Gambling in a Computationally Expensive Casino: Algorithm Selection as a Bandit Problem

Matteo Gagliolo[*†] **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

Automating algorithm selection and parameter tuning is an old dream of the AI community, which has been brought closer to reality in the last decade. Most available techniques are either *oblivious*, with no knowledge transfer across different problems; or are based on a model of algorithm performance, learned in a separate *offline* training sequence, which is often prohibitively expensive. We describe recent work in which the problem is treated in a fully *online* setting. A model of algorithm performance can be learned *and* used to reduce the cost of learning it. The resulting *exploration-exploitation* trade-off can be treated in the context of Bandit problems.

Keywords: algorithm selection, algorithm portfolios, online learning, life-long learning, bandit problem, adversarial setting, expert advice, survival analysis, randomized algorithms, restart strategies, satisfiability, constraint programming, performance modeling.

1 Introduction

Most problems in AI can be solved by more than one algorithm. Most algorithms feature a number of parameters that have to be set. Both choices can dramatically affect the quality of the solution, and the time spent obtaining it. Algorithm Selection [32], or *Meta-Learning* [36] techniques, address these questions in a machine learning setting. Based on a training set of performance data for a large number of problem instances, a model is learned that maps (*problem*, *algorithm*) pairs to expected performance. The model is later used to select and run, for each new problem, only the algorithm that is expected to give the best results. This approach, though preferable to the still far more popular "trial and error", poses several problems. The training set is assumed to be representative of future instances. The computational cost of the initial training phase, which obviously requires solving each training problem repeatedly, at least once for each of the algorithms, can be high enough to make algorithm selection impractical.

One way of reducing the cost of learning a performance model, is to use the model itself to guide the training phase [9, 10, 12, 8]. There is an obvious trade-off arising in this context, between the *exploration* of algorithm performances on different problem instances, aimed at improving the model, and *exploitation* of the best algorithm/problem combinations, based on current model's predictions. In this paper, we address this trade-off in the context of bandit problems, summarizing recent results from [12, 8].

2 Related work

Algorithm selection techniques can be described according to different orthogonal features:

Optimisation vs. decision problems. A first distinction needs to be made among *decision* problems, where a binary criterion for recognizing a solution is available; and *optimisation* problems, where different levels of solution quality can be attained, measured by an *objective* function [18]. Conversions among the two kinds are possible.
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 [20]); or repeated for each instance (*per instance* selection).
Static vs. dynamic selection. A further independent distinction [29] can be made among *static* algorithm selection, in which any decision on the 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.
Offline 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* or *life-long* learning [31] techniques, where the criteria are updated at every instance solution.

A seminal paper in this field is [32], in which offline, per instance algorithm selection is first advocated, both for decision and optimisation problems. More recently, similar concepts have been proposed, with different terminology (algorithm *recommendation*, *ranking*, *model selection*), by the *Meta-Learning* community [6, 36, 14]. Usually, meta-learning research deals with optimisation problems, and is focused on maximizing solution quality, without taking into account the computational aspect. Works on *Empirical Hardness Models* [23, 28] are instead applied to decision problems, and focus on obtaining accurate models of runtime performance, conditioned on numerous features of the problem instances, as well as on parameters of the solvers [20]. The models are used to perform algorithm selection on a per instance basis, and are learned offline: online selection is advocated in [20]. Literature on algorithm portfolios [19, 15, 30] is usually focused on choice criteria for building the set of candidate solvers, such that their areas of good performance don't 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 [21], 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; the model is learned offline. In a Reinforcement Learning [35] setting, algorithm selection can be formulated as a Markov Decision Process: in [22], 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 [29], 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 are per set, offline.

"Low-knowledge" oblivious approaches can be found in [3, 4], 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 [4], the selection process is iterated: machine time shares are based on a recency-weighted average of performance improvements. In [7] we adopted a similar approach. 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.

More references can be found in [7, 9, 10, 8], and Sect. 4.

3 The adversarial bandit problem

In its most basic form [33], the *multi-armed bandit* problem is faced by a gambler, playing a sequence of trials against a K-armed slot machine. At each trial, the gambler chooses one of the

available arms, whose rewards are randomly generated from different *stationary* distributions. The gambler can then receive the corresponding reward r_k, and, in the *full information* game, observe the rewards that he would have gained pulling any of the other arms. The aim of the game is to minimize the *regret*, defined as the difference between the cumulative reward of the best arm, and the one earned by the gambler. A bandit problem solver (BPS) can be described as mapping from the history of the observed rewards $r_k \in [0, 1]$ for each arm k, to a probability distribution $\mathbf{p} = (p_1, ..., p_K)$, from which the choice for the successive trial will be picked.

In recent works, the original restricting assumptions have been progressively relaxed, allowing for *non-stationary* reward distributions, *partial* information (only the reward for the pulled arm is observed), and *adversarial* bandits, that can set their rewards in order to deceive the player. In [1], 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, proving interesting bounds on the regret.

4 Algorithm selection as a bandit problem

Consider now a sequence $B = \{b_1, ..., b_M\}$ of M problem instances, and a set of K algorithms $A = \{a_1, a_2, ..., a_K\}$, such that each b_m can be solved by at least one a_k. It is straightforward to describe static algorithm selection in a K-armed bandit setting, where "pick arm k" means "run algorithm a_k on next problem instance". For decision problems, runtime t_k can be treated as a *loss*, to be minimized; or a reward could be set, for example as $r_k := 1/t_k$. For optimisation problems, the quality of the obtained solution could be the reward. In both cases, the reward would be generated by a rather complex mechanism, i.e., the algorithms a_k themselves, running on the current problem, so the bandit problem would fall into the adversarial setting. The information would be partial: the runtime for other algorithms would not be available. 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 [12], where we presented an online method for learning a per set estimate of an optimal restart strategy[1] (GAMBLER, see Alg. 1). The method consists in alternating the universal strategy of [26], and an estimated optimal strategy, again based on [26]. The estimate is performed according to a nonparametric model of runtime distribution on the set of instances, updated at every solution. Here the bandit problem solver (EXP3 from [1]) is used at an upper level: the two arms are the two restart strategies, proposing different restart thresholds for the same randomized algorithm. The reward for each solved instance was given based on the logarithm of the total time t_k spent by the winning strategy k, including unsuccessful runs.

Alternative per instance, but oblivious, approaches, can be built considering more refined reward attributions. If the aim of selection is only to maximize solution quality, a same problem can be solved multiple times, eventually keeping only the best found solution. The selection problem can then 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 observed on a sequence of rounds. Solvers for this game are used in [5, 34] to implement oblivious per instance selection from a set of multi-start optimisation techniques: each problem is treated independently, and multiple runs of the available solvers are allocated, to maximize performance quality.

In a variant of this game, machine time can be subdivided into intervals δt: "pick arm k" would mean "resume algorithm a_k on current problem instance, for a time δt, then pause it". Reward could be attributed as before, $r_k := 1/t_k$, t_k being the *total* runtime of the winning algorithm.

Information would again be partial: more precisely, in this case it would be *incomplete*, or *censored*,[2] as a *lower bound* on performance, and a corresponding *upper bound* on reward, would be available for the other algorithms. The bandit would be a *non-oblivious* adversary, as the result of each arm

[1] A restart strategy consists in executing a sequence of runs of a randomized algorithm, in order to solve a given problem instance, stopping each run j after a time $T(j)$ if no solution is found, and restarting the algorithm with a different random seed.

[2] *Censored sampling* is a commonly used technique in lifetime distribution estimation (see, e. g., [27]), 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. See also [9, 10, 11, 12, 8].

Algorithm 1 GAMBLER(M) Gambling Restart for algorithm s, interleaving K strategies.

set t_{min}, t_{max}, K
initialize EXP3 (M, K), $\mathbf{p}$
for each problem $1, ..., M$ **do**
  set $t_k := 0$, $j_k := 0$, $k = 1, ..., K$
  repeat
    pick $k \sim \mathbf{p}$
    run s with cutoff $T_k(j_k + 1)$
    update counter $j_k := j_k + 1$, timer $t_k := t_k + \min\{t_s, T_k(j_k)\}$
    if problem solved **then**
      observe reward $r_k := \frac{\log t_{max} - \log t_k}{\log t_{max} - \log t_{min}}$
    else
      observe reward $r_k := 0$
    end if
    let EXP3 update $\mathbf{p}$
  until problem solved
  update $\mathcal{M}$ based on collected runtime data
end for

pull would depend on previous pulls of the same arm. Also in this case, oblivious per instance selection can be implemented, if some indicator of current performance can be obtained at runtime, after pausing each a_k. The *increment* in this quantity, can be used to attribute reward for the last step. This approach is followed in the per instance, oblivious, dynamic selection technique presented in [4]: the simple *recency-weighted average* of performance increment used there can be seen as a simple solver for a time-varying bandit problem (see, e.g., [35], Sect. 2.6).

For a very small δt, and a large number of arm pulls, the expected value of time spent executing a_k would be proportional to p_k. And, typically, bounds on regret for a BPS are proved based on expected values. The game described above would then be equivalent to a *static* portfolio [9, 29, 10, 30], running the algorithms a_k in parallel, allocating time to a_k proportionally to s_k, such that for any portion of time spent t, $s_k t$ is used by a_k. The $\mathbf{p}$ of the BPS can be used as the *share* value $\mathbf{s} = (s_1, ..., s_K)$, , $s_k \geq 0$, $\sum_k s_k = 1$, and can be updated after a problem instance is solved. Again, the resulting selection technique is *static*, *per set*, only profitable if one of the algorithms dominates the others on all problem instances. A less restrictive, and more interesting hypothesis, is that there is one of a set of *time allocators* (TA) [7, 9, 10, 8] whose performance dominates the others. A TA can be an arbitrary function mapping the current history of collected performance data for each a_k, to a share $\mathbf{s}$. The simplest example is the *uniform* time allocator, assigning a constant $\mathbf{s} = (1/K, ..., 1/K)$. In previous work, we presented examples of heuristic oblivious [7] and non-oblivious [9] model-based allocators. More sound TAs are proposed in [8], based on minimization of expected solution time, or of a *quantile* of solution time, and on maximization of solution probability within a give time *contract*.

At this higher level, one can use a BPS to select among different time allocators, TA$^{(1)}$, TA$^{(2)}$,..., working on a same algorithm set $\mathcal{A}$. In this case, "pick arm n" means "use time allocator TA$^{(n)}$ 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$^{(n)}$ that is best at allocating time to algorithms in $\mathcal{A}$ on a *per instance* basis. If the BPS allows for time-varying reward distributions, it can also deal with time allocators that are *learning* to allocate time: this is precisely the situation of a model-based allocator, whose model $\mathcal{M}$ is being learned online, based on results on the sequence of problems met so far.

A more refined alternative is suggested by the bandit problem with *expert* advice, as described in [1, 2]. 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 advices, in order to generate the distribution for the lower level arms. To play this two-level game, Auer *et al.* [1] propose an

algorithm called EXP4 , featuring bounds on regret relative to the performance of the best *expert*, provided that the *uniform* expert $(1/K, ..., 1/K)$ is included in the set.

In our case, 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. The *partial* information on the reward at the lower level (based on the runtime of the a_k first to solution) is translated to *full* information at the upper level, based on the $\mathbf{s}^{(n)}$ proposed by each $\text{TA}^{(n)}$. The bound of EXP4 on the regret w.r.t. the best TA can be achieved including the uniform TA in the set.

A straightforward extension can be made, to the case of *dynamic* algorithm portfolios [7, 9, 29], in which a sequence of machine time slots $\Delta t(0), \Delta t(1), ...$ is allocated during the solution of a single problem, and each $\text{TA}^{(n)}$ can update its proposed $\mathbf{s}^{(n)}(j)$ for each subsequent $\Delta t(j)$. In this case, the normalized value of $\sum_j \mathbf{s}^{(n)}(j)\Delta t(j)$ is used in place of $\mathbf{s}^{(n)}$ to update $\mathbf{p}$ of EXP4 . The resulting "Gambling" Time Allocator (GAMBLETA) [8] is described in Alg. 2, again based on a logarithmic reward.

Algorithm 2 GAMBLETA (M) Gambling Time Allocator.

Algorithm set $\mathcal{A}$ with K algorithms; N TA; M problem instances.
let EXP4 (K, N, M) initialize $\mathbf{p} \in [0, 1]^N$
set t_{min}, t_{max}, initialized model $\mathcal{M}$
for each problem $b_1, b_2, ..., b_M$ **do**
  while b_m not solved **do**
    update Δt
    for each time allocator $\text{TA}^{(1)}, ..., \text{TA}^{(N)}$ **do**
      update $\mathbf{s}^{(n)} = \text{TA}^{(n)}(\mathcal{M})$, $\mathbf{s}^{(n)} \in [0, 1]^K$
    end for
    evaluate mix $\mathbf{s} = \sum_{n=1}^{N} p_n \mathbf{s}^{(n)}$
    run $\mathcal{A}$ with share $\mathbf{s}$, for a maximum time Δt
  end while
  observe reward $r_k := \frac{\log t_{max} - \log t_k}{\log t_{max} - \log t_{min}}$ for winner a_k
  let EXP4 update $\mathbf{p}$
  update $\mathcal{M}$ based on collected runtime data
end for

5 Experiments

In [8] we present experiments with two small algorithm sets ($K = 2$), but long and challenging problem sequences. In the first experiment, a local search and a complete SAT solver (respectively, G2-WSAT [25] and Satz-Rand [16], a randomized version of Satz [24]) are controlled by GAMBLETA during the solution of a sequence of random satisfiable and unsatisfiable problems (benchmarks `uf-*`, `uu-*` from [17], 1899 instances in total). Local search (LS) 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 GAMBLETA over an ideal "oracle", which can predict satisfiability of an instance, and run only the fastest algorithm, is 14%. Satz-Rand alone could 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 (a) plots the evolution of cumulative time along the problem sequence. In the second experiment from [8], we compare with results of a static algorithm selection approach [23], controlling two solvers (CASS and CPLEX) on a set of 10664 combinatorial Auction Winner Determination problems, available online. In [23], an overhead of 8% on the oracle is reported, obtained after a training sequence that cost several years of CPU-time. GAMBLETA achieved an overhead of 4%, which includes training

SAT-UNSAT	GAMBLETA	$2.88 \times 10^{10} \pm 1.06 \times 10^{8}$
	ORACLE	$2.53 \times 10^{10} \pm 5.17 \times 10^{7}$
	CUMOVH	0.138 ± 0.00324
WDP	GAMBLETA	$1.12 \times 10^{8} \pm 1.81 \times 10^{5}$
	ORACLE	$1.08 \times 10^{8} \pm 7.61 \times 10^{-8}$
	CUMOVH	0.0381 ± 0.00167

Table 1: Performance of GAMBLETA , evaluated averaging over 50 runs, each time with a different random order of instances. 95% confidence intervals. SAT-UNSAT and Winner Determination Problem (WDP) benchmarks. The cumulative performance $\sum_j t_G(j)$ of GAMBLETA and of the ORACLE $\sum_j t_O(j)$ are reported, $t_O(j) = min_k\{t_k(j)\}$, along with the cumulative overhead of GAMBLETA , with respect to the ORACLE (CUMOVH), $(\sum_j t_G(j) - \sum_j t_O(j))/\sum_j t_O(j)$.

time, as it is a fully online technique.[3] Table 5 reports results[4] of 50 runs, each time with a different random order of instances.

In [12] we present experiments with GAMBLER , controlling the restart threshold of Satz-Rand on 9 sets of structured graph-coloring (GC) problems [13], available from [17], each composed of 100 instances encoded in CNF 3 SAT format. This algorithm/benchmark combination is particularly interesting as the *heavy-tailed* behavior [16] of Satz-Rand differs for the various sets of instances[5] [13, 11]. In figure 1 (b) we present, for each problem set $0, ..., 8$, the total computation time for GAMBLER (G), and the comparison terms: Satz-Rand without restart (S), the universal strategy from [26] (U), and a *lower bound* on the performance of a single optimal restart strategy, evaluated *a posteriori* for the whole set ($L^*(set)$), and for each instance ($L^*(inst)$). On all sets, GAMBLER scores fairly against $L^*(set)$, and U is between 3 and 5 times worst.

6 Conclusions

We presented recent promising results of a "bandit" approach to algorithm selection. In both cases, a bandit problem solver is used at an upper level, to integrate the proposals of different time allocators (GAMBLETA), and different restart strategies (GAMBLER). In this latter case, the bound on performance of the universal strategy [26], combined with the bound on regret for EXP3 , result in a worst-case bound the performance of GAMBLER [12]. For GAMBLETA , only the regret w.r.t. the best time allocator is minimized: nothing can be guaranteed about the performance overhead on the *per instance* best algorithm.

Future research will explore the use of different BPS, starting from the alternatives described in [2]; and different reward schemes. A more ambitious goal is to reformulate the bandit problem with *censored* rewards, arising in the context of algorithm portfolios.

References

[1] Peter Auer, Nicolò Cesa-Bianchi, Yoav Freund, and Robert E. Schapire. Gambling in a rigged casino: the adversarial multi-armed bandit problem. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, pages 322–331. IEEE Computer Society Press, Los Alamitos, CA, 1995.

[2] Peter Auer, Nicolò Cesa-Bianchi, Yoav Freund, and Robert E. Schapire. The nonstochastic multiarmed bandit problem. *SIAM J. Comput.*, 32(1):48–77, 2002.

[3] On this latter point we have to remark that in [23] no attempt was made at reducing the training cost, the interest of the authors being more focused on the precision of the estimated models of performance.

[4] For the SAT-UNSAT benchmark, we modified the original code of the two solvers, adding a counter that is incremented at every loop in the code. All runtimes reported for this benchmark are expressed in these loop cycles: on a 2.4 GHz machine, 10^9 cycles take about 1 minute. A different random seed was used for each run. For the WDP problems, time is expressed in seconds, based on the runtime data available online [23]. The solvers are deterministic, and runs differ only in the order of the sequences.

[5] A *heavy-tailed* runtime distribution $F(t)$ is characterized by a Pareto tail, i.e., $F(t) \rightarrow_{t\rightarrow\infty} 1 - Ct^{-\alpha}$. In practice, this means that most runs are relatively short, but the remaining few can take a very long time. Plots of the RTD and restart cost for each set of this benchmark can be found in [11].

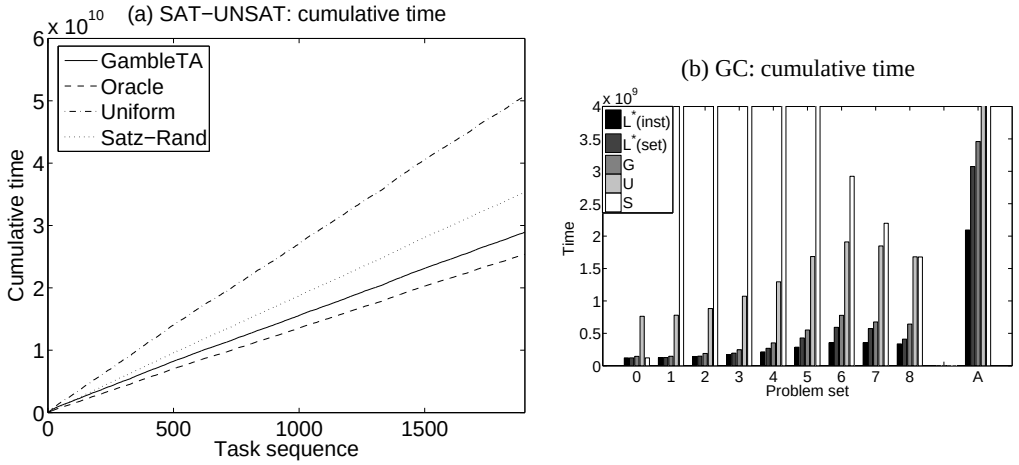

Figure 1: (a): Cumulative time spent by GAMBLETA on the SAT-UNSAT problem set ($10^9 \approx 1$ min.). Upper 95% confidence bounds on 50 runs, with random reordering of the problems. ORACLE is the lower bound on performance. UNIFORM is the $(0.5,0.5)$ share. SATZ-RAND is the single algorithm. (b): Experiments with GAMBLER (G) and Satz-Rand (S) on the GC benchmarks. Time to solve each set of problems. U is the universal restart strategy; $L^*(set)$, 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. The difference of these two bounds is an indirect measure of the heterogeneity of the RTD of the instances in each set. We also run experiments with all the 900 instances grouped as a single set (labeled A in the graph). In problem sets 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). Upper 95% confidence bounds estimated on 20 runs, with random reordering of the problems, and different random seeds for Satz-Rand.

[3] Christopher J. Beck and Eugene C. Freuder. Simple rules for low-knowledge algorithm selection. In *CPAIOR*, pages 50–64, 2004.

[4] T. Carchrae and J. C. Beck. Applying machine learning to low knowledge control of optimization algorithms. *Computational Intelligence*, 21(4):373–387, 2005.

[5] 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.

[6] Johannes Fürnkranz. On-line bibliography on meta-learning, 2001. EU ESPRIT METAL Project (26.357): A Meta-Learning Assistant for Providing User Support in Machine Learning Mining.

[7] M. Gagliolo, V. Zhumatiy, and J. Schmidhuber. Adaptive online time allocation to search algorithms. In J. F. Boulicaut et al., editors, *Machine Learning: ECML 2004.*, pages 134–143. Springer, 2004. — Extended tech. report available at `http://www.idsia.ch/idsiareport/IDSIA-23-04.ps.gz`.

[8] Matteo Gagliolo and Jürgen Schmidhuber. Learning dynamic algorithm portfolios. AI&MATH 2006 Special Issue, to appear.

[9] Matteo Gagliolo and Jürgen Schmidhuber. A neural network model for inter-problem adaptive online time allocation. In Włodzisław Duch et al., editors, *ICANN 2005, Proceedings, Part 2*, pages 7–12. Springer, 2005.

[10] Matteo Gagliolo and Jürgen Schmidhuber. Dynamic algorithm portfolios. Ninth International Symposium on Artificial Intelligence and Mathematics, 2006.

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

[12] Matteo Gagliolo and Jürgen Schmidhuber. Learning restart strategies. In *IJCAI 2007 — Twentieth International Joint Conference on Artificial Intelligence*, 2007. To appear.

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

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

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

[16] 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.

[17] H. H. Hoos and T. Stützle. SATLIB: An Online Resource for Research on SAT. In I.P.Gent et al., editors, *SAT 2000*, pages 283–292, 2000. `http://www.satlib.org`.

[18] Holger H. Hoos and Thomas Stützle. Local search algorithms for SAT: An empirical evaluation. *Journal of Automated Reasoning*, 24(4):421–481, 2000.

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

[20] Frank Hutter and Youssef Hamadi. Parameter adjustment based on performance prediction: Towards an instance-aware problem solver. Technical Report MSR-TR-2005-125, Microsoft Research, Cambridge, UK, December 2005.

[21] Henry A. Kautz, Eric Horvitz, Yongshao Ruan, Carla P. Gomes, and Bart Selman. Dynamic restart policies. In *AAAI/IAAI*, pages 674–681, 2002.

[22] Michail G. Lagoudakis and Michael L. Littman. Algorithm selection using reinforcement learning. In *Proc. 17th ICML*, pages 511–518. Morgan Kaufmann, 2000.

[23] Kevin Leyton-Brown, Eugene Nudelman, and Yoav Shoham. Learning the empirical hardness of optimization problems: The case of combinatorial auctions. In *ICCP: International Conference on Constraint Programming (CP), LNCS*, 2002. Runtime data available at `http://www.cs.ubc.ca/~kevinlb/downloads/db-data.zip`.

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

[25] Chu Min Li and Wenqi Huang. Diversification and determinism in local search for satisfiability. In *SAT2005*, pages 158–172. Springer, 2005.

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

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

[28] Eugene Nudelman, Kevin Leyton-Brown, Holger H. Hoos, Alex Devkar, and Yoav Shoham. Understanding random sat: Beyond the clauses-to-variables ratio. In *CP*, pages 438–452, 2004.

[29] Marek Petrik. Statistically optimal combination of algorithms. Presented at SOFSEM 2005 - 31st Annual Conference on Current Trends in Theory and Practice of Informatics, 2005.

[30] Marek Petrik and Shlomo Zilberstein. Learning static parallel portfolios of algorithms. Ninth International Symposium on Artificial Intelligence and Mathematics., 2006.

[31] Lorien Pratt and Sebastian Thrun. Guest editors' introduction. *Machine Learning*, 28:5, 1997. Special Issue on Inductive Transfer.

[32] J. R. Rice. The algorithm selection problem. In Morris Rubinoff and Marshall C. Yovits, editors, *Advances in computers*, volume 15, pages 65–118. Academic Press, New York, 1976.

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

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

[35] R. Sutton and A. Barto. *Reinforcement learning: An introduction*. Cambridge, MA, MIT Press, 1998.

[36] Ricardo Vilalta and Youssef Drissi. A perspective view and survey of meta-learning. *Artif. Intell. Rev.*, 18(2):77–95, 2002.