Learning Restart Strategies

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

Abstract

Restart strategies are commonly used for minimizing the computational cost of randomized algorithms, but require prior knowledge of the run-time distribution in order to be effective. We propose a portfolio of two strategies, one fixed, with a provable bound on performance, the other based on a model of run-time distribution, updated as the two strategies are run on a sequence of problems. Computational resources are allocated probabilistically to the two strategies, based on their performances, using a well-known K-armed bandit problem solver. We present bounds on the performance of the resulting technique, and experiments with a satisfiability problem solver, showing rapid convergence to a near-optimal execution time.

Keywords: Randomized algorithms, restart strategies, performance modeling, heavy-tailed distributions, algorithm portfolios, satisfiability, lifelong-learning, adversarial multi-armed bandit problem.

1 Introduction

When trying to communicate across a lossy communication channel with unbounded delays, how long should we wait before resending an unanswered message? In multimodal function optimization using gradient descent, how many steps should we take before starting from a new random initial point? In randomized search over a tree, how deep should we go before abandoning all hopes of finding the goal, and starting the search anew? In similar situations, it is desirable to minimize the time to solve a given problem instance, with a given algorithm, whose run-time is a random variable with a possibly unknown distribution. A *restart strategy* can be used, to generate a sequence of "cutoff" times, at which the algorithm is restarted if unsuccessful.

It has been proved that knowledge of the run-time distribution (RTD) for a given algorithm/problem combination[1] can be used to determine an optimal *uniform* strategy, based on a constant cutoff [Luby *et al.*, 1993]. In the same paper, the authors argue that such a strategy is of little practical interest, as a model of performance is usually unavailable. They then present a *universal* restart strategy, whose performance is worse than the optimal strategy by a logarithmic factor.

These two strategies are based on opposite hypotheses about the availability of the RTD. As a third alternative, a run-time sample for a *set* of problem instances could be collected, and used to learn a *model* of the underlying RTD. There are two potential obstacles to this approach. First, gathering a meaningful sample of run-time data can be computationally expensive, especially in the very case where a restart strategy would be most useful, i. e., when the run-time exhibits huge variations [Gomes *et al.*, 2000]. Second, a given problem set might contain instances with significantly different RTDs: the obtained model would capture the overall behavior of the algorithm on the set, but the corresponding restart strategy would be suboptimal for each instance. Problems met *after* the initial training phase could also be captured poorly by the estimated RTD, again leading to a suboptimal strategy.

Both obstacles are not difficult to circumvent. Training cost can be reduced by using a small *censored* sample of run-time values, obtained aborting runs that exceed some cutoff time. This obviously has an impact on the model's precision, but, the requirements in this sense are less strict than one would expect, and a coarse model already allows to evaluate a near-optimal strategy [Gagliolo and Schmidhuber, 2006b]. The second problem is unavoidable in a *worst case* setting; but in practical situations we do not expect the RTD of each instance of a problem set to be uncorrelated, and a sub-optimal uniform strategy based on the set RTD can still have a great advantage over the large bound of the universal.

In this paper we show how Luby's universal and uniform restart can be effectively interleaved, to solve a set of problem instances. The uniform strategy is based on a non-parametric model of the RTD on the problem set, which is updated every time a new problem instance is solved, in an *online* fashion. The resulting *exploration* vs. *exploitation* tradeoff, is treated as a bandit problem: a solver from [Auer *et al.*, 1995] is used to allocate runs among the two strategies, such that, as the model converges, the uniform strategy is favored if the RTD

[1] In the following, for the sake of readability, we will often refer to the RTD of a problem *instance*, meaning the RTD of different runs of the randomized algorithm of interest on that instance; and the RTD of a problem *set*, meaning the RTD of different runs of the randomized algorithm of interest, each run on a different instance, uniform randomly picked without replacement from the set.

of the set is close to the RTD of most instances, and worst-case bounds on performance are preserved otherwise.

In the following, restart strategies (Sect. 2) are briefly introduced, followed by a discussion of an "adversarial bandit" approach to algorithm selection (Sect. 3), which will be used in our strategy (Sect. 4). We give references to other related work in Sect. 5. Sect. 6 presents experiments with a randomized satisfiability problem solver. Sect. 7 discusses potential impact, and viable improvements, of the presented approach.

2 Optimal restart strategies

A restart strategy consists in executing a sequence of runs of a randomized algorithm, in order to solve a same problem instance, stopping each run r after a time $T(r)$ if no solution is found, and restarting the algorithm with a different random seed; it can be operationally defined by a function $T : \mathbb{N} \to \mathbb{R}^+$ producing the sequence of thresholds $T(r)$ employed. Luby *et al.* [1993] proved that the optimal restart strategy is *uniform*, i. e. , one in which a constant $T(r) = T$ is used to bound each run. They show that in this case the expected value of the total run-time t_T can be evaluated as

$$E(t_T) = \frac{T - \int_0^T F(\tau)d\tau}{F(T)} \tag{1}$$

where $F(t)$ is the cumulative distribution function (CDF) of the run-time for an unbounded run of the algorithm, i. e., a function quantifying the probability that the problem is solved before time t.

If such distribution is known, an optimal cutoff time T^* can be evaluated minimizing (1). Otherwise, the authors suggest a universal non-uniform restart strategy, with cutoff sequence $\{1, 1, 2, 1, 1, 2, 4, 1, ...\}$,[2] proving that its performance t_U is bounded with high probability with respect to the *expected* run-time $E(t_{T^*})$ of the optimal uniform strategy, as

$$t_U <= 192E(t_{T^*})(logE(t_{T^*}) + 5) \tag{2}$$

3 Algorithm selection as a bandit problem

Consider now a sequence $B = \{b_1, ..., b_M\}$ of problem instances, and a set of algorithms $A = \{a_1, a_2, ..., a_K\}$, such that each b_m is solvable by at least one a_k. Without loss of generality, we assume that the execution of each a_k can be subdivided in sequential steps, and indicate with $t_k(r)$, $r = 1, 2, ...$, the duration of the r-th step of the k-th algorithm on the current problem. Each a_k has its own mechanism for determining $t_k(r)$: e. g., for an iterative algorithm, $t_k(r)$ can be the cost of the r-th loop . In this setting, one generally wants to solve the incoming problem instances as fast as possible. This could mean identifying the single best algorithm for the whole set, or for each instance (algorithm *selection* [Rice, 1976]); or, in the more general setting of *algorithm portfolios* [Huberman *et al.*, 1997], the optimal schedule for interleaving the execution of different algorithms, each performing well on different subsets of B.

[2]The sequence is composed of powers of 2: when 2^{j-1} is used twice 2^j is the next. More precisely, $r = 1, 2, ..., T(r) := 2^{j-1}$ if $r = 2^j - 1$; $T(r) := T(r - 2^{j-1} + 1)$ if $2^{j-1} \leq r < 2^j - 1$

In both cases, an obvious trade-off is faced, between *exploration* of the performance of the various a_k, and *exploitation* of the solver(s) estimated to be fastest. A well-known, and well aged, paradigm for addressing such a trade-off is the *multi-armed bandit* problem. In its most basic form [Robbins, 1952], it is faced by a greedy gambler, playing a sequence of trials against a K-armed slot machine, each trial consisting of a choice of one of K available arms, whose rewards are randomly generated from different stationary distributions. The gambler can then receive the corresponding reward, and regret what he would have gained, if only he had pulled any of the other arms. The aim of the game is to minimize this regret, defined as the difference between the expected cumulative reward of the best arm, and the one cashed in by the gambler. Generally speaking, a bandit problem solver (BPS) can be described as a mapping from the entire history of observed rewards x_k for each arm k to a probability distribution $\mathbf{p} = (p_1, ..., p_K)$, from which the choice for the successive trial is picked.

In recent works, the original restricting assumptions have been progressively relaxed. In the *partial information* case, only the reward of the pulled arm is revealed to the gambler. In the *non-oblivious adversarial* setting, the rewards for a given trial can be an arbitrary function of the entire history of the game, and are thus allowed to be deceptive, but have to be set before the current choice is made. Based on these pessimistic hypotheses, Auer *et al.* [1995] devised a probabilistic gambling scheme (**Exp3**), whose regret on a finite number of trials is bounded with high probability as $O(M^{2/3}(K \log K)^{1/3})$, K being the number of arms, M the number of trials (see Alg. 1). The algorithm features a fixed lower bound γ on the exploration probability, and a parameter α controlling the amount of exploitation. Both can be set based on M to obtain the optimal regret rate above.[3]

Algorithm 1 Exp3(M) by Auer et al.

set $\alpha = ((4K \log K)/M)^{1/3}$
set $\gamma = \min\{1, ((K \log K)/(2M))^{1/3}\}$
initialize $s_k := 0$ for $k = 1, ..., K$;
for each trial **do**
  set $p_k \propto (1 + \alpha)^{s_k}$, $\sum_{k=1}^{K} p_k = 1$
  pick arm k with probability $\hat{p}_k := (1 - \gamma)p_k + \gamma/K$
  observe reward $x_k \in [0, 1]$
  update $s_k := s_k + \frac{x_k \gamma}{\hat{p}_k K}$
end for

The *selection* of a best algorithm for a *set* of problems can be naturally described in a K-armed bandit setting,[4] where

[3]The original formulation is based on a finite upper bound g on the cumulative reward of the best arm, which is at most M as each reward is in $[0, 1]$. A variant (**Exp3.1**) of the algorithm is proposed if M is unknown. The authors also present a better bound on the *expected* reward.

[4]The cases of selection on an per-instance basis, and of algorithm portfolios, are more difficult to treat: bandit problem solvers usually provide bounds on regret with respect to a single arm, but the best algorithm might be different for each problem.

"pick arm k" means "execute the next step of algorithm a_k" (Alg. 2).

Algorithm 2 Algorithm selection using BPS

```
initialize BPS, p.
for each problem do
  set t_k := 0, r_k := 0, k = 1, ..., K
  repeat
    pick k ~ p
    execute step r_k + 1 of a_k
    update timer r_k := r_k + 1, t_k := t_k + t_k(r_k)
    if problem solved then
      observe reward x_k := 1/t_k
    else
      observe reward x_k := 0
    end if
    update p = BPS(k, x_k)
  until problem solved
end for
```

Note that this simple scheme falls in the partial information, non-oblivious adversarial setting, as we only observe reward for successful algorithms, and this reward depends on previous choices: for example, if a_k takes R steps to solve a problem, we will not see any reward for the first $R-1$ pulls. Our adversary is not malevolent, but it can be deceptive, especially if the $t_k(r)$ vary among algorithms:[5] even so, any bounds on performance of BPS will hold, and given our definition of reward, the expected performance of (Alg. 2) will converge to the one of the best solver for the set of problems.

4 Mixing restart strategies

Traditional statistical tests assess the goodness of a model measuring the fit of the corresponding probability density function (pdf) along the whole spectrum of observed samples. In our case, the sole purpose of the model $\hat{F}$ of the RTD is to set up a restart strategy, in order to gain on future performance, so the only quantity of practical interest is the loss in performance induced by the mismatch of our model. In [Gagliolo and Schmidhuber, 2006b] we studied the impact of increasingly censored sampling[6] on the performance of an estimated uniform restart strategy. This impact is quite low, due to the fact that (1) depends only on the lower quantiles of the distribution: a rough model of F, obtained from a heavily censored sample from the lower portion of the time scale, suffices to evaluate a near-optimal strategy, allowing to reduce its training cost.

The practical implementation of such a technique requires some censoring strategy to limit the cost of solving the training instances. Assuming no a priori information about T^*, an ideal candidate is the universal strategy of Luby (Sect 2), as it invests equal portions of run-time on a set of exponentially spaced restart thresholds. Note that the restart thresholds of unsuccessful runs can be exploited in the form of censored samples. Such a simple scheme would allow to keep the performance bound of the universal strategy during training, and use the gathered data to estimate $\hat{F}$, and the corresponding strategy $\hat{T}$, to be exploited for future problem instances. To limit the cost of learning, and allow exploitation of the model to start as soon as possible, we will instead adopt an *online*, or *life-long learning* approach. We will use the "bandit" algorithm selection scheme described in Sect. 3 at an upper level, to control allocation of computational resources. In this case, the arms are the two restart strategies, each generating a sequence of restart thresholds $T_k(r_k)$, to bound a same algorithm s; "pull arm k" simply means "start s on the current problem, with threshold $T_k(r_k)$". If the problem is not solved, $t_k(r_k) = T_k(r_k)$, and $x_k := 0$; if it is solved in a time $t_s <= T_k(r_k)$, $t_k(r_k) = t_s$, and a positive reward is observed.

Our strategy GAMBLER (Alg. 3) starts by solving the first problem using the universal strategy alone. After a problem instance is solved, the solution time of the successful run, and all the collected censored samples from the unsuccessful runs, can be used to update the estimate $\hat{F}$, and evaluate a new $\hat{T}$, to be used on next problem. To model F, we will use the non-parametric product-limit estimator by Kaplan and Meier [1958]. It guarantees large sample convergence to an arbitrary distribution, and can account for censored samples as well. The non-parametric form is particularly appealing, as it does not require any a-priori assumptions about the RTD, it is fast to update, and the resulting CDF is a step-wise function, which makes the evaluation of the integral in (1) inexpensive.

To use **Exp3** as the BPS we only need to normalize the rewards. We can do so by setting a lower and upper bound on t_s, $t_{min} \leq t_s \leq t_{max}$. In order to limit the impact of this choice on the convergence rate of **Exp3**, we adopt a logarithmic reward scheme: upon solution of a problem during a restart with strategy k, $r_k := (\log t_{max} - \log t_k)/(\log t_{max} - \log t_{min})$, t_k being the total time spent by strategy k on that problem, including unsuccesful runs. In this way 1 is the maximum reward, that would be obtained by a strategy that solved a problem in a time t_{min}. Note that the use of a logarithm allows to set t_{min} and t_{max}, respectively, to a very small and a very large value, such that only the knowledge of a very loose bound on t_s is required.[7]

Let us now analyze how the bound (2) combines with the optimal regret of **Exp3**. In the following, $E(y)$ is the expected value of a random variable y, t_k represents solution time of strategy k on a single problem, R_k the number of restarts performed, $p_k \in [0,1]$ the probability of picking a restart strategy, and indices T^*, U, $\hat{T}$, will label quantities related to

[5] Consider a situation in which a_1 solves a problem in one step of time 10, and gets reward 0.1, while a_2 would have solved the same problem with 10 steps of duration 0.1 each, scoring reward 1.

[6] *Censored sampling* is a commonly used technique in lifetime distribution estimation (see, e. g., [Nelson, 1982]), which allows to reduce the duration of a sequence of experiments, simply aborting runs exceeding a time threshold. The information carried by these runs can still be used for modeling, both in the parametric and non-parametric settings.

[7] Also, the bound is not required to hold: if we always set $t_s := \max\{t_s, t_{min}\}$, **Exp3** will not be able to discriminate among two algorithms that always have $t_s \leq t_{min}$ but will otherwise work; if we always restart algorithms exceeding t_{max}, **Exp3** will still work if $F^*(t_{max}) > 0$). In both cases we can keep $x_k \in [0,1]$.

Algorithm 3 GAMBLER(M) restart strategy for algorithm s

set t_{min}, t_{max}
initialize **Exp3**(M), $p_U = 1$.
for each problem $1, ..., M$ **do**
  set $t_k := 0$, $r_k := 0$, $k = 1, 2$
  repeat
    pick $k \sim \mathbf{p}$
    run s with cutoff $T_k(r_k + 1)$
    update timer $r_k := r_k + 1$, $t_k := t_k + \min\{t_s, T_k(r_k)\}$
    if problem solved **then**
      observe reward $x_k := \frac{\log t_{max} - \log t_k}{\log t_{max} - \log t_{min}}$
    else
      observe reward $x_k := 0$
    end if
    update $\mathbf{p}$ =**Exp3**(k, x_k)
  until problem solved
end for

the unknown optimal uniform, universal, and estimated optimal uniform strategies, respectively, while G will identify our novel strategy GAMBLER, interleaving the execution of the latter two. On a given problem, for which the RTD of our algorithm s is $F(t)$, a restart strategy with thresholds $T(r)$ can be viewed as sequence of independent Bernoulli processes, with success probabilities $F(T(r))$. The number of restarts R required to solve the problem is then distributed with pdf $p(R) = \prod_{r=1}^{R-1}(1 - F(T(r)))F(T(R))$: for a uniform strategy $T(r) = T$, $p(R_T)$ is geometric, with $E(R_T) = 1/F(T)$. For any deterministic strategy, the expected time to solve a problem is $E_{T(r)}(t) = \sum_{r=1}^{E(R)} T(r)$, a monotonic function of $E(R)$. For U this implies that the bound (2) can be translated into a bound on R_U, with the same high probability. Given our simple reward scheme **Exp3** will keep a constant $\mathbf{p} = (p_U, p_{\hat{T}})$ during solution of a single problem. Consider now the worst-case setting in which $\hat{T}$ is such that $F(\hat{T}) = 0$, i. e., the uniform restart $\hat{T}$ will *never* solve the problem. If U spends R_U restarts, in the meantime another $R_{\hat{T}}$ restarts will have been wasted on $\hat{T}$, with $E(R_{\hat{T}}) = E(R_U)(1 - p_U)/p_U$. As **Exp3** always keeps $p_U \geq \gamma/2$, the expected performance of GAMBLER on this problem will then be bounded by $E(t_U + R_U\hat{T}(2 - \gamma)/\gamma)$ with high probability, and upper bounds on t_U and R_U will also guarantee an upper bound on t_G.

5 Originality and related work

Restart strategies are particularly effective if the RTD of the controlled algorithm exhibits "heavy" tails, , i. e., is Pareto for both very large and very small values of time. Such behavior is observed for backtracking search on structured underconstrained problems in [Hogg and Williams, 1994; Gomes *et al.*, 2000], but also in other problem domains, such as computer networks, as in [van Moorsel and Wolter, 2004]. An alternative solution is to run multiple copies of the same algorithm in parallel (algorithm *portfolios* [Huberman *et al.*, 1997; Gomes and Selman, 2001]). In [Luby *et al.*, 1993], Luby presents *parallel* restart strategies, which represent the natural combination with the portfolio approach: its solution is not adaptive, as it simply consists in restarting each algorithm with the universal strategy U. The literature on meta-learning and algorithm selection [Giraud-Carrier *et al.*, 2004; Rice, 1976], algorithm portfolios [Huberman *et al.*, 1997; Gomes and Selman, 2001], *anytime* algorithms [Boddy and Dean, 1994], provides many examples of the application of performance modeling to resource allocation. Most works in these field adopt a more traditional *offline* approach, in which run-time samples are collected during an often impractically long initial training phase, and the obtained model is used without further tuning on subsequently met problems. As discussed in the introduction, this also implies stronger assumptions about the representativeness of the training set, and does not guarantee upper bounds on execution time.

The Max K-armed bandit problem is a variation on the original formulation, in which the estimated reward is only updated for the arm obtaining the maximum reward value, on a set of plays. Solvers for this game are used in [Cicirello and Smith, 2005; Streeter and Smith, 2006] to maximize performance quality, on a single problem instance.

Allocation of resources is usually set before each problem instance solution, and possibly updated afterward. *Dynamic* approaches, in which feedback information from the algorithms is exploited to adapt resource allocation *during* problem solution, represent a notable recent trend. In [Lagoudakis and Littman, 2000; Petrik, 2005], algorithm selection is modeled as a reinforcement learning problem, a more general setting than the bandit problem proposed in (Sect. 3). Models of performance conditional on the dynamic state after a fixed execution time are used in [Kautz *et al.*, 2002; Wu, 2006] to implement dynamic context-sensitive restart policies for SAT solvers: also in this case the main difference of our approach is the online learning setting. [Gagliolo and Schmidhuber, 2006a] present a heuristic dynamic online learning approach to resource allocation, in which a conditional model of performance is trained and progressively exploited during training.

6 Experiments

The experiments were conducted using Satz-Rand [Gomes *et al.*, 2000], a version of Satz [Li and Anbulagan, 1997] in which random noise influences the choice of the branching variable. Satz is a modified version of the complete DPLL procedure, in which the choice of the variable on which to branch next follows an heuristic ordering based on first and second level unit propagation. Satz-rand differs in that, after the list is formed, the next variable to branch on is randomly picked among the top h fraction of the list. We present results with the heuristic starting from the most constrained variables, as suggested in [Li and Anbulagan, 1997], and noise parameter set to 0.4.

A benchmark from SATLIB was used, consisting of different sets of "morphed" graph-coloring problems [Gent *et al.*, 1999]: each graph is composed of the set of common edges among two random graphs, plus fractions p and $1-p$ of the remaining edges for each graph, chosen as to form regular ring lattices. Each of the 9 problem sets contains 100 instances,

generated with a logarithmic grid of 9 different values for the parameter p, from 2^0 to 2^{-8}, to which we henceforth refer with labels $0, ..., 8$. Satz-Rand has an initialization cost which depends only on the size of the problem, which in this case is the same for all sets. This cost was found to be always bigger than 10^5. We then set $t_{min} = 10^5$, $t_{max} = 10^{12}$ (about 100 minutes on our machine), and, to be fair with the universal strategy, we multiplied and shifted its $T_U(r)$ as $t_{min}(1 + T_U(r))$. With $M = 100$, parameters of **Exp3** are determined as $\alpha = 0.38$, $\gamma = 0.19$.

Each experiment was repeated 20 times with different random seeds, and a different random order of the instances. For comparison purposes, we repeated the experiments running the original algorithm, without restarts (labeled S) and the universal strategy alone (U). To compare with the ideal performance of T^*, we evaluated, *a posteriori* for each run, the T that minimized the cost of the problem set ($T_L(set)$), and the cost of each instance ($T_L(inst)$), based on the *actual* runtime outcomes. Note that these can be different for each run, and their performances are *lower bounds* on the performances of the optimal T^*, evaluated from the unknown actual RTD of, respectively, the problem set, and each problem instance.[8] The difference of these two bounds is particularly interesting, as it is an indirect measure of the heterogeneity of the RTD of the instances in each set. To show the effect of a more heterogeneous problem set, we also run experiments with all the 900 instances grouped as a single set (labeled A in the graph; $M = 900$, $\alpha = 0.18$, $\gamma = 0.09$), again randomly mixed. In figure 6 we present, for each set, the total computation time[9] for our mixture (G), and the comparison terms. Upper 95% confidence bounds are given, estimated on the 20 runs. Plots of the RTD and restart cost for each set can be found in [Gagliolo and Schmidhuber, 2006b].

The results are quite impressive, and further confirm that the estimated restart strategy $\hat{T}$ is not very sensitive to the fit of the model $\hat{F}$. In a typical run, between 40% and 80% of the sample is censored, as there is only one uncensored runtime for each task; the fit of the model is visibly bad, and it is limited to the lower portion of the time scale, near or below $\hat{T}$, but $\hat{T}$ itself quickly converges close enough to T^*, giving a near-optimal strategy.

Problems in 0 are easy. Satz-rand solves all instances in a similar time, any larger T is an optimal restart value, i.e., restarts are never executed, and the performance of a single copy S is the same as the ones of the optimal restarts. Also our GAMBLER quickly learns that, while U has to reach this value for every problem, resulting in a five times worst performance. In problems 1 to 5 we see the effect of a heavy-tailed RTD. S solves each set with times between 1.6×10^{10} (1) and 4.6×10^{12} (3). In practice, only a few "unlucky" runs have very long completion times, but this is enough to penalize its performance dramatically. GAMBLER scores fairly against the lower bounds, and U is between 3 and 5 times worst. From problem 6 on, we see that the heavy tail effect is less marked, but the two lower bounds diverge noticeably: instances in these sets present more heterogeneous RTDs, and the instance-optimal $T^*(inst)$ may vary of more than one order of magnitude. Here U is only 2.5-3 times worse than GAMBLER. The worst performance of GAMBLER compared to $L^*(set)$ can be seen on problem 8, where t_G is about 1.5 times $t_{L^*(set)}$. Here most problems require a small threshold T, and a few require one about ten times larger: in this situation also threshold $T^*(set)$ varies visibly among different runs, and $\hat{T}$ sometimes overestimates the optimal threshold. Results on the whole set of problems A are better than expected: $t_G/t_{L^*(set)}$ is about 1.12, but the performance compared to $t_{L^*(inst)}$ is obviously worst (1.65). This is natural, as both GAMBLER and $T^*(set)$ cannot discriminate different restarts for each problem. U completes the set at 1.17×10^9, about 3.4 times worse than GAMBLER.

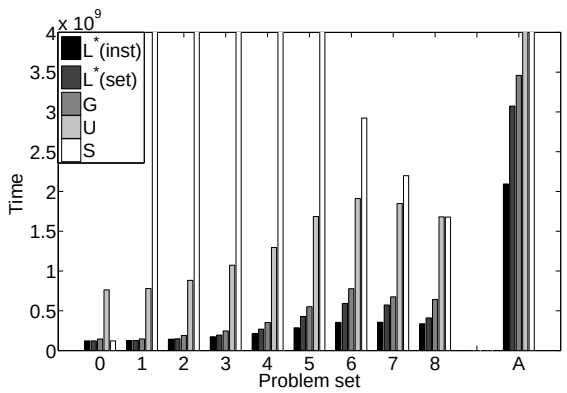

Figure 1: Experiments with Satz-Rand. Time to solve each set of problems ($10^9 \approx 1$ min.). G is our mixed strategy GAMBLER, whose performance is initially as U, the universal restart strategy, and quickly converges near $L^*(set)$, a lower bound on the performance of the unknown optimal restart strategy for the whole set. $L^*(inst)$ is instead a lower bound on the performance of a distinct optimal restart strategy for each instance. A is the heterogeneous set obtained mixing all sets $0, ..., 8$. Upper 95% confidence bounds estimated on 20 runs.

7 Conclusions and future work

We presented a novel restart strategy (GAMBLER), combining the universal and optimal strategies by [Luby *et al.*, 1993], based on a bandit approach to algorithm selection. GAMBLER takes the best of two worlds: it preserves the upper bounds of the universal strategy in a worst-case setting, and it proved to be effective in a realistic application, with only a small overhead over an ideal lower bound. It can save a few orders of magnitude in computation time for algorithms with a heavy-tailed RTD, and, unlike the universal strategy, it does not worsen the performance otherwise. It has a negligible overhead compared to the controlled algorithm, and

[8] As $E(t_{T^*}) = \min_T\{E(t_T)\} \geq E(\min_T\{t_T\}) = E(t_L)$ in both cases.

[9] As we are also conducting experiments with parallel portfolios of heterogeneous algorithms, and thus we need a common measure of time, we modified the original code of Satz-rand adding a counter, that is incremented at every loop in the code. The resulting time measure was consistent with the number of backtracks. All results reported are in these loop cycles: on a 2.4 GHz machine, 10^9 cycles take about one minute.

does not require the tuning of any parameter, nor any a-prior assumption about the RTD, excluding very loose bounds on execution time, which in the presented experiments differed of 7 orders of magnitude. It does not require an initial training phase.

The parameters of **Exp3** are automatically set based on the number of problems to be solved. If this number is not known in advance, **Exp3.1** can be used instead.

The obtained strategy can be applied to any randomized algorithm, including, e. g., evolutionary algorithms, stochastic local search, gradient descent from a random initial value. Its main limitation is that it relies on the assumption that the problems to be solved display a similar RTD, but we saw in practice that this requirement is not strict, and the observed performance was consistently better than the one of the universal strategy alone.

The strategy has a modular structure: it would allow inclusion of more strategies, with an $O(K \log K)$ impact on performance, guaranteed by **Exp3**. Or more algorithms, each with its own RTD model. We particularly expect improvements from the use of conditional models, based on problem features, as they should allow for a distinction among different sub-classes of problems, sharing a similar RTD. A further step could be taken including dynamic information from the actual execution, as exemplified in [Kautz *et al.*, 2002; Gagliolo and Schmidhuber, 2006a].

Acknowledgments. The authors would like to thank Faustino Gomez for precious comments on the draft of this paper. This work was supported by SNF grant 200020-107590/1.

References

[Auer *et al.*, 1995] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. Gambling in a rigged casino: the adversarial multi-armed bandit problem. In *Proc. 36th Annual Symposium on Foundations of Computer Science*, pages 322–331. IEEE, 1995.

[Boddy and Dean, 1994] Mark Boddy and Thomas L. Dean. Deliberation scheduling for problem solving in time-constrained environments. *Artificial Intelligence*, 67(2):245–285, 1994.

[Cicirello and Smith, 2005] Vincent A. Cicirello and Stephen F. Smith. The max k-armed bandit: A new model of exploration applied to search heuristic selection. In *AAAI*, pages 1355–1361, 2005.

[Gagliolo and Schmidhuber, 2006a] Matteo Gagliolo and Jürgen Schmidhuber. Dynamic algorithm portfolios. In *AI & MATH*, 2006.

[Gagliolo and Schmidhuber, 2006b] Matteo Gagliolo and Jürgen Schmidhuber. Impact of censored sampling on the performance of restart strategies. In *CP 2006* , pages 167–181. Springer, 2006.

[Gent *et al.*, 1999] Ian Gent, Holger H. Hoos, Patrick Prosser, and Toby Walsh. Morphing: Combining structure and randomness. In *Proc. of AAAI-99*, pages 654–660, 1999.

[Giraud-Carrier *et al.*, 2004] Christophe Giraud-Carrier, Ricardo Vilalta, and Pavel Brazdil. Introduction to the special issue on meta-learning. *Machine Learning*, 54(3):187–193, 2004.

[Gomes and Selman, 2001] Carla P. Gomes and Bart Selman. Algorithm portfolios. *Artificial Intelligence*, 126(1–2):43–62, 2001.

[Gomes *et al.*, 2000] Carla P. Gomes, Bart Selman, Nuno Crato, and Henry Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *J. Autom. Reason.*, 24(1-2):67–100, 2000.

[Hogg and Williams, 1994] Tad Hogg and Colin P. Williams. The hardest constraint problems: a double phase transition. *Artif. Intell.*, 69(1-2):359–377, 1994.

[Huberman *et al.*, 1997] B. A. Huberman, R. M. Lukose, and T. Hogg. An economic approach to hard computational problems. *Science*, 275:51–54, 1997.

[Kaplan and Meyer, 1958] E.L. Kaplan and P. Meyer. Nonparametric estimation from incomplete samples. *J. of the ASA*, 73:457–481, 1958.

[Kautz *et al.*, 2002] H. Kautz, E. Horvitz, Y. Ruan, C. Gomes, and B. Selman. Dynamic restart policies, 2002.

[Lagoudakis and Littman, 2000] Michail G. Lagoudakis and Michael L. Littman. Algorithm selection using reinforcement learning. In *Proc. ICML*, pages 511–518. Morgan Kaufmann, 2000.

[Li and Anbulagan, 1997] Chu-Min Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *IJCAI97*, pages 366–371, 1997.

[Luby *et al.*, 1993] Michael Luby, Alistair Sinclair, and David Zuckerman. Optimal speedup of las vegas algorithms. *Inf. Process. Lett.*, 47(4):173–180, 1993.

[Nelson, 1982] Wayne Nelson. *Applied Life Data Analysis*. John Wiley, New York, 1982.

[Petrik, 2005] Marek Petrik. Statistically optimal combination of algorithms, 2005. Presented at SOFSEM 2005.

[Rice, 1976] J. R. Rice. The algorithm selection problem. In *Advances in computers*, volume 15, pages 65–118. Academic Press, New York, 1976.

[Robbins, 1952] H. Robbins. Some aspects of the sequential design of experiments. *Bulletin of the AMS*, 58:527–535, 1952.

[Streeter and Smith, 2006] Matthew J. Streeter and Stephen F. Smith. An asymptotically optimal algorithm for the max k-armed bandit problem. In *AAAI*, 2006.

[van Moorsel and Wolter, 2004] Aad P. A. van Moorsel and Katinka Wolter. Analysis and algorithms for restart. In *QEST '04*, pages 195–204, 2004. IEEE.

[Wu, 2006] H. Wu. Randomization and restart strategies. Master's thesis, University of Waterloo, School of Computer Science, 2006.