

Dynamic Algorithm Portfolios

Matteo Gagliolo, Jürgen Schmidhuber

`{matteo, juergen}@idsia.ch`



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

Static vs. Dynamic: an example

A student wants to study some new algorithm.

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

He then starts the chosen algorithms in parallel on his machine.

He might then leave them running, or check their outputs (e.g. their *learning curves*) from time to time.

If he does so, he could save time by periodically *adjusting the priorities* of the running algorithms, according to their actual performance.

Problem statement

Consider:

- a sequence B of problem instances b_1, b_2, \dots, b_m
- a set A of solvers a_1, a_2, \dots, a_n
- termination criterion for each b
- for each b exist at least one a that solves it ($t(b, a) < \infty$)

We want to minimize the time to solve B

Typical approach: trial and error...

Lower bound:

$$t_{best} = \sum_{b \in B} \min_{a \in A} t_{sol}(b, a)$$

Brute force approach: run all a in parallel until the fastest one solves b

$$t_{brute} = n \sum_{b \in B} \min_{a \in A} t_{sol}(b, a) = nt_{best}$$

Expensive, but allows to identify fastest a

More refined: find the “fastest” a according to some criterion (e.g. on average, min-max)

- split B into disjoint B_{train} and B_{test}
- solve each $b \in B_{train}$ with each $a \in A$, storing $t_{sol}(b, a)$
- solve B_{test} with a that was fastest on B_{train}

Issues:

- usually no single best
- is B_{train} representative of B_{test} ?
- training time $\sum_{b \in B} \sum_{a \in A} t_{sol}(b, a) \gg t_{brute}(B_{train}, A)$

Algorithm selection/Meta-learning: learn to select a single a for each $b \in B_{test}$ (Rice '76)

- split B into disjoint B_{train} and B_{test}
- solve each $b \in B_{train}$ with each $a \in A$, storing $t_{sol}(b, a)$
- learn a model of performance $(b, a) \rightarrow t_{sol}(b, a)$
- solve each $b \in B_{test}$ with expected fastest a

Issues:

- no single best \leftarrow different a for each b
- is B_{train} representative of B_{test} ?
- training time $\sum_{b \in B} \sum_{a \in A} t_{sol}(b, a) \gg t_{brute}(B_{train}, A)$
- *is performance predictable?*

Static Algorithm Portfolios: learn to select a *subset* A_r of A for each $b \in B_{test}$.
Run its elements in parallel. (Gomes & Selman, AI '01)

- split B into disjoint B_{train} and B_{test}
- solve each $b \in B_{train}$ with each $a \in A$, storing $t_{sol}(b, a)$
- learn a model of performance $(b, a) \rightarrow t_{sol}(b, a)$
- solve each $b \in B_{test}$ with the r expected fastest a *in parallel* ($r < n$)

Issues:

- is B_{train} representative of B_{test} ?
- long training time
- is performance predictable? ← more algorithms running
- *choice of r ?*

Most existing meta-learning techniques *first* select an algorithm *then* run it.

- is B_{train} representative of B_{test} ? → how can I detect an unusual b ?
- long training time → how can I detect a slow a ?
- is performance predictable? → can I predict it based on static features?

Idea: maybe I should exploit *runtime* information!

Dynamic Algorithm Portfolios: learn to *allocate time* to elements in A
(Adaptive Online Time Allocation - ECML '04, ICANN '05)

Dynamic *state information* x for each (b, a) execution

Life-long learning:

- solve each $b \in B$ running all $a \in A$ in parallel, periodically *updating priorities* ...
- ... according to a model of performance $(b, a, x) \rightarrow t_{sol}(b, a, x)$...
- update the model after the solution of each b

Issues:

- is B_{train} representative of B_{test} ? ← detect failures of the model at runtime
- training time ← use model during training, detect slow a at runtime
- is performance predictable? ← dynamic features should make prediction easier
- *model not reliable in the beginning*
- *model precision vs. training time trade-off*
- *model training time*

Other dynamic approaches

- Incremental Machine Learning (Solomonoff, IDSIA techrep '03)
- Bayesian approach (Horvitz et al., UAI '01)
- Adaptive algorithm combination (Petrik, SOFSEM '05)
- Parameterless GA (Harick & Lobo, GECCO '99)
- Algorithm Selection using RL (Lagoudakis & Littman, ICML '00)
- Anytime algorithm monitoring (Hansen & Zilberstein, AAI '96)

Learning to predict t

Idea: prediction based on current state: $t = f(b, a, x)$

Naive approach: learn the mapping $(b, a, x) \rightarrow \tau$ using collected $t(b, a, x)$ as a training set

Problem: which target for unsuccessful algorithms?

Correct t might be very small, or ∞ !

Model precision vs. training time trade-off

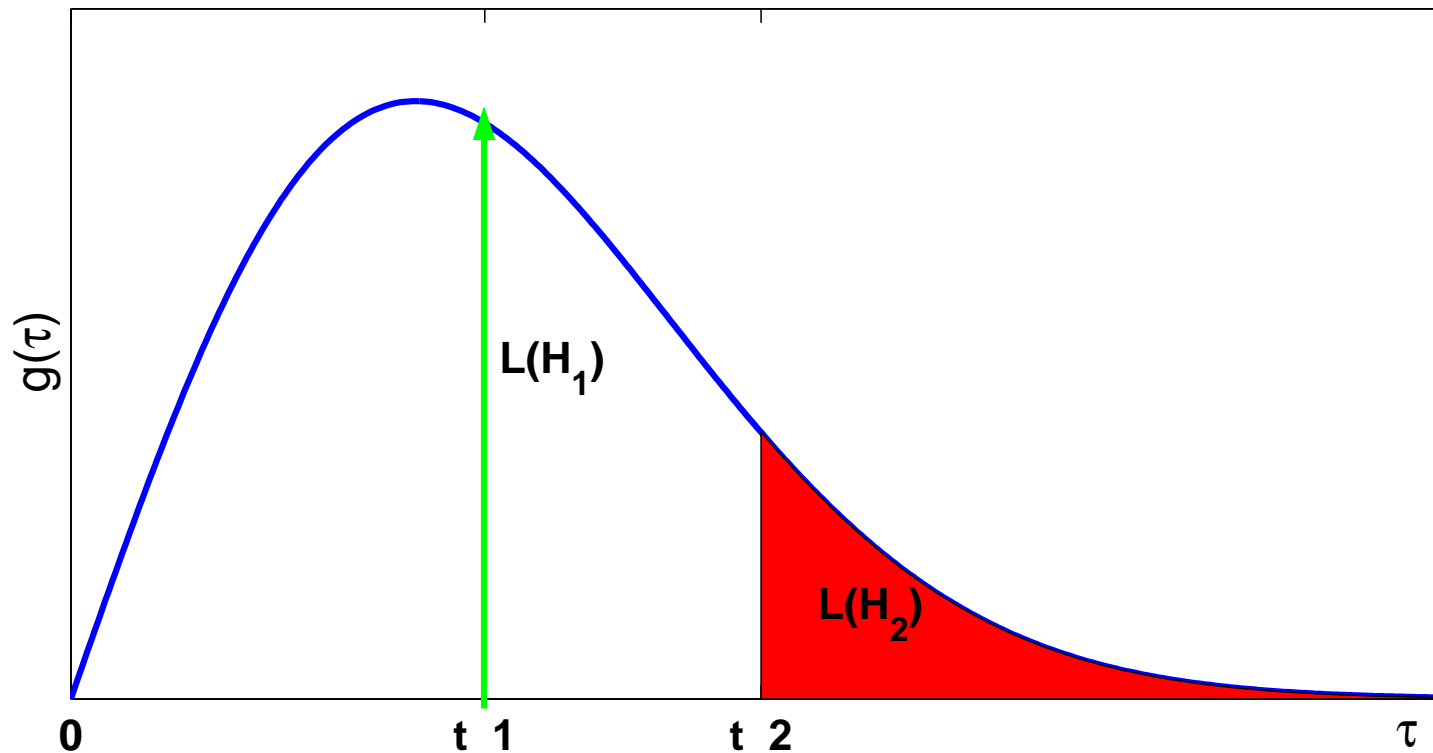
- assign arbitrary high value \rightarrow **incorrect model**
- collect more $t(b, a, x)$ after the fastest a has ended \rightarrow **additional overhead**

Learning the distribution of t

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

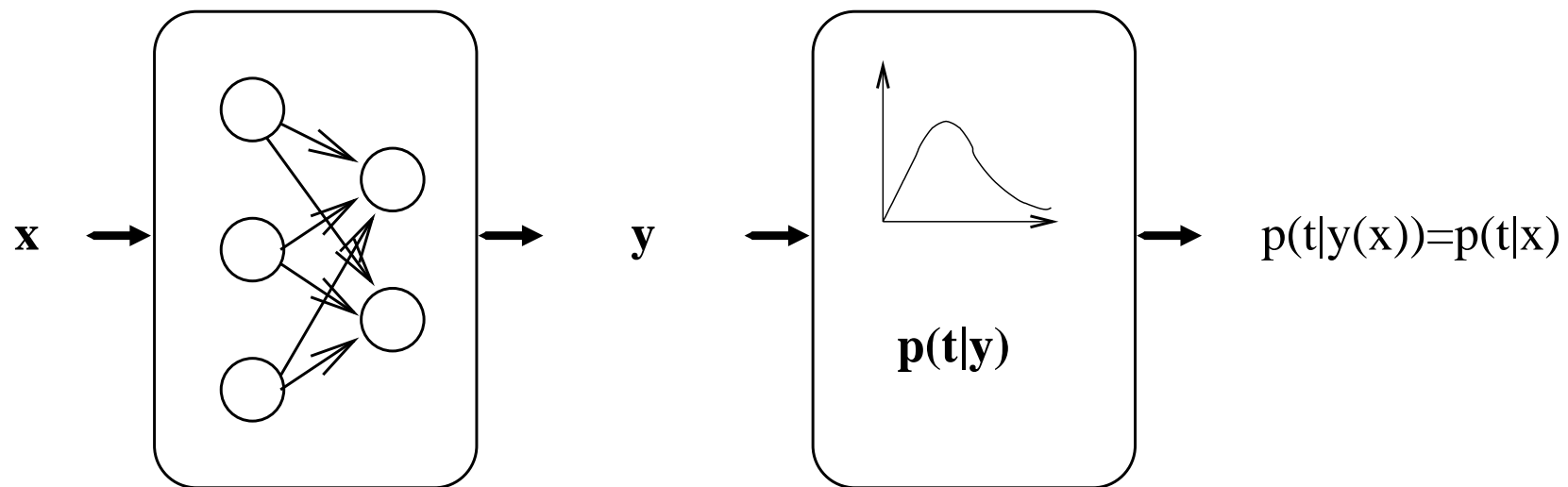
Allows learning from unsuccessful algorithms as well (*censored sampling*).

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



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 another parametric model
- train the parametric model by gradient descent, maximizing the likelihood of the training set



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. 21 instances, roughly sorted by difficulty: 18 for training, 3 for testing.
 - 000 \rightarrow 2
 - 001, 010, 100 \rightarrow 1
 - 011, 110, 101 \rightarrow 0
 - 111 \rightarrow 3
- Random satisfiable CNF3SAT problems from www.satlib.org: 40 training instances, 9 test instances; 20, 50 and 75 clauses

Algorithms

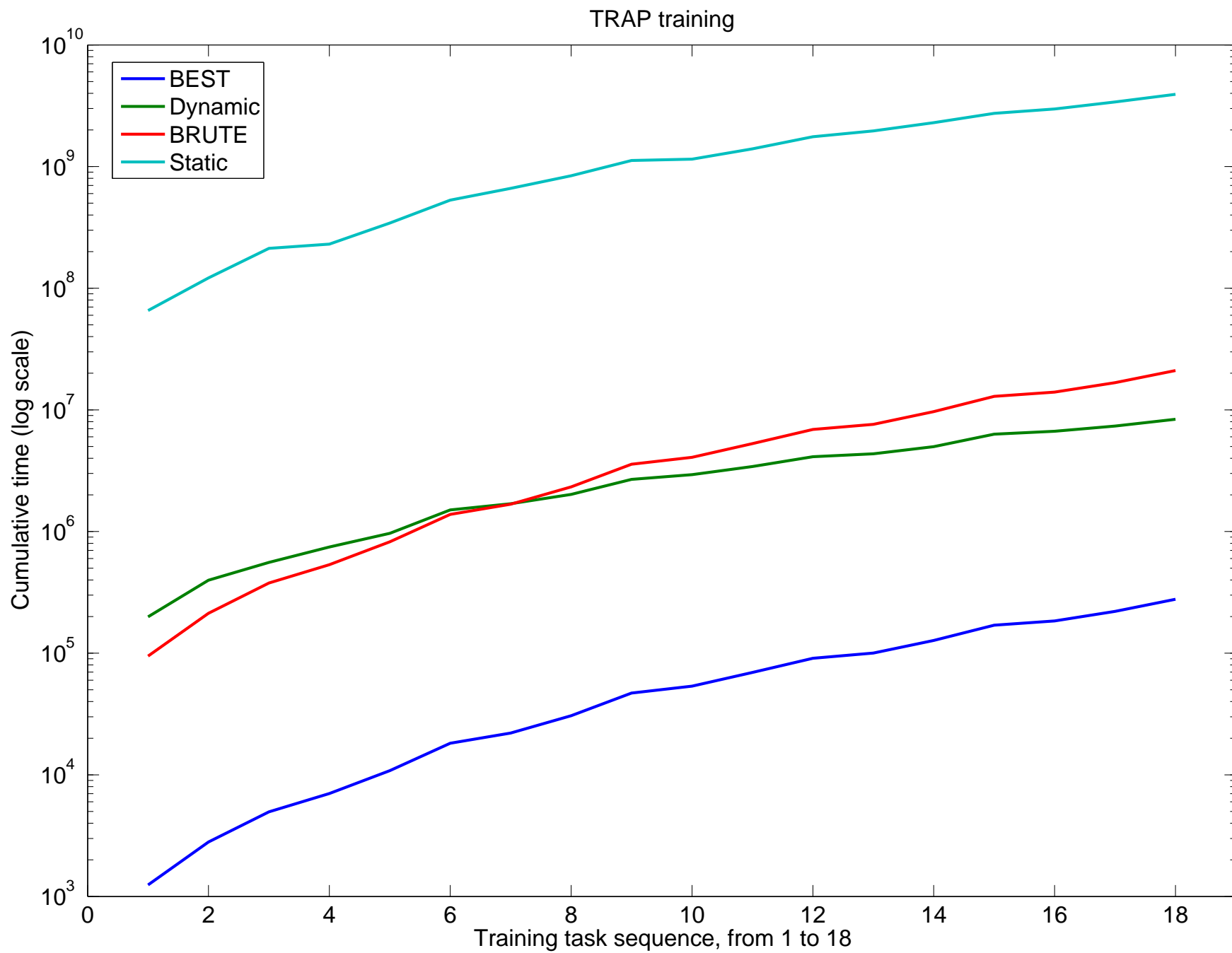
76 Simple Genetic Algorithms obtained combining the following parameters:

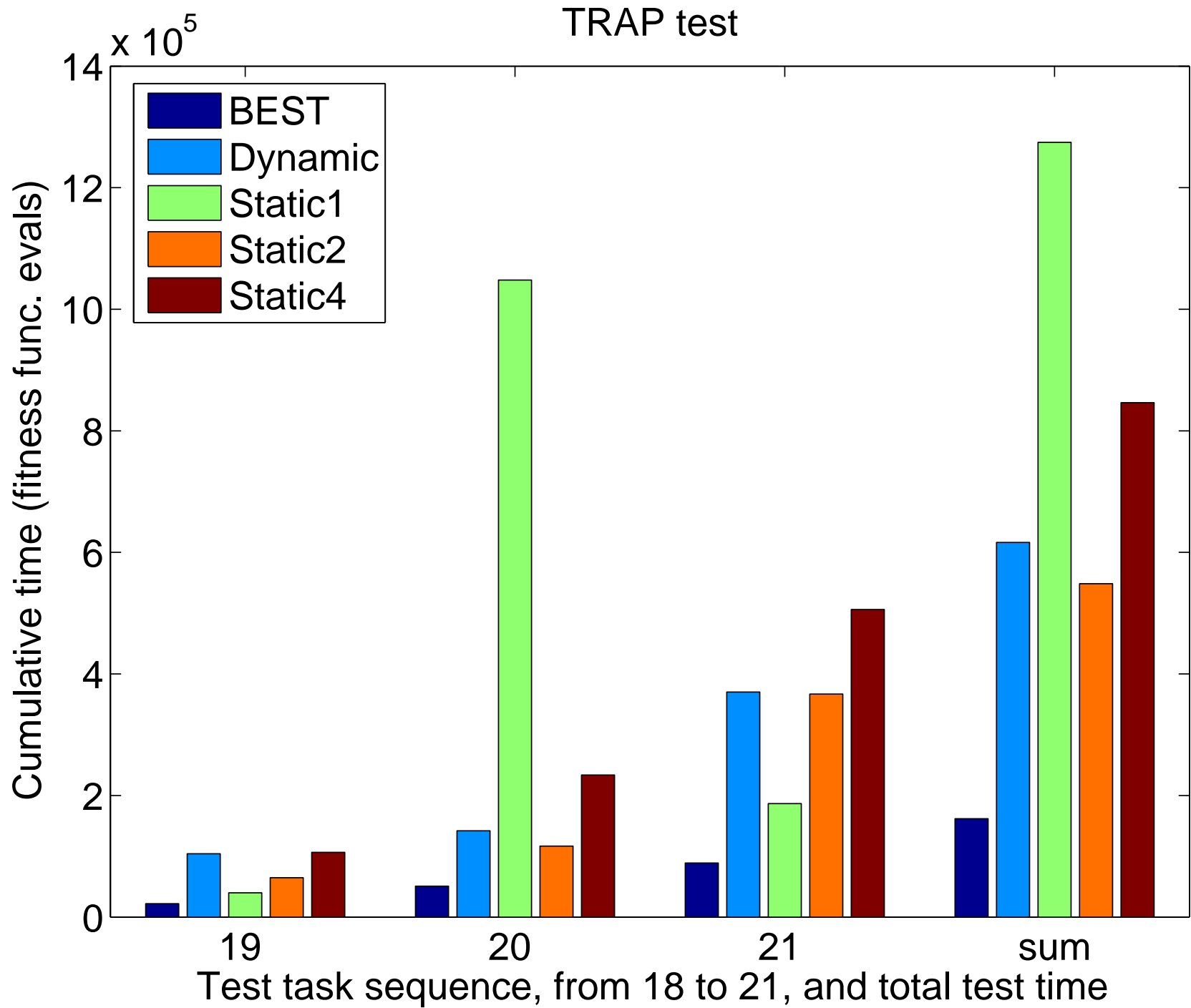
- population size: 2, 4, 8, ..., 2^{19}
- mutation rate: 0 or $0.7/mn$
- crossover operator: uniform or one-point

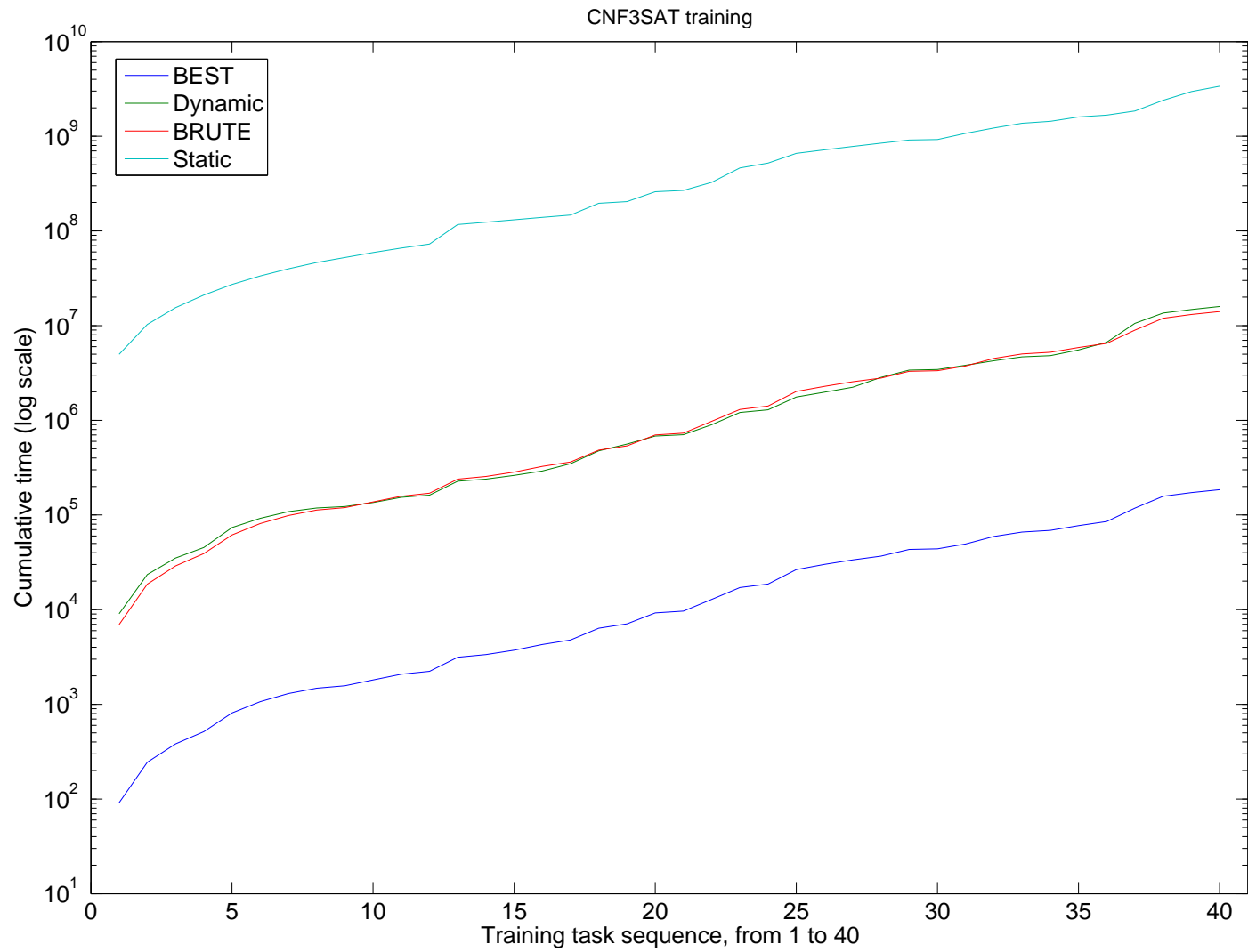
Comparison term: static algorithm portfolio of sizes 1, 2, 4

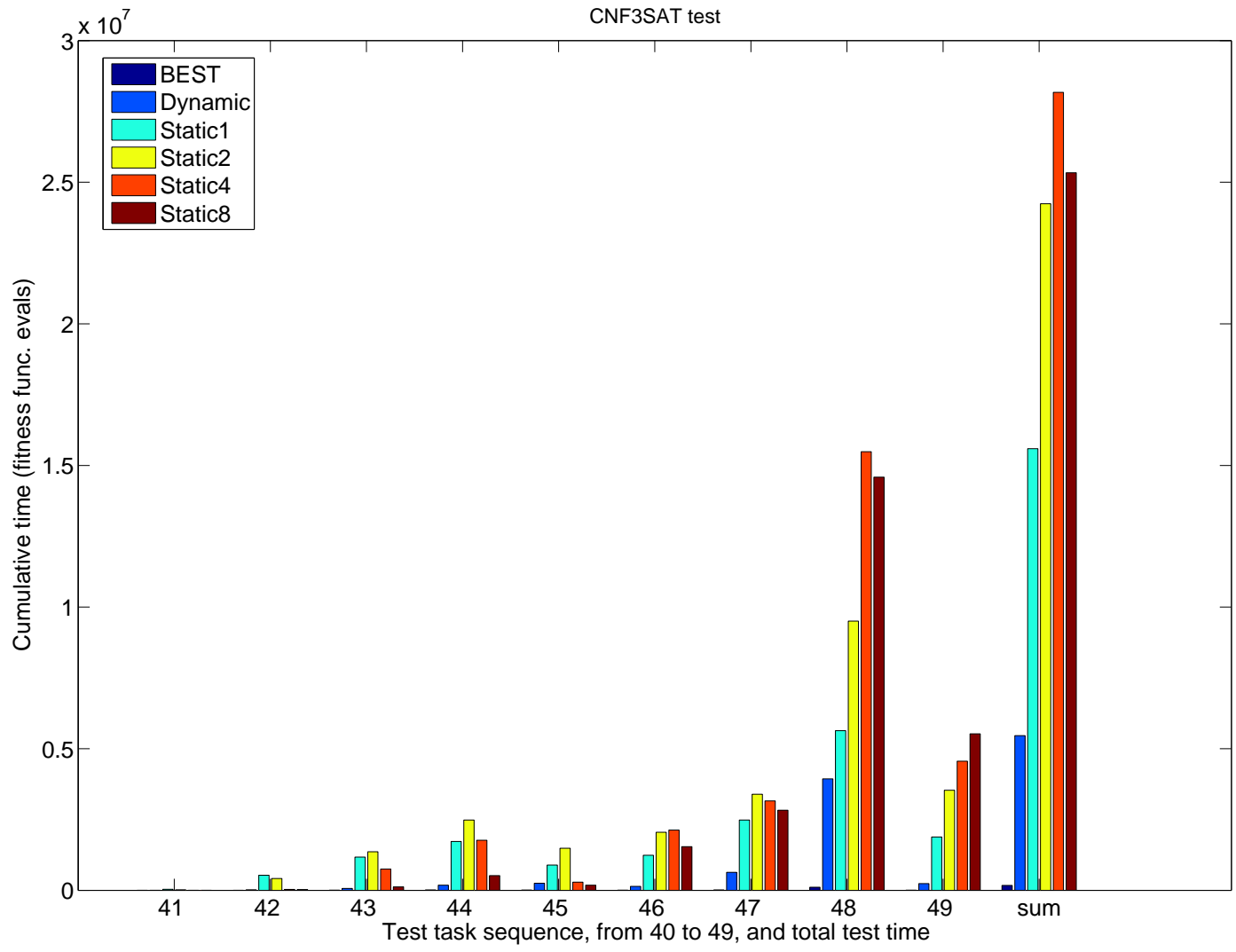
State information \mathbf{x} (9 dimensions),
sampled with exponential period ($t = 2, 4, 8, \dots$)

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









Conclusions

- Performance comparable to static portfolios
- Saves more than 2 orders of magnitude in training time!

Issues:

- is B_{train} representative of B_{test} ? ← detect failures of the model at runtime?
- long training time ← use model during training
- is performance predictable? ← dynamic features make prediction easier
- *model not reliable in the beginning* ← heuristic
- *model precision vs. training time trade-off* ← heuristic
- *model training time*

Future work

- train the model online (currently batch mode)
- application to other A 's and B 's
- non-sequential B
- non-heuristic time allocation and exploration
- predict time and next state to detect model failure
- adaptive A
- feature selection

Long-term goal

Let the user pick:

- a set of problems of unknown difficulty
- a redundant set of algorithms
- a redundant set of features

... and let the machine think about the rest...

AOTA pseudocode

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

AOTA pseudocode

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

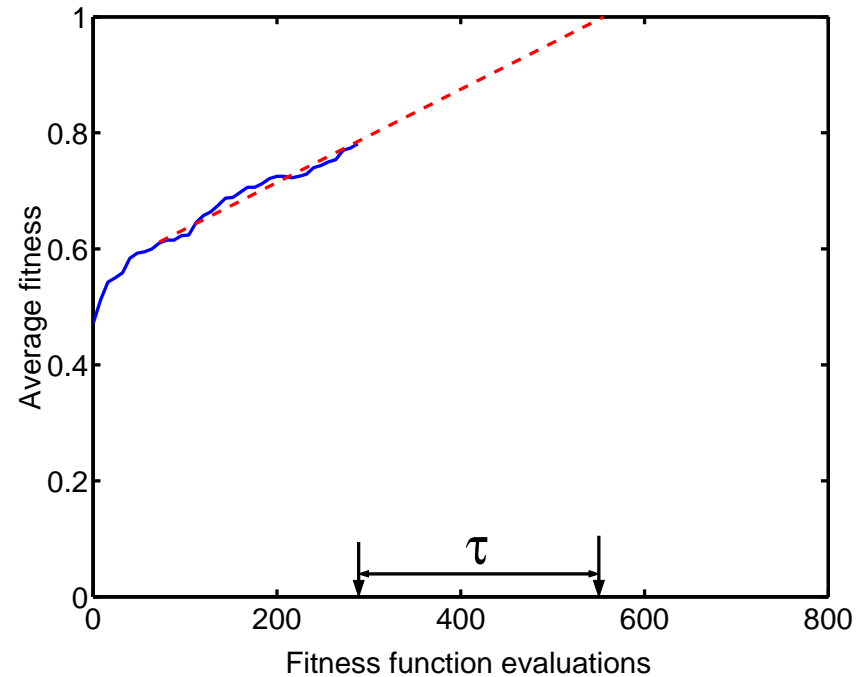
update f_τ based on H

End

Our previous heuristic 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



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 $p(\mathbf{w}|H) \propto \mathcal{L}(H|\mathbf{w})p(\mathbf{w})$

Learning the distribution of τ

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 :

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

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 :

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

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

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 quadratic expansions of \mathbf{x} of the form $w_0 + \sum_i w_i x_i + \sum_{i,j} w_{i,j} x_i x_j$.

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!

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

Example for $m = 3$:

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

Experiments

Problems

49 random satisfiable CNF3SAT problems from `www.satlib.org`, sorted a posteriori based on their difficulty.

- 20 instances with 20 clauses (uf20)
- 18 instances with 50 clauses (uf50)
- 11 instances with 75 clauses (uf75)

Last 9 instances used for testing: uf20-010, uf50-03, uf50-014, uf75-013, uf50-017, uf50-04, uf50-08, uf75-05, uf50-012