Towards Life-Long Meta Learning

Matteo Gagliolo † **Jürgen Schmidhuber** †‡
†IDSIA, Galleria 2, 6928 Manno-Lugano, Switzerland
‡TU Munich, Boltzmannstr. 3, 85748 Garching, München, German
{matteo,juergen}@idsia.ch

Abstract

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 its current time to solution, estimated according to a parametric model that is trained *and* used while solving a sequence of problems.

1 Motivation

Meta-Learning techniques typically require a long training phase, during which a large number of problems is repeatedly solved with each of the available algorithms, in order to learn a mapping from (*problem*,*algorithm*) pairs to expected performance, to be used for algorithm selection. This approach poses a number of problems. It presumes that such a mapping can be learned at all, i.e. that the actual performance of an algorithm on a given problem will be predictable with enough precision before even starting it. 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. The *Algorithm Portfolio* paradigm [1] consists 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 is more robust, as it's less likely that performance estimates will be wrong for all selected algorithms, but it requires the same expensive training procedure, and also involves 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 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 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 [2, 3] we termed this approach *Adaptive Online Time Allocation* (AOTA).

2 Previous work

A number of interesting "dynamic" exceptions to the otherwise static algorithm selection paradigm can be met in literature (see the techrep version of [2] for a more exhaustive bibliography). In [4], 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* [5], 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" [6] is a fixed heuristic time allocation technique for Genetic Algorithms. In a Reinforcement Learning setting, algorithm selection can be formulated as a Markov Decision Process: in [7], 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 [8], a set of deterministic algorithms is considered, and, under some limitations, static and dynamic algorithm selection techniques based on dynamic programming are presented.

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 obviously gives 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 (Fig. 3). 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.

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 is 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.

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 based on {H_i}
End
```

4 Example AOTAs and experiments

In [2] 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. The alternative we presented in [3] is inspired by *censored sampling* for lifetime distribution estimation, and consists in learning a parametric model $g(\tau|\mathbf{x}; \mathbf{w})$ of the conditional probability density function (pdf) of the residual time τ. 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. The model was obtained by training a neural network to map $\mathbf{x}$ values to the two parameters of an Extreme Value distribution of the time to solution, on data collected while solving a sequence B of 21 deceptive problems, with a set A of 76 different Genetic Algorithms. In Fig. 2 we compare the NN model (NN-AOTA) from [3] with a simpler one, a quadratic expansions of $\mathbf{x}$ of the form $w_0 + \sum_i w_i x_i + \sum_{i,j} w_{i,j} x_i x_j$ (L2-AOTA). The average ratio between the time spent by the whole portfolio and the (usually different at each run and on each task) best element in the set was about 11 for the fixed f_τ and 8 for the adaptive f_τ AOTA. This latter would be e.g. the performance of an *already trained* "static" Algorithm Portfolio that picked, for each problem, 8 of the 76 algorithms, always including the fastest: to fairly compare with such a technique, though, we should also consider its additional training time.

We advocate the use of Dynamic Algorithm Portfolios with sets of computationally expensive algorithms. For faster ones, a more refined approach should also take into account the cost of updating the model. The model f_τ was trained on all historic data gathered so far, in a "batch learning" approach: for longer problem sequences, an online method would obviously be preferable, in order to obtain a scalable *life-long* meta learning technique. In future work we plan to address these and other limitations; ongoing experiments focus on

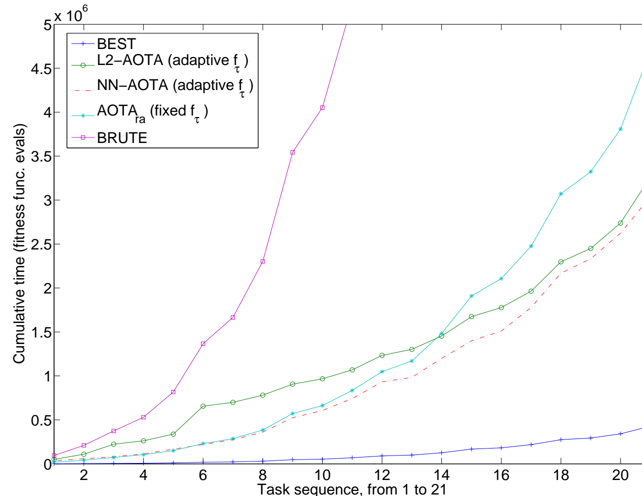

Figure 2: The cumulative time spent on the sequence of tasks by the adaptive f_τ method, with a neural network (NN-AOTA) and a quadratic model (L2-AOTA), compared with the fixed f_τ from [2] this time with ranking f_P (AOTA$_{ra}$). Also shown are the the performances of the (usually different at each run and on each task) fastest solver of the set, (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 (BRUTE), i.e. running all the algorithms in parallel until one solves the problem, which leaves the figure and completes the task sequence at time 3.3×10^7. Time is measured in fitness function evaluations, values shown are upper 95% confidence limits calculated on 20 runs.

alternative parametric models, and different algorithm set/problem sequence combinations.

References

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

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

[3] M. Gagliolo and J. Schmidhuber. A neural network model for inter-problem adaptive online time allocation. In W. Duch et al., editor, *ICANN 2005, Proceedings, Part 2*, pages 7–12, 2005.

[4] E. Horvitz, Y. Ruan, C. P. Gomes, H. A. Kautz, B. Selman, and D. Maxwell Chickering. A bayesian approach to tackling hard computational problems. In *UAI '01*, pages 235–244, 2001.

[5] E. A. Hansen and S. Zilberstein. Monitoring and control of anytime algorithms: A dynamic programming approach. *Artificial Intelligence*, 126(1–2):139–157, 2001.

[6] G. R. Harick and F. G. Lobo. A parameter-less genetic algorithm. In W. Banzhaf et al., editor, *GECCO*, volume 2, 1999.

[7] M. G. Lagoudakis and M. L. Littman. Algorithm selection using reinforcement learning. In *Proc. 17th ICML*, pages 511–518, 2000.

[8] M. Petrik. Statistically optimal combination of algorithms. SOFSEM 2005.