is performed, after which the selection criteria are kept fixed; and *online* techniques, where the criteria can be updated every time an instance is solved.

A seminal paper in the field of algorithm selection is [1], in which offline, per instance selection is first proposed, for both decision and optimization problems. More recently, similar concepts have been proposed, with different terminology (algorithm *recommendation*, *ranking*, *model selection*), in the *Meta-Learning* community [12,2,13]. Research in this field usually deals with optimization problems, and is focused on maximizing solution quality, without taking into account the computational aspect. Work on *Empirical Hardness Models* [14,15] is instead applied to decision problems, and focuses on obtaining accurate models of runtime performance, conditioned on numerous features of the problem instances, as well as on parameters of the solvers [10]. The models are used to perform algorithm selection on a per instance basis, and are learned offline: online selection is advocated in [10]. Literature on algorithm portfolios [16,17,18] is usually focused on choice criteria for building the set of candidate solvers, such that their areas of good performance do not overlap, and optimal static allocation of computational resources among elements of the portfolio.

A number of interesting dynamic exceptions to the static selection paradigm have been proposed recently. In [19], algorithm performance modeling is based on the behavior of the candidate algorithms during a predefined amount of time, called the *observational horizon*, and dynamic context-sensitive restart policies for SAT solvers are presented. In both cases, the model is learned offline. In a Reinforcement Learning [20] setting, algorithm selection can be formulated as a Markov Decision Process: in [21], the algorithm set includes sequences of recursive algorithms, formed dynamically at run-time solving a sequential decision problem, and a variation of Q-learning is used to find a dynamic algorithm selection policy; the resulting technique is per instance, dynamic and online. In [11], a set of deterministic algorithms is considered, and, under some limitations, static and dynamic schedules are obtained, based on dynamic programming. In both cases, the method presented is per set, offline.

An approach based on runtime distributions can be found in [22,23], for parallel independent processes and shared resources respectively. The runtime distributions are assumed to be known, and the expected value of a cost function, accounting for both wall-clock time and resources usage, is minimized. A task switching schedule is evaluated offline, using a branch-and-bound algorithm to find the optimal one in a tree of possible schedules. Unfortunately, the computational complexity of the tree search grows exponentially in the number of processes.

Finding a per set optimal task switching schedule is proved to be NP-hard in [24]. Based on this work, [25] propose a greedy 4-approximation of the optimal schedule, and use it as a basis for an online algorithm. [26] introduces per instance selection, based on discrete features.

"Low-knowledge" oblivious approaches can be found in [27,28], in which various simple indicators of current solution improvement are used for algorithm selection, in order to achieve the best solution quality within a given time contract. In [28], the selection process is dynamic: machine time shares are based on a recency-weighted average of performance improvements. We adopted a similar approach in [3], where we considered algorithms with a scalar state, that had to reach a target value. The time to solution was estimated based on a shifting-window linear extrapolation of the learning curves.

For optimization problems, if selection is only aimed at maximizing solution quality, the same problem instance can be solved multiple times, keeping only the best solution. In this case, algorithm selection can be represented as a *Max* K-armed bandit problem, a variant of the game in which the reward attributed to each arm is the maximum payoff on a set of rounds. Solvers for this game are used in [29,30] to implement oblivious per instance selection from a set of multi-start optimization techniques: multiple runs of the available solvers are allocated, to maximize solution quality on a single problem instance.

Further references can be found in [31,5].

3 Algorithm Selection as a Bandit Problem

In its most basic form [32], the *multi-armed bandit* problem is faced by a gambler, playing a sequence of trials against an N-armed slot machine. At each trial, the gambler chooses one of the available arms, whose losses are randomly generated from different *stationary* distributions. The gambler incurs in the corresponding loss, and, in the *full information* game, she can observe the losses that would have been paid pulling any of the other arms. A more optimistic formulation can be made in terms of positive rewards. The aim of the game is to minimize the *regret*, defined as the difference between the cumulative loss of the gambler, and the one of the best arm. A bandit problem solver (BPS) can be described as a mapping from the history of the losses observed so far, to a probability distribution $\mathbf{p} = (p_1, ..., p_N)$ over the N arms, which will be used to pick an arm at the subsequent trial.

More recently, the original restricting assumptions have been progressively relaxed, allowing for *non-stationary* loss distributions, *partial* information (only the loss for the pulled arm is observed), and *adversarial* bandits that can set their losses in order to deceive the player. In [33,6], a reward game is considered, and no statistical assumptions are made about the process generating the rewards, which are allowed to be an arbitrary function of the entire history of the game (*non-oblivious* adversarial setting). Based on these pessimistic hypotheses, the authors describe probabilistic gambling strategies for the full and the partial information games.

Let us now see how to represent algorithm selection for *decision* problems as a bandit problem, with the aim of minimizing solution time. Consider a sequence $\mathcal{B} = \{b_1, \dots, b_M\}$ of M instances of a decision problem, for which we want to minimize solution time, and a set of K algorithms $\mathcal{A} = \{a_1, \dots, a_K\}$, such that each b_m can be solved by each a_k. It is straightforward to describe static algorithm selection in a multi-armed bandit setting, where "pick arm k" means "run algorithm a_k on next problem instance". Runtimes t_k can be viewed as losses, generated by a rather complex mechanism, i.e., the algorithms a_k themselves, running on the current problem. The information is partial, as the runtime for other algorithms is not available, unless we decide to solve the same problem instance again. In a worst case scenario one can receive a "deceptive" problem sequence, starting with problem instances on which the performance of the algorithms is misleading, so this bandit problem should be considered adversarial. As BPS typically minimize the regret with respect to a single arm, this approach would allow to implement *per set* selection, of the overall best algorithm. An example can be found in [7], where we presented an online method for learning a per set estimate of an optimal restart strategy.

Unfortunately, per set selection is only profitable if one of the algorithms dominates the others on all problem instances. This is usually not the case: it is often observed in practice that different algorithms perform better on different problem instances. In this situation, a per instance selection scheme, which can take a different decision for each problem instance, can have a great advantage.

One possible way of exploiting the nice theoretical properties of a BPS in the context of algorithm selection, while allowing for the improvement in performance of per instance selection, is to use the BPS at an upper level, to select among alternative algorithm selection techniques. Consider again the algorithm selection problem represented by $\mathcal{B}$ and $\mathcal{A}$. Introduce a set of N *time allocators* (TA$_j$) [3,5]. Each TA$_j$ can be an arbitrary function, mapping the current history of collected performance data for each a_k, to a share $\mathbf{s}^{(j)} \in [0,1]^K$, with $\sum_{k=1}^{K} s_k = 1$. A TA is used to solve a given problem instance executing all algorithms in $\mathcal{A}$ in parallel, on a single machine, whose computational resources are allocated to each a_k proportionally to the corresponding s_k, such that for any portion of time spent t, $s_k t$ is used by a_k, as in a *static* algorithm portfolio [16]. The runtime before a solution is found is then $\min_k\{t_k/s_k\}$, t_k being the runtime of algorithm a_k.

A trivial example of a TA is the *uniform* time allocator, assigning a constant $\mathbf{s} = (1/K, ..., 1/K)$. Single algorithm selection can be represented in this framework by setting a single s_k to 1. Dynamic allocators will produce a time-varying share $\mathbf{s}(t)$. In previous work, we presented examples of heuristic oblivious [3] and non-oblivious [4] allocators; more sound TAs are proposed in [5], based on non-parametric models of the runtime distribution of the algorithms, which are used to minimize the expected value of solution time, or a *quantile* of this quantity, or to maximize solution probability within a give time *contract*.

At this higher level, one can use a BPS to select among different time allocators, TA$_j$, TA$_2$..., working on a same algorithm set $\mathcal{A}$. In this case, "pick arm j" means

Algorithm 1. GAMBLETA $(\mathcal{A}, \mathcal{T}, BPS)$ Gambling Time Allocator.

```
Algorithm set $\mathcal{A}$ with $K$ algorithms;
A set $\mathcal{T}$ of $N$ time allocators TA$_j$;
A bandit problem solver BPS
$M$ problem instances.

initialize BPS $(N, M)$
for each problem $b_i, i = 1, \ldots, M$ do
    pick time allocator $I(i) = j$ with probability $p_j(i)$ from BPS.
    solve problem $b_i$ using TA$_I$ on $\mathcal{A}$
    incur loss $l_{I(i)} = \min_k\{t_k(i)/s_k^{(I)}(i)\}$
    update BPS
end for
```

"use time allocator TA$_j$ on $\mathcal{A}$ to solve next problem instance". In the long term, the BPS would allow to select, on a *per set* basis, the TA$_j$ that is best at allocating time to algorithms in $\mathcal{A}$ on a *per instance* basis. The resulting "Gambling" Time Allocator (GAMBLETA) is described in Alg. 1.

If BPS allows for non-stationary arms, it can also deal with time allocators that are *learning* to allocate time. This is actually the original motivation for adopting this two-level selection scheme, as it allows to combine in a principled way the exploration of algorithm behavior, which can be represented by the uniform time allocator, and the exploitation of this information by a model-based allocator, whose model is being learned online, based on results on the sequence of problems met so far. If more time allocators are available, they can be made to compete, using the BPS to explore their performances. Another interesting feature of this selection scheme is that the initial requirement that each algorithm should be capable of solving each problem can be relaxed, requiring instead that at least one of the a_k can solve a given b_m, and that each TA$_j$ can solve each b_m: this can be ensured in practice by imposing a $s_k > 0$ for all a_k. This allows to use interesting combinations of complete and incomplete solvers in $\mathcal{A}$ (see Sect. 5). Note that any bound on the regret of the BPS will determine a bound on the regret of GAMBLETA with respect to the best time allocator. Nothing can be said about the performance w.r.t. the best algorithm. In a worst-case setting, if none of the time allocator is effective, a bound can still be obtained by including the uniform share in the set of TAs. In practice, though, per-instance selection can be much more efficient than uniform allocation, and the literature is full of examples of time allocators which eventually converge to a good performance.

The original version of GAMBLETA (GAMBLETA4 in the following) [5] was based on a more complex alternative, inspired by the bandit problem with *expert* advice, as described in [33,6]. In that setting, two games are going on in parallel: at a lower level, a partial information game is played, based on the probability distribution obtained *mixing* the advice of different *experts*, represented as probability distributions on the K arms. The experts can be arbitrary functions of the history of observed rewards, and give a different advice for each trial. At a higher level, a *full information* game is played, with the N experts playing the roles of the different arms. The probability distribution $\mathbf{p}$ at this level is not used to pick a single expert, but to *mix* their advises, in order to generate the distribution for the lower level arms. In GAMBLETA4, the time allocators play the role of the experts, each suggesting a different $\mathbf{s}$, on a per instance basis; and the arms of the lower level game are the K algorithms, to be run in parallel with the mixture share. EXP4 [33,6] is used as the BPS. Unfortunately, the bounds for EXP4 cannot be extended to GAMBLETA4 in a straightforward manner, as the loss function itself is not convex; moreover, EXP4 cannot deal with unbounded losses, so we had to adopt an heuristic reward attribution instead of using the plain runtimes.

A common issue of the above approaches is the difficulty of setting reasonable upper bounds on the time required by the algorithms. This renders a straightforward application of most BPS problematic, as a known bound on losses is usually assumed, and used to tune parameters of the solver. Underestimating this bound can invalidate the bounds on regret, while overestimating it can produce an excessively "cautious" algorithm, with a poor performance. Setting in advance a good bound is particularly difficult when dealing with algorithm runtimes, which can easily exhibit variations of several order of magnitudes among different problem instances, or even among different runs on a same instance [34].

Some interesting results regarding games with *unbounded* losses have recently been obtained. In [8,35], the authors consider a full information game, and provide two algorithms which can adapt to unknown bounds on signed rewards. Based on this work, [36] provide a Hannan consistent algorithm for losses whose bound grows in the number of trials i with a known rate i^ν, $\nu < 1/2$. This latter hypothesis does not fit well our situation, as we would like to avoid any restriction on the sequence of problems: a very hard instance can be met first, followed by an easy one. In this sense, the hypothesis of a constant, but unknown, bound is more suited: in GAMBLETA, this unkown bound would correspond to the worst performance of the worst time allocator. In [8], Cesa-Bianchi *et al.* also introduce an algorithm for loss games with partial information (EXP3LIGHT), which requires losses to be bound, and is particularly effective when the cumulative loss of the best arm is small. In the next section we introduce a variation of this algorithm that allows it to deal with an unknown bound on losses.

4 An Algorithm for Games with an Unknown Bound on Losses

Here and in the following, we consider a partial information game with N arms, and M trials; an index (i) indicates the value of a quantity used or observed at trial $i \in \{1, \dots, M\}$; j indicate quantities related to the j-th arm, $j \in \{1, \dots, N\}$; index E refers to the loss incurred by the bandit problem solver, and $I(i)$ indicates the arm chosen at trial (i), so it is a discrete random variable with value in $\{1, \dots, N\}$; r, u will represent quantities related to an *epoch* of the game, which consists of a sequence of 0 or more consecutive trials; $\log$ with no index is the natural logarithm.

EXP3LIGHT [8, Sec. 4] is a solver for the bandit loss game with partial information. It is a modified version of the weighted majority algorithm [37], in which the cumulative

Algorithm 2. EXP3LIGHT $(N, M, \mathcal{L})$ A solver for bandit problems with partial information and a known bound $\mathcal{L}$ on losses.

```
$N$ arms, $M$ trials
losses $l_j(i) \in [0, \mathcal{L}]\ \forall\, i = 1, ..., M, j = 1, \dots, N$
initialize epoch $r = 0$, $L_E = 0$, $\tilde{L}_j(0) = 0$.
initialize $\eta_r$ according to (1)
for each trial $i = 1, ..., M$ do
    set $p_j(i) \propto \exp(-\eta_r \tilde{L}_j(i-1)/\mathcal{L})$, $\sum_{j=1}^{N} p_j(i) = 1$.
    pick arm $I(i) = j$ with probability $p_j(i)$.
    incur loss $l_E(i) = l_{I(i)}(i)$.
    evaluate unbiased loss estimates:
    $\tilde{l}_{I(i)}(i) = l_{I(i)}(i)/p_{I(i)}(i)$, $\tilde{l}_j = 0$ for $j \neq I(i)$
    update cumulative losses:
    $L_E(i) = L_E(i-1) + l_E(i)$,
    $\tilde{L}_j(i) = \tilde{L}_j(i-1) + \tilde{l}_j(i)$, for $j = 1, \dots, N$
    $\tilde{L}^*(i) = min_j \tilde{L}_j(i)$.
    if $(\tilde{L}^*(i)/\mathcal{L}) > 4^r$ then
        start next epoch $r = \lceil \log_4(\tilde{L}^*(i)/\mathcal{L}) \rceil$
        update $\eta_r$ according to (1)
    end if
end for
```

losses for each arm are obtained through an unbiased estimate[2]. The game consists of a sequence of epochs $r = 0, 1, \dots$: in each epoch, the probability distribution over the arms is updated, proportional to $\exp(-\eta_r \tilde{L}_j)$, $\tilde{L}_j$ being the current unbiased estimate of the cumulative loss. Assuming an upper bound 4^r on the smallest loss estimate, η_r is set as:

$$\eta_r = \sqrt{\frac{2(\log N + N \log M)}{(N 4^r)}} \tag{1}$$

When this bound is first trespassed, a new epoch starts and r and η_r are updated accordingly.

The original algorithm assumes losses in $[0, 1]$. We first consider a game with a known finite bound $\mathcal{L}$ on losses, and introduce a slightly modified version of EXP3LIGHT (Algorithm 2), obtained simply dividing all losses by $\mathcal{L}$. Based on Theorem 5 from [8], the following is trivial to prove:

Theorem 1. *If $L^*(M)$ is the loss of the best arm after M trials, and $L_E(M) = \sum_{i=1}^{M} l_{I(i)}(i)$ is the loss of* EXP3LIGHT $(N, M, \mathcal{L})$*, the expected value of its regret is bounded as:*

[2] For a given round, and a given arm with loss l and pull probability p, the estimated loss $\tilde{l}$ is l/p if the arm is pulled, 0 otherwise. This estimate is unbiased in the sense that its expected value, with respect to the process extracting the arm to be pulled, equals the actual value of the loss: $E\{\tilde{l}\} = pl/p + (1-p)0 = l$.

Algorithm 3. EXP3LIGHT-A (N, M) A solver for bandit problems with partial information and an unknown (but finite) bound on losses.

```
$N$ arms, $M$ trials,
losses $l_j(i) \in [0, \mathcal{L}]\ \forall\, i = 1, ..., M, j = 1, \dots, N$
**unknown** $\mathcal{L} < \infty$
initialize epoch $u = 0$, EXP3LIGHT $(N, M, 2^u)$
**for** each trial $i = 1, ..., M$ **do**
    pick arm $I(i) = j$ with probability $p_j(i)$ from EXP3LIGHT.
    incur loss $l_E(i) = l_{I(i)}(i)$.
    **if** $l_{I(i)}(i) > 2^u$ **then**
        start next epoch $u = \lceil \log_2 l_{I(i)}(i) \rceil$
        **restart** EXP3LIGHT $(N, M - i, 2^u)$
    **end if**
**end for**
```

$$\begin{aligned} & E\{L_E(M)\} - L^*(M) \\ & \leq 2\sqrt{6\mathcal{L}(\log N + N\log M)NL^*(M)} \\ & + \mathcal{L}[2\sqrt{2\mathcal{L}(\log N + N\log M)N} \\ & + (2N+1)(1+\log_4(3M+1))]. \end{aligned} \tag{2}$$

We now introduce a simple variation of Algorithm 2 which does not require the knowledge of the bound $\mathcal{L}$ on losses, and uses Algorithm 2 as a subroutine.

EXP3LIGHT-A (Algorithm 3) is inspired by the doubling trick used in [8] for a *full* information game with unknown bound on losses. The game is again organized in a sequence of epochs $u = 0, 1, \dots$: in each epoch, Algorithm 2 is *restarted*, resetting all loss estimates, and using a bound $\mathcal{L}_u = 2^u$; a new epoch is started with the appropriate u whenever a loss larger than the current $\mathcal{L}_u$ is observed.

Theorem 2. *If $L^*(M)$ is the loss of the best arm after M trials, and $\mathcal{L} < \infty$ is the unknown bound on losses, the expected value of the regret of* EXP3LIGHT-A (N, M) *is bounded as:*

$$\begin{aligned} & E\{L_E(M)\} - L^*(M) \leq \\ & 4\sqrt{3\lceil \log_2 \mathcal{L}\rceil \mathcal{L}(\log N + N\log M)NL^*(M)} \\ & + 2\lceil \log_2 \mathcal{L}\rceil \mathcal{L}[\sqrt{4\mathcal{L}(\log N + N\log M)N} \\ & + (2N+1)(1+\log_4(3M+1)) + 2] \end{aligned} \tag{3}$$

The proof is given in the appendix. The regret obtained by EXP3LIGHT-A is $O(\sqrt{\mathcal{L}N\log(M)L^*(M)})$, which can be useful in a situation in which $\mathcal{L}$ is high but L^* is relatively small, as we expect in our time allocation setting if the algorithms exhibit huge variations in runtime, but at least one of the TAs eventually converges to a good performance. We can then use EXP3LIGHT-A as a BPS for selecting among different time allocators in GAMBLETA (Algorithm 1).

5 Experiments

The set of time allocator used in the following experiments is the same as in [5], and includes the uniform allocator, along with nine other *dynamic* allocators, optimizing different quantiles of runtime, based on a nonparametric model of the runtime distribution that is updated after each problem is solved. We first briefly describe these time allocators, inviting the reader to refer to [5] for further details and a deeper discussion. A separate model $F_k(t|\mathbf{x})$, conditioned on features $\mathbf{x}$ of the problem instance, is used for each algorithm a_k. Based on these models, the runtime distribution for the whole algorithm portfolio $\mathcal{A}$ can be evaluated for an arbitrary share $\mathbf{s} \in [0,1]^K$, with $\sum_{k=1}^{K} s_k = 1$, as

$$F_{\mathcal{A},\mathbf{s}}(t) = 1 - \prod_{k=1}^{K} [1 - F_k(s_k t)]. \tag{4}$$

Eq. (4) can be used to evaluate a quantile $t_{\mathcal{A},\mathbf{s}}(\alpha) = F^{-1}_{\mathcal{A},\mathbf{s}}(\alpha)$ for a given solution probability α. Fixing this value, time is allocated using the share that minimizes the quantile

$$\mathbf{s} = \arg\min_{\mathbf{s}} F^{-1}_{\mathcal{A},\mathbf{s}}(\alpha). \tag{5}$$

Compared to minimizing expected runtime, this time allocator has the advantage of being applicable even when the runtime distributions are improper, i. e. $F(\infty) < 1$, as in the case of incomplete solvers. A *dynamic* version of this time allocator is obtained updating the share value periodically, conditioning each F_k on the time spent so far by the corresponding a_k; allocation of multiple CPUs is considered in [38].

Rather than fixing an arbitrary α, we used nine different instances of this time allocator, with α ranging from 0.1 to 0.9, in addition to the uniform allocator, and let the BPS select the best one.

We present experiments for the algorithm selection scenario from [5], in which a local search and a complete SAT solver (respectively, G2-WSAT [39] and Satz-Rand [34]) are combined to solve a sequence of random satisfiable and unsatisfiable problems (benchmarks `uf-*`, `uu-*` from [40], 1899 instances in total). As the clauses-to-variable ratio is fixed in this benchmark, only the number of variables, ranging from 20 to 250, was used as a problem feature $\mathbf{x}$. Local search algorithms are more efficient on satisfiable instances, but cannot prove unsatisfiability, so are doomed to run forever on unsatisfiable instances; while complete solvers are guaranteed to terminate their execution on all instances, as they can also prove unsatisfiability.

For the whole problem sequence, the overhead of GAMBLETA3 (Algorithm 1, using EXP3LIGHT-A as the BPS) over an ideal "oracle", which can predict and run only the fastest algorithm, is 22%. GAMBLETA4 (from [5], based on EXP4) seems to profit from the mixing of time allocation shares, obtaining a better 14%. Satz-Rand alone can solve all the problems, but with an overhead of about 40% w.r.t. the oracle, due to its poor performance on satisfiable instances. Fig. 1 plots the evolution of cumulative time, and cumulative overhead, along the problem sequence.

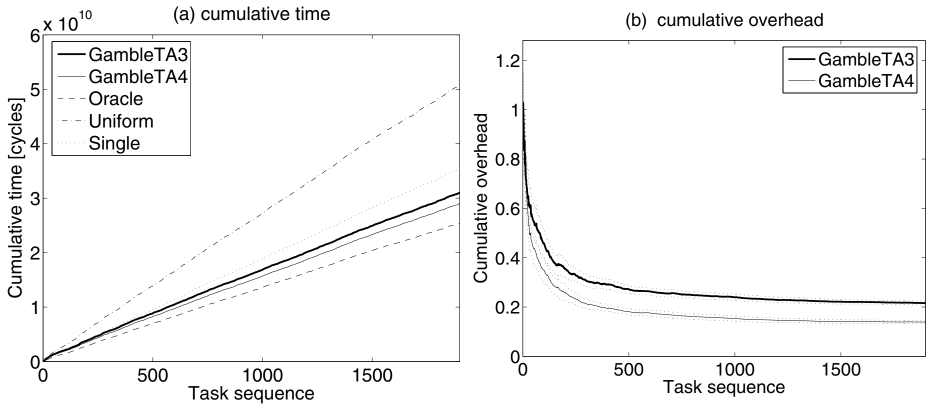

Fig. 1. (a): Cumulative time spent by GAMBLETA3 and GAMBLETA4 [5] on the SAT-UNSAT problem set ($10^9 \approx 1$ min.). Upper 95% confidence bounds on 20 runs, with random reordering of the problems. ORACLE is the lower bound on performance. UNIFORM is the (0.5,0.5) share. SATZ-RAND is the per-set best algorithm. (b): The evolution of cumulative overhead, defined as $(\sum_j t_G(j) - \sum_j t_O(j))/\sum_j t_O(j)$, where t_G is the performance of GAMBLETA and t_O is the performance of the oracle. Dotted lines represent 95% confidence bounds.

6 Conclusions

We introduced EXP3LIGHT-A, a bandit problem solver for loss games with partial information and an unknown bound on losses, extending the work of [8]. Based on this, we proposed a simpler version of GAMBLETA [5], an online algorithm portfolio selection method. The use of EXP3LIGHT-A avoids the setting of any additional parameter, and provides a bound on regret with respect to the best element from the set of time allocators. The choices of the algorithm set, and of the time allocators to use, are still left to the user. Any existing algorithm (portfolio) selection technique, including oblivious ones, can be included in the set of N allocators, with an impact $O(\sqrt{N})$ on the regret: the overall performance of GAMBLETA is guaranteed to converge to the one of the best time allocator.

We also presented preliminary experiments, observing a slight degradation in performance compared to the heuristic version of GAMBLETA presented in [5] (based on EXP4), which requires to set a maximum runtime in advance, and cannot be provided of a bound on regret.

According to [41], a bound for the original EXP3LIGHT can be proved for an adaptive η_r (1), in which the total number of trials M is replaced by the current trial i. This should allow for a potentially more efficient variation of EXP3LIGHT-A, in which EXP3LIGHT is not restarted at each epoch, and can retain the information on past losses.

Acknowledgments. We would like to thank Nicolò Cesa-Bianchi for contributing the proofs for EXP3LIGHT and useful remarks on his work, and Faustino Gomez for his comments on a draft of this paper. This work was supported by the Hasler foundation with grant n. 2244.

References

1. Rice, J.R.: The algorithm selection problem. In: Rubinoff, M., Yovits, M.C. (eds.) Advances in Computers, vol. 15, pp. 65–118. Academic Press, New York (1976)
2. Vilalta, R., Drissi, Y.: A perspective view and survey of meta-learning. Artif. Intell. Rev. 18(2), 77–95 (2002)
3. Gagliolo, M., Zhumatiy, V., Schmidhuber, J.: Adaptive online time allocation to search algorithms. In: Boulicaut, J.-F., Esposito, F., Giannotti, F., Pedreschi, D. (eds.) ECML 2004. LNCS (LNAI), vol. 3201, pp. 134–143. Springer, Heidelberg (2004)
4. Gagliolo, M., Schmidhuber, J.: A neural network model for inter-problem adaptive online time allocation. In: Duch, W., Kacprzyk, J., Oja, E., Zadrożny, S. (eds.) ICANN 2005. LNCS, vol. 3697, pp. 7–12. Springer, Heidelberg (2005)
5. Gagliolo, M., Schmidhuber, J.: Learning dynamic algorithm portfolios. Annals of Mathematics and Artificial Intelligence 47(3-4), 295–328 (2006); AI&MATH 2006 Special Issue
6. Auer, P., Cesa-Bianchi, N., Freund, Y., Schapire, R.E.: The nonstochastic multiarmed bandit problem. SIAM J. Comput. 32(1), 48–77 (2003)
7. Gagliolo, M., Schmidhuber, J.: Learning restart strategies. In: Veloso, M.M. (ed.) IJCAI 2007 – Twentieth International Joint Conference on Artificial Intelligence, January 2007, vol. 1, pp. 792–797. AAAI Press, Menlo Park (2007)
8. Cesa-Bianchi, N., Mansour, Y., Stoltz, G.: Improved second-order bounds for prediction with expert advice. In: Auer, P., Meir, R. (eds.) COLT 2005. LNCS (LNAI), vol. 3559, pp. 217–232. Springer, Heidelberg (2005)
9. Hoos, H.H., Stützle, T.: Local search algorithms for SAT: An empirical evaluation. Journal of Automated Reasoning 24(4), 421–481 (2000)
10. Hutter, F., Hamadi, Y.: Parameter adjustment based on performance prediction: Towards an instance-aware problem solver. Technical Report MSR-TR-2005-125, Microsoft Research, Cambridge, UK (December 2005)
11. Petrik, M.: Statistically optimal combination of algorithms. Presented at SOFSEM 2005 - 31st Annual Conference on Current Trends in Theory and Practice of Informatics (2005)
12. Fürnkranz, J.: On-line bibliography on meta-learning. In: EU ESPRIT METAL Project (26.357): A Meta-Learning Assistant for Providing User Support in Machine Learning Mining (2001)
13. Giraud-Carrier, C., Vilalta, R., Brazdil, P.: Introduction to the special issue on meta-learning. Machine Learning 54(3), 187–193 (2004)
14. Leyton-Brown, K., Nudelman, E., Shoham, Y.: Learning the empirical hardness of optimization problems: The case of combinatorial auctions. In: Van Hentenryck, P. (ed.) CP 2002. LNCS, vol. 2470, p. 556. Springer, Heidelberg (2002)
15. Nudelman, E., Leyton-Brown, K., Hoos, H.H., Devkar, A., Shoham, Y.: Understanding random sat: Beyond the clauses-to-variables ratio. In: Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, pp. 438–452. Springer, Heidelberg (2004)
16. Huberman, B.A., Lukose, R.M., Hogg, T.: An economic approach to hard computational problems. Science 275, 51–54 (1997)
17. Gomes, C.P., Selman, B.: Algorithm portfolios. Artificial Intelligence 126(1-2), 43–62 (2001)
18. Petrik, M., Zilberstein, S.: Learning static parallel portfolios of algorithms. In: Ninth International Symposium on Artificial Intelligence and Mathematics (2006)
19. Kautz, H.A., Horvitz, E., Ruan, Y., Gomes, C.P., Selman, B.: Dynamic restart policies. In: AAAI/IAAI, pp. 674–681 (2002)
20. Sutton, R., Barto, A.: Reinforcement learning: An introduction. MIT Press, Cambridge (1998)
21. Lagoudakis, M.G., Littman, M.L.: Algorithm selection using reinforcement learning. In: Proc. 17th ICML, pp. 511–518. Morgan Kaufmann, San Francisco (2000)

22. Finkelstein, L., Markovitch, S., Rivlin, E.: Optimal schedules for parallelizing anytime algorithms: the case of independent processes. In: Eighteenth national conference on Artificial intelligence, pp. 719–724. AAAI Press, Menlo Park (2002)
23. Finkelstein, L., Markovitch, S., Rivlin, E.: Optimal schedules for parallelizing anytime algorithms: The case of shared resources. Journal of Artificial Intelligence Research 19, 73–138 (2003)
24. Sayag, T., Fine, S., Mansour, Y.: Combining multiple heuristics. In: Durand, B., Thomas, W. (eds.) STACS 2006. LNCS, vol. 3884, pp. 242–253. Springer, Heidelberg (2006)
25. Streeter, M.J., Golovin, D., Smith, S.F.: Combining multiple heuristics online. In: Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, Vancouver, British Columbia, Canada, July 22-26, pp. 1197–1203. AAAI Press, Menlo Park (2007)
26. Streeter, M., Smith, S.F.: New techniques for algorithm portfolio design. In: UAI 2008: Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence (2008)
27. Beck, C.J., Freuder, E.C.: Simple rules for low-knowledge algorithm selection. In: CPAIOR, pp. 50–64 (2004)
28. Carchrae, T., Beck, J.C.: Applying machine learning to low knowledge control of optimization algorithms. Computational Intelligence 21(4), 373–387 (2005)
29. Cicirello, V.A., Smith, S.F.: The max k-armed bandit: A new model of exploration applied to search heuristic selection. In: Twentieth National Conference on Artificial Intelligence, pp. 1355–1361. AAAI Press, Menlo Park (2005)
30. Streeter, M.J., Smith, S.F.: An asymptotically optimal algorithm for the max k-armed bandit problem. In: Twenty-First National Conference on Artificial Intelligence. AAAI Press, Menlo Park (2006)
31. Smith-Miles, K.A.: Cross-disciplinary perspectives on meta-learning for algorithm selection. ACM Computing Surveys 41(1), 1–25 (2008)
32. Robbins, H.: Some aspects of the sequential design of experiments. Bulletin of the AMS 58, 527–535 (1952)
33. Auer, P., Cesa-Bianchi, N., Freund, Y., Schapire, R.E.: Gambling in a rigged casino: the adversarial multi-armed bandit problem. In: Proceedings of the 36th Annual Symposium on Foundations of Computer Science, pp. 322–331. IEEE Computer Society Press, Los Alamitos (1995)
34. Gomes, C.P., Selman, B., Crato, N., Kautz, H.: Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. J. Autom. Reason. 24(1-2), 67–100 (2000)
35. Cesa-Bianchi, N., Mansour, Y., Stoltz, G.: Improved second-order bounds for prediction with expert advice. Machine Learning 66(2-3), 321–352 (2007)
36. Allenberg, C., Auer, P., Györfi, L., Ottucsák, G.: Hannan consistency in on-line learning in case of unbounded losses under partial monitoring. In: Balcázar, J.L., Long, P.M., Stephan, F. (eds.) ALT 2006. LNCS (LNAI), vol. 4264, pp. 229–243. Springer, Heidelberg (2006)
37. Littlestone, N., Warmuth, M.K.: The weighted majority algorithm. Inf. Comput. 108(2), 212–261 (1994)
38. Gagliolo, M., Schmidhuber, J.: Towards distributed algorithm portfolios. In: Corchado, J.M., et al. (eds.) International Symposium on Distributed Computing and Artificial Intelligence 2008 (DCAI 2008). Advances in Soft Computing, vol. 50, pp. 634–643. Springer, Heidelberg (2008)
39. Li, C.M., Huang, W.: Diversification and determinism in local search for satisfiability. In: Bacchus, F., Walsh, T. (eds.) SAT 2005. LNCS, vol. 3569, pp. 158–172. Springer, Heidelberg (2005)
40. Hoos, H.H., Stützle, T.: SATLIB: An Online Resource for Research on SAT. In: Gent, I.P., et al. (eds.) SAT 2000, pp. 283–292 (2000), `http://www.satlib.org`
41. Cesa-Bianchi, N.: Personal Communication (2008)

Appendix

A.1 Proof of Theorem 2

This follows the proof technique employed in [8, Theorem 4]. Be i_u the last trial of epoch u, i. e. the first trial at which a loss $l_{I(i)}(i) > 2^u$ is observed. Write cumulative losses during an epoch u, *excluding* the last trial i_u, as $L^{(u)} = \sum_{i=i_{u-1}+1}^{i_u-1} l(i)$, and let $L^{*(u)} = \min_j \sum_{i=i_{u-1}+1}^{i_u-1} l_j(i)$ indicate the optimal loss for this subset of trials. Be $U = u(M)$ the *a priori* unknown epoch at the last trial. In each epoch u, the bound (2) holds with $\mathcal{L}_u = 2^u$ for all trials except the last one i_u, so noting that $\log(M-i) \leq \log(M)$ we can write:

$$\begin{aligned} &E\{L_E^{(u)}\} - L^{*(u)} \leq \\ &2\sqrt{6\mathcal{L}_u(\log N + N\log M)NL^{*(u)}} \\ &+ \mathcal{L}_u[2\sqrt{2\mathcal{L}_u(\log N + N\log M)N} \\ &+ (2N+1)(1+\log_4(3M+1))]. \end{aligned} \tag{6}$$

The loss for trial i_u can only be bound by the next value of $\mathcal{L}_u$, evaluated *a posteriori*:

$$E\{l_E(i_u)\} - l^*(i_u) \leq \mathcal{L}_{u+1}, \tag{7}$$

where $l^*(i) = \min_j l_j(i)$ indicates the optimal loss at trial i.

Combining (6,7), and writing $i_{-1} = 0$, $i_U = M$, we obtain the regret for the whole game:[3]

$$\begin{aligned} &E\{L_E(M)\} - \sum_{u=0}^{U} L^{*(u)} - \sum_{u=0}^{U} l^*(i_u) \\ &\leq \sum_{u=0}^{U}\{2\sqrt{6\mathcal{L}_u(\log N + N\log M)NL^{*(u)}} \\ &+ \mathcal{L}_u[2\sqrt{2\mathcal{L}_u(\log N + N\log M)N} \\ &+ (2N+1)(1+\log_4(3M+1))]\} \\ &+ \sum_{u=0}^{U} \mathcal{L}_{u+1}. \end{aligned}$$

The first term on the right hand side of (8) can be bounded using Jensen's inequality

$$\sum_{u=0}^{U} \sqrt{a_u} \leq \sqrt{(U+1)\sum_{u=0}^{U} a_u}, \tag{8}$$

[3] Note that all cumulative losses are counted from trial $i_{u-1}+1$ to trial $i_u - 1$. If an epoch ends on its first trial, (6) is zero, and (7) holds. Writing $i_U = M$ implies the worst case hypothesis that the bound $\mathcal{L}_U$ is exceeded on the last trial. Epoch numbers u are increasing, but not necessarily consecutive: in this case the terms related to the missing epochs are 0.

with

$$\begin{aligned} a_u &= 24\mathcal{L}_u(\log N + N\log M)NL^{*(u)} \\ &\leq 24\mathcal{L}_{U+1}(\log N + N\log M)NL^{*(u)}. \end{aligned} \tag{9}$$

The other terms do not depend on the optimal losses $L^{*(u)}$, and can also be bounded noting that $\mathcal{L}_u \leq \mathcal{L}_{U+1}$.

We now have to bound the number of epochs U. This can be done noting that the maximum observed loss cannot be larger than the unknown, but finite, bound $\mathcal{L}$, and that

$$U + 1 = \lceil \log_2 max_i l_{I(i)}(i) \rceil \leq \lceil \log_2 \mathcal{L} \rceil, \tag{10}$$

which implies

$$\mathcal{L}_{U+1} = 2^{U+1} \leq 2\mathcal{L}. \tag{11}$$

In this way we can bound the sum

$$\sum_{u=0}^{U} \mathcal{L}_{u+1} \leq \sum_{u=0}^{\lceil \log_2 \mathcal{L} \rceil} 2^u \leq 2^{1+\lceil \log_2 \mathcal{L} \rceil} \leq 4\mathcal{L}. \tag{12}$$

We conclude by noting that

$$\begin{aligned} L^*(M) &= min_j L_j(M) \\ &\geq \sum_{u=0}^{U} L^{*(u)} + \sum_{u=0}^{U} l^*(i_u) \geq \sum_{u=0}^{U} L^{*(u)}. \end{aligned} \tag{13}$$

Inequality (8) then becomes:

$$\begin{aligned} &E\{L_E(M)\} - L^*(M) \\ &\leq 2\sqrt{6(U+1)\mathcal{L}_{U+1}(\log N + N\log M)NL^*(M)} \\ &+ (U+1)\mathcal{L}_{U+1}[2\sqrt{2\mathcal{L}_{U+1}(\log N + N\log M)N} \\ &+ (2N+1)(1+\log_4(3M+1))] + 4\mathcal{L}. \end{aligned}$$

Plugging in (10, 11) and rearranging we obtain (3). □