

Matteo Gagliolo

IDSIA — Istituto Dalle Molle
di Studi sull'Intelligenza
Artificiale, Lugano,
Switzerland

matteo@idsia.ch

University of Lugano, Faculty
of Informatics, Switzerland

www.idsia.ch/~matteo

Static vs. Dynamic: an example. A student wants to study some new algorithm. He first picks a set of variants and/or different parametrization 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

- 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 ($t_{sol}(b, a) < \infty$)

We want to minimize the time to solve B

Typical approach: trial and error...

Brute force approach: run all a in parallel until the fastest one solves b . Expensive, but allows to identify fastest 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)

- solve each $b \in B_{test}$ with the r expected fastest a **in parallel**

Issues:

- is performance predictable? \leftarrow 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} ?
 \rightarrow **how can I detect an unusual b ?**
- long training time \rightarrow **how can I detect a slow a ?**
- is performance predictable?
 \rightarrow **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 (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 **dynamic** 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} ? \leftarrow **detect failures of the model at runtime**
- training time \leftarrow **exploit model during training, detect slow a at runtime**
- is performance predictable?
 \leftarrow **dynamic features should make prediction easier**
- **model not reliable in the beginning**
- **model precision vs. training time trade-off**

Other dynamic approaches

- Incremental Machine Learning (Solomonoff, IDSIA techrep '03)
- Dynamic restart strategies (Kautz et al., AAAI '02)
- Adaptive algorithm combination (Petrik, SOFSEM '05)
- Parameterless GA (Harick & Lobo, GECCO '99)
- Algorithm Selection using RL (Lagoudakis & Littman, ICML '00)
- Anytime algorithm monitoring (Hansen & Zillberstein, AAAI '96)

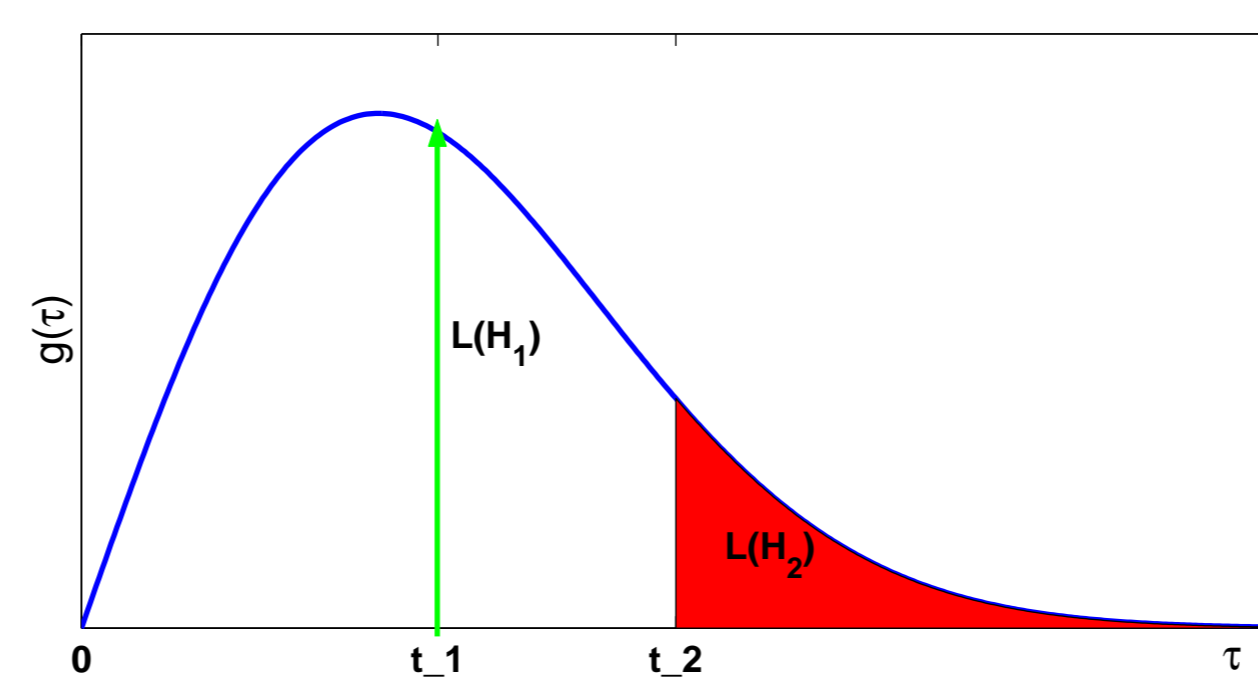
Learning to predict t

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

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



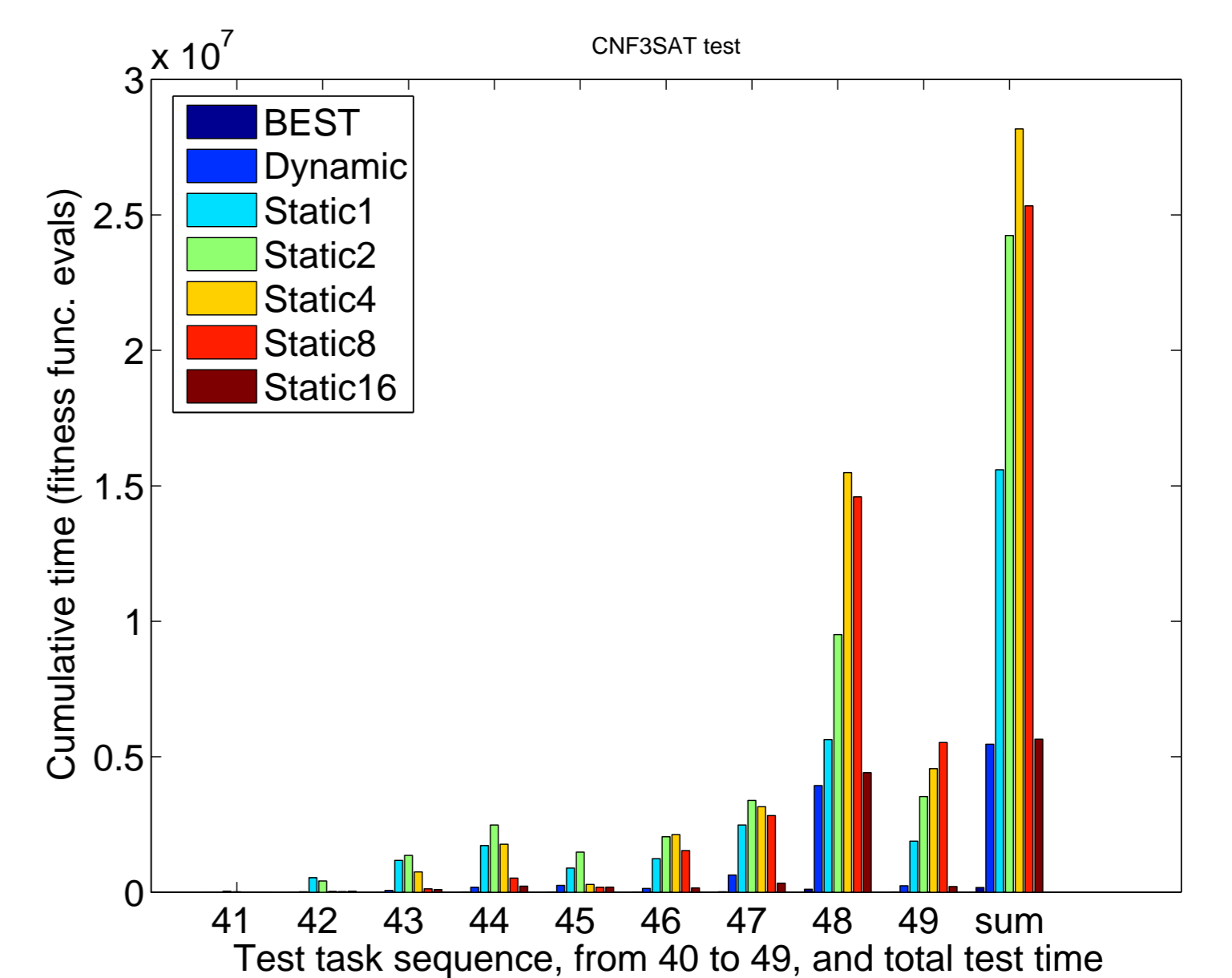
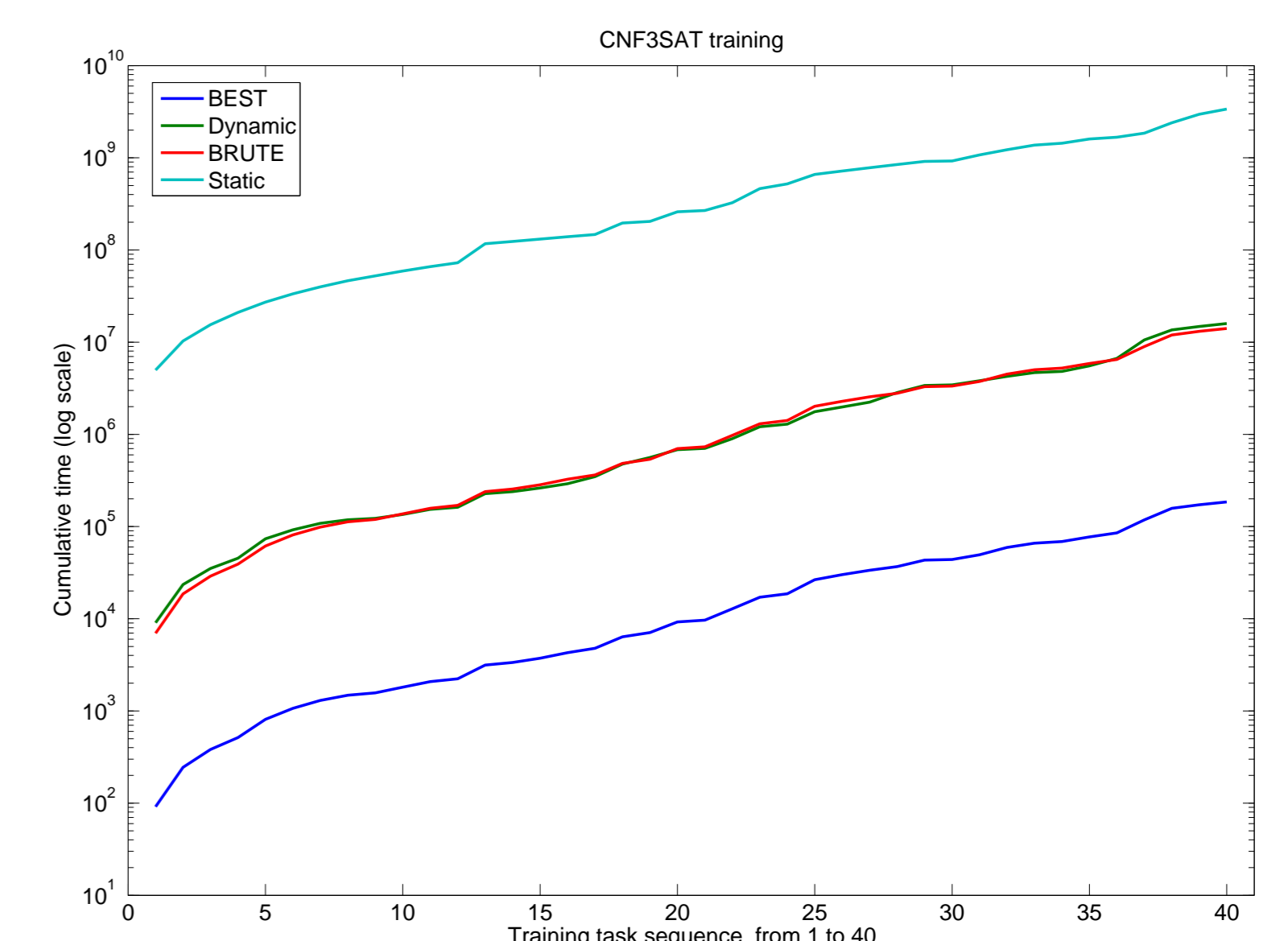
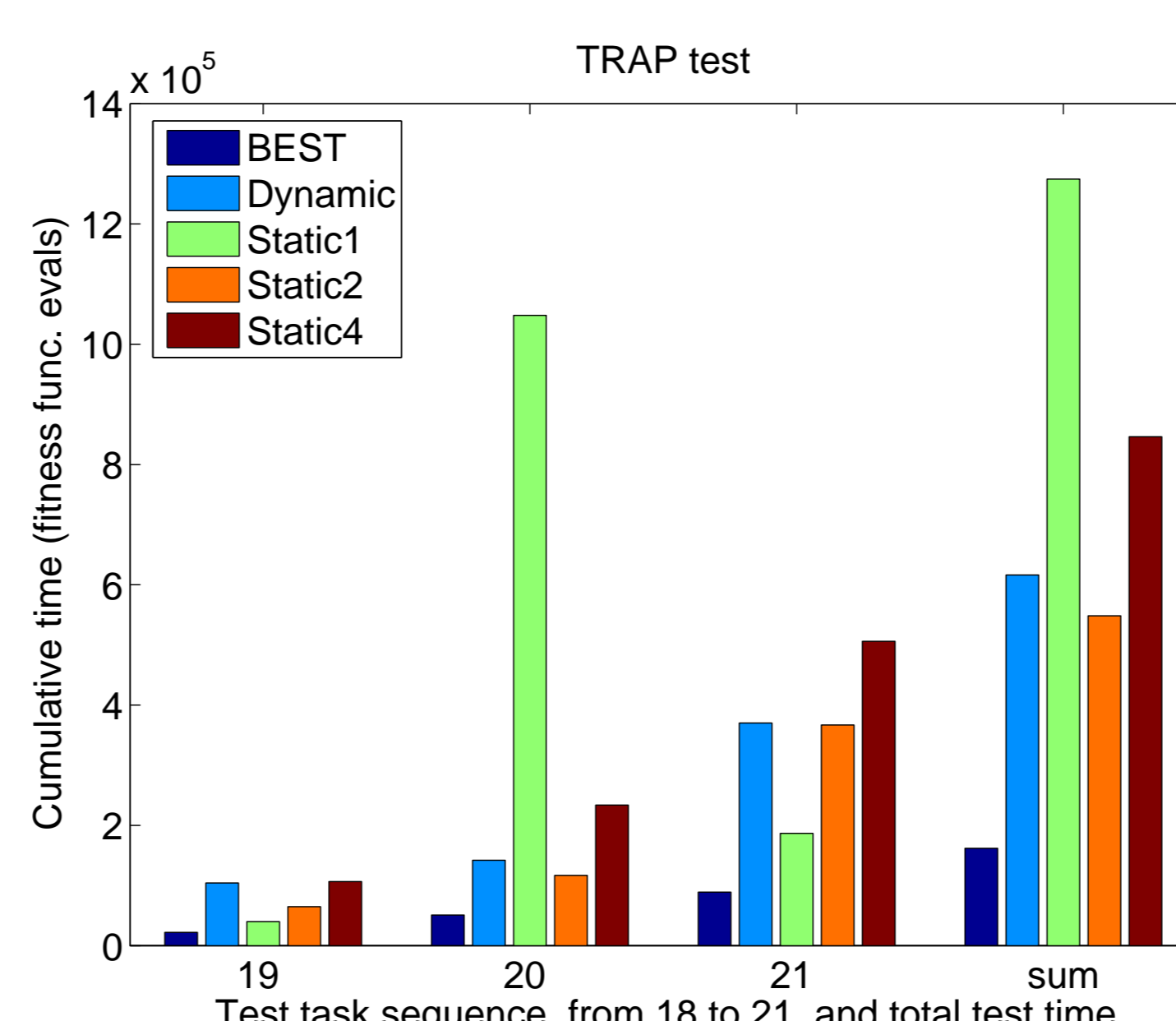
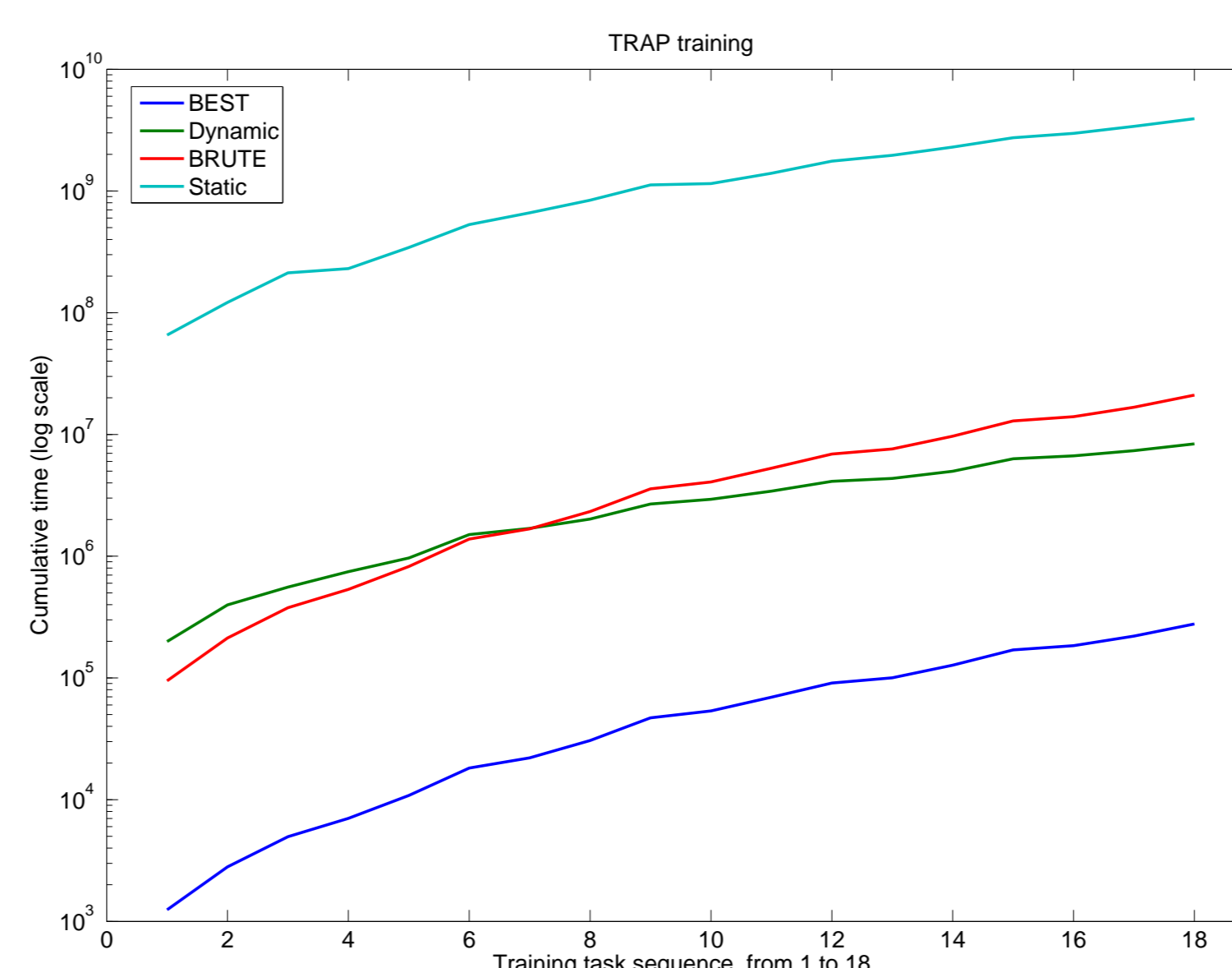
Experiments

- "Trap" problem (Harick & Lobo, GECCO '99): find the bitstring with all bits 1, guided by a deceptive fitness function
- Random satisfiable CNF3SAT problems from SATLIB: 40 training instances, 9 test instances; 20, 50 and 75 clauses

Algorithms: 76 Simple Genetic Algorithms obtained combining different population sizes (2, 4, 8, ..., 2^{19}), mutation rates (0 or $0.7/mn$), crossover operator (uniform or one-point)

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

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



Conclusions

- Performance comparable to static portfolios, without a-priori choice of portfolio size
- Saves more than 2 orders of magnitude in training time!

Issues:

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

Ongoing

- non-parametric models
- restart strategies (CP '06)
- non-heuristic time allocation and exploration
- predict time and next state to detect model failure

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...

This work was funded by SNF under grant 200020-107590/1