

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

Matteo Gagliolo, Jürgen Schmidhuber
{matteo, juergen}@idsia.ch



*IDSIA – Istituto Dalle Molle
di Studi sull'Intelligenza Artificiale*
Lugano, Switzerland
<http://www.idsia.ch>

Roadmap

- What do we mean by AOTA
- Why/when would that be useful
- Previous/related work
- Inter-Problem AOTA
- Experiments

An example AOTA: the Student

A PhD student wants to study some new algorithm.

He first chooses a set of variants and/or different parameterizations of the algorithm, and some benchmark problem.

He then runs the chosen algorithms in parallel on his machine, checking their outputs (e.g. their *learning curves*) from time to time.

He could then *adjust the priorities* of the running algorithms *online* according to their performance.

He will soon realize that this task is so dull that even a machine can do it..

Consider:

- a sequence B of problems b_1, b_2, \dots, b_m , with precise stopping criteria
- a set A of iterative algorithms a_1, a_2, \dots, a_n , that can be
 - applied to the solution of the problems in B
 - paused and resumed at any time, at a negligible cost
 - queried, at a negligible cost, for state information $\mathbf{x} \in R^d$ related to their progress in solving current b

We want to minimize the time to solve B

Typical approaches: trial and error, brute force

Meta-learning: learn to select a *single element* or a *subset* of A
(to be executed in parallel, or in a given order)

More general: learn to *allocate time* to elements in A

In short:

- most existing meta-learning techniques *first* select an algorithm *then* run it
- with AOTA the choice is repeatedly updated *at runtime*

Adaptive Time Allocation

Time can be allocated:

- **offline**: a fixed execution schedule is chosen before starting the selected algorithm(s)
- **online**: time is allocated to the $a_i \in A$ *dynamically*, according to the evolution of their states x_i

Allocation strategy can be:

- *fixed* for any problem (*intra-problem* adaptation)
- *learned* by solving a *sequence* of problems (*inter-problem* adaptation)

Why/when AOTA ?

- some algorithms (esp. stochastic) can exhibit large performance fluctuations
- potentially large computational advantage over parallel execution
- no training data is available for new algorithms
- time to collect training data can be huge
- new problems might not be well represented in training set

In all these cases we should have a feedback on actual performance of the a 's

Previous/Related work

Offline, inter-problem adaptation:

- Algorithm selection (Rice '76)
- Algorithm recommendation (Soares, ML '04)
- Algorithm portfolios (Gomes & Selman, AI '01)
- Anytime algorithms scheduling (Boddy & Dean, AI '94)
- Time-limited planning (Horvitz & Zilberstein, AI '01; Russel & Wefald, AI '91)
- Optimal Ordered Problem Solver (Schmidhuber, ML '04)
- Racing algorithms (Moore & Lee, ICML '94; Birattari et al., GECCO '02)

Previous/Related work

Limited online adaptation:

- Incremental Machine Learning (Solomonoff, IDSIA techrep '03)
- Parameterless GA (Harick & Lobo, GECCO '99)
- Algorithm Selection using RL (Lagoudakis & Littman, ICML '00)
- Bayesian approach (Horvitz et al., UAI '01)
- Anytime algorithm monitoring (Hansen & Zilberstein, AI '01)
- Adaptive algorithm combination (Petrik, SOFSEM '05)

Previous/Related work

online

offline

intra

- Our AOTA (ECML '04)
- Parameterless GA

- A-priori choice

inter

- Incremental ML
- Algorithm Selection using RL
- Bayesian approach
- Anytime alg. monitoring
- Adaptive alg. combination

- Algorithm selection
- Algorithm portfolios
- Anytime alg. scheduling
- Time-limited planning
- OOPS
- Racing

Our AOTA

Problem sequence $B = b_1, \dots, b_m$, algorithm set $A = \{a_i, \dots, a_n\}$,
bias $P_A = \{p_1, \dots, p_n\}, p_i \geq 0, \sum_{i=1}^n p_i = 1$. Machine time sliced in small Δt .

For each problem $b_k, k = 1..m$

While (b_k not solved)

update P_A

For each $i = 1..n$

run a_i for time $p_i \Delta t$

End

End

End

Our AOTA

Problem sequence $B = b_1, \dots, b_m$, algorithm set $A = \{a_i, \dots, a_n\}$,
bias $P_A = \{p_1, \dots, p_n\}, p_i \geq 0, \sum_{i=1}^n p_i = 1$, algorithm state (\mathbf{x}_i, t_i) ,
histories $H_i = \{(\mathbf{x}_i^{(r)}, t_i^{(r)}), r = 0, \dots, h_i\}$, $H = \cup_i H_i$,
estimated time to solution $\tau_i = t_{i,sol} - t_i$.

For each problem $b_k, k = 1, \dots, m$

For each problem $b_k, k = 1..m$

While (b_k not solved)

update P_A

For each $i = 1..n$

run a_i for time $p_i \Delta t$

End

End

End

initialize $\tau_i, i = 1..n$

While (b_k not solved)

$P_A = f_P(\{\tau_i\})$

For each $i = 1..n$

run a_i for $p_i \Delta t$

$H_i = H_i \cup (\mathbf{x}_i, t_i)$

$\tau_i = f_\tau(H_i)$

End

End

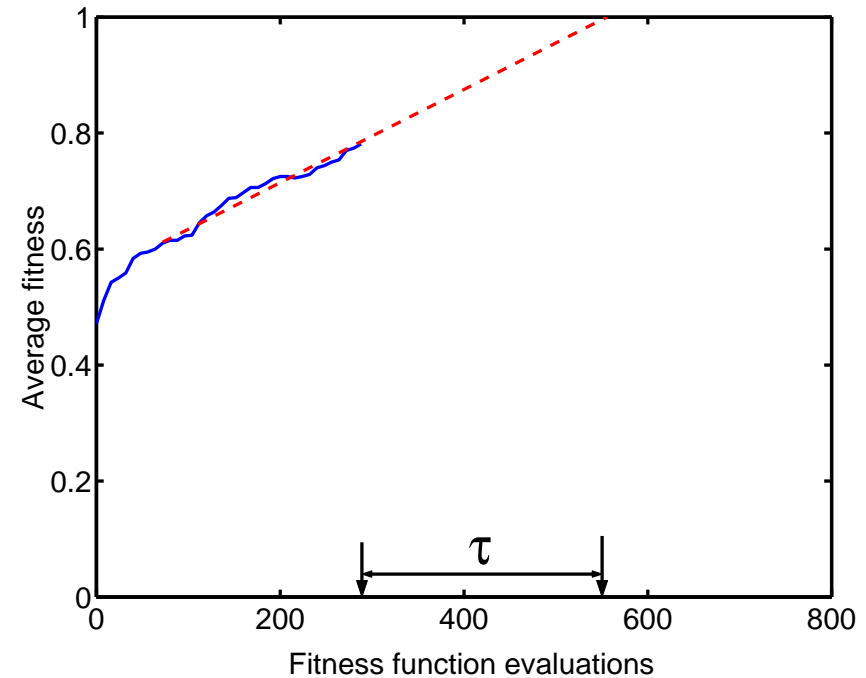
update f_τ based on H

End

Our previous intra-problem AOTA (ECML '04)

Scalar state x , that has to reach a target value
(H_i is a *learning curve*)

- from current H_i , predict time $t_{i,sol}$ at which x_i would reach the target value
- evaluate time to solution $\tau_i = t_{i,sol} - t_i$
- prediction based on a shifting window linear regression
- *intra-problem* approach - f_τ is fixed



Inter problem AOTA: learning to predict τ

Idea: prediction based on current state only: $\tau = f_{\tau}(H) = f_{\tau}(\mathbf{x}, t)$

Naive approach: learn the mapping $(\mathbf{x}, t) \rightarrow \tau$ using H as a training set:
correct target values can be evaluated a posteriori as $\tau_i = t_{i,sol} - t_i$.

Problem: which target τ for unsuccessful algorithms?

Correct τ might be very small, or ∞ !

- assign arbitrary high value \rightarrow incorrect model
- continue running other a after the fastest has ended \rightarrow additional overhead

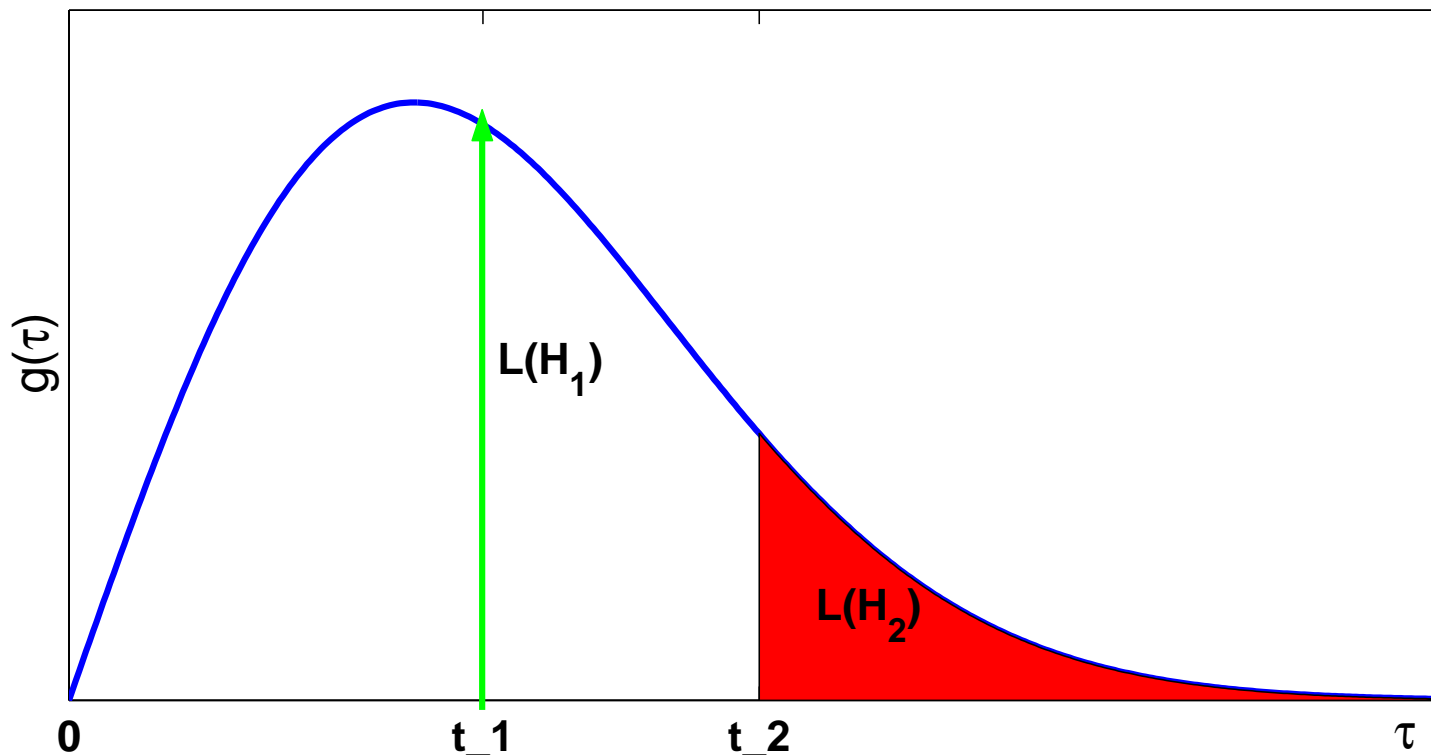
This did not work..

Inter problem AOTA: learning the distribution of τ

Better: learn a *parametric model* $g(\tau|\mathbf{x}, t; \mathbf{w})$ of the conditional probability density function (pdf) of the time to solution τ . Parameter \mathbf{w} can be learned with **Maximum Likelihood** or **Bayesian** approach.

Allows learning from unsuccessful algorithms as well (*censored sampling*).

Graphical example: a_1 successful at time t_1 , a_2 still unsuccessful at time t_2



Inter problem AOTA: learning the distribution of τ

- Maximum Likelihood approach: maximize $\mathcal{L}(H|\mathbf{w}) = \prod_{i \in I} \mathcal{L}(H_i|\mathbf{w})$
- Bayesian approach: maximize $p(\mathbf{w}|H) \propto \mathcal{L}(H|\mathbf{w})p(\mathbf{w})$

Be $H_i = \{(\mathbf{x}_i^{(r)}, t_i^{(r)}), r = 0, \dots, h_i\}$

A posteriori: a_i solved the problem $\Rightarrow t_i^{(h_i)} = t_{i,sol}$.

For each element of H_i : $\tau_i^{(r)} = t_i^{(h_i)} - t_i^{(r)}$

Likelihood of H_i :

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

Inter problem AOTA: learning the distribution of τ

Be $H_i = \{(\mathbf{x}_i^{(r)}, t_i^{(r)}), r = 0, \dots, h_i\}$

A posteriori: a_i did not solve the problem $\Rightarrow t_i^{(h_i)} < t_{i,sol} \leq +\infty$

For each element r of H_i : $\tau_i^{(r)} > t_i^{(h_i)} - t_i^{(r)}$

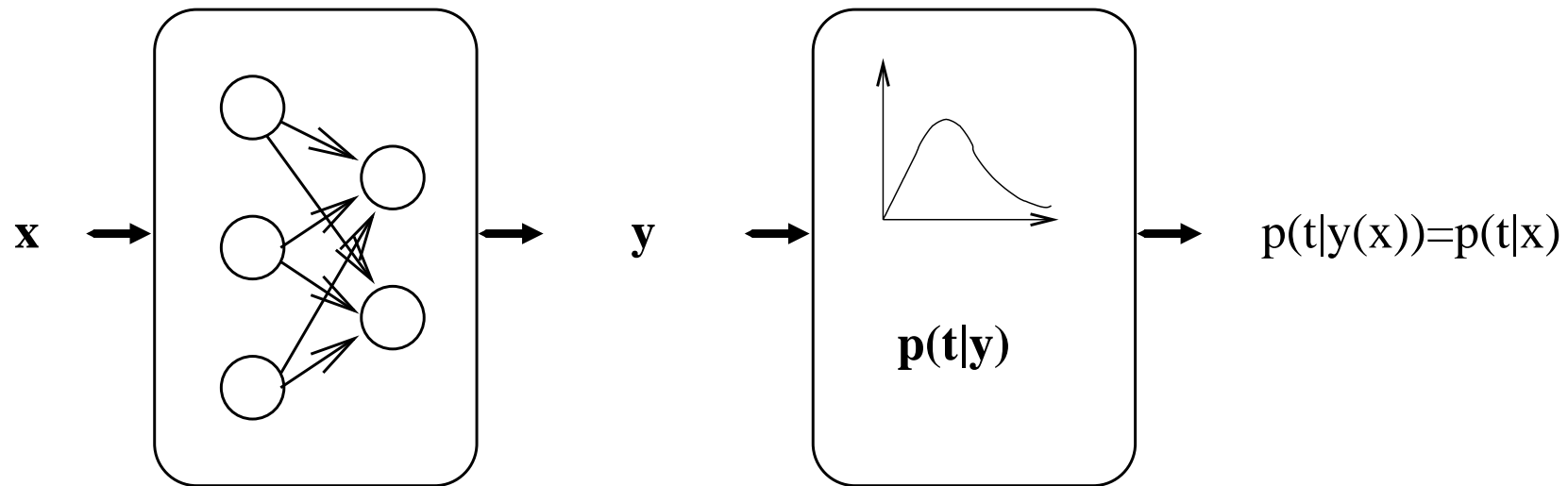
Likelihood of H_i :

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

$$G(\tau|\mathbf{x}; \mathbf{w}) = \int_0^{\tau} g(\xi|\mathbf{x}; \mathbf{w})d\xi$$

Modeling a conditional distribution $p(t|\mathbf{x})$ (Bishop '95)

- choose a parametric $p(t|\mathbf{y})$
- \mathbf{x} input and \mathbf{y} output of a Feed-Forward Neural Network
- train FFNN by gradient descent, maximizing the likelihood of the training set



Inter problem AOTA: learning the distribution of τ

Parametric model $g(\log \tau | \mathbf{x}, t; \mathbf{w})$: **Extreme Value** distribution

$$g(l) = \frac{1}{\delta} e^{\{[(l-\eta)/\delta] - e^{(l-\eta)/\delta}\}}$$

on the logarithms $l = \log \tau$ of time values.

Parameters $\eta(\mathbf{x}; \mathbf{w})$ and $\delta(\mathbf{x}; \mathbf{w})$ are the two outputs of a Feed-Forward Neural Network, with two separate hidden layers of 32 units.

Weights \mathbf{w} are learned maximizing the Bayesian posterior $p(\mathbf{w} | H)$, using a Cauchy prior $p(w) = 1/(1 + w^2)$.

Ranking f_P

Idea: algorithms sorted from fastest to slowest get $1/2, 1/4, \dots, 1/2^n$ of current slice Δt

Problem: during early tasks the model is incorrect!

Solution: $p_i = \left(2 - \frac{\log(m+1-k)}{\log(m)}\right)^{-r_i}$, k current task, m last task, r_i current rank of a_i .

- task 1: uniform distribution
- ...
- task m : $p_i = 1/2^{r_i}$

Experiments

Problems

“Trap” problem (Harick & Lobo, GECCO '99): find the bitstring with all bits 1, guided by a deceptive fitness 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 $m - q - 1$ if $q < m$ bits are 1.

Example for $m = 3$:

- 000 \rightarrow 2
- 001, 010, 100 \rightarrow 1
- 011, 110, 101 \rightarrow 0
- 111 \rightarrow 3

21 instances, roughly sorted by difficulty, from $m = 2, n = 15$ to $m = 4, n = 24$.

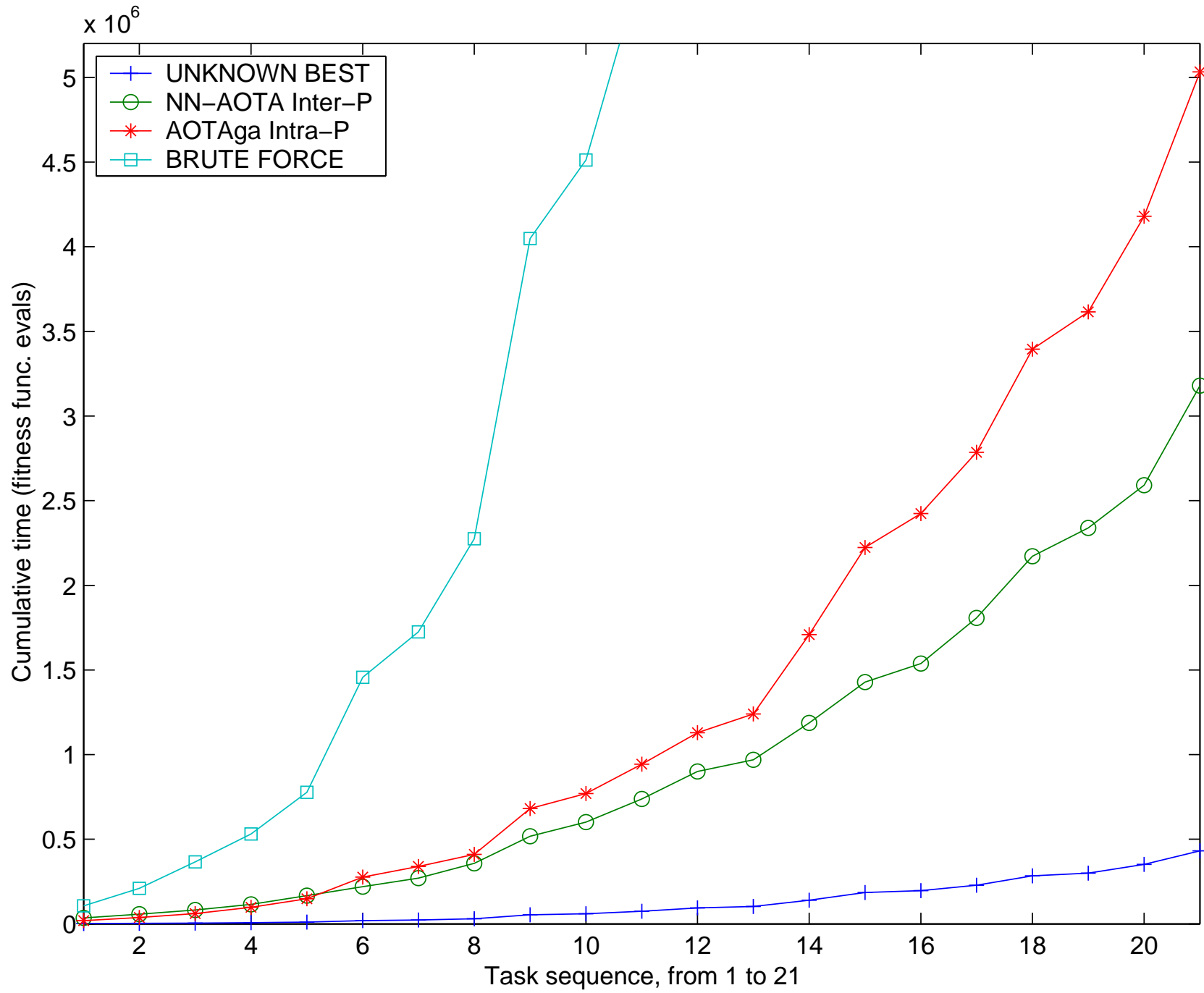
Algorithms

76 Simple Genetic Algorithms obtained combining the following parameters:

- population size: 2, 4, 8, .., 2^{19}
- mutation rate: 0 or $0.7/mn$
- crossover operator: uniform or one-point

State information \mathbf{x} (13 dimensions)

- problem features: genome length, block size
- algorithm features: parameter values
- algorithm state:
 - best and average fitness, and time
 - their last variation
 - their trend



Conclusions

- Intra-problem AOTA (ECML '04): some prior knowledge prewired
- Inter-problem AOTA: less prior knowledge, better performance
- Still generic and algorithm independent

Future work:

- alternatives to FFNN
- other choices for f_P
- adaptive A
- application to other A 's and B 's