The best of both worlds: Machine learning meets logical reasoning

Holger H. Hoos

LIACS
Universiteit Leiden
The Netherlands

CS Department
University of British Columbia
Canada

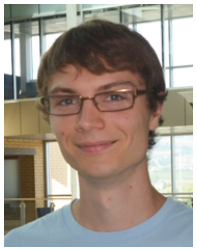
Hybris 2018
2018/05/07

Frank Hutter
UBC

Lin Xu
UBC

Zongxu Mu
UBC

Chris Thornton
UBC


Yasha Pushak
UBC

Marius Schneider
U. Potsdam

Lars Kotthoff
UBC

Thomas Stützle
U. Libre de Bruxelles

Kevin Leyton-Brown
UBC

Yoav Shoham
Stanford U.

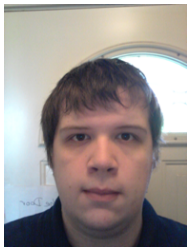
Eugene Nudelman
Stanford U.

Alan Hu
UBC

Domagoj Babić
UBC

Torsten Schaub
U. Potsdam

Benjamin Kaufmann
U. Potsdam


James Styles
UBC

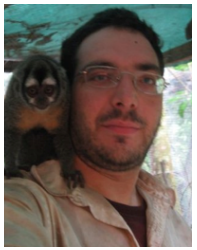
Chris Fawcett
UBC

Alfonso Gerevini
U. di Brescia

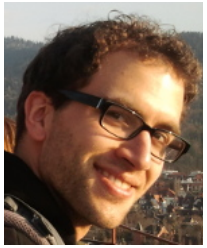
Alessandro Saetti
U. di Brescia

Mauro Vallati
U. di Brescia

Matle Helmert
U. Basel


Erez Karpas
Technion

Gabriele Röger
U. Freiburg


Jendrik Seipp
U. Freiburg

Chuan Luo
U. Leiden

Chris Cameron
UBC

Jérémy Dubois-Lacoste
U. Libre de Bruxelles

Pascal Kerschke
U. Münster

Jakob Bossek
U. Münster

Bernd Bischl
LMU München

Heike Trautmann
U. Münster

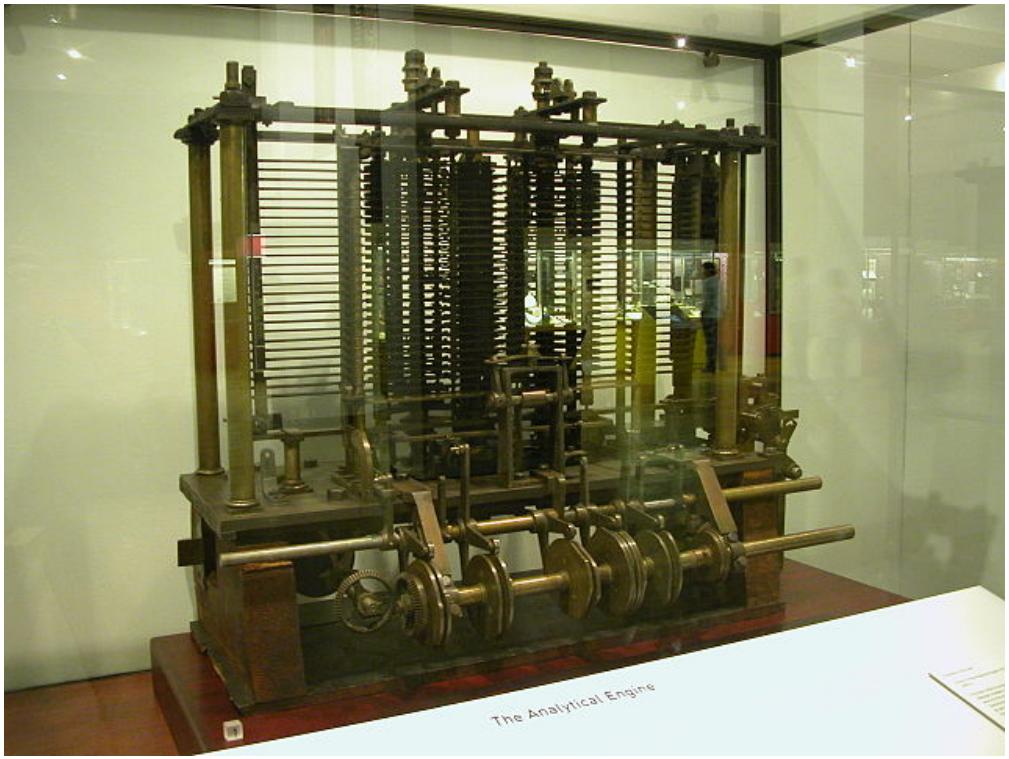
Donald Knuth
Stanford U.

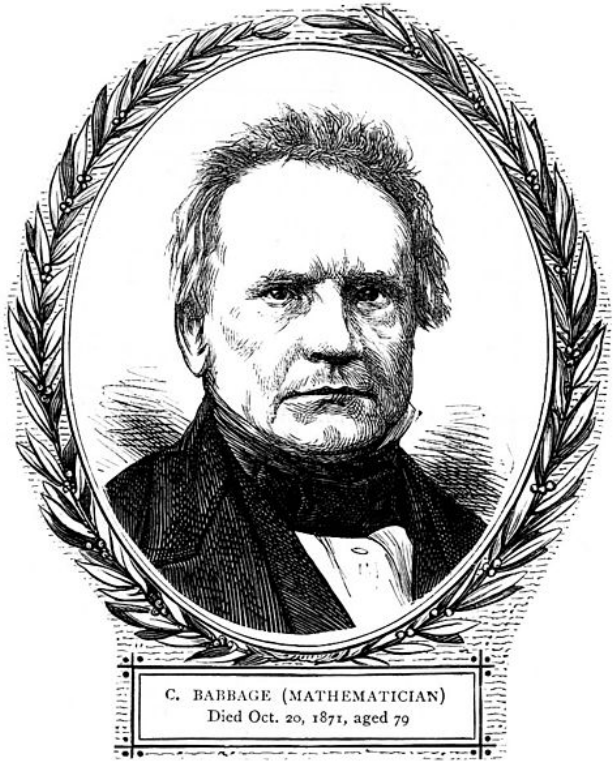
The nature of computation

Clear, precise instructions – flawlessly executed

$\rightsquigarrow$ algorithm

The age of machines





"As soon as an Analytical Engine exists, it will necessarily guide the future course of the science.

(Charles Babbage, 1864)

The age of computation

BBC Mobile News | Sport | Weather | Travel | TV | Radio

NEWS TECHNOLOGY

Home | US & Canada | Latin America | UK | Africa | Asia | Europe | Mid-East | Business | Health | Sci/Environment | Tech

22 August 2011 Last updated at 20:42 ET

1.4K Share

When algorithms control the world

By Jane Wakefield
Technology reporter

If you were expecting some kind of warning when computers finally get smarter than us, then think again.

There will be no soothing HAL 9000-type voice informing us that our human services are now surplus to requirements.

In reality, our electronic overlords are already taking control, and they are doing it in a far more subtle way than science fiction would have us believe.

Their weapon of choice - the algorithm.

Behind every smart web service is some even smarter web code. From the web retailers - calculating what books and films we might be interested in, to Facebook's friend finding and image tagging services, to the search engines that guide us around the net.

Algorithms are spreading their influence around the globe

It is these invisible computations that increasingly control how we interact with our electronic world.

At last month's TEDGlobal conference, algorithm expert Kevin Slavin delivered one of the tech show's most "sit up and take notice" speeches where he warned that the "maths that computers use to decide stuff" was infiltrating every aspect of our lives.

Related Stories

Are search engines skewing objectivity?

Robot reads minds to train itself

"The maths that computers use to decide stuff [is] infiltrating every aspect of our lives."

- financial markets
- social interactions
- cultural preferences
- artistic production
- ...

The machine learning (ML) revolution

manually constructed algorithms

$\rightsquigarrow$

automatic adaptation to given set / distribution of inputs
through optimisation of performance metric (loss minimisation)

machine learning procedures
= meta-algorithms (procedures for optimising algorithm)

ML is old ...

- Alan Turing (1950): Computing machinery and intelligence
- Farley and Clark (1954): Simulation of Self-Organizing Systems by Digital Computer
- Arthur Samuel (1959): Some Studies in Machine Learning Using the Game of Checkers
- Paul Werbos (1974): Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences
- ...

... automated reasoning is even older:

- Ramon Llull (1232–1315):
  mechanical devices for producing "all possible knowledge"
- Gottfried Wilhelm Leibniz (1646–1716):
  calculus ratiocinator
- A. Newell, H.A. Simon, J.C. Shaw (1955–56):
  logic theorist
- H.A. Simon, J.C. Shaw, A. Newell (1959):
  general problem solver
- ...

Complementary pillars of (artificial) intelligence:

- logical inference
  $\rightsquigarrow$ automated reasoning, theorem proving,
  SAT, CP, logic programming, (classical) planning, ...
- statistical inference
  $\rightsquigarrow$ (statistical) machine learning,
  reasoning under uncertainty, ...

Here: Use statistical inference (ML)
to analyse & improve logical inference

1

Challenge:

Given algorithm A, input i, predict $p(A, i)$ = performance of A on i

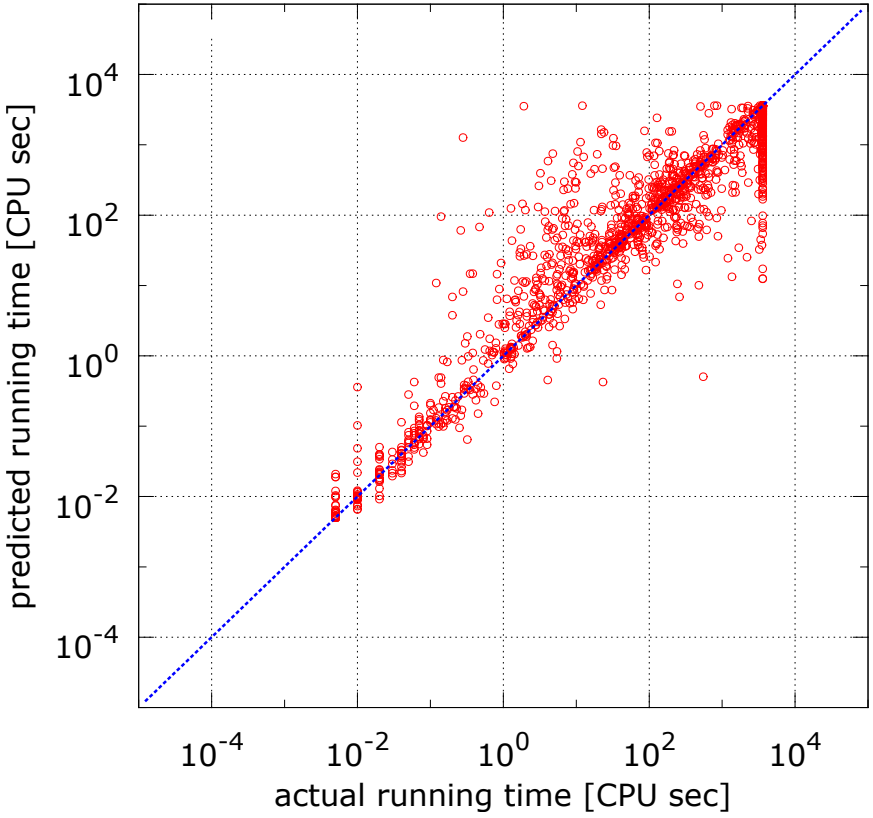
Key idea:

Learn mapping $i \mapsto p(A, i)$ from observed performance data
= empirical performance model (EPM)

Need:

- (cheap) features of inputs
- effective method to learn EPM
- (lots of) training data

MiniSAT 2.0 on SAT Competition Benchmarks
Random Forest Model



Spearman correlation coefficient = 0.90

Further details: Hutter, Xu, HH, Leyton-Brown (2014)

Similar success for state-of-the-art algorithms for:

- mixed integer programming (CPLEX)
- travelling salesperson problem (TSP)

Can exploit this for:

- benchmark generation
- automatic algorithm selection
- automatic algorithm configuration

Hutter, Xu, HH, Leyton-Brown (2014); Leyton-Brown, HH, Hutter, Xu (2014)

Much bigger challenge:

Performance prediction for inputs *unlike* those seen before
= *beyond* distribution underlying training set

$\rightsquigarrow$ extrapolation, transfer learning

[This slide was not used during the presentation; it was added to make it easier to follow the slide deck.]

Example:
Running time of state-of-the-art TSP solver (Concorde) on prominent class of benchmarks

input size	*running time [CPU sec]*
100	0.7
200	3.6
300	10.6
400	25.9
500	50.2
1000	601.6
2000	14 065.6

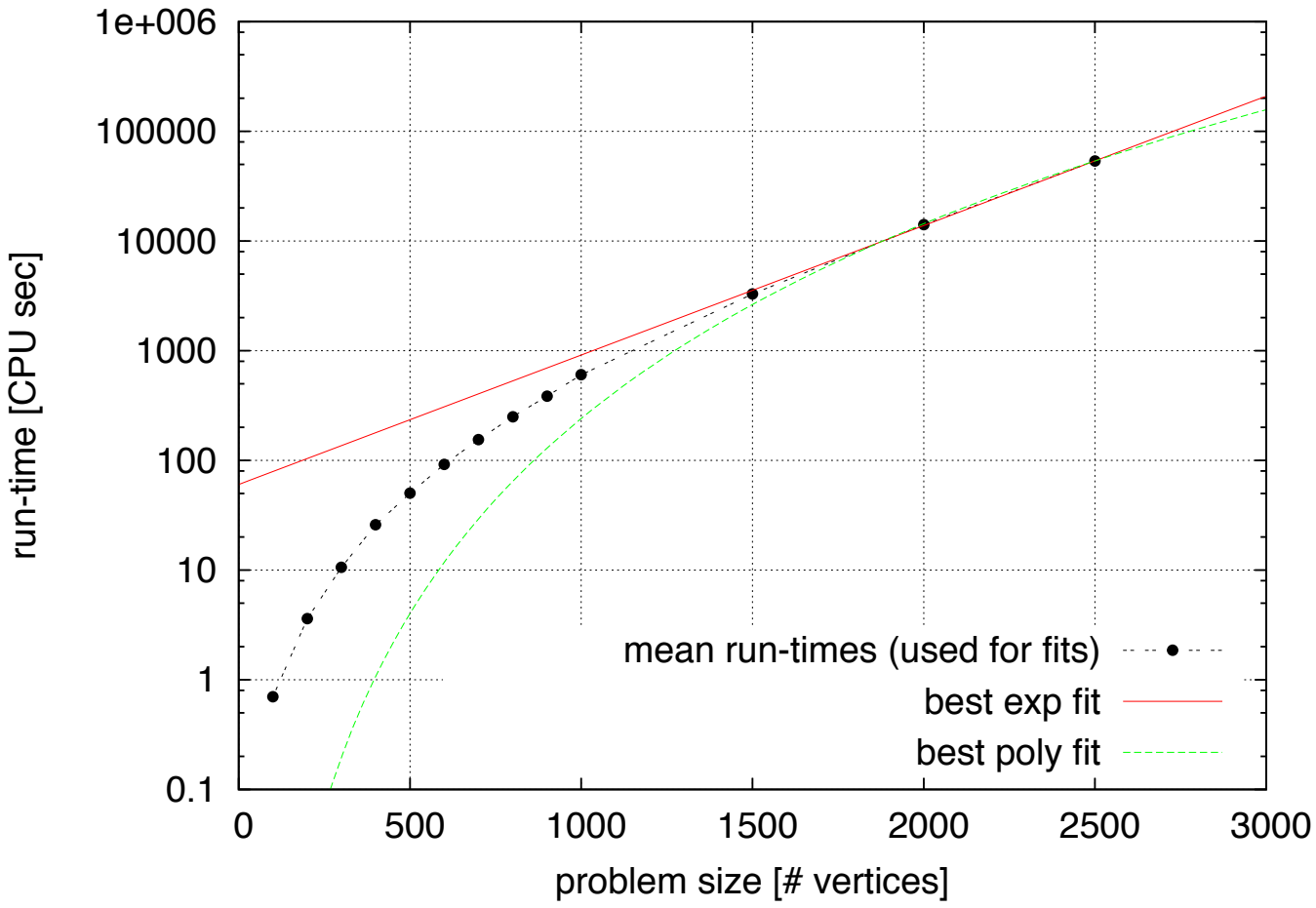
[This slide was not used during the presentation; it was added to make it easier to follow the slide deck.]

Simple, very natural idea:
Model scaling using functions; chosen based on different hypotheses on scaling (perhaps informed by known complexity result).

Scaling models:

- $a \cdot b^n$ (exponential)
- $a \cdot n^b$ (polynomial)

Model fitting



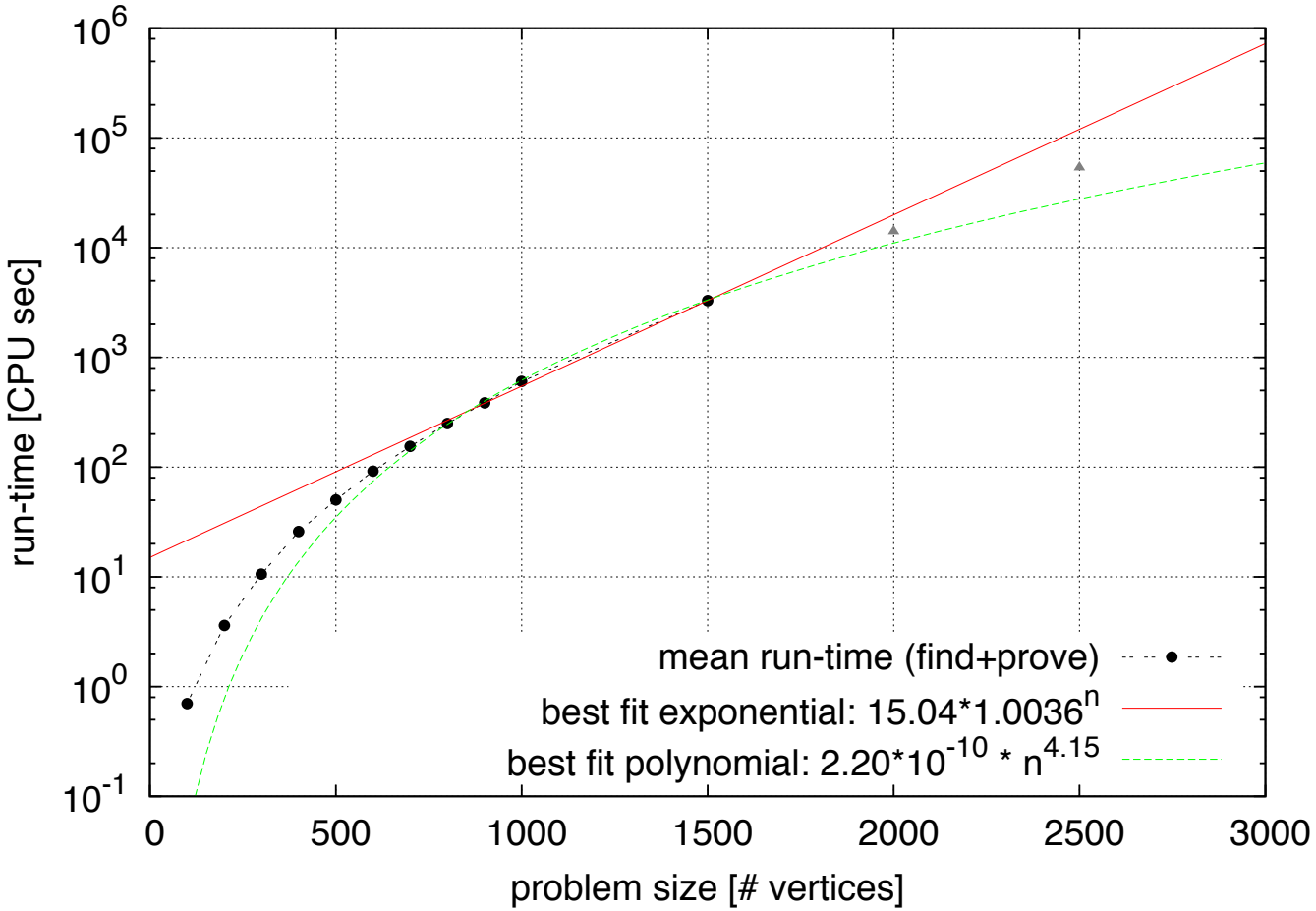
RMSE: exp = 197.58, poly = 227.05

[This slide was not used during the presentation; it was added to make it easier to follow the slide deck.]

Unsurprisingly, the exponential model gives
a (slightly) better fit to the training data.

Of course, using the scientific method, we should
challenge our models by assessing their predictions
on previously unseen data.
Extrapolation challenges particularly interesting
and difficult.

Extrapolation challenge



RMSE (test): exp = 5820.66, poly = 3058.22

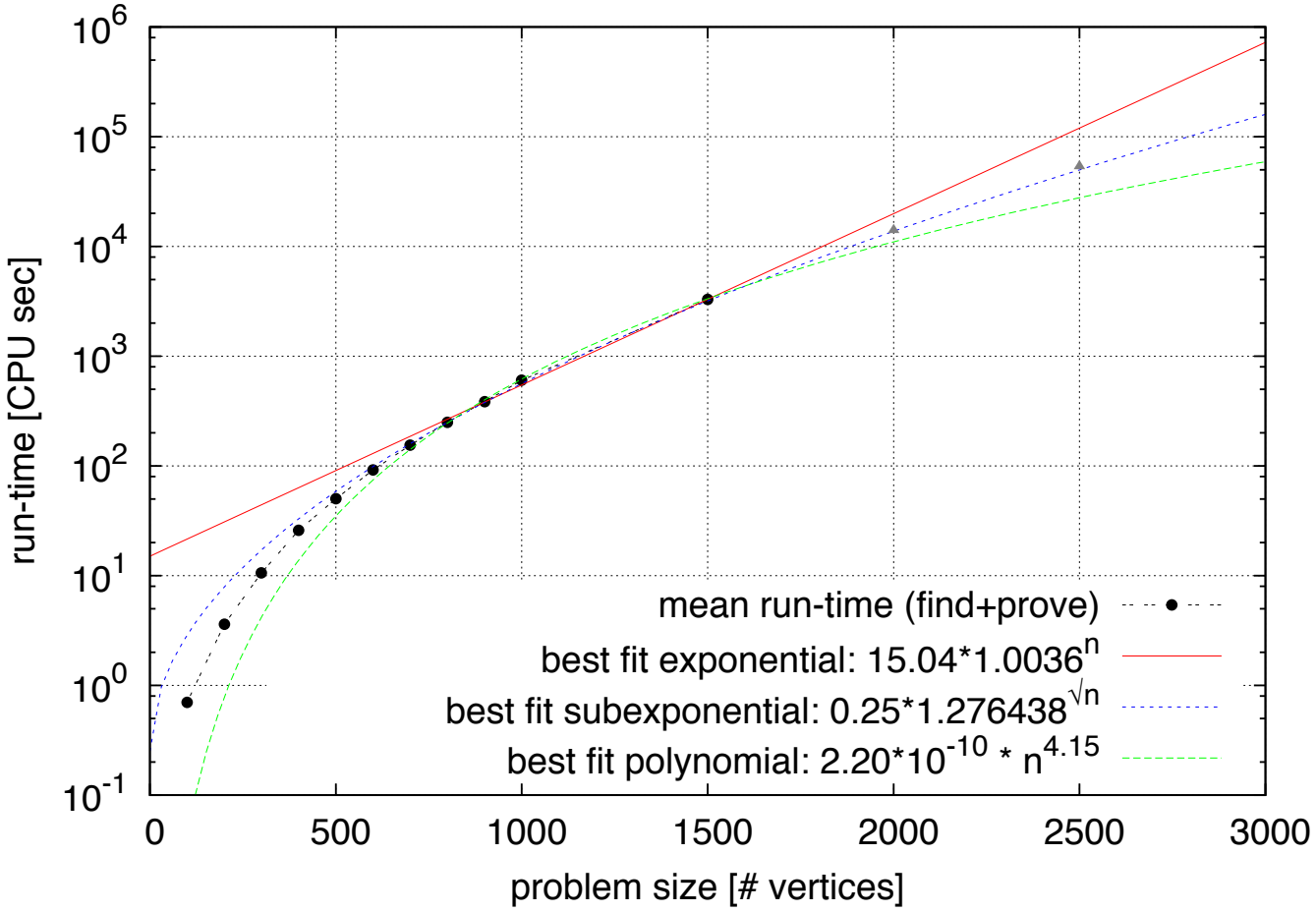
[This slide was not used during the presentation; it was added to make it easier to follow the slide deck.]

Surprise!
The polynomial model gives a better fit
(quantitatively and qualitatively).

Scaling models:

- $a \cdot b^n$ (exponential)
- $a \cdot n^b$ (polynomial)
- $a \cdot b^{\sqrt{n}}$ (root-exponential)

Extrapolation challenge



RMSE (test): exp = 5820.66, poly = 3058.22, root-exp = 329.79

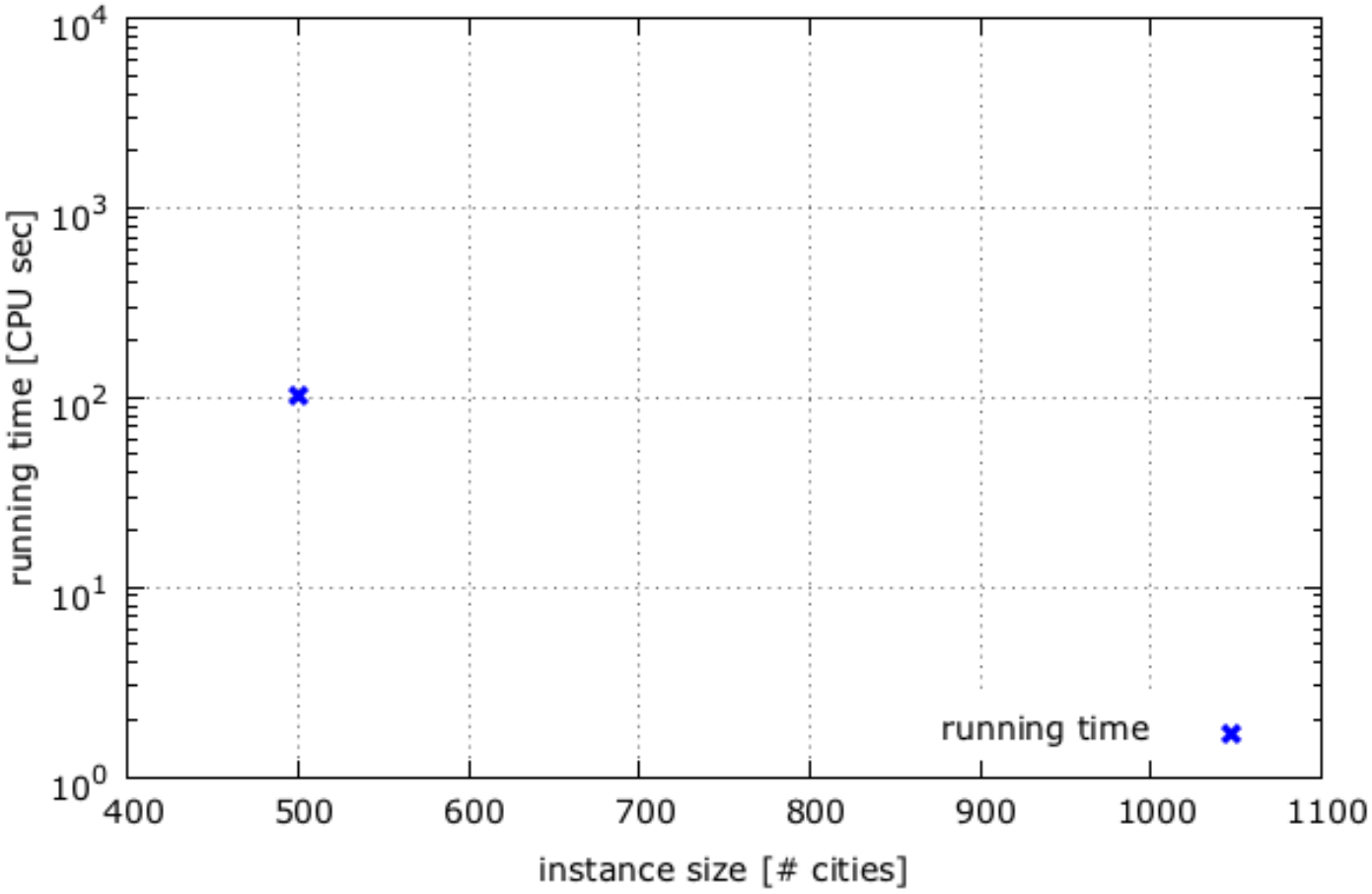
How close is close enough?

$\rightsquigarrow$ **want:** principled method for validating empirical scaling models

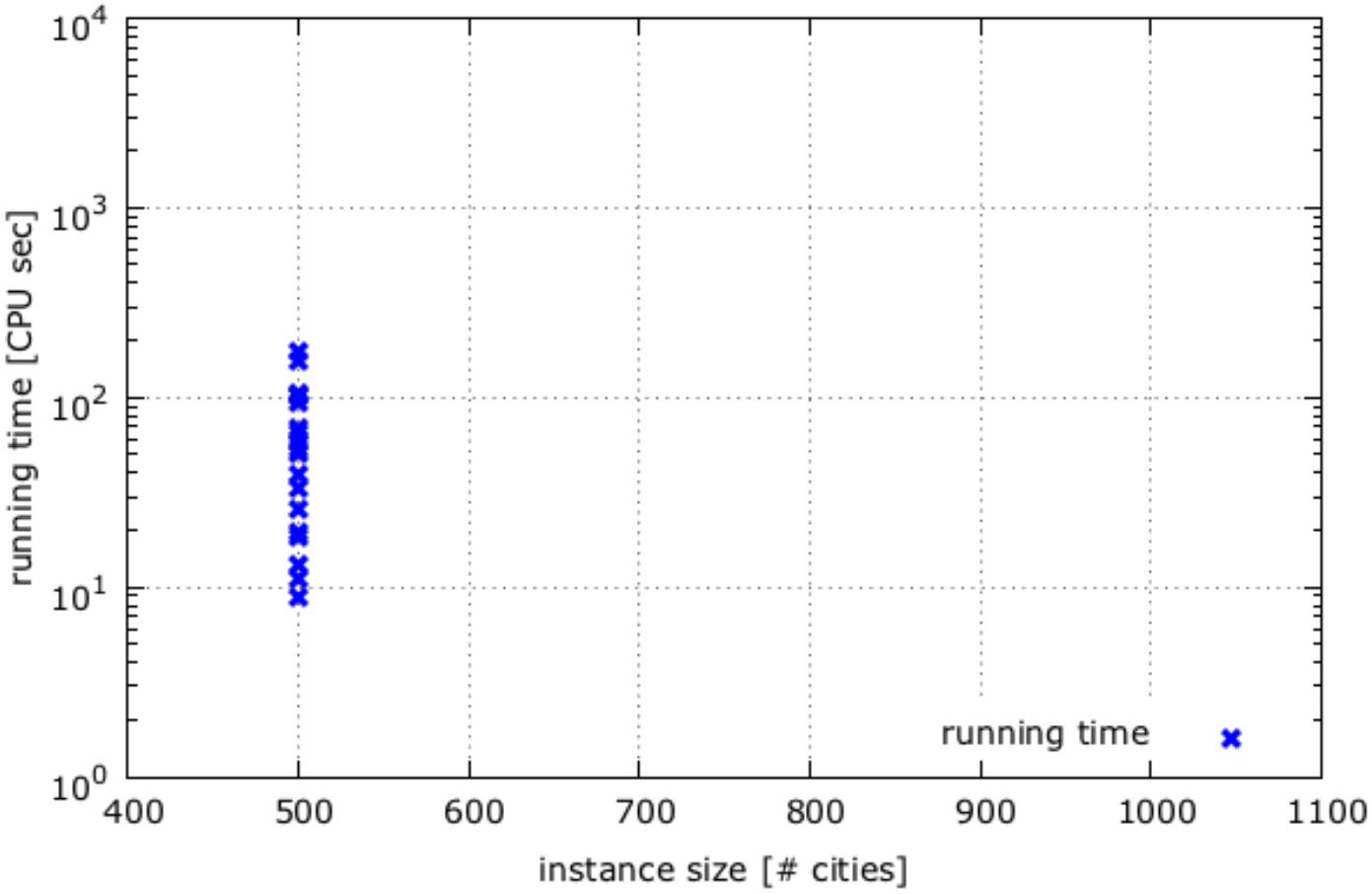
[This slide was not used during the presentation; it was added to make it easier to follow the slide deck.]

Remember: the data used for fitting scaling models is based on runs on many instances for each instance size, and so is the data used for assessing predictions obtained from the model.

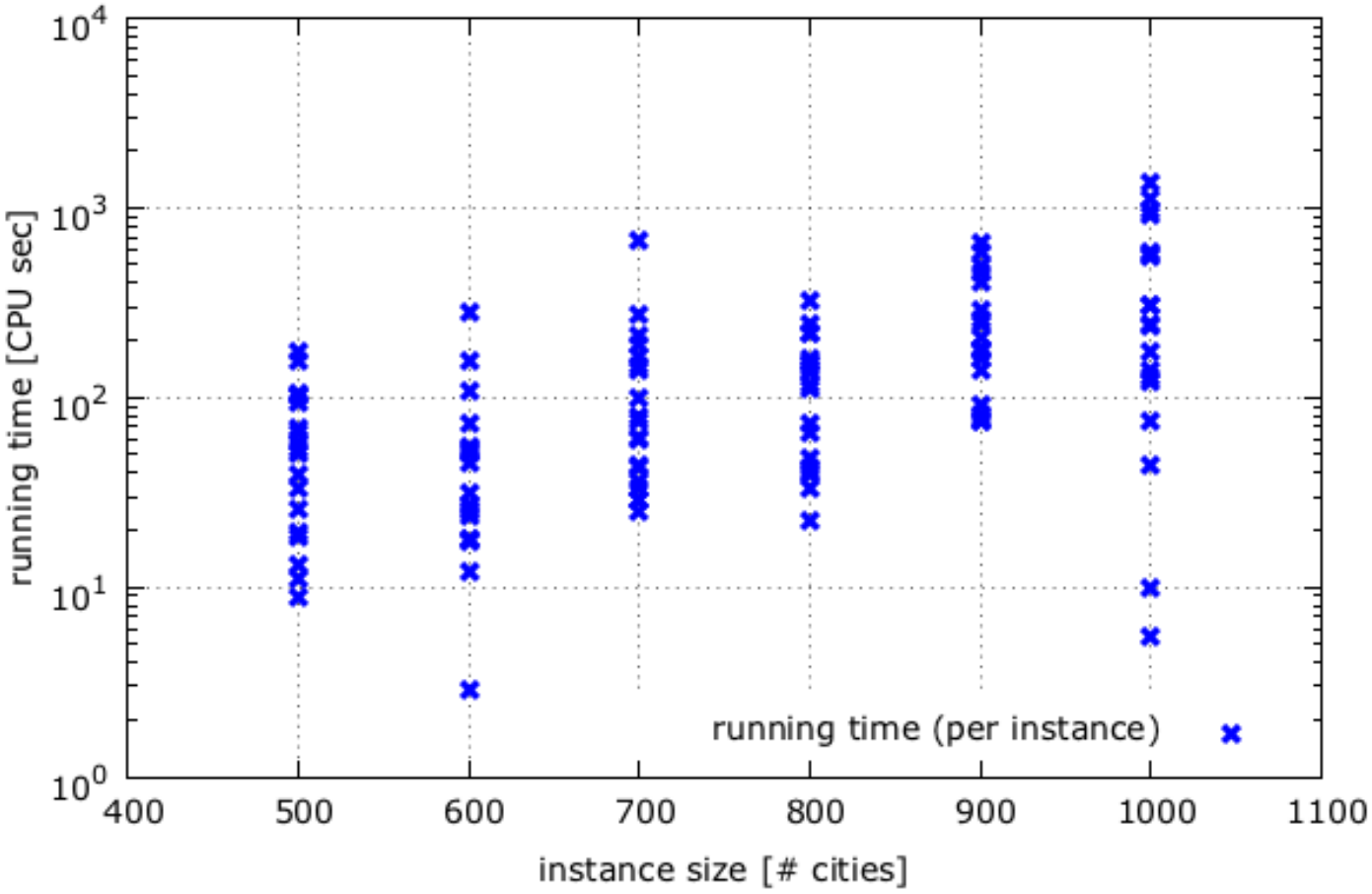
Solver running times



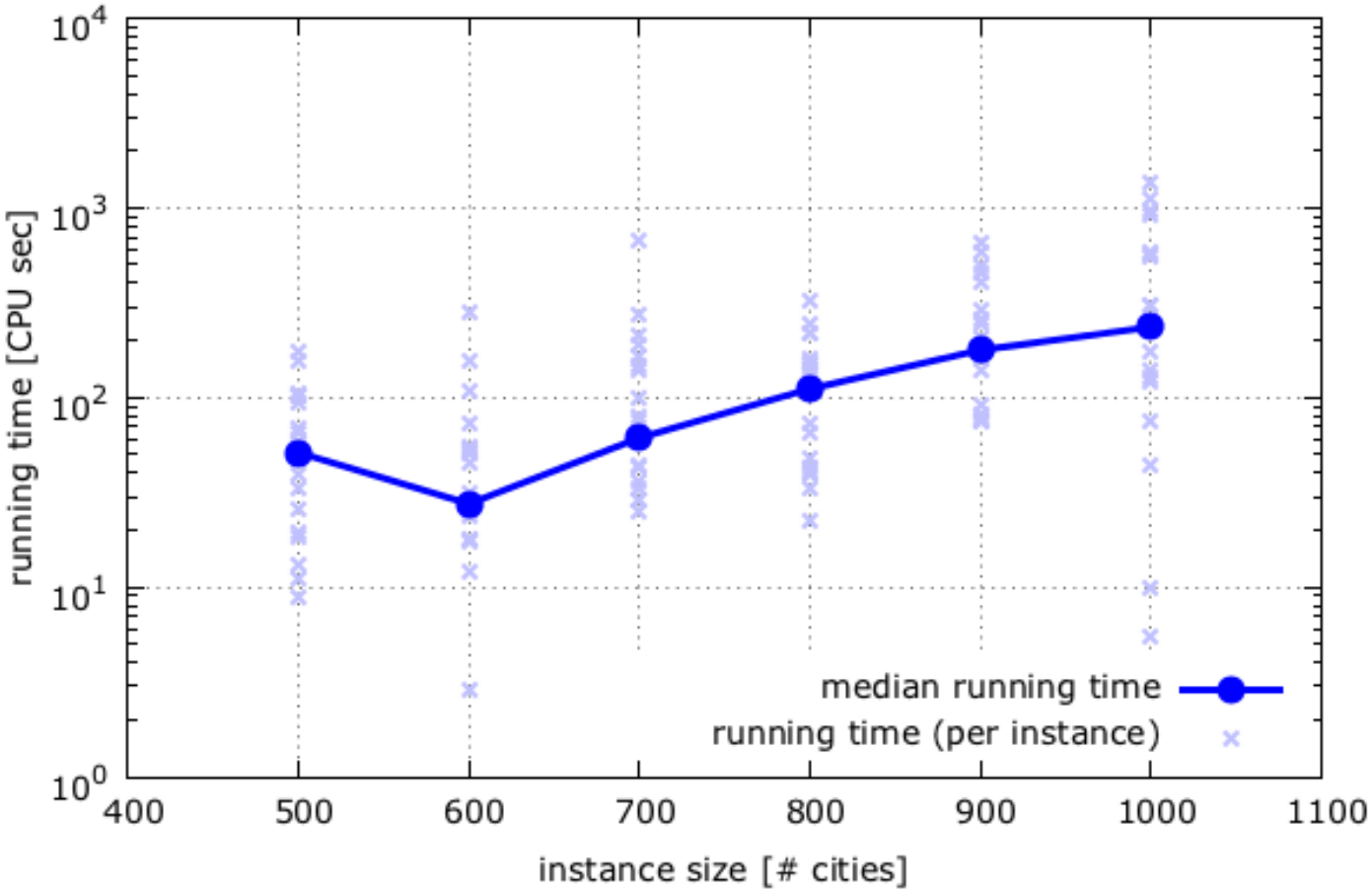
Solver running times



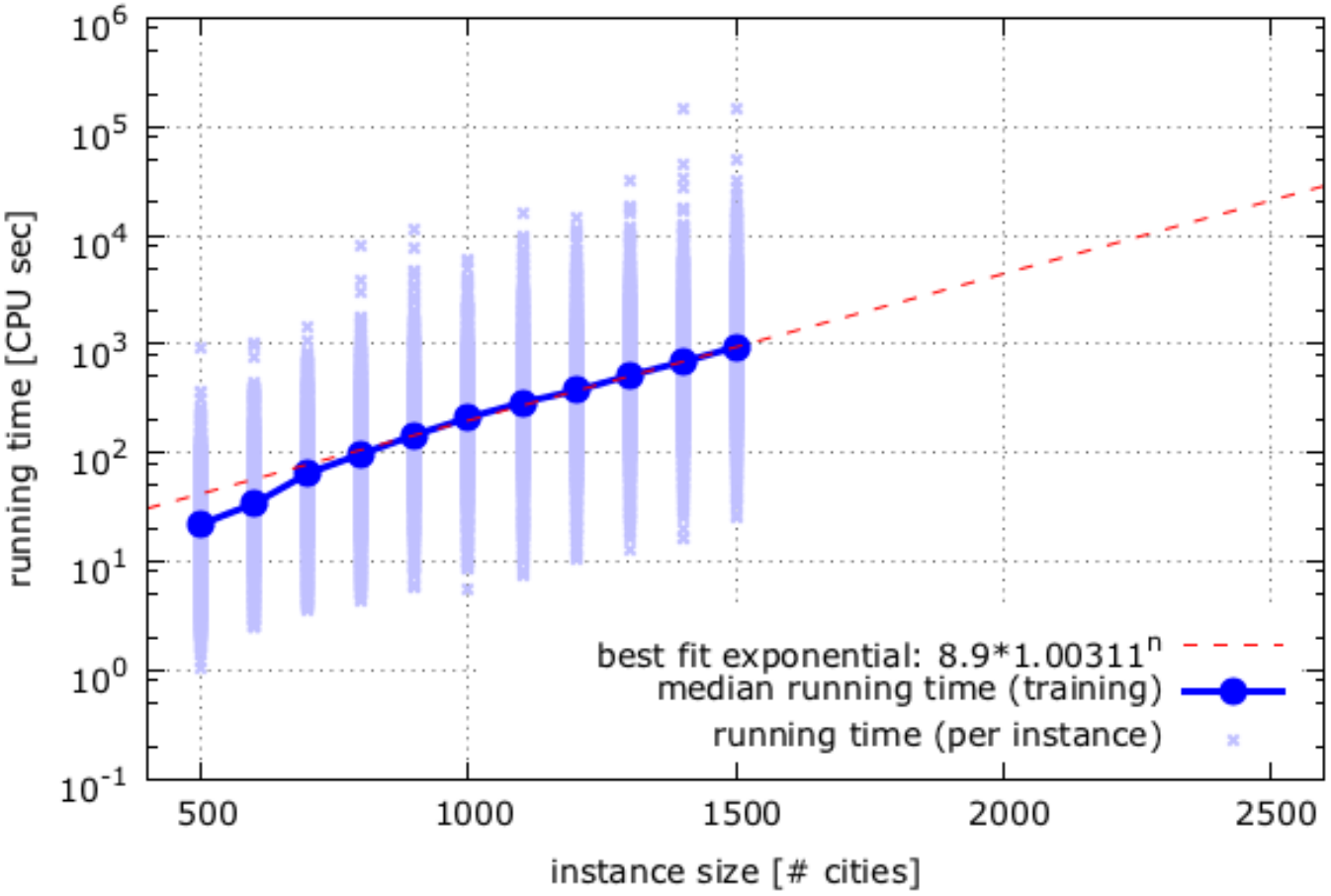
Solver running times



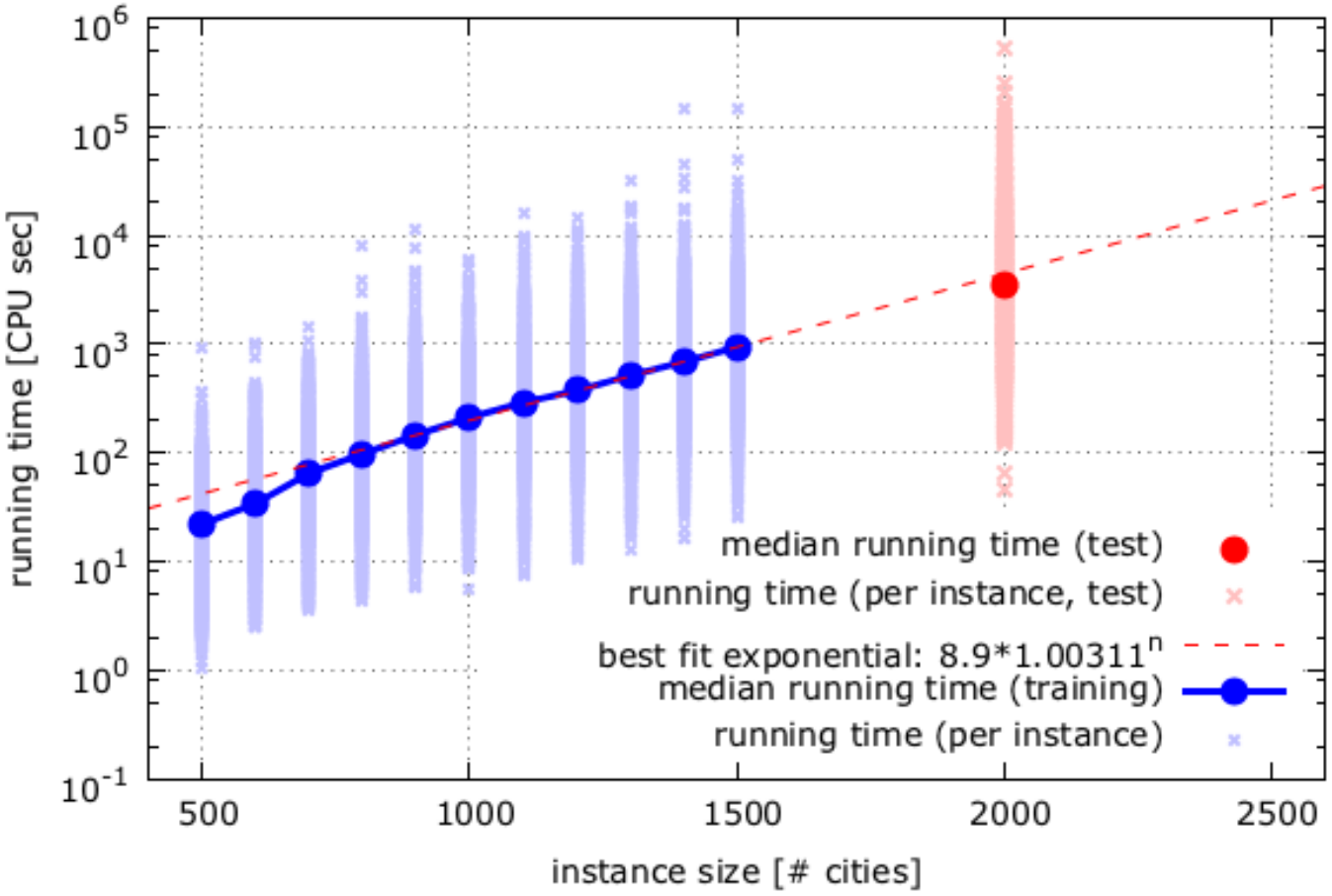
Solver running times



Solver running times + scaling model



Solver running times + scaling model



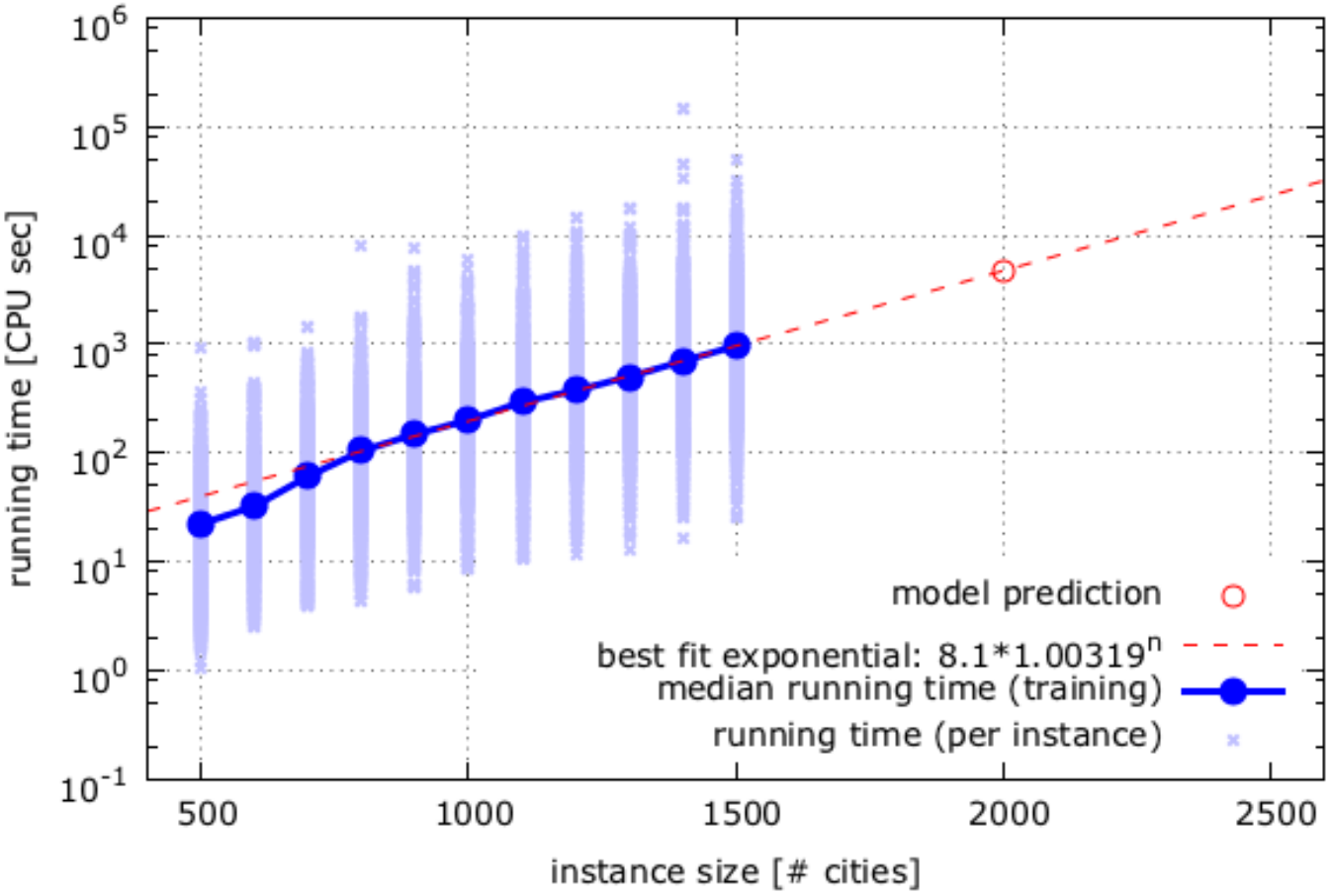
Dependence of empirical scaling models on given set (= sample) of instances?

$\rightsquigarrow$ model validation using bootstrap confidence intervals (HH 2009)

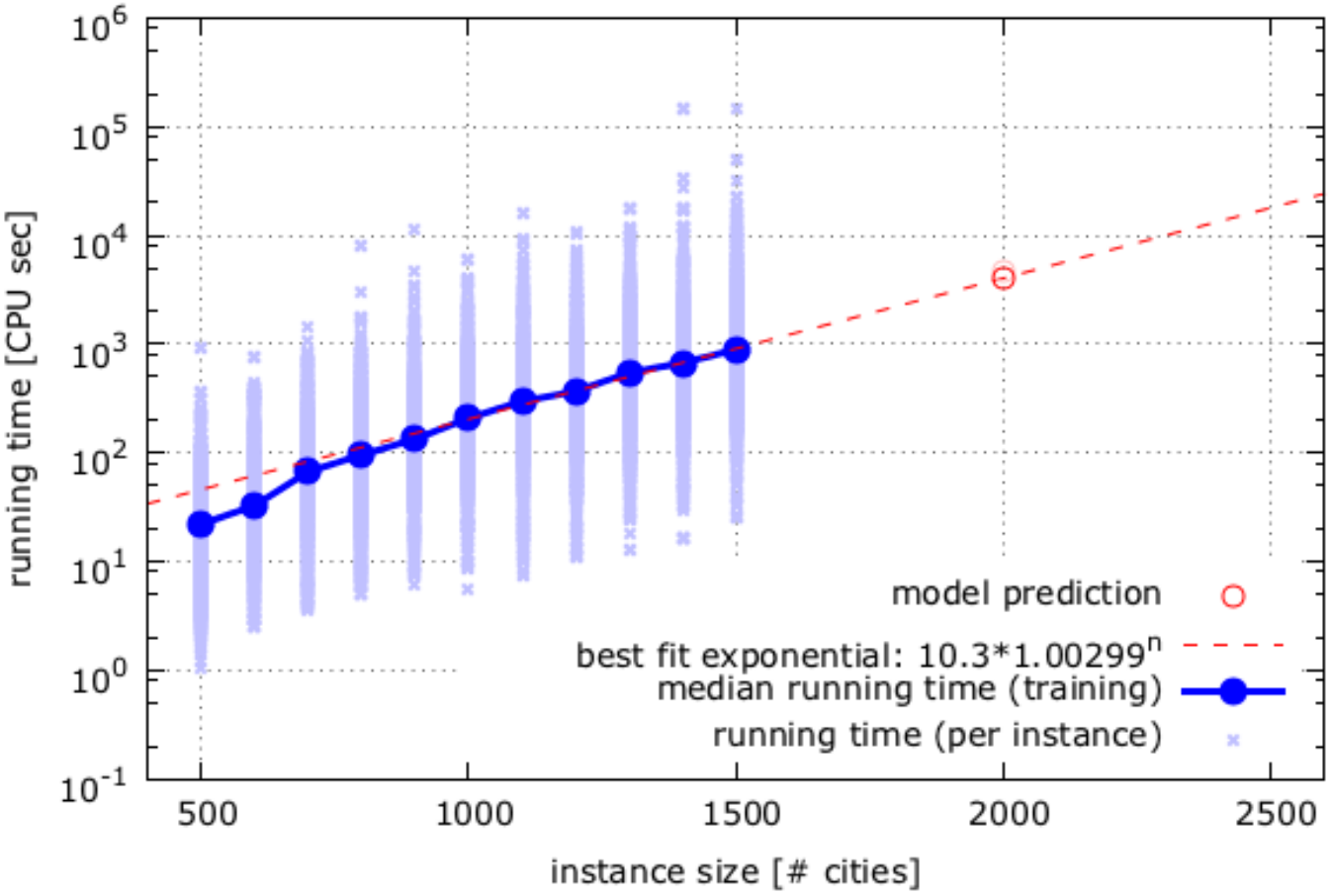
[This slide was not used during the presentation; it was added to make it easier to follow the slide deck.]

Key observation:
Scaling model depends on the sets of instances for each n; resampling those sets will lead to a set of models and therefore sets of predictions.

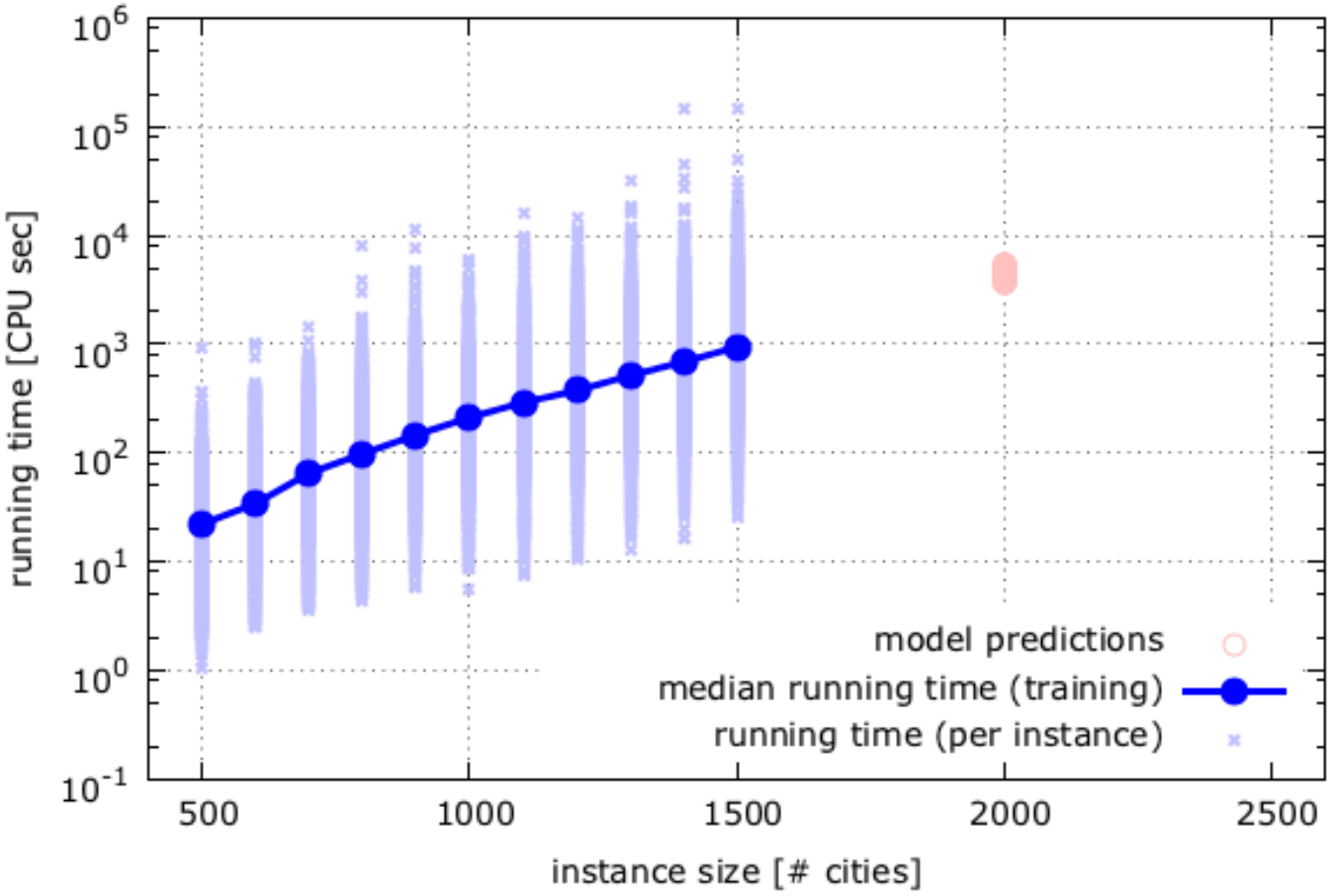
Solver running times + scaling model



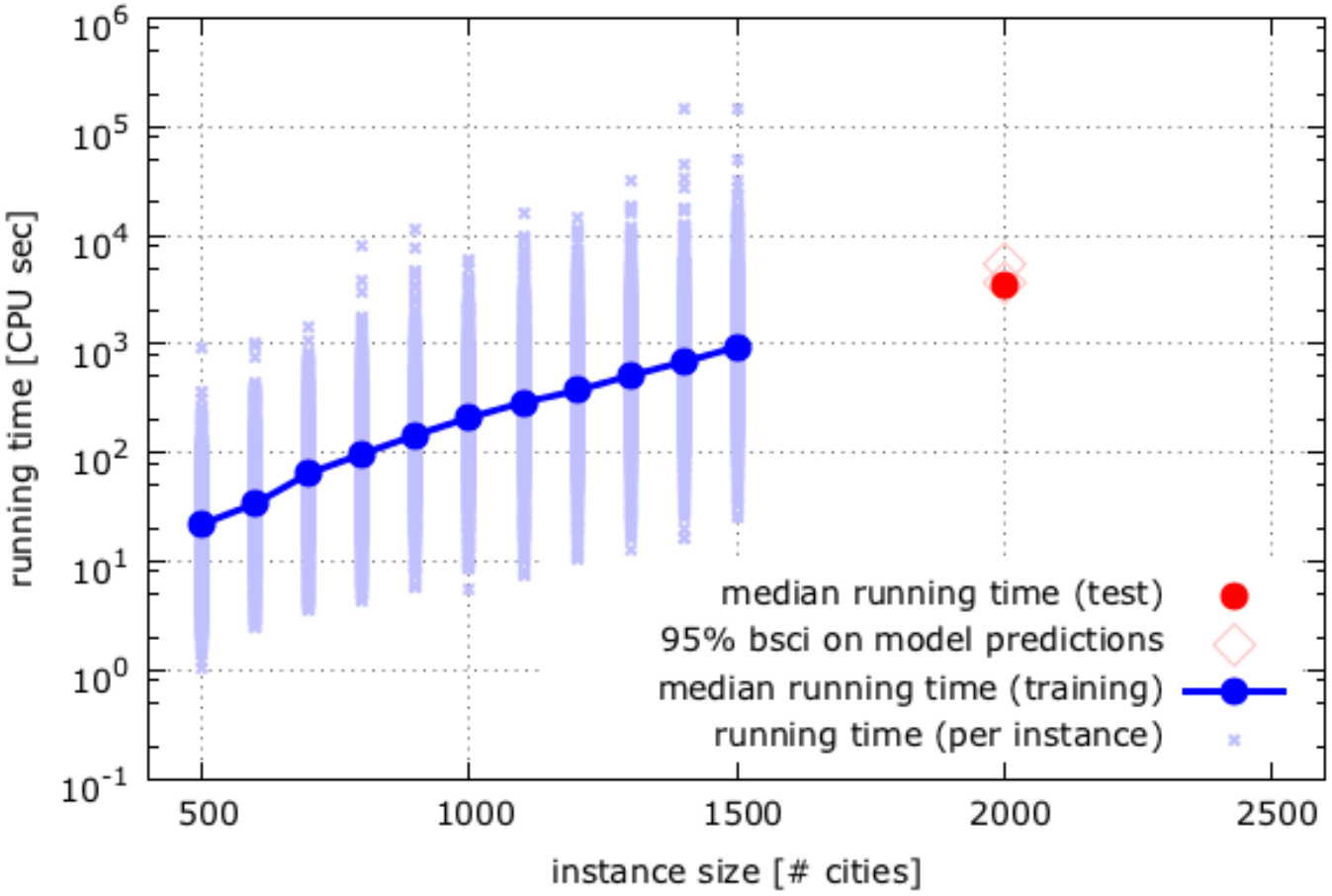
Solver running times + scaling model



Solver running times + scaling model



Solver running times + scaling model



ESA: Fully automated empirical scaling analysis (Mu & HH 2015)
$\rightsquigarrow$ www.cs.ubc.ca/labs/beta/Projects/ESA

Interesting results for TSP
(HH & Stützle 2014; Dubois-Lacoste, HH, Stützle 2015;
Mu, Dubois-Lacoste, HH, Stützle – to appear)

Even more interesting results for SAT
(Mu & HH 2015)

Empirical Scaling of Complete SAT Solvers on Phase-Transition Random 3-SAT

kcnfs:	$4.30 \times 10^{-5} \times 1.03411^{n}$
march_hi:	$4.93 \times 10^{-5} \times 1.03295^{n}$
march_br:	$1.02 \times 10^{-5} \times 1.03048^{n}$

trained on $n = 200..400$,
passed challenge at $n = 450..550$

[This slide was not used during the presentation; it was added to make it easier to follow the slide deck.]

From our set of models we can also obtain confidence intervals for model parameters, which indicate uncertainty associated with the paramater values obtained from model fitting.

Empirical Scaling of Complete SAT Solvers on Phase-Transition Random 3-SAT

kcnfs	$[3.33 \times 10^{-5}, 1.08 \times 10^{-4}] \times [1.03136, 1.03476]^n$
march_hi	$[2.90 \times 10^{-5}, 1.73 \times 10^{-4}] \times [1.02928, 1.03433]^n$
march_br	$[2.61 \times 10^{-5}, 1.09 \times 10^{-4}] \times [1.03064, 1.03466]^n$

trained on $n = 200..400$,
passed challenge at $n = 450..550$

Empirical Scaling of Incomplete SAT Solvers on Phase-Transition Random 3-SAT

$$\text{WalkSAT/SKC:} \quad [2.58 \times 10^{-12}, 8.64 \times 10^{-10}] \times n^{[2.80,3.77]}$$

$$\text{probSAT:} \quad [5.00 \times 10^{-12}, 1.03 \times 10^{-8}] \times n^{[2.40,3.67]}$$

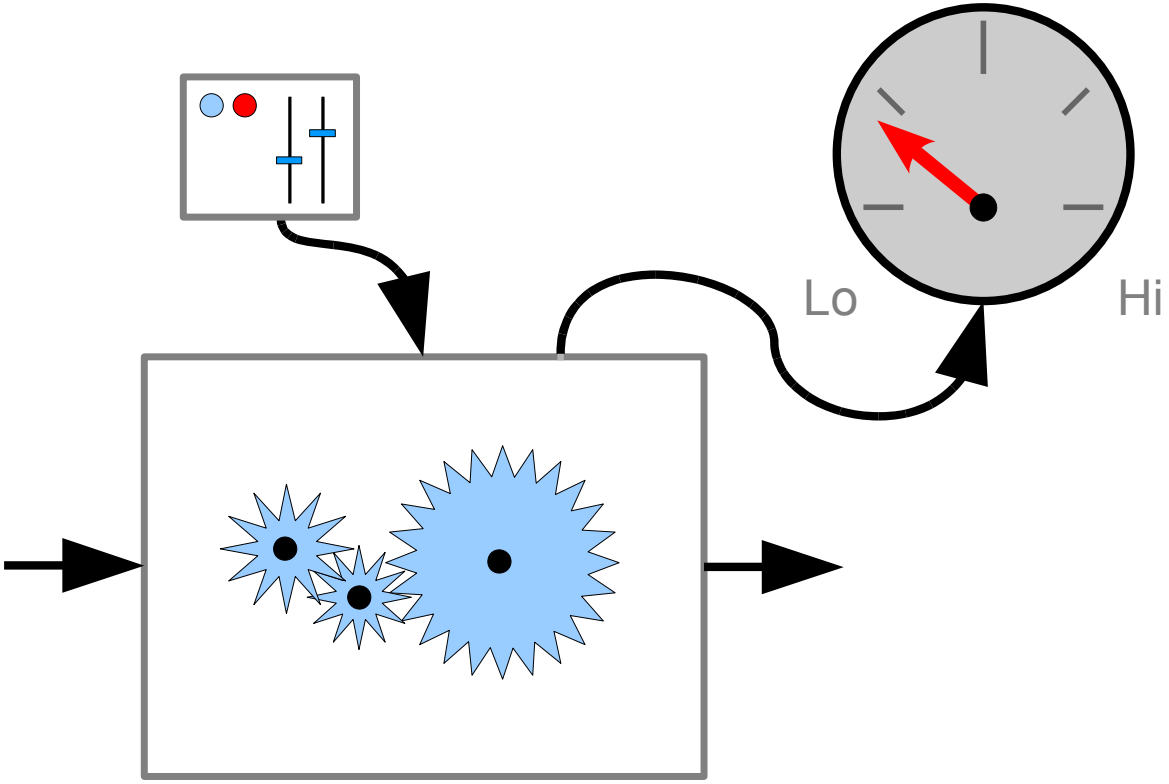
$$\text{BalancedZ:} \quad [3.66 \times 10^{-11}, 4.54 \times 10^{-9}] \times n^{[2.60,3.42]}$$

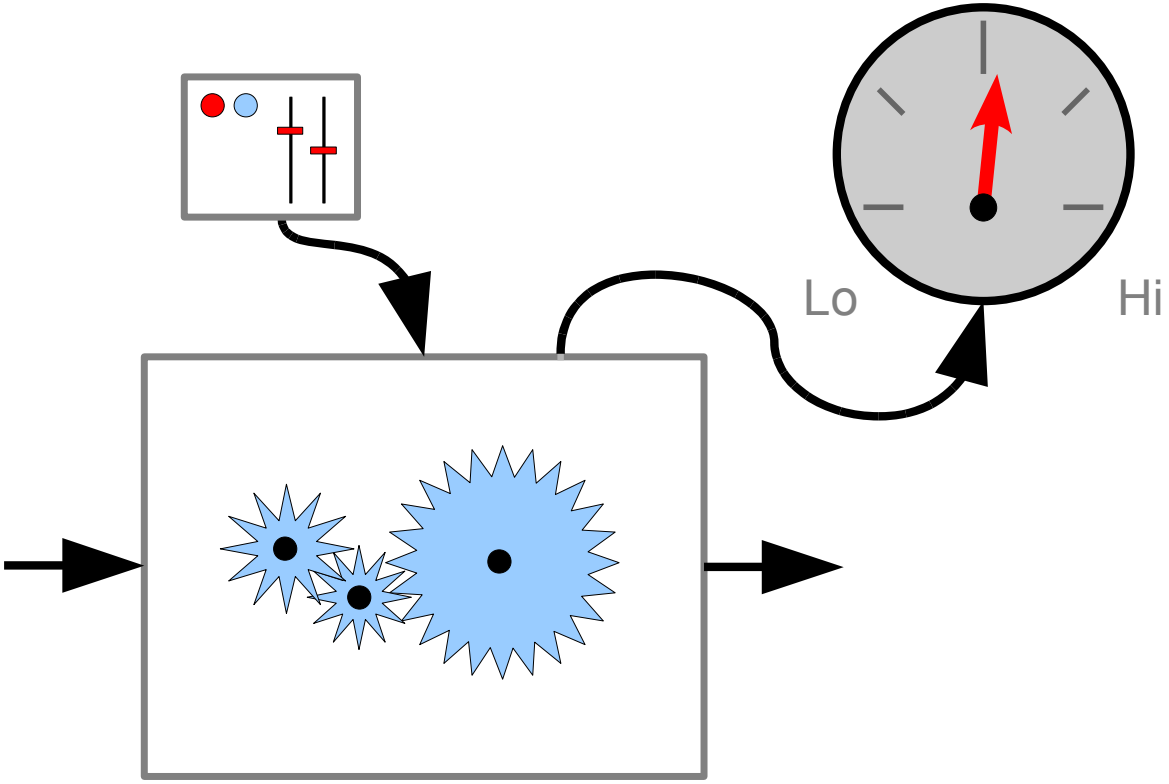
trained on $n = 200..500$,
passed challenge at $n = 600..1000$

2

[This slide was not used during the presentation; it was added to make it easier to follow the slide deck.]

Many (most?) algorithms have parameters
that impact their performance.


Lo
Hi


Lo
Hi

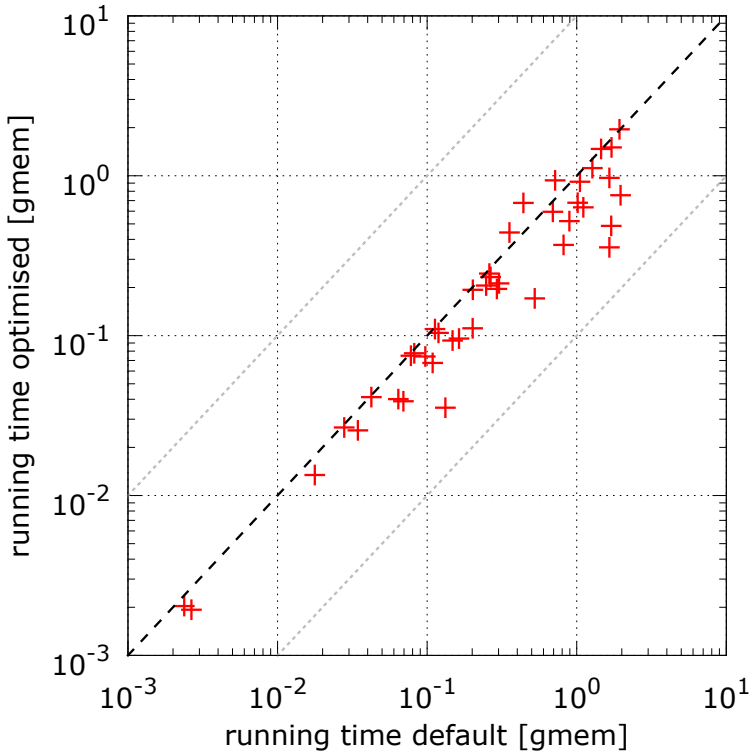
"Parameter optimization for general broad-spectrum use is a daunting task [...]

How could then *any* set of defaults be recommended, without an enormous expense of time and money? Fortunately, there's a way out of this dilemma, thanks to advances in the theory of learning."

Donald Knuth, The Art of Computer Programming,
Vol. 4, Fascicle 6, p. 125

Knuth's sat13 (7.2.2.2C) on diverse set of instances

(very easy to medium; trained on very easy only)

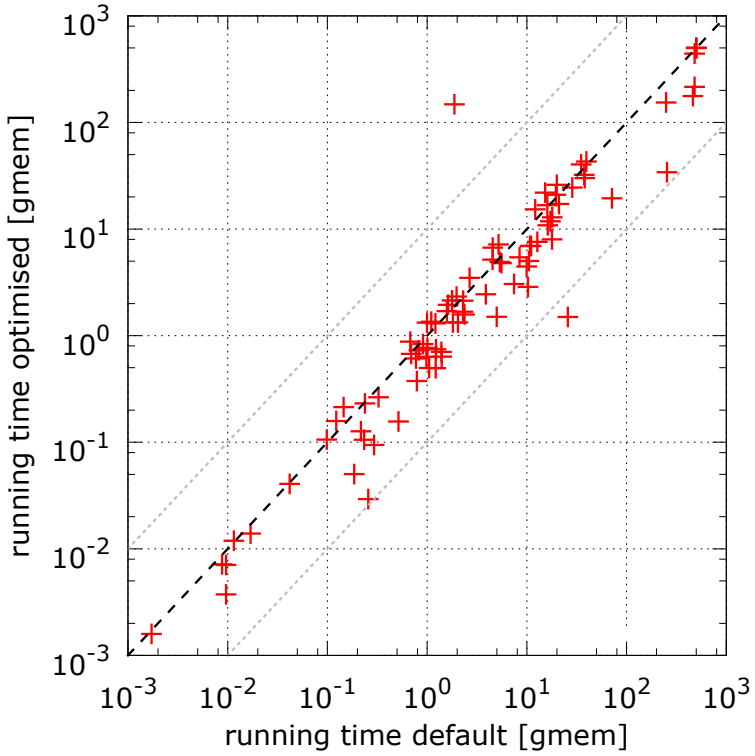


mean running time 0.572 → 0.402 gmems;

geometric average speedup: 1.414-fold

Knuth's sat13 (7.2.2.2C) on diverse set of instances

(TAOCP testing instances; trained on very easy)



mean running time 47.4 → 36.9 gmems;

geometric average speedup: 1.357-fold

[This slide was not used during the presentation; it was added to make it easier to follow the slide deck.]

Don's algorithm has few parameters,
and he optimised the default values manually.
The performance differences seen here are
large compared to those, e.g., between
gold and silver medal winners in
SAT competitions.

The algorithm configuration problem

Given:

- parameterised target algorithm A
  with configuration space C
- set of (training) inputs I
- performance metric m
  (w.l.o.g. to be minimised)

Want: $c^* \in \arg\min_{c \in C} m(A[c], I)$

Algorithm configuration approaches:

- Advanced sampling methods

  (*e.g.*, REVAC, REVAC++ – Nannen & Eiben 2006–09)

- Racing

  (*e.g.*, F-Race – Birattari, Stützle, Paquete, Varrentrapp 2002;
  Iterative F-Race – Balaprakash, Birattari, Stützle 2007;
  irace package – López-Ibáñez, Dubois-Lacoste, Stützle, Birattari 2011;
  irace+capping – Péres-Cáceres, López-Ibáñez, HH, Stützle, Birattari 2017)

- Model-free search

  (*e.g.*, ParamILS – Hutter, HH, Stützle 2007;
  Hutter, HH, Leyton-Brown, Stützle 2009)

- Sequential model-based optimisation

  (*e.g.*, SPO – Bartz-Beielstein 2006; SMAC – Hutter, HH, Leyton-Brown 2011–12)

Algorithm configuration approaches:

- Advanced sampling methods

  (*e.g.*, REVAC, REVAC++ – Nannen & Eiben 2006–09)

- Racing

  (*e.g.*, F-Race – Birattari, Stützle, Paquete, Varrentrapp 2002;
  Iterative F-Race – Balaprakash, Birattari, Stützle 2007;
  irace package – López-Ibáñez, Dubois-Lacoste, Stützle, Birattari 2011;
  irace+capping – Péres-Cáceres, López-Ibáñez, HH, Stützle, Birattari 2017)

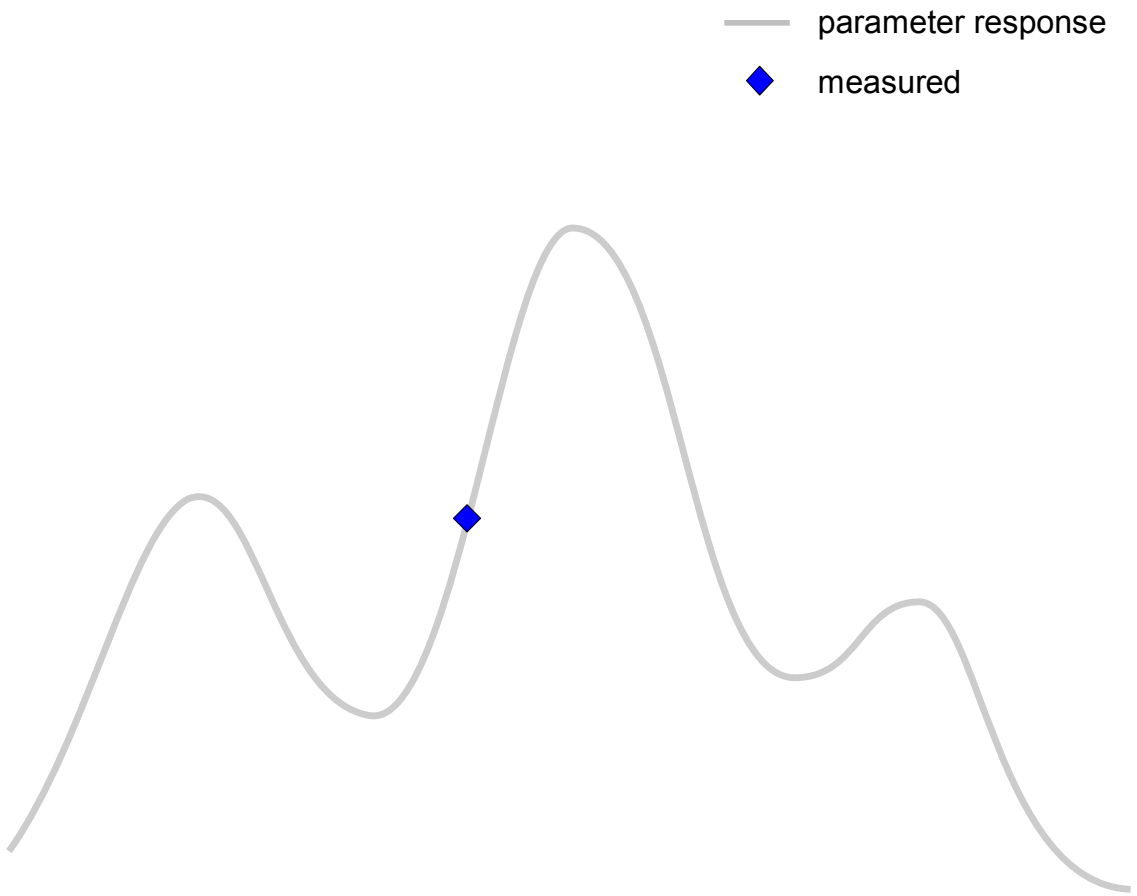
- Model-free search

  (*e.g.*, ParamILS – Hutter, HH, Stützle 2007;
  Hutter, HH, Leyton-Brown, Stützle 2009)

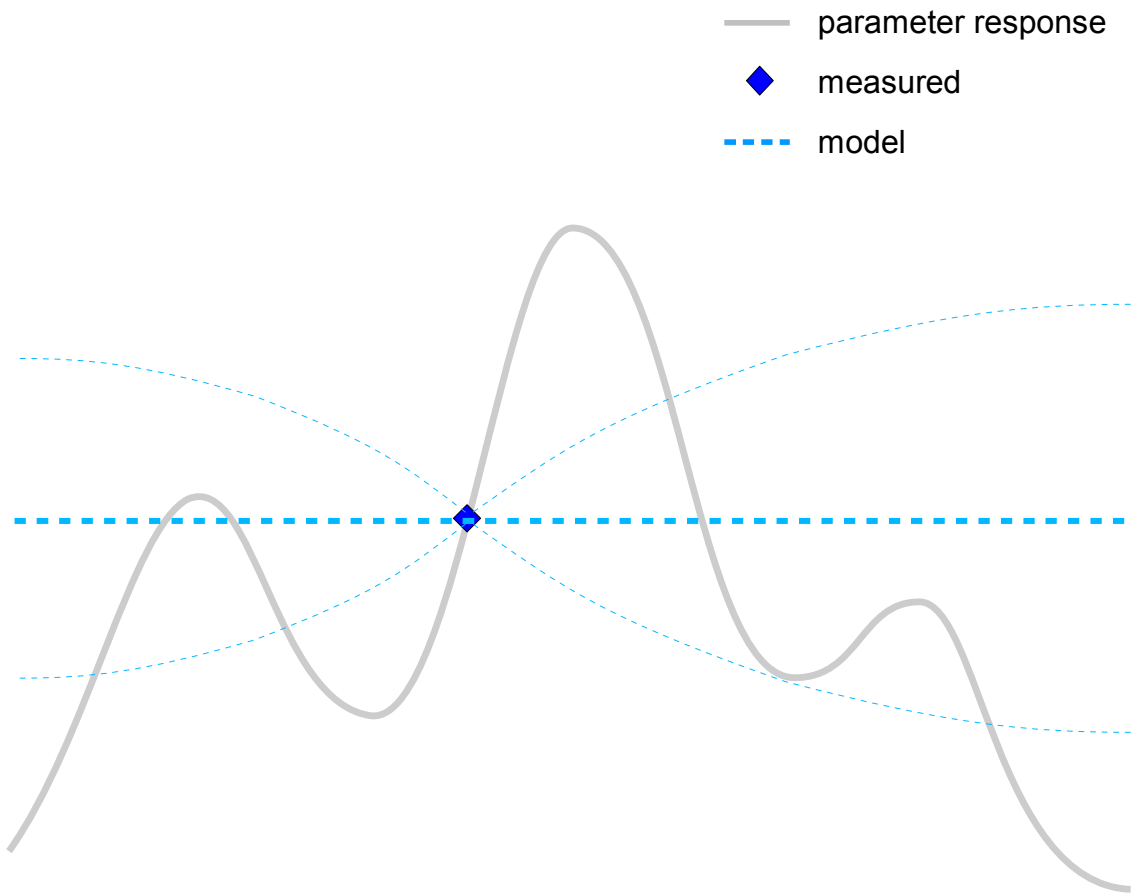
- Sequential model-based optimisation

  (*e.g.*, SPO – Bartz-Beielstein 2006; SMAC – Hutter, HH, Leyton-Brown 2011–12)

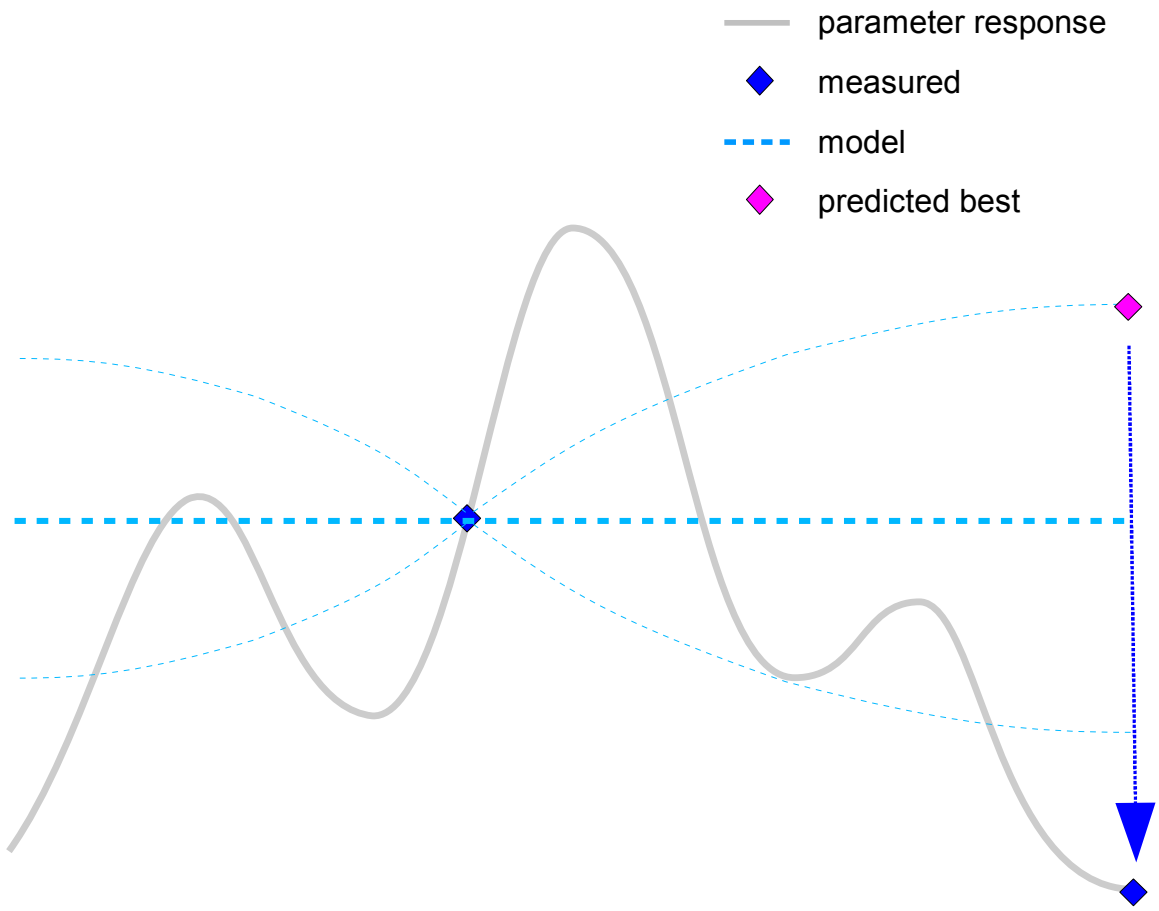
Sequential Model-based Optimisation



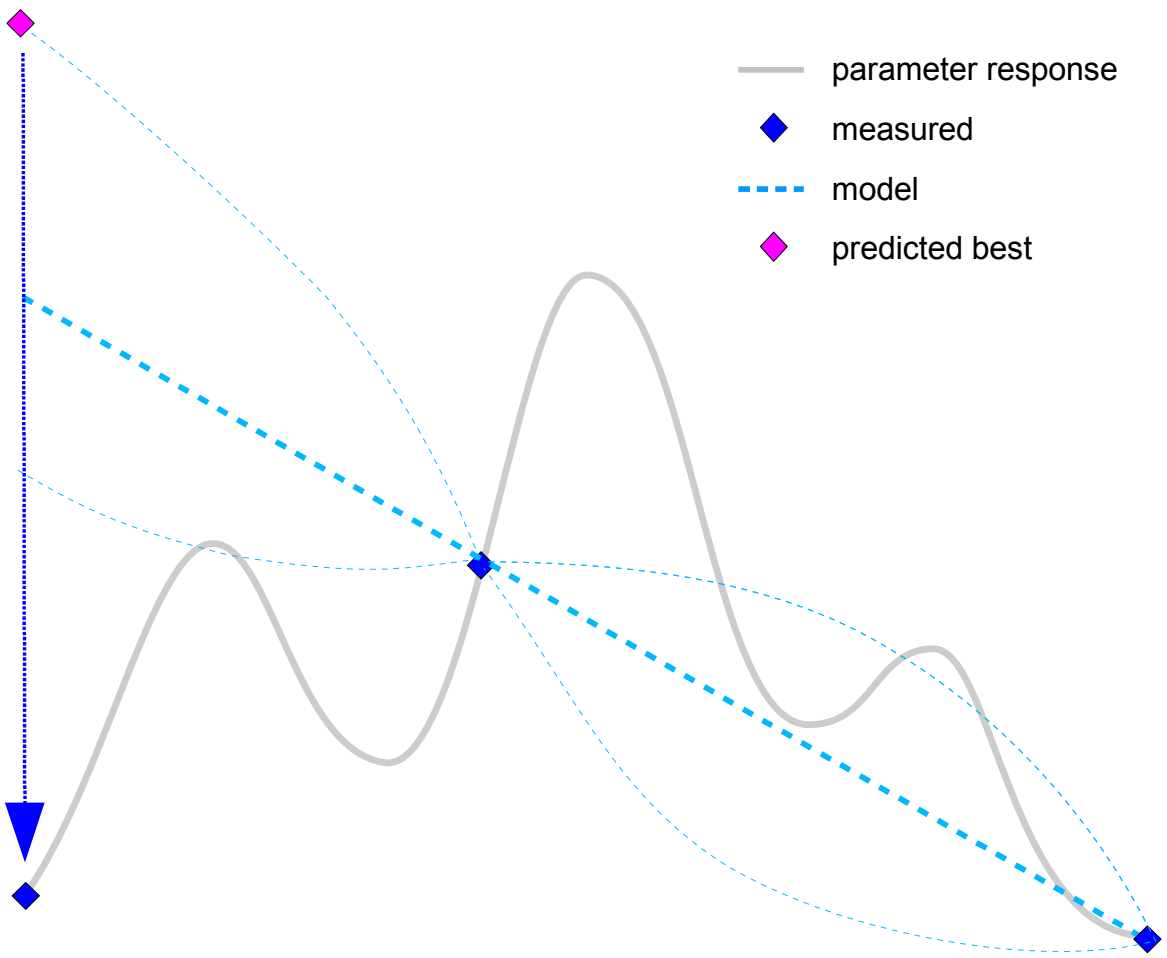
Sequential Model-based Optimisation



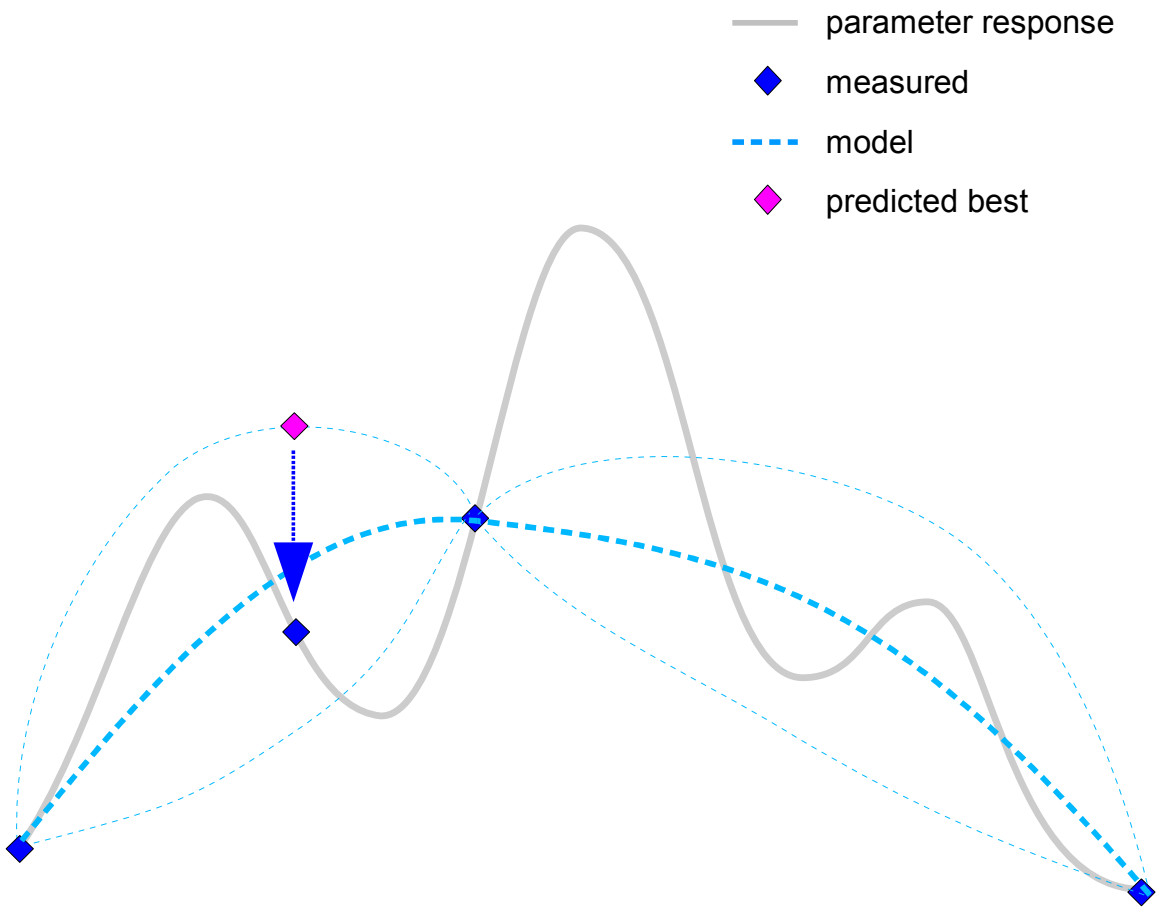
Sequential Model-based Optimisation



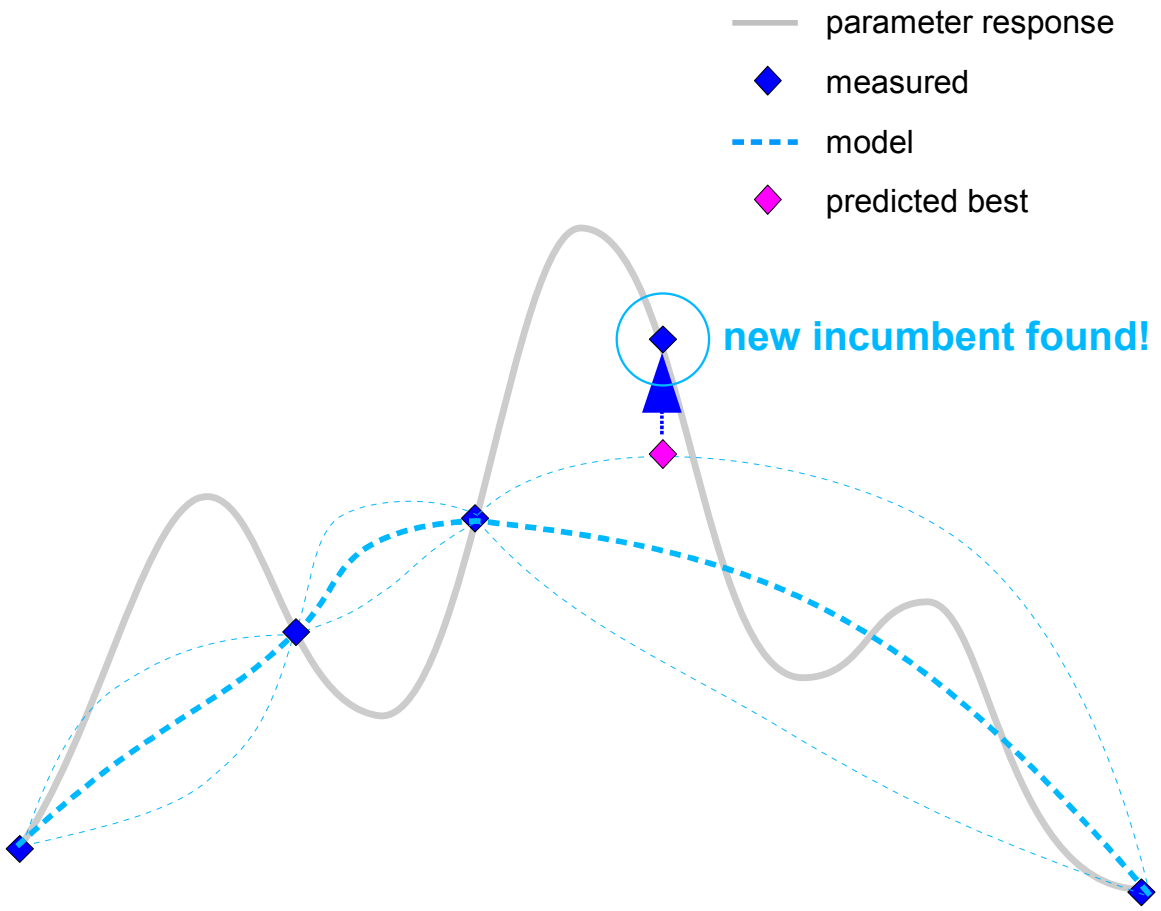
Sequential Model-based Optimisation



Sequential Model-based Optimisation



Sequential Model-based Optimisation



Sequential Model-based Algorithm Configuration (SMAC)

Hutter, HH, Leyton-Brown (2011)

- uses *random forest model* to predict performance of parameter configurations
- predictions based on algorithm parameters and instance features, aggregated across instances
- finds promising configurations based on *expected improvement criterion*, using multi-start local search and random sampling
- initialisation with single configuration (algorithm default or randomly chosen)

Excellent results on SAT:

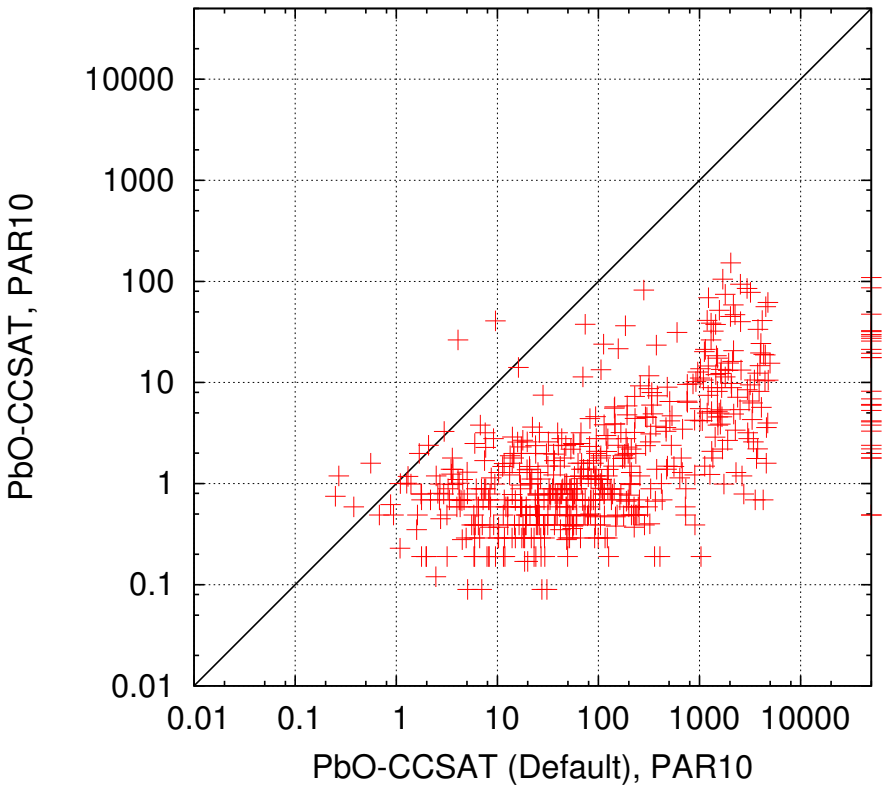
- Spear: 41 parameters, 4.5...500× speed-up
  Hutter, Babić, HH, Hu (2007)
- Captain Jack: state-of-the-art on random, 'industrial-like' benchmarks
  Tompkins, Balint, HH (2011)
- SATenstein: 42 parameters, state-of-the-art on broad range of random and structured SAT instances
  KhudaBukhsh, Xu, HH, Leyton-Brown (2009, 2016)
- PbO-CCSAT: 23 parameters, state-of-the-art on broad range of recent structured SAT instances
  Luo, HH, Cai (under review)

[This slide was not used during the presentation; it was added to make it easier to follow the slide deck.]

PbO-CCSAT is based on configuration checking,
PTN is a very prominent, recent class of
benchmarks taken from work by Heule et al. (2016-17)

PbO-CCSAT on Boolean Pythagorean Triple problem (PTN)

(performance on test instances not used for configuration)

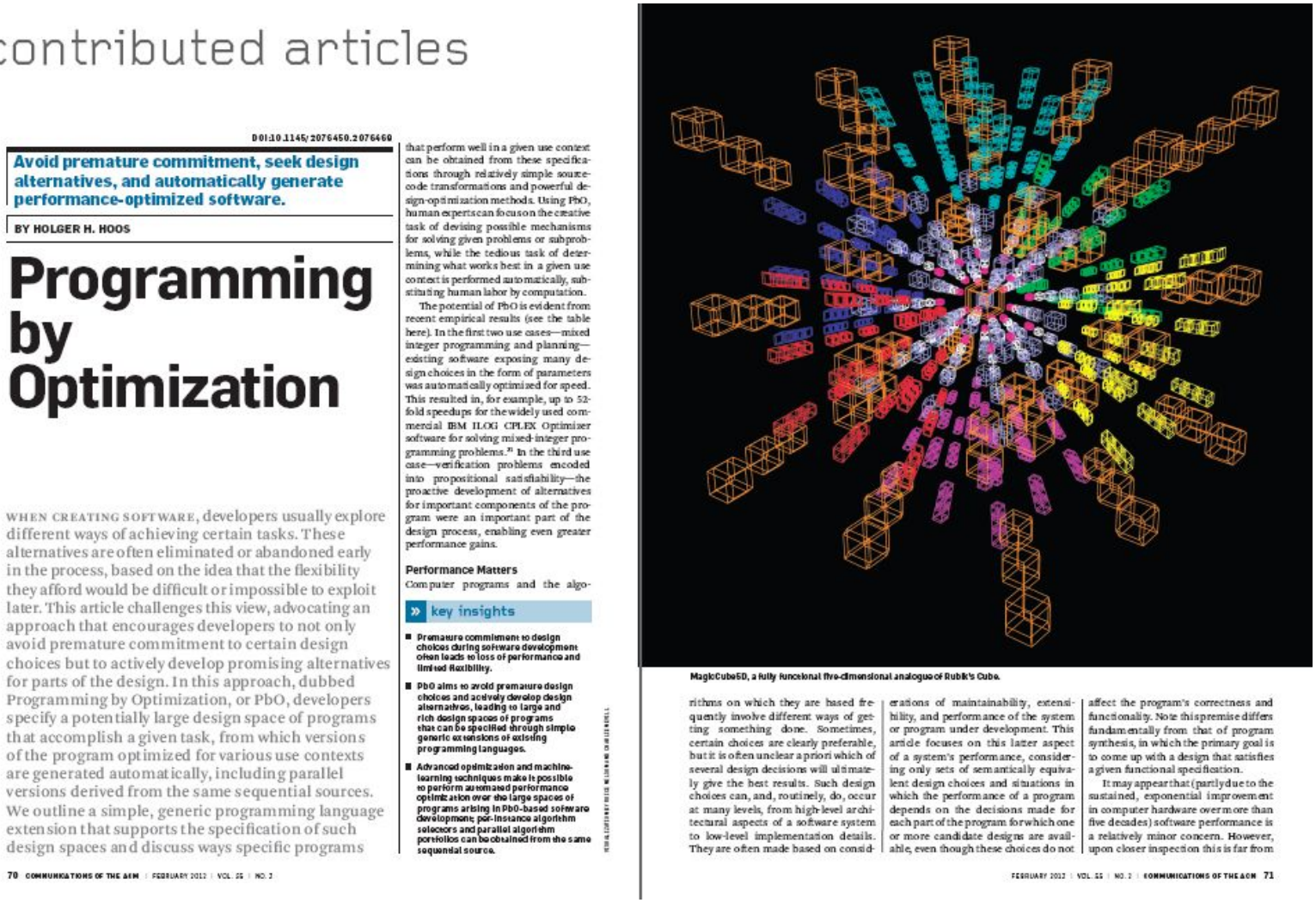


running time (PAR10) 1404.7 → /6.1 CPU sec;

average speedup: 230-fold

... and beyond:

- AI Planning (LPG): 62 parameters, 3...118× speed-up
  Vallati, Fawcett, Gerevini, HH, Saetti (2011)
- Mixed integer programming (CPLEX):
  76 parameters, 2...52× speed-up
  Hutter, Leyton-Brown, HH (2010)
- Machine learning (Auto-WEKA): 768 parameters
  Thornton, Hutter, HH, Leyton-Brown (2013);
  Kotthoff, Thornton, HH, Leyton-Brown (2017)
- garbage collection in Java
  Lengauer & Mössenböck (2014)
- bike sharing rebalancing
  Dell'Amico *et al.* (2016)



contributed articles

DOI:10.1145/2076450.2076460

Avoid premature commitment, seek design alternatives, and automatically generate performance-optimized software.

BY HOLGER H. HOOS

Programming by Optimization

WHEN CREATING SOFTWARE, developers usually explore different ways of achieving certain tasks. These alternatives are often eliminated or abandoned early in the process, based on the idea that the flexibility they afford would be difficult or impossible to exploit later. This article challenges this view, advocating an approach that encourages developers to not only avoid premature commitment to certain design choices but to actively develop promising alternatives for parts of the design. In this approach, dubbed Programming by Optimization, or PbO, developers specify a potentially large design space of programs that accomplish a given task, from which versions of the program optimized for various use contexts are generated automatically, including parallel versions derived from the same sequential sources. We outline a simple, generic programming language extension that supports the specification of such design spaces and discuss ways specific programs that perform well in a given use context can be obtained from these specifications through relatively simple source-code transformations and powerful design-optimization methods. Using PbO, human experts can focus on the creative task of devising possible mechanisms for solving given problems or subproblems, while the tedious task of determining what works best in a given use context is performed automatically, substituting human labor by computation.

The potential of PbO is evident from recent empirical results (see the table here). In the first two use cases—mixed integer programming and planning—existing software exposing many design choices in the form of parameters was automatically optimized for speed. This resulted in, for example, up to 52-fold speedups for the widely used commercial IBM ILOG CPLEX Optimizer software for solving mixed-integer programming problems.[21] In the third use case—verification problems encoded into propositional satisfiability—the proactive development of alternatives for important components of the program were an important part of the design process, enabling even greater performance gains.

Performance Matters

Computer programs and the algorithms on which they are based frequently involve different ways of getting something done. Sometimes, certain choices are clearly preferable, but it is often unclear a priori which of several design decisions will ultimately give the best results. Such design choices can, and, routinely, do, occur at many levels, from high-level architectural aspects of a software system to low-level implementation details. They are often made based on considerations of maintainability, extensibility, and performance of the system or program under development. This article focuses on this latter aspect of a system's performance, considering only sets of semantically equivalent design choices and situations in which the performance of a program depends on the decisions made for each part of the program for which one or more candidate designs are available, even though these choices do not affect the program's correctness and functionality. Note this premise differs fundamentally from that of program synthesis, in which the primary goal is to come up with a design that satisfies a given functional specification.

It may appear that (partly due to the sustained, exponential improvement in computer hardware over more than five decades) software performance is a relatively minor concern. However, upon closer inspection this is far from

» key insights

- Premature commitment to design choices during software development often leads to loss of performance and limited flexibility.
- PbO aims to avoid premature design choices and actively develop design alternatives, leading to large and rich design spaces of programs that can be specified through simple generic extensions of existing programming languages.
- Advanced optimization and machine-learning techniques make it possible to perform automated performance optimization over the large spaces of programs arising in PbO-based software development; per-instance algorithm selectors and parallel algorithm portfolios can be obtained from the same sequential source.

MagicCube5D, a fully functional five-dimensional analogue of Rubik's Cube.

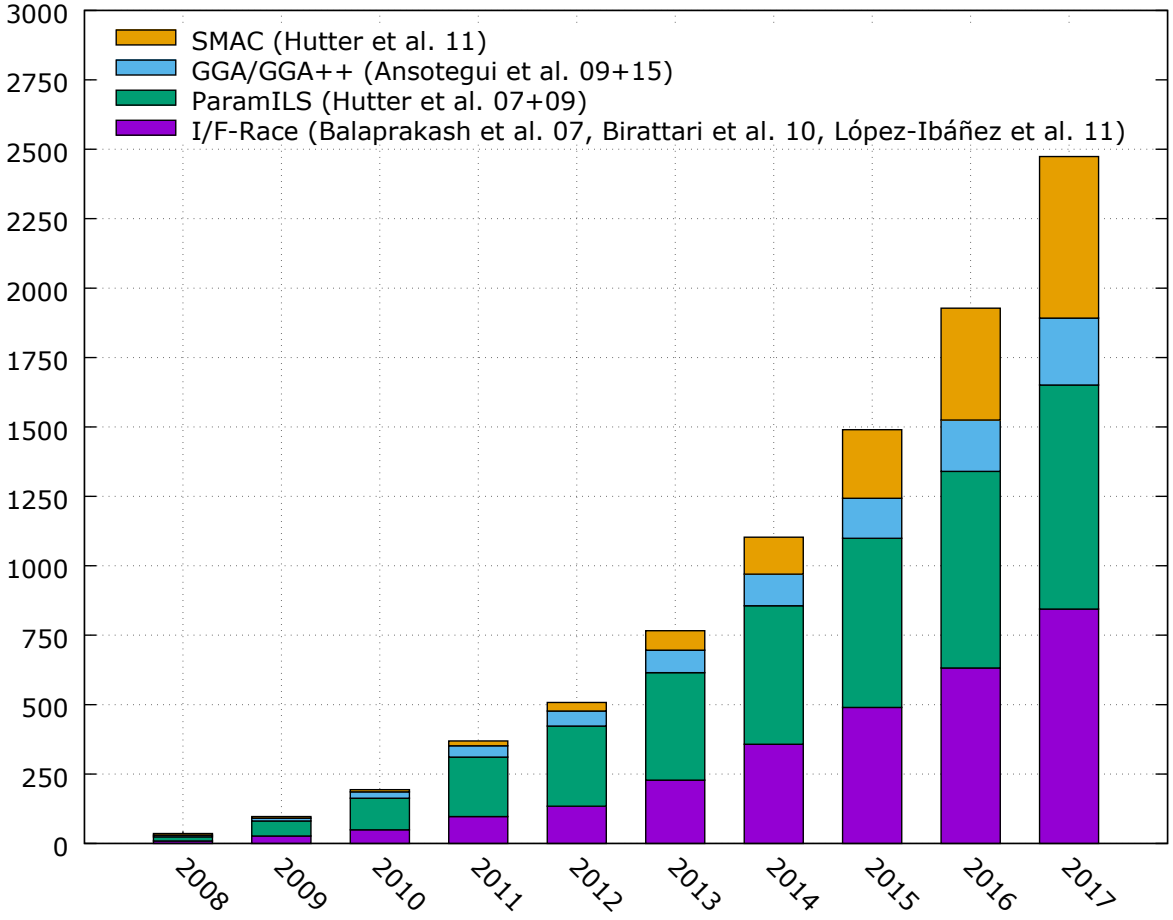
70 COMMUNICATIONS OF THE ACM | FEBRUARY 2012 | VOL. 55 | NO. 2

FEBRUARY 2012 | VOL. 55 | NO. 2 | COMMUNICATIONS OF THE ACM 71

Communications of the ACM, 55(2), pp. 70–80, February 2012

www.prog-by-opt.net

Total citations for key publications on automated algorithm configuration



(Data from Google Scholar; year *vs* total # citations up that year)

3

[This slide was not used during the presentation; it was added to make it easier to follow the slide deck.]

There is no single best solver for any problem that has been subject to even somewhat intense study. Instead, different solvers are best on different kinds of instances.

SAT 2017 Competition

Organizers	Marijn Heule, Matti Järvisalo, Tomáš Balyo
Slides	Slides used at SAT 2017
Proceedings	Descriptions of the solvers and benchmarks
Benchmarks	Available here
Solvers	Available here

	Gold	Silver	Bronze	Gold	Silver	Bronze	Gold	Silver	Bronze
	Agile Track			Main Track			Random Track		
SAT+UNSAT	CaDiCaL Agile, CaDiCaL NoProof	Glu_VC	Glucose 4.1	Maple LCM Dist, Maple LCM, MapleLRB LCMOccRestart, MapleLRB LCM	MapleCOMSPS LRB VSIDS 2, MapleCOMSPS LRB VSIDS	COMiniSATPS Pulsar	YalSAT	tch glucose3	Score2SAT
	Parallel Track			No-Limit Track			Incremental Library Track		
SAT+UNSAT	Syrup24, Syrup48	Plingeling	Painless MapleCOMSPS	COMiniSATPS Pulsar	MapleCOMPSPS LRB VSIDS 2, MapleCOMPSPS LRB VSIDS	CaDiCaL NoProof	AbcdSAT	Glucose	Riss

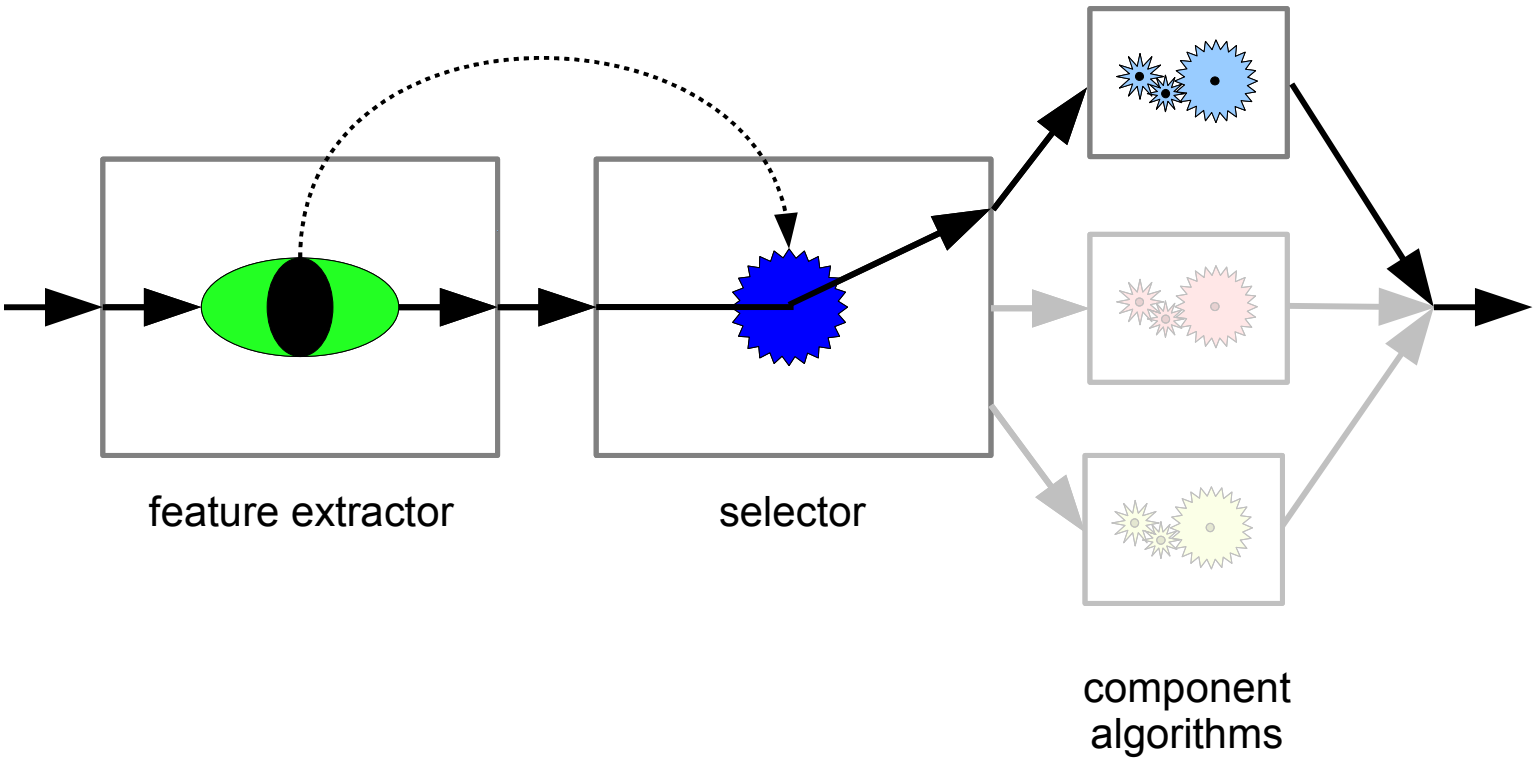
Per-instance algorithm selection (Rice 1976):

- *Given:* set S of algorithms for a problem, problem instance π
- *Objective:* select from S the algorithm expected to solve π *most efficiently*, based on (cheaply computable) *features* of π

Note:

Best case performance bounded by oracle, which selects the best $s \in S$ for each π = *virtual best solver (VBS)*

Per-instance algorithm selection



Key components:

- set of (state-of-the-art) *solvers*
- set of cheaply computable, informative *features*
- efficient procedure for mapping features to solvers (*selector*)
- *training data*
- procedure for building good selector based on training data (*selector builder*)

"The overall champion in 2007 was SATzilla, which was actually not a separate SAT solver but rather a program that knew how to choose intelligently between *other* solvers an any given instance. [...]

This 'portfolio' approach, which tunes itself nicely to the characteristics of vastly different sets of clauses, has continued to dominate the international competitions ever since.

Of course portfolio solvers rely on the existence of 'real' solvers, invented independently and bug-free, which shine with respect to particular classes of problems. And of course the winner of the competition may not be the best actual system for practical use."

Donald Knuth, The Art of Computer Programming, Vol. 4, Fascicle 6, p. 132*f.*

Methods for per-instance selection:

- **classification-based:** predict the best solver, *e.g.*, using . . .
  - decision trees
    (Guerri & Milano 2004)
  - case-based reasoning
    (Gebruers *et al.* 2004)
  - (weighted) k-nearest neighbours
    (Malitsky *et al.* 2011; Kadioglu *et al.* 2011)
  - pairwise cost-sensitive decision forests + voting
    (Xu, Hutter, HH, Leyton-Brown 2012)
- **regression-based:** predict running time for each solver, select the one predicted to be fastest
  (Leyton-Brown *et al.* 2003; Xu, Hutter, HH, Leyton-Brown 2007–9)

Instance features:

- Use generic and problem-specific features that correlate with performance and can be computed (relatively) cheaply:
  - number of clauses, variables, ...
  - constraint graph features
  - local & complete search probes
- Use as features statistics of distributions,
  e.g., variation coefficient of node degree in constraint graph
- Consider combinations of features (*e.g.*, pairwise products $\rightsquigarrow$ quadratic basis function expansion)

SATzilla 2007–2009:

(Xu, Hutter, HH, Leyton-Brown 2008)

- use state-of-the-art complete (DPLL/CDCL) and incomplete (local search) SAT solvers
- extract (up to) 84 polytime-computable instance features
- use ridge regression on selected features to predict solver run-times from instance features
  (one model per solver)
- run solver with best predicted performance

Some bells and whistles:

- use pre-solvers to solve ‘easy’ instances quickly
- build run-time predictors for various types of instances, use classifier to select best predictor based on instance features.
- predict time required for feature computation; if that time is too long (or error occurs), use back-up solver
- use method by Schmee & Hahn (1979) to deal with censored run-time data

$\rightsquigarrow$ prizes in 5 of the 9 main categories of the 2009 SAT Solver Competition (3 gold, 2 silver medals)

SATzilla 2011–2012:

(Xu, Hutter, HH, Leyton-Brown 2012)

- uses cost-based decision forests to directly select solver based on features
- one predictive model for each pair of solvers (which is better?)
- majority voting (over pairwise predictions) to select solver to be run

$\rightsquigarrow$ 1st prizes in 2 of the 3 main tracks, 2nd in the 3rd main track, 1st in the sequential portfolio track of the 2012 SAT Challenge

Successful beyond SAT:

- Max-SAT (see, *e.g.*, Malitsky, Mehta, OSullivan (2013)
- QBF (see, *e.g.*, Pulina, Tacchella (2009)
- CSP (see, *e.g.*, OMahony *et al.* 2008)
- ASP (see, *e.g.*, Gebser *et al.* (2012)
- AI planning (see, *e.g.*, Helmert, Röger, Karpas (2011)
- ...

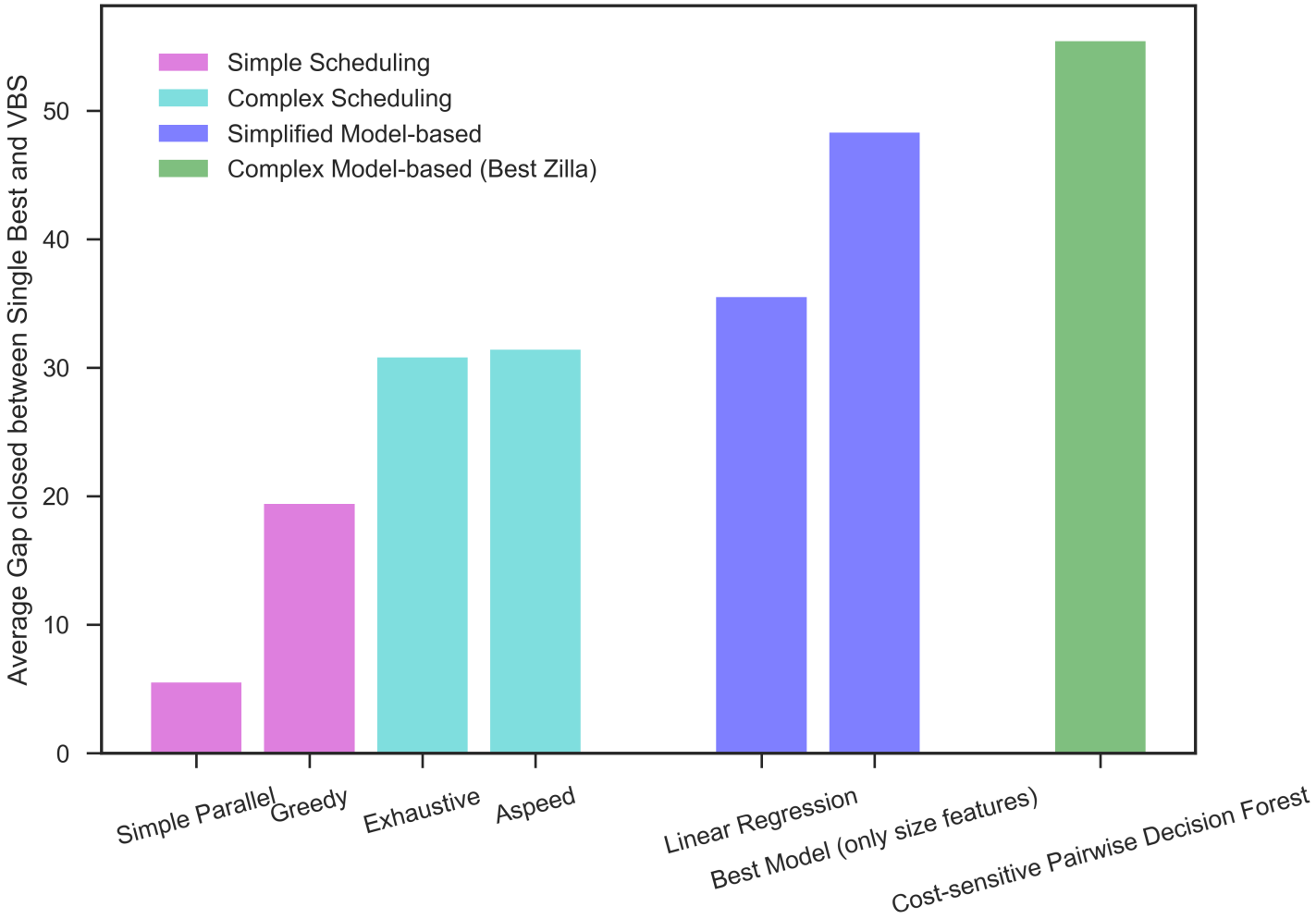
ASlib: Algorithm selection benchmark library (Bischl *et al.* 2016)

ICON Algorithm Selection Challenge (Kotthoff 2015)

OASC: Open Algorithm Selection Challenge (Lindauer *et al.* 2017)

Portfolio-based methods: How much sophistication is needed?

Cameron, Xu, Fréchette, HH, Hutter, Leyton-Brown (in preparation)



Combining configuration and selection:

- performance complementarity between different configurations of given solver
  $\rightsquigarrow$ selection over automatically determined configurations
  (Xu, HH, Leyton-Brown 2010; Kadioglu *et al.* 2010)

- many design choices in selector construction, different selectors perform best in different applications
  $\rightsquigarrow$ AutoFolio = automatic configuration of algorithm selectors
  (Lindauer, HH, Hutter, Schaub 2015)

Recap

1. How long? (performance prediction)
2. Which parameter settings? (algorithm configuration)
3. Which solver? (algorithm selection)
4. The road ahead

The state of the art in solving X ...

- ... is not defined by a single solver / solver configuration
- ... requires use of / interplay between multiple heuristic mechanisms
- ... has been substantially advanced by ML

Competitions ...

- ... have helped advance the state of the art in many fields (SAT, AI planning, machine learning, ...)
- ... suffer from noise, low statistical significance
  Fawcett, Vallati, HH, Gerevini (in preparation)
- ... are mostly focussed on single solvers, broad-spectrum performance
- ... often don't reflect state of the art
- ... typically don't provide effective incentive to improve state of the art

A new kind of competition:

- solvers submitted to competition platform
- state-of-the-art selector built based on all solvers
- solver contributions to overall performance assessed based on (relative) marginal contribution
  (Xu, Hutter, HH, Leyton-Brown 2012)
- full credit for contributions to selector performance goes to component solver authors

$\rightsquigarrow$ Sparkle (Luo & HH – underway)

[This slide was not used during the presentation; it was added to make it easier to follow the slide deck.]

In the Olympics, all the glory goes to the
gold medal winners;
the same holds for (traditional) solver competitions.
Instead, we cut the gold medal into pieces
of size proportional to the contribution to overal
(team) performance.

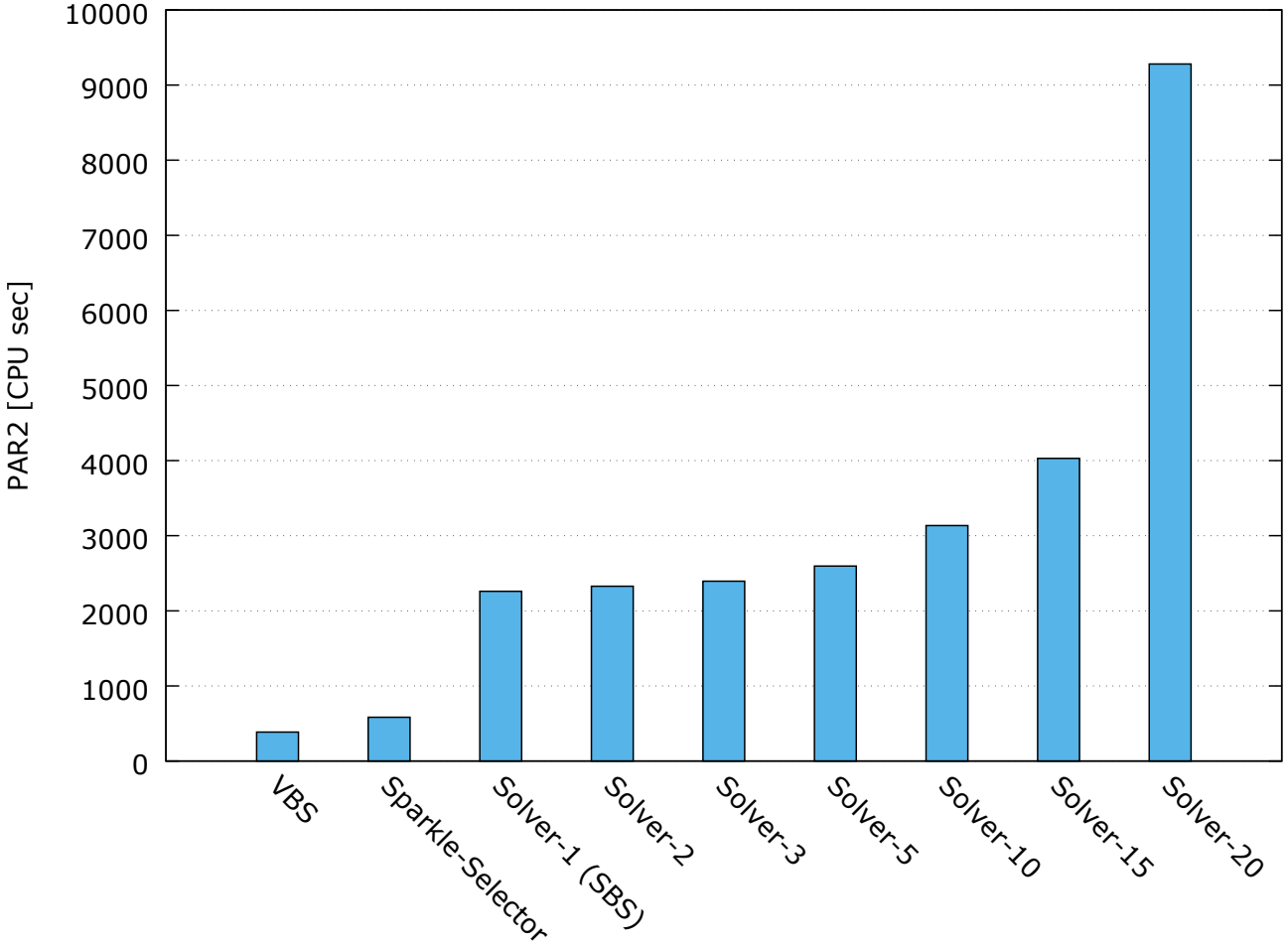
1
2
3

XXXI Olimpíada Rio 2016

Sparkle SAT Challenge 2018

- Part of FLoC Olympic Games, coordinated with 2018 SAT Competition
- launched March 2018, leader board phase 5–15 April, final results @ FLoC 2018
- 20 solvers submitted

Final leader board results (on training data)

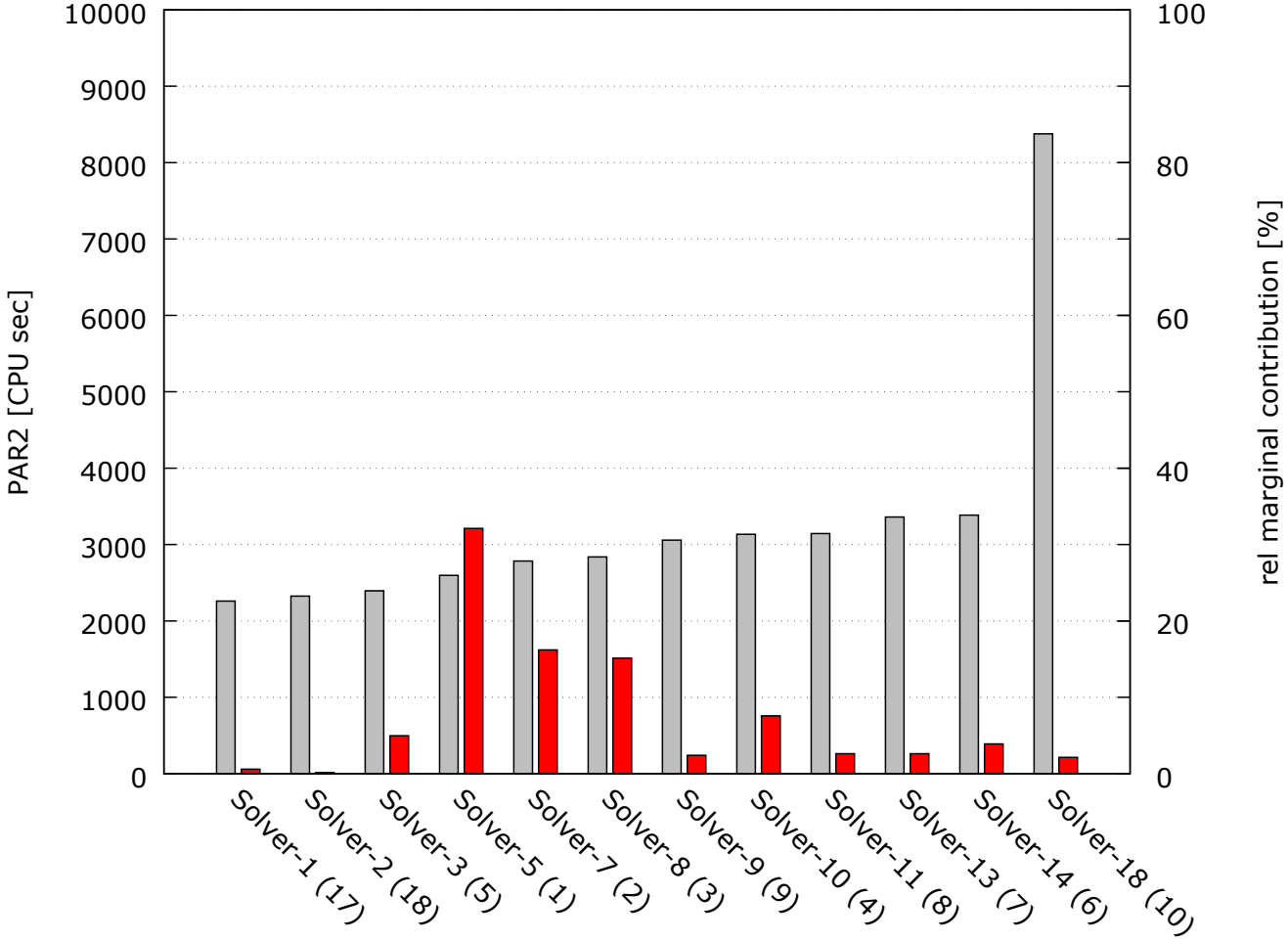


[This slide was not used during the presentation; it was added to make it easier to follow the slide deck.]

Notice that there is a large performance gap between VBS and the single best solver (SBS).

The selector constructed by Sparkle is very close in performance to the VBS and much better than the single best solver (SBS) - thanks to AutoFolio, which constructs the selector.

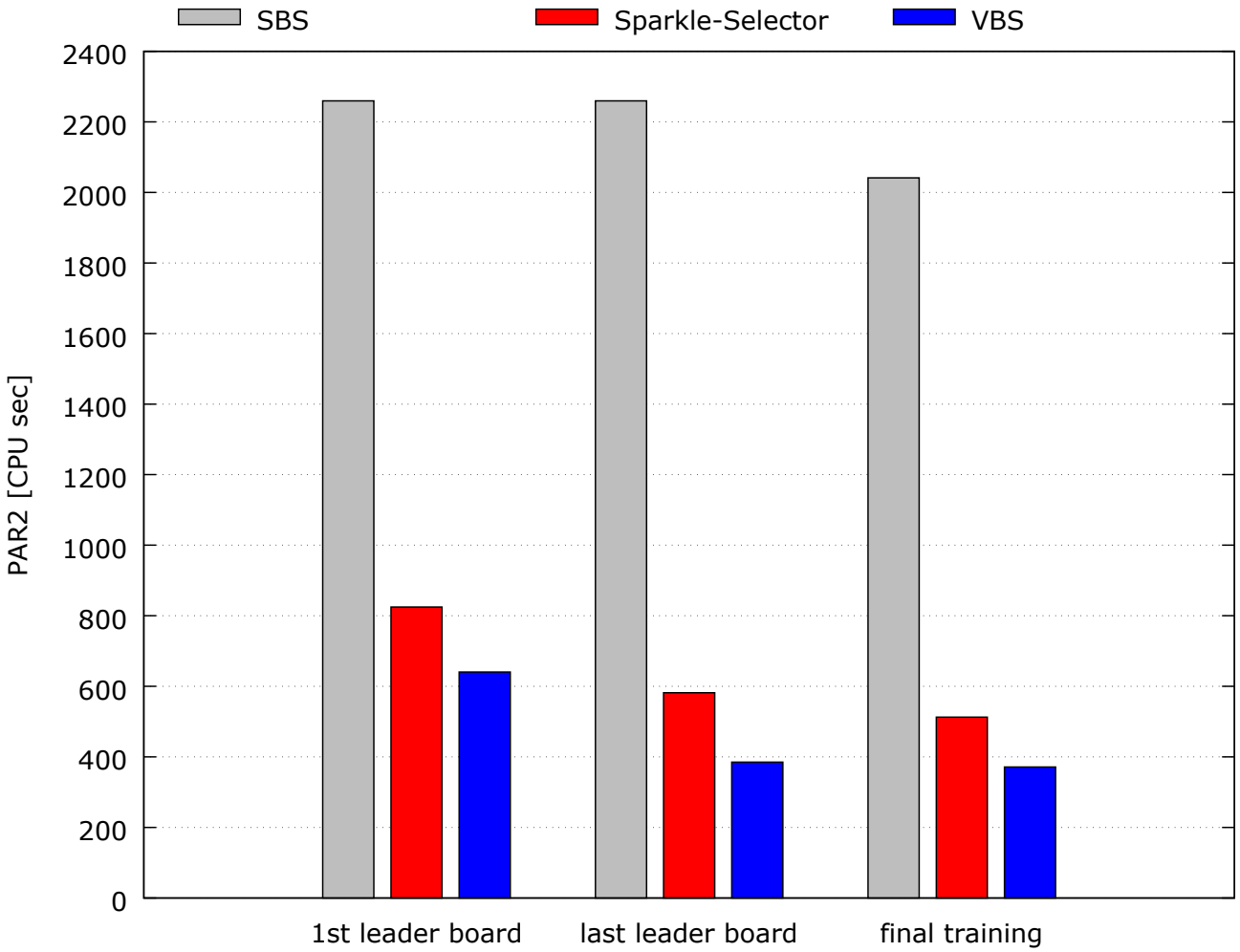
Final leader board results (on training data)



[This slide was not used during the presentation; it was added to make it easier to follow the slide deck.]

The best stand-alone solvers contribute very little to the Sparkle selector.

Improvement over time (on training data)



[This slide was not used during the presentation; it was added to make it easier to follow the slide deck.]

During the leader board phase,
there was no improvement in SBS,
but the VBS improved by a factor of 1.5;
most of this improvement was also
seen in the Sparkle selector.

Just before the deadline, a few solvers
were submitted, which lead to an
improvement (factor ~1.1) in SBS,
but didn't make the VBS much better.
Yet, the Sparkle selector improved by
a factor ~1.1, just like the SBS.

[This slide was not used during the presentation; it was added to make it easier to follow the slide deck.]

This shows that Sparkle was effective in incentivising solver authors to improve the state of the art in SAT solving as represented by the Sparkle selector rather than just the SBS.

Furthermore, submitting a solver at the last minute - so that no one else can exploit knowledge on its (aggregate) performance, as revealed during the leader board phase of the Sparkle SAT Challenge - might make sense form the SBS perspective, but seems unlikely to help in doing well in our challenge.

But this is all on training data ... stay tuned for official results on test data (@ FLoC 2018).

Advantages over traditional competition:

- makes it easier to gain recognition for specialised techniques
- better reflects and makes accessible state of the art
- provides incentive to improve true state of the art

Further use of Sparkle:

- continuous solver evaluation (as community service)
- specialised application contexts (reduced solver sets)
- experimentation platform for algorithm selection, configuration, programming by optimisation (PbO)

Take-home message:

- Reasoning + learning:
  crucial for AI, advanced problem solving
- Machine learning $=\subset$ automated performance optimisation
  $\subset$ automated algorithm design
- Meta-algorithmic techniques (configurators, selectors, ...):
  powerful, useful, readily available;
  key to better SAT, SMT, CP, ASP, ..., ML, AI
- Next: Make those easily accessible + broadly usable
  $\rightsquigarrow$ Sparkle (community effort)

Bigger picture:

- Part of broader effort: Automation of AI (AutoAI)
  (prominent special case: AutoML)
  $\rightsquigarrow$ advancement + democratisation of AI
- Sparkle, PbO use key AI techniques + (lots of) computation to leverage human ingenuity + intuition
- Human-centred AI:
  AI that augments, not replaces, human intelligence