A Neural Network Model for Inter-problem Adaptive Online Time Allocation

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

[1] IDSIA, Galleria 2, 6928 Manno-Lugano, Switzerland
[2] TU Munich, Boltzmannstr. 3, 85748 Garching, München, Germany
{matteo, juergen}@idsia.ch

Abstract. One aim of Meta-learning techniques is to minimize the time needed for problem solving, and the effort of parameter hand-tuning, by automating algorithm selection. The predictive model of algorithm performance needed for task often requires long training times. We address the problem in an *online* fashion, running multiple algorithms in parallel on a sequence of tasks, continually updating their relative priorities according to a neural model that maps their current state to the expected time to the solution. The model itself is updated at the end of each task, based on the actual performance of each algorithm. Censored sampling allows us to train the model effectively, without need of additional exploration after each task's solution. We present a preliminary experiment in which this new *inter*-problem technique learns to outperform a previously proposed *intra*-problem heuristic.

1 Problem Statement

A typical machine learning scenario involves a (possibly inexperienced) practitioner trying to cope with a set of problems, that could be solved, in principle, using one element of a set of available algorithms. While most users still solve such dilemmas by trial and error, or by blindly applying some unquestioned rule-of-thumb, the steadily growing area of *Meta-Learning* [1] research is devoted to automating this process. Apart from a few notable exceptions (e.g. [2,3,4,5], see [6], of which we adopt the notation and terminology, for a commented bibliography), most existing techniques amount to the selection of a single candidate solver (e.g. *Algorithm recommendation* [7]), or a small subset of the available algorithms to be run in parallel with the same priority (e.g. *Algorithm portfolio selection* [8]). This approach usually requires a long training phase, which can be prohibitive if the algorithms at hand are computationally expensive; it also assumes that the algorithm runtimes can be predicted *offline*, based on problem features, and do not exhibit large fluctuations. In more complex cases, where the difficulty of the problems cannot be precisely predicted a priori, a more robust approach would be to run the candidate solvers in parallel, adapting their priorities *online* according to their actual performance. We termed this *Adaptive Online Time Allocation* (AOTA) in [6], in which we further distinguish between *intra*-problem AOTA, where the prediction of algorithm performance is made according to some heuristic based on a-priori knowledge about the algorithm's behavior; and *inter*-problem AOTA, in which a time allocation strategy is learned by collecting experience on a sequence of tasks.

W. Duch et al. (Eds.): ICANN 2005, LNCS 3697, pp. 7–12, 2005.

In this work we present an *inter*-problem approach for training a parametric model of algorithm runtimes, and give an example of how this model can be used to allocate time online, comparing its performance with the simple *intra*-problem heuristic from [6].

2 A Parametric Model for Inter-problem AOTA

Consider a finite algorithm set A containing n algorithms a_i, $i \in I = \{1, \dots, n\}$, applied to the solution of the same problem and running according to some time allocation procedure. Let t_i be the time spent on a_i; $\mathbf{x}_i$ a feature vector, possibly including information about the current problem, the algorithm a_i itself (e.g. its kind, the values of its parameters), and its current state d_i; $H_i = \{(\mathbf{x}_i^{(r)}, t_i^{(r)}), r = 0, \dots, h_i\}$ a set of collected samples of these pairs; $H = \cup_{i \in I} H_i$ the historic experience set relative to the entire A.

In order to allocate machine time efficiently, we would like to map each pair in each H_i to the time τ_i still left before a_i reaches the solution. If we are allowed to learn such mapping by solving a sequence of related tasks, we can, for a successful algorithm a_i that solved the problem at time $t_i^{(h_i)}$, *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 would increase the overhead.

The alternative we present here is inspired by *censored sampling* for lifetime distribution estimation [9], 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 the time allocation for a while after the first algorithm solves the current task, such that we end up having one or more successful algorithms a_i, with indices $i \in I_s \subseteq I$, 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

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

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)

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

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 $\mathfrak{L}(H) = \prod_{i\in I}\mathfrak{L}(H_i)$, 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 to force the model to have a realistic shape, we can use some known parametric lifetime model, such as a Weibull distribution [9], with pdf $g(\tau|\mathbf{x},t;\mathbf{w}) = \lambda^\beta\beta\tau^{\beta-1}e^{-(\lambda\tau)^\beta}$ and express the dependency on $\mathbf{x}$ and $\mathbf{w}$ in its two parameters $\lambda = \lambda(\mathbf{x};\mathbf{w})$,$\beta = \beta(\mathbf{x};\mathbf{w})$. In the example we present here, these will be the two outputs of a feed-forward neural network, which will be trained by back-propagation minimizing the negative logarithm of $\mathfrak{L}(H)$, whose derivatives are easily obtainable, in a fashion that is commonly used for modelling conditional distributions (see e.g. [10], par 6.4).

From the time allocation perspective, one advantage of this approach is that it allows to learn also from the unsuccessful algorithms, suffering less from the trade-off between the accuracy of the learned model, and the time spent on learning it.

3 An Example Application

If the estimated model g was the correct one, the time allocation task would be trivial, as we could allocate all resources to the expected fastest algorithm, i.e., the one with lower expected run time $\int_0^{+\infty}\tau g(\tau|\mathbf{x})d\tau$, periodically re-checking which algorithm is to be selected given the current states $\{\mathbf{x}_i\}$. In practice, however, the predictive power of the model depends on the how the current task compares to the ones solved so far, so trusting it completely would be too risky. In preliminary experiments, we adopted a time allocation technique similar to the one in ([6]), slicing machine time in small intervals ΔT, and sharing each ΔT among elements of A according to a distribution $P_A = \{p_i\}$; the latter is updated at each step based on the current model g, which is re-trained at the end of each task on the whole history H collected so far, as follows:

```
for each problem r
   while (r not solved)
      update {τ_i} based on current g and current {x_i}:
         τ_i = ∫_0^{+∞} τ g(τ|x_i) dτ
      update P_A = {p_i} based on {τ_i}
      for each i = 1..n
         run a_i for a time p_i ΔT
         update x_i
      end
   end
   update H
   update g maximizing 𝔏(H)
end
```

To model g we used an Extreme Value distribution[1] on the logarithms of time values, with parameters $\eta(\mathbf{x};\mathbf{w})$ and $\delta(\mathbf{x};\mathbf{w})$ being the two outputs of a feedforward neural network, with two separate hidden layers of 32 units each, whose weights are obtained by minimizing the negative logarithm of the Bayesian posterior $p(\mathbf{w}|H)$ obtained in Sect. 2, using 20% of the current history H as a validation set, and a Cauchy distribution $p(w) = 1/1 + w^2$ as a prior.

At each cycle of the time allocation, the current expected time τ_i to the solution is evaluated for each a_i from $g(\tau|\mathbf{x}_i;\mathbf{w})$; these values are ranked in ascending order, and the current time slice is allocated proportionally to $(\frac{\log(m+1-j)}{\log(m)})^{-r_i}$, r_i being the current rank of a_i, m the total number of tasks, j the index of current task (from 1 to m). 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 a sharing pattern in which the expected fastest solver gets half of the current time slice, the second one gets one quarter, and so on. We ran some preliminary tests, using the algorithm set A_3 from [6], a set of 76 simple generational Genetic Algorithms [11], 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). We applied these solvers to a sequence of artificial deceptive problems, such as the "trap" described in [3], 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. We generated a sequence of 21 different problems, varying the genome length from 30 to 96 and the size of the deceptive block from 2 to 4. The problems were first sorted by genome length, then by block size, such that the resulting sequence is roughly sorted by difficulty (see Table 1). The feature vector $\mathbf{x}$ included two problem features (genome length and block size), the algorithm parameters, the current best and average fitness values, together with their last variation and their current trend, the time spent and its last increment, for a total of 13 inputs.

We compared the presented inter-problem AOTA with the intra-problem AOTA$_{ga}$, the most competitive from [6], in which the $\{\tau_i\}$ were heuristically estimated based on a simple linear extrapolation of the learning curve. In figure 1 we show the significant improvement over AOTA$_{ga}$, which by itself already greatly reduces computation time with respect to a brute-force approach.

4 Conclusions and Future Work

The purpose of this work was to show that a parametric model of algorithm performance can be learned and used to allocate time efficiently, without requiring a long training phase. Thanks to the model, the system was able to learn the bits of a-priori knowledge that we had to pre-wire in the *intra*-problem AOTA$_{ga}$: for example, the fact that increases in the average fitness are an indicator of potentially good performance. Along the sequence of tasks, the model gradually became more reliable, and NN-AOTA was

[1] If τ is Weibull distributed, $l = \log\tau$ has Extreme Value distribution $g(l) = \frac{1}{\delta}e^{\{[(l-\eta)/\delta]-e^{(l-\eta)/\delta}\}}$, with parameters $\delta = 1/\beta$, $\eta = -\log\lambda$. The distribution of the logarithm of residual times was used to learn a common model for a set of tasks whose solution times have different orders of magnitude.

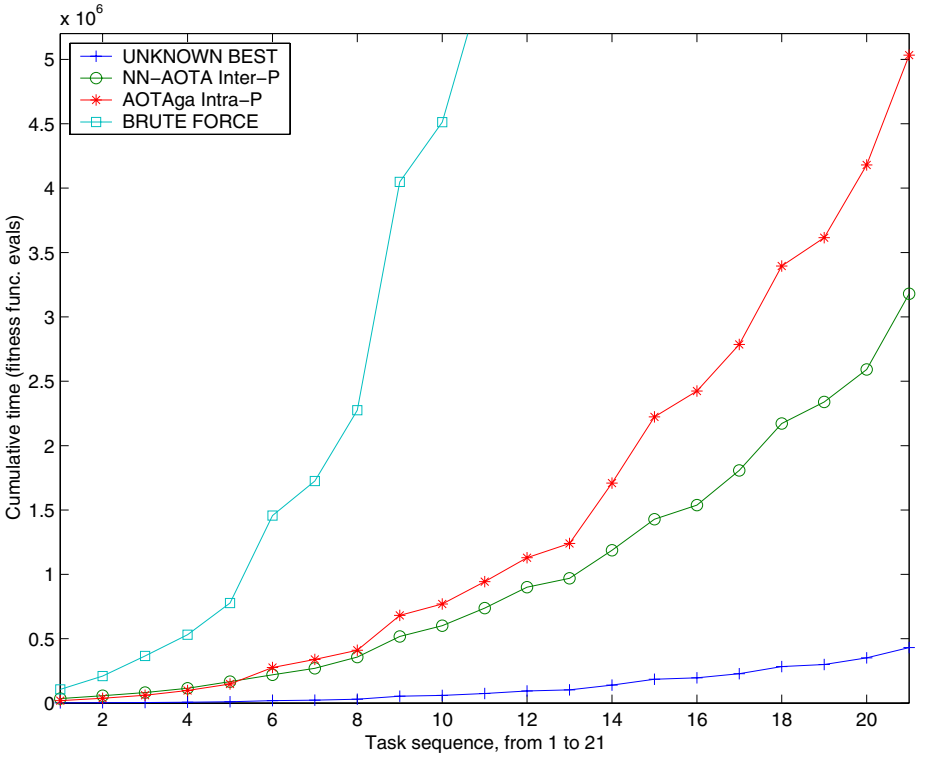

Fig. 1. A comparison between the presented method, labeled NN-AOTA Inter-P, and the intra-problem AOTA$_{ga}$, on a sequence of 21 tasks. Also shown are the the performances of the (a priori unknown, different for each problem and for each random seed) fastest solver of the set (which would be the performance of an ideal AOTA with foresight), labeled UNKNOWN BEST, and the estimated performance of a brute force approach (running all the algorithms in parallel until one solves the problem), labeled BRUTE FORCE, which leaves the figure and completes the task sequence at time 3.3×10^7. The cumulative time spent on the sequence of tasks, i.e. the total time spent in solving the current and all previous tasks, is plotted against current task index. Time is measured in fitness function evaluations; values shown are upper 95% confidence limits calculated on 20 runs.

Table 1. The 21 trap problems used, each listed with its block size m and number of blocks n

	m	n		m	n		m	n
1)	2	15	8)	3	16	15)	4	18
2)	3	8	9)	4	12	16)	2	40
3)	4	6	10)	2	30	17)	3	28
4)	2	20	11)	3	20	18)	4	21
5)	3	12	12)	4	15	19)	2	45
6)	4	9	13)	2	35	20)	3	32
7)	2	25	14)	3	24	21)	4	24

finally able to outperform AOTA$_{ga}$. In spite of the size of the network used, the obtained model is not very accurate, due to the variety of the algorithms behavior on the different tasks; still, it is discriminative enough to be used to rank the algorithms according to their expected runtimes.

The neural network can be replaced by any parametric model whose learning algorithm is based on gradient descent: in future work, we plan to test a more complex mixture model [12], in order to obtain more accurate predictions, and even better performances.

As the obtained model is continuous, and can give predictions also before starting the algorithms (i.e. for $t_i = 0$), it could in principle be used to adapt also the algorithm set A to the current task, guiding the choice of a set of promising points in parameter space.

Acknowledgements. This work was supported by SNF grant 16R1GSMLR1.

References

1. Vilalta, R., Drissi, Y.: A perspective view and survey of meta-learning. Artif. Intell. Rev. **18** (2002) 77–95
2. 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.
3. 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
4. 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
5. 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
6. 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`.
7. Fürnkranz, J., Petrak, J., Brazdil, P., Soares, C.: On the use of fast subsampling estimates for algorithm recommendation. Technical Report TR-2002-36, Österreichisches Forschungsinstitut für Artificial Intelligence, Wien (2002)
8. Gomes, C.P., Selman, B.: Algorithm portfolios. Artificial Intelligence **126** (2001) 43–62
9. Nelson, W.: Applied Life Data Analysis. John Wiley, New York (1982)
10. Bishop, C.M.: Neural networks for pattern recognition. Oxford University Press (1995)
11. Holland, J.H.: Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor (1975)
12. Jacobs, R.A., Jordan, M.I., Nowlan, S.J., Hinton, G.E.: Adaptive mixtures of local experts. Neural Computation **3** (1991) 79–87