Dynamic Algorithm Portfolios

Matteo Gagliolo [*]and Jürgen Schmidhuber [*†]
[*]IDSIA, Galleria 2, 6928 Manno (Lugano), Switzerland
[†]TU Munich, Boltzmannstr. 3, 85748 Garching, München, Germany

Abstract

Traditional Meta-Learning requires long training times, and is often focused on optimizing performance quality, neglecting computational complexity. Algorithm Portfolios are more robust, but present similar limitations. We reformulate algorithm selection as a *time allocation* problem: all candidate algorithms are run in parallel, and their relative priorities are continually updated based on runtime information, with the aim of minimizing the time to reach a desired performance level. Each algorithm's priority is set based on its current time to solution, estimated according to a parametric model that is trained *and* used while solving a sequence of problems, gradually increasing its impact on the priority attribution. The use of *censored sampling* allows to train the model efficiently.

1 Motivation

Most solvable AI problems can be addressed by more than one algorithm; most AI algorithms feature a number of parameters that have to be set. Both choices can dramatically affect the quality of the obtained solution, and the time spent obtaining it. Algorithm Selection, or *Meta-Learning*, techniques [1, 2] typically address these questions by solving a large number of problems with each of the available algorithms, in order to learn a mapping from (*problem*,*algorithm*) pairs to expected performance. The obtained mapping is later used to select and run, for each new problem, only the algorithm that is expected to give the best results.

This approach, tough being preferable to the far more popular "trial and error", poses a number of problems. It presumes that such a mapping can be learned at all, i.e., that the actual algorithm performance on a given problem will be predictable with enough precision before even starting the algorithm — often not the case with stochastic algorithms, whose performance can exhibit large fluctuations among different runs. It also assumes problem instances met during the training phase to be statistically representative of successive ones. For these reasons, there usually is no way to detect a relevant discrepancy between expected and actual performance of the chosen algorithm. It also neglects computational complexity issues: ranking between algorithms is often based solely on the expected *quality* of the performance, and the time spent during the training phase is not even considered, although it can be large enough to cancel any practical advantage of algorithm selection.

An alternative, inspired by the *Algorithm Portfolio* paradigm [3], could consist in selecting a *subset* of the available algorithms, to be run in parallel, with the same priority, until the fastest one solves the problem. This simple scheme would be more robust, as it is less likely that performance estimates would be wrong for all selected

algorithms, but it would also involve an additional overhead, due to the "brute force" parallel execution of all candidate solvers.

In our view, a crucial weakness of these approaches is that they don't exploit any feedback from the actual execution of the chosen algorithms. We try to move a step in this direction, introducing *Dynamic Algorithm Portfolios*. Instead of *first* choosing a portfolio *then* running it, we iteratively *allocate* a time slice, sharing it among all the available algorithms, and *update the relative priorities* of the algorithms, based on their current state, in order to favor the most promising ones. Instead of basing the priority attribution only on performance quality, we fix a target performance, and try to minimize the time to reach it. To this aim, we search for a mapping from (*problem*,*algorithm*,*current algorithm state*) triples to *expected time* to reach the desired performance quality. To further reduce computational complexity, we focus on *lifelong-learning* techniques that drop the artificial boundary between training and usage, exploiting the mapping during training, and including training time in performance evaluation. In [4] we termed this approach *Adaptive Online Time Allocation* (AOTA), and introduced an example of a fixed heuristic mapping; in [5] we proposed a method to learn a probabilistic mapping while solving a problem sequence. In the following we briefly present some related work (Sect. 2); describe the AOTA framework (Sect. 3) and its current instantiations (Sect. 4). Sect. 5 reports new experimental results of a comparison with a static approach. Sect. 6 concludes the article with directions for future work.

2 Previous work

A number of interesting "dynamic" exceptions to the static algorithm selection paradigm can be found in literature (see the tech report version of [4] for a more exhaustive bibliography). In [6], algorithm recommendation is based on the performance of the candidate algorithms during a predefined amount of time, called the *observational horizon*. In *anytime algorithm monitoring* [7], the *dynamic performance profile* of a planning technique is updated according to its performance, in order to stop the planning phase when further improvements in the actions planned are not worth the time spent in evaluating them. The "Parameterless GA" [8] is a fixed heuristic time allocation technique for Genetic Algorithms. In [9], a system solves function inversion and time-limited optimization problems by searching in a space of problem solving techniques, allocating time to them according to their probabilities, and updating the probabilities according to positive and negative results on a sequence of problems. In a Reinforcement Learning [10] setting, algorithm selection can be formulated as a Markov Decision Process: in [11], 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 an online algorithm selection policy; in [12], a set of deterministic algorithms is considered, and, under some limitations, static and dynamic algorithm selection techniques based on dynamic programming are presented. Success Story algorithms [13] can undo policy modifications that did not improve the reward rate.

Literature on algorithm portfolios is usually focused on choice criteria for building the set of candidate solvers such that their areas of good performance don't overlap. The portfolio is then executed in parallel [3] or used as a pool for algorithm selection [14]. Other interesting research areas that can be related to "static" algorithm selection include landmarking [15], anytime algorithm scheduling [16], time limited planning [17], bandit problem solving [18], racing [19], search in program space [20].

3 AOTA framework

Consider a *sequence* B of m problem instances $b_1, b_2, \ldots, b_m$, roughly sorted in increasing order of difficulty, and featuring precise stopping criteria (e.g., search problems in which the solution is known to exist and can be recognized; optimization problems in which a reachable target value for performance is given); and a set A of n algorithms $a_1, a_2, \ldots, a_n$, that can be applied to the solution of the problems in B, paused and resumed at any time, and queried, at a negligible cost, for state information $\mathbf{d} \in \mathbb{R}^d$ related to their progress in solving the current instance. We aim at minimizing the time to solve the whole problem sequence B. To describe the state of a Dynamic Algorithm Portfolio (DAP), let t_i be the time already spent on a_i, τ_i the current *estimate* of the time still needed by a_i to solve the current problem, $\mathbf{x}_i$ a feature vector, possibly including information about the current problem instance, the algorithm a_i itself (e.g., its kind, the values of its parameters), and its current state $\mathbf{d}_i$; $H_i = \{(\mathbf{x}_i^{(r)}, t_i^{(r)}), r = 0, \ldots, h_i\}$ a set of collected samples of these pairs, f_τ a model that maps histories H_i to estimated τ_i.

If the model f_τ was precise enough, we would not need to run more than one algorithm, the a_i that is mapped to a lower τ before its start ($t_i = 0$): it is instead more realistic to assume that the model's estimates are rough, but can be improved by collecting more data in H_i, i.e., by getting more run-time feedback on the actual performance of a_i on current problem instance. We then introduce a set of nonnegative scalars $P_A = \{p_1, .., p_n\}, p_i \geq 0, \sum_{i=1}^{n} p_i = 1$, that represent the current *bias* of the portfolio, slice machine time with a small interval Δt, and iteratively share each time slice between the algorithms proportionally to the current bias. Before each iteration, the bias is updated according to a function f_P of $\{\tau_i\}$, that should obviously give more time to expected faster a_i (i.e., the ones with a low τ_i); after a share $p_i \Delta t$ has expired, τ_i is updated based on current H_i.In *intra-problem* AOTA, the predictive model f_τ is fixed; in *inter-problem* AOTA, f_τ itself is adaptive, and gets updated after each problem's solution.

Figure 1: A pseudocode for inter-problem AOTA

```
For each problem b_k
    initialize {τ_i}
    While (b_k not solved)
        update P_A = f_P({τ_i})
        For each algorithm a_i
            run a_i for p_i Δt
            update H_i = H_i ∪ (x_i, t_i)
            update τ_i = f_τ(H_i)
        End
    End
    update f_τ = F{H_i}
End
```

Figure 1 displays pseudocode for the AOTA framework: example instantiations for f_τ, f_P and $\mathcal{F}$ are given in the next section.

4 Example AOTAs

In [4] we presented a fixed heuristic f_τ. We considered algorithms with a scalar state x, that had to reach a target value: H_i in this case is a simple *learning curve*. Through a shifting window linear regression, we extrapolated for each i the time $t_{i,sol}$ at which the current learning curve H_i would reach the target value, in order to estimate the time to solution $\tau_i = t_{i,sol} - t_i$. Even though the estimates were obviously optimistic, they were updated so often that the overall performance of the intra-problem AOTA was remarkably good; its obvious limitations were that it required some prior knowledge about the algorithms, and a simple relationship between the learning curve and the time to solution.

What if we instead want to *learn* a potentially complex mapping f_τ from scratch? For a successful algorithm a_i that solved the problem at time $t_i^{(h_i)}$, we can *a posteriori* evaluate the correct $\tau_i^{(r)} = t_i^{(h_i)} - t_i^{(r)}$ for each pair $(\mathbf{x}_i^{(r)}, t_i^{(r)})$ in H_i. In a first tentative experiment, that led to poor results, these values were used as targets to learn a regression from pairs $(\mathbf{x}, t)$ to residual time values τ. The main problem with this approach is which τ values to choose as targets for the *unsuccessful* algorithms. Assigning them heuristically would penalize with high τ values algorithms that were stopped on the point of solving the task, or give incorrectly low values to algorithms that cannot solve it; obtaining more exact targets τ by running more algorithms until the end, and "capping" runs of unsuccessful or poorly performing algorithms to high time values, as in [14], would allow to obtain more precise models, but at the expense of increasing the overhead of training it.

The alternative we presented in [5] is inspired by *censored sampling* for lifetime distribution estimation [21], and consists in learning a parametric model $g(\tau|\mathbf{x}_i, t_i; \mathbf{w})$ of the conditional probability density function (pdf) of the residual time τ. To see how the model can be trained, imagine we continue running the portfolio for a while after the first algorithm solves the current task, such that we end up having one or more successful algorithms a_i, for whose H_i the correct targets $\tau_i^{(r)}$ can be evaluated as above. Assuming each $\tau_i^{(r)}$ to be the outcome of an independent experiment, including t in $\mathbf{x}$ to ease notation, if $p(\mathbf{x})$ is the (unknown) pdf of the $\mathbf{x}_i^{(r)}$ we can write the likelihood of H_i as

$$\mathfrak{L}_{succ}(H_i) = \prod_{r=0}^{h_i-1} g(\tau_i^{(r)}|\mathbf{x}_i^{(r)}; \mathbf{w}) p(\mathbf{x}_i^{(r)}) \tag{1}$$

For the unsuccessful algorithms, the final time value $t_i^{(h_i)}$ recorded in H_i is a lower bound on the unknown, and possibly infinite, time to solve the problem, and so are the $\tau_i^{(r)}$, so to obtain the likelihood we have to integrate (1)

$$\mathfrak{L}_{fail}(H_i) = \prod_{r=0}^{h_i-1} [1 - G(\tau_i^{(r)}|\mathbf{x}_i^{(r)}; \mathbf{w})] p(\mathbf{x}_i^{(r)}) \tag{2}$$

where $G(\tau|\mathbf{x}; \mathbf{w}) = \int_0^\tau g(\xi|\mathbf{x}; \mathbf{w}) d\xi$ is the conditional cumulative distribution function (cdf) corresponding to g.

We can then search the value of $\mathbf{w}$ that maximizes $\mathfrak{L}(H) = \prod_i \mathfrak{L}(H_i)$, or, in a Bayesian approach, maximize the posterior $p(\mathbf{w}|H) \propto \mathfrak{L}(H|\mathbf{w}) p(\mathbf{w})$. Note that in both cases the logarithm of these quantities can be maximized, and terms not in $\mathbf{w}$ can be dropped.

To prevent overfitting, and force the model to have a realistic shape, we can use some known parametric lifetime model, such as an Extreme Value distribution $g(l|\mathbf{x}, t; \mathbf{w}) = \frac{1}{\delta} e^{\{[(l-\eta)/\delta] - e^{(l-\eta)/\delta}\}}$ on the logarithm $l = \log \tau$ of time values [21] and express the dependency on $\mathbf{x}$ and $\mathbf{w}$ in its two parameters $\eta = \eta(\mathbf{x}; \mathbf{w})$, $\delta = \delta(\mathbf{x}; \mathbf{w})$: these can in turn be the two outputs of a more complex parametric model, with input $\mathbf{x}$, and parameter $\mathbf{w}$, whose value can be optimized by gradient descent, minimizing the negative logarithm of the resulting $\mathfrak{L}(H)$, in a fashion that is common for modeling conditional distributions (see, e.g., [22], par 6.4).

One advantage of this approach is that it fully exploits the state history information gathered, as it allows to learn from the unsuccessful algorithms as well.

For f_P, one reasonable heuristic, that gave good results, consists in assigning $1/2$ of the current time slice to the expected fastest algorithm (i.e., the one with lowest τ_i), $1/4$ to the second fastest, and so on. This heuristic cannot be directly applied to inter-problem AOTA, though, as the model would obviously be unreliable during the first problems of the sequence. In this case it would be better to start the problem sequence with a "brute force" f_P ($p_i = 1/n$), and vary it gradually towards the above described "ranking" f_P ($p_i = 2^{-r_i}$, r_i being the current rank of a_i based on $\{\tau_i\}$).

5 Experiments

We present experimental results with the same A and B as in [5]: A a set of 76 simple generational Genetic Algorithms [23], differing in population size ($2^i, i = 1..19$), mutation rate (0 or $0.7/L$, L being the genome length) and crossover operator (uniform or one-point, with rate 0.5 in both cases); B a sequence 21 of artificial deceptive problems, such as the "trap" described in [8], consisting of n copies of an m-bit trap function: each m-bit block of a bitstring of length nm gives a fitness contribution of m if all its bits are 1, and of $m - q$ if $q < m$ bits are 1. The genome length varies from 30 to 96 and the size m of the deceptive block from 2 to 4. The problems were sorted based on a rough estimate of their difficulty, from 15 blocks of size 2 to 24 blocks of size 4. The feature vectors $\mathbf{x}$ included two problem features (genome length and block size), the algorithm parameters, the current best and average fitness values and the time spent, together with their last variations, for a total of 11 inputs.

To model g we used an Extreme Value distribution on the logarithms of time values with parameters $\eta(\mathbf{x}; \mathbf{w})$ and $\delta(\mathbf{x}; \mathbf{w})$ being quadratic expansions of $\mathbf{x}$ of the form $w_0 + \sum_i w_i x_i + \sum_{i,j} w_{i,j} x_i x_j$. The weights were obtained by maximizing the Bayesian posterior described in sect. 4, using a Cauchy distribution $p(w) = 1/(1 + w)^2$ as a prior.

As for f_P, p_i was set proportionally to $(2 - \frac{\log(m+1-j)}{\log(m)})^{-r_i}$, r_i being the current rank of a_i in order of increasing τ_i, j the index of current task, m the total number of tasks. In this way the distribution of time is uniform during the first task (when the model is still untrained), and tends through the task sequence to the "ranking" f_P described in Sect. 4 ($p_i = 2^{-r_i}$), in which the expected fastest solver gets half of the current time slice. After the solution of each task, a further fraction of the time spent is allocated to the remaining algorithms, gathering more data in their histories in order to improve the model: this fraction is also varied during the task sequence, from 1 to 0, linearly. Results were similar to the ones obtained in [5] with a slightly different AOTA, employing a Neural Network as parametric model.

To compare with a more traditional algorithm selection technique, analogous to,

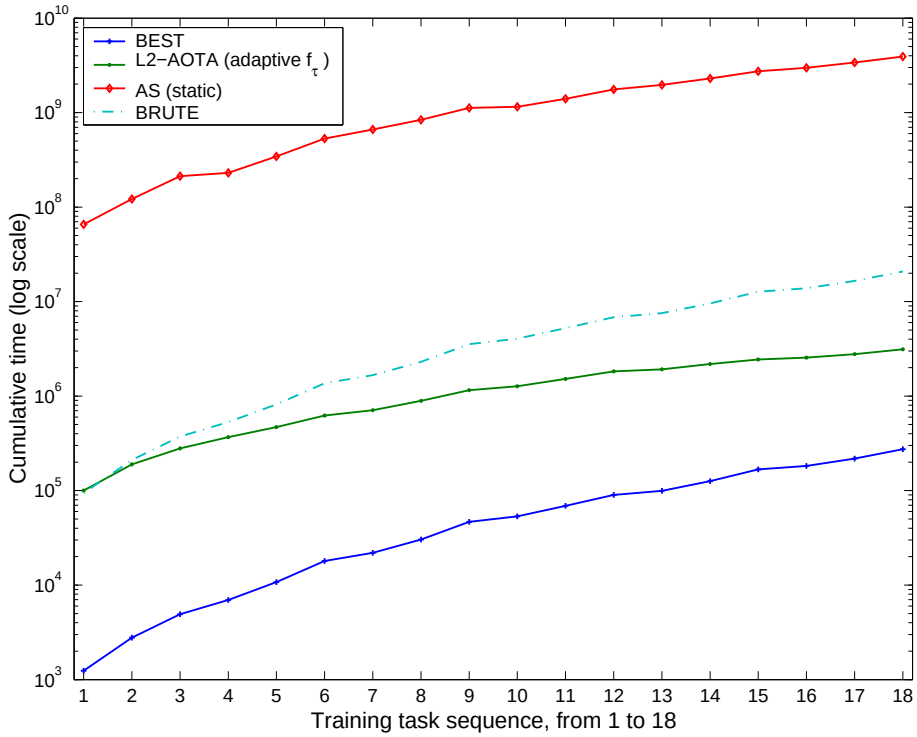

Figure 2: The cumulative time, i.e., the time spent so far, on the sequence of training tasks by the adaptive f_τ method, labeled L2-AOTA, compared with the static algorithm selection AS, which requires solving each problem with every algorithm; the best algorithm in the set (BEST), and the brute-force approach (BRUTE). Time is measured in fitness function evaluations, values shown are upper 95% confidence limits calculated on 20 runs. A logarithmic scale is used on time values.

e.g., [14], we also trained a static model of algorithm performance, gathering runtime data for all 76 algorithms on the first 18 problems: this data was then used to perform a simple linear regression from a quadratic expansion of problem and algorithm features to the logarithm of runtime values, capping runs of the unsuccessful algorithms. Dynamic features related to the state of the algorithms were obviously not considered in this case. The obtained model was then used to select and run a single algorithm for each of the remaining 3 problems.

Figure 2 compares training time of this static approach, labeled AS, with the one of the adaptive f_τ approach, labeled L2-AOTA; we also display the performance of the (usually different at each run) fastest solver of the set, labeled BEST, which would be the performance of an ideal algorithm selection with "foresight" of the correct τ_i values at $t_i = 0$; and the estimated performance of a brute force approach, i.e., running all the algorithms in parallel until one solves the problem. Note that this latter is much more effective than the training of 'AS', which requires solving each problem with every algorithm.

Figure 3 plots the performance on the three remaining problems, with deceptive block sizes of 2,3 and 4 respectively. Learning was turned off in $L2-AOTA$ to allow a comparison with AS. The latter had a better performance on tasks 19 and 21,

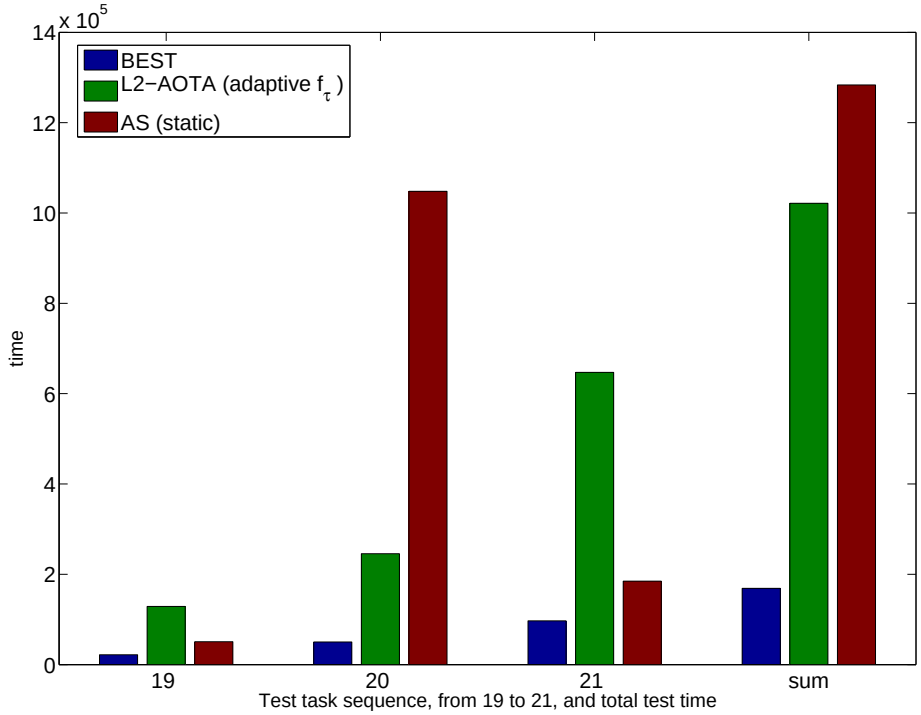

Figure 3: The times spent on the last three tasks of the sequence, and their sum. The overall performance of the dynamic approach is better, notwithstanding a difference of three orders of magnitude in training time.

but much worse on task 20, for which the algorithm runtimes were apparently more difficult to predict, so the overall performance of $L2-AOTA$ was better, even though its training time was more than three orders of magnitude faster.

6 Conclusions and future work

The main novelties of our method consist in the "lifelong-learning" approach to algorithm selection, and in the use of censored sampling to update a parametric conditional distribution of algorithm runtime. The particular bias update strategy f_P adopted allowed to use the model *while* training it, gradually increasing its impact on the time allocation as more problems were solved, and more historic data gathered, and practically saving computation time by limiting the learning process to "interesting" parts of the training space; The idea of performing some sort of algorithm selection based on runtime interaction with the algorithms is not completely new (see Sect. 2), but we consider it as an interesting line of research.

We advocate online time allocation with sets of computationally expensive algorithms, whose performance is not easily predictable. For faster algorithms, a more refined approach should also take into account the cost of updating the model; for domains in which algorithm performance can be easily predicted, traditional "offline" algorithm selection techniques can be sufficient, even though the "online" approach can still be useful to reduce training time.

At the moment, the main limitations of our approach, that still prevent its use in more challenging situations, are the use of "batch" learning for training the model, which should be replaced by an online technique, as it obviously limits the number of problems; and the need of sorting the problems by difficulty. This latter can be relaxed, e.g., by performing a round robin on the available tasks, trying to solve each of them for some amount of time, until the easiest one is solved by brute force, after which the model is first updated, and so on. Other prior knowledge, is required for the choice of the algorithms in A and of the features to include in $\mathbf{x}$. The parametric model employed can give predictions also before starting the algorithms (i.e., for $t_i = 0$), so it could in principle be used to adapt a redundant algorithm set A to the current problem, guiding the choice of a set of promising points in parameter space, or even to pick a single algorithm when the selection is easy enough. *Feature selection* techniques could also help automate the procedure. Another limitation is that the user has to set a target performance quality for each problem. While this might be natural for certain problems, in many cases it would be preferable to set instead a parameter quantifying how much extra time should be spent relative to the improvement in the solution. This would require learning a more complex model of *performance profile*, as in [7].

In future work we plan to address these limitations; ongoing research is focussed on online training techniques, in order to allow for larger experiments with different algorithm set/problem sequence combinations.

Acknowledgements. This work was supported by SNF grant 200020-107590/1.

References

[1] Rice, J.R.: The algorithm selection problem. In Rubinoff, M., Yovits, M.C., eds.: Advances in computers. Volume 15. Academic Press, New York (1976) 65–118

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

[3] Gomes, C.P., Selman, B.: Algorithm portfolios. Artificial Intelligence **126** (2001) 43–62

[4] Gagliolo, M., Zhumatiy, V., Schmidhuber, J.: Adaptive online time allocation to search algorithms. In Boulicaut, J.F., Esposito, F., Giannotti, F., Pedreschi, D., eds.: Machine Learning: ECML 2004. Proceedings of the 15th European Conference on Machine Learning, Pisa, Italy, September 20-24, 2004, Springer (2004) 134–143 — Extended tech. report available at `http://www.idsia.ch/idsiareport/IDSIA-23-04.ps.gz`.

[5] 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.: Artificial Neural Networks: Formal Models and Their Applications - ICANN 2005, 15th International Conference, Warsaw, Poland, September 11-15, 2005, Proceedings, Part 2. Volume 3697 of Lecture Notes in Computer Science., Springer (2005) 7–12

[6] Horvitz, E., Ruan, Y., Gomes, C.P., Kautz, H.A., Selman, B., Chickering, D.M.: A bayesian approach to tackling hard computational problems. In: UAI '01: Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc. (2001) 235–244

[7] Hansen, E.A., Zilberstein, S.: Monitoring and control of anytime algorithms: A dynamic programming approach. Artificial Intelligence **126** (2001) 139–157

[8] Harick, G.R., Lobo, F.G.: A parameter-less genetic algorithm. In Banzhaf, W., Daida, J., Eiben, A.E., Garzon, M.H., Honavar, V., Jakiela, M., Smith, R.E., eds.: Proceedings of the Genetic and Evolutionary Computation Conference. Volume 2., Orlando, Florida, USA, Morgan Kaufmann (1999) 1867

[9] Solomonoff, R.J.: Progress in incremental machine learning. Technical Report IDSIA-16-03, IDSIA (2003)

[10] Kaelbling, L., Littman, M., Moore, A.: Reinforcement learning: a survey. Journal of AI research **4** (1996) 237–285

[11] Lagoudakis, M.G., Littman, M.L.: Algorithm selection using reinforcement learning. In: Proc. 17th International Conf. on Machine Learning, Morgan Kaufmann, San Francisco, CA (2000) 511–518

[12] Petrik, M.: Statistically optimal combination of algorithms (2005) Presented at SOFSEM 2005 - 31st Annual Conference on Current Trends in Theory and Practice of Informatics - Liptovsky Jan, Slovak Republic, January 22 - 28.

[13] Schmidhuber, J., Zhao, J., Wiering, M.: Shifting inductive bias with success-story algorithm, adaptive Levin search, and incremental self-improvement. Machine Learning **28** (1997) 105–130 — Based on: Simple principles of metalearning. TR IDSIA-69-96, 1996.

[14] Leyton-Brown, K., Nudelman, E., Shoham, Y.: Learning the empirical hardness of optimization problems: The case of combinatorial auctions. In: ICCP: International Conference on Constraint Programming (CP), LNCS. (2002)

[15] Pfahringer, B., Bensusan, H., Giraud-Carrier, C.: Meta-learning by landmarking various learning algorithms. In: Proceedings of the Seventeenth International Conference on Machine Learning, ICML'2000, Morgan Kaufmann (2000) 743–750

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

[17] Horvitz, E.J., Zilberstein, S.: Computational tradeoffs under bounded resources (editorial). Artificial Intelligence **126** (2001) 1–4 Special Issue.

[18] Berry, D.A., Fristedt, B.: Bandit Problems: Sequential Allocation of Experiments. Chapman and Hall, London (1985)

[19] Moore, A.W., Lee, M.S.: Efficient algorithms for minimizing cross validation error. In: Proceedings of the 11th International Conference on Machine Learning, Morgan Kaufmann (1994) 190–198

[20] Schmidhuber, J.: Optimal ordered problem solver. Machine Learning **54** (2004) 211–254 Short version in *NIPS 15*, p. 1571–1578, 2003.

[21] Nelson, W.: Applied Life Data Analysis. John Wiley, New York (1982)

[22] Bishop, C.M.: Neural networks for pattern recognition. Oxford University Press (1995)

[23] Holland, J.H.: Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor (1975)