Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms

Chris Thornton, Frank Hutter,
Holger H. Hoos, Kevin Leyton-Brown

Department of Computer Science
University of British Columbia
Canada

COSEAL Workshop Münster, Germany
2013/07/29

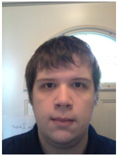
Chris Thornton
UBC

Frank Hutter
UBC

Kevin Leyton-Brown
UBC


Chris Fawcett
UBC


Marius Schneider
U. Potsdam


James Styles
UBC


Thomas Stützle
U. Libre de Bruxelles


Alan Hu
UBC


Domagoj Babić
UBC


Marco Chiarandini
U. Southern Denmark


Torsten Schaub
U. Potsdam


Benjamin Kaufmann
U. Potsdam


Martin Müller
U. of Alberta

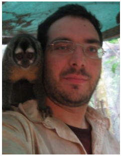

Thomas Barz-Beielstein
FH Köln


Alfonso Gerevini
U. di Brescia

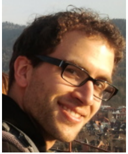

Alessandro Saetti
U. di Brescia


Mauro Vallati
U. di Brescia


Matle Helmert
U. Freiburg


Erez Karpas
Technion


Gabriele Röger
U. Freiburg


Jendrik Seipp
U. Freiburg

Fundamental problem:

Which of many available algorithms (models) applicable to given machine learning problem to use, and with which hyper-parameter settings?

Example: WEKA contains 39 classification algorithms,
3×8 feature selection methods

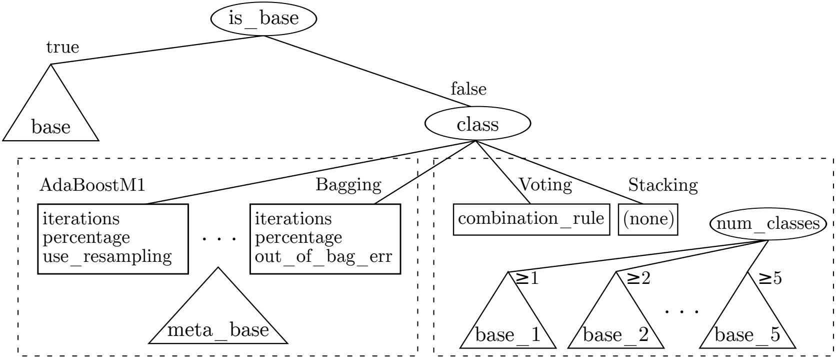
Key idea:

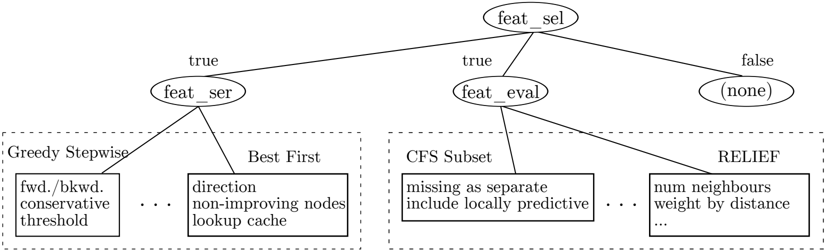
simultaneously solve algorithm selection
$+$ hyperparameter optimisation problem

1

Auto-WEKA approach:

- model space of *all* combinations of classification algorithms, feature selection methods as *single* parametric algorithm
- select between the $39 \times 3 \times 8$ algorithms using high-level categorical choices
- consider hyper-parameters for each algorithm


is_base
true
false
base
class
AdaBoostM1
Bagging
Voting
Stacking
iterations
percentage
use_resampling
. . .
iterations
percentage
out_of_bag_err
combination_rule
(none)
num_classes
meta_base
≥1
≥2
≥5
. . .
base_1
base_2
base_5


feat_sel
true
true
false
feat_ser
feat_eval
(none)
Greedy Stepwise
Best First
CFS Subset
RELIEF
fwd./bkwd.
conservative
threshold
. . .
direction
non-improving nodes
lookup cache
missing as separate
include locally predictive
. . .
num neighbours
weight by distance
...

Auto-WEKA approach:

- model space of *all* combinations of classification algorithms, feature selection methods as *single* parametric algorithm
- select between the $39 \times 3 \times 8$ algorithms using high-level categorical choices
- consider hyper-parameters for each algorithm
- solve resulting algorithm configuration problem using general-purpose configurator

Automated configuration process:

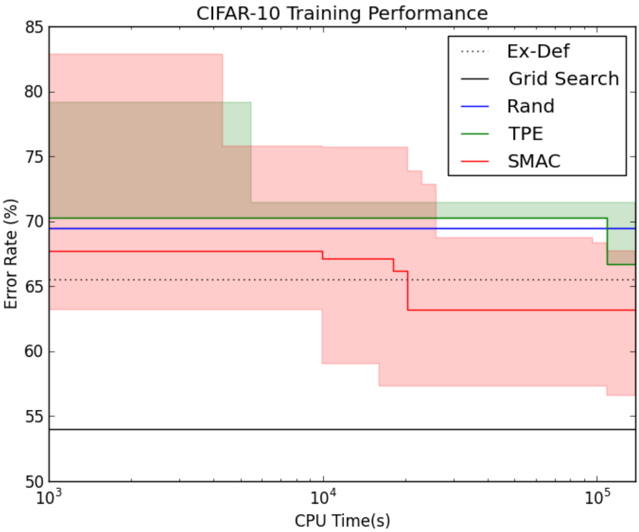
- configurator: SMAC (Hutter, HH, Leyton-Brown 2011–13)
- performance objective: cross-validated mean error rate
- time budget: 4×30 CPU hours

Selected results (mean error rate)

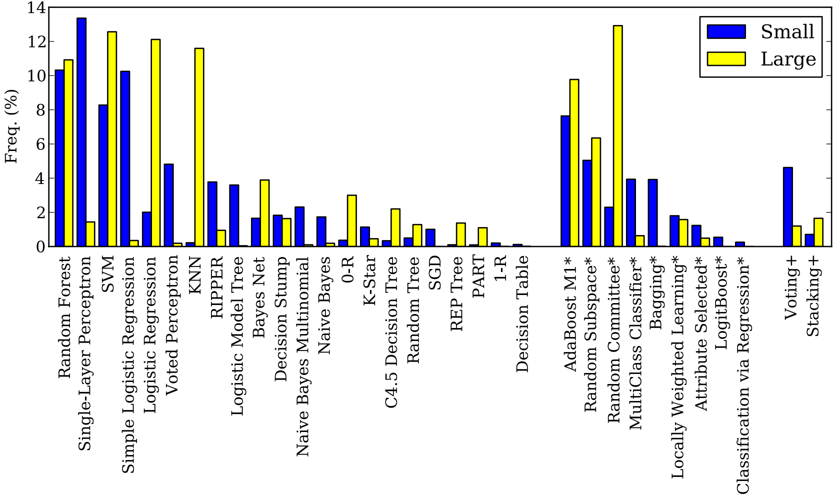
					Auto-WEKA	
Dataset	#Instances	#Features	#Classes	Best Def.	TPE	SMAC
Semeion	1115+478	256	10	8.18	8.26	**5.08**
KR-vs-KP	2237+959	37	2	0.31	0.54	**0.31**
Waveform	3500+1500	40	3	14.40	**14.23**	14.42
Gisette	4900+2100	5000	2	2.81	3.94	**2.24**
MNIST Basic	12k+50k	784	10	5.19	12.28	**3.64**
CIFAR-10	50k+10k	3072	10	64.27	66.01	**61.15**

Auto-WEKA better than full grid search in 15/21 cases

Further details: KDD-13 paper (to appear)


CIFAR-10 Training Performance
Error Rate (%)
CPU Time(s)
85
80
75
70
65
60
55
50
10^3
10^4
10^5
Ex-Def
Grid Search
Rand
TPE
SMAC

Which classifiers were chosen by Auto-WEKA?



Some results for regression problems (RMSE)

Dataset	#Instances	#Features	Best Def.	Auto-WEKA TPE	Auto-WEKA SMAC
Forest Fires	362 + 155	12	**63.55**	63.73	64.36
Crime	1396 + 598	126	0.1404	**0.1356**	0.1376
Abalone	2924 + 1253	8	2.130	**2.072**	2.101
Parkinsons – Motor	4113 + 1762	20	0.6323	0.5627	**0.4047**
Parkinsons – Total	4113 + 1762	20	0.7999	0.3837	**0.1606**
COIL	5822 + 4000	85	0.2328	0.2471	**0.2317**

Auto-WEKA ...

- beats oracle (optimal) choice from large set of ML algorithms with default hyper-parameter settings
- beats full grid search over all algorithms, hyper-parameters; also beats random search (Bergstra & Bengio 12)
- effectively solves combined algorithm selection
  + hyper-parameter optimisation problem
  on standard 4-core machine in less than 1.5 days

Note:

- general-purpose algorithm configurator (SMAC) outperforms best method from ML literature (TPE, Bergstra *et al.* 2011)

2

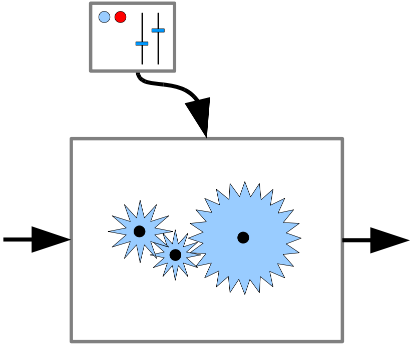
Algorithm configuration

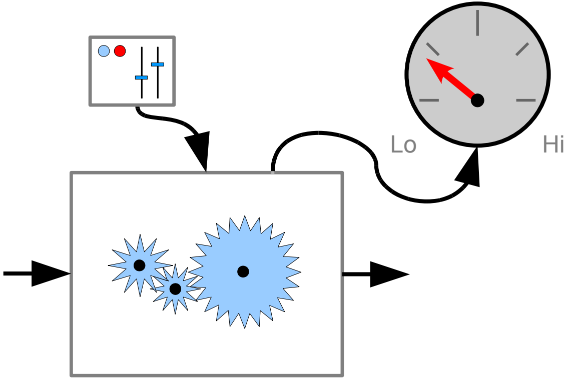
Observation: Many algorithms have parameters (sometimes hidden / hardwired) whose settings affect performance

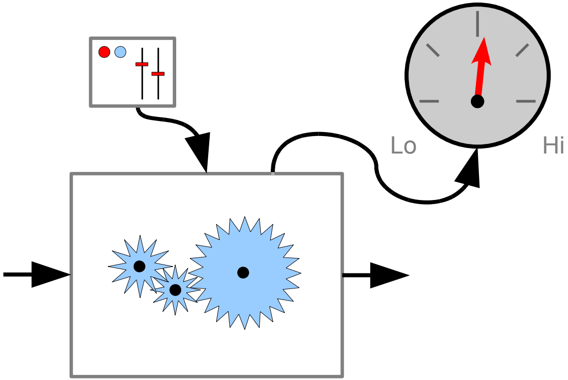
Challenge: Find parameter settings that achieve good / optimal performance on given type of input data

Example: IBM ILOG CPLEX

- widely used industrial optimisation software
- exact solver, based on sophisticated branch & cut algorithm and numerous heuristics
- 159 parameters, 81 directly control search process
- find parameter settings that solve MIP-encoded wildlife corridor construction problems as fast as possible




Lo
Hi


Lo
Hi

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 is challenging:

- size of configuration space
- parameter interactions
- discrete / categorical parameters
- conditional parameters
- performance varies across inputs (problem instances)
- evaluating poor configurations can be very costly
- censored algorithm runs

$\rightsquigarrow$ standard optimisation methods are insufficient

Algorithm configuration approaches

- Sampling methods
  (*e.g.*, REVAC, REVAC++ – Nannen & Eiben 06–09)

- Racing
  (*e.g.*, F-Race – Birattari, Stützle, Paquete, Varrentrapp 02;
  Iterative F-Race – Balaprakash, Birattari, Stützle 07)

- Model-free search
  (*e.g.*, ParamILS – Hutter, HH, Stützle 07;
  Hutter, HH, Leyton-Brown, Stützle 09;
  GGA – Ansótegui, Sellmann, Tierney09)

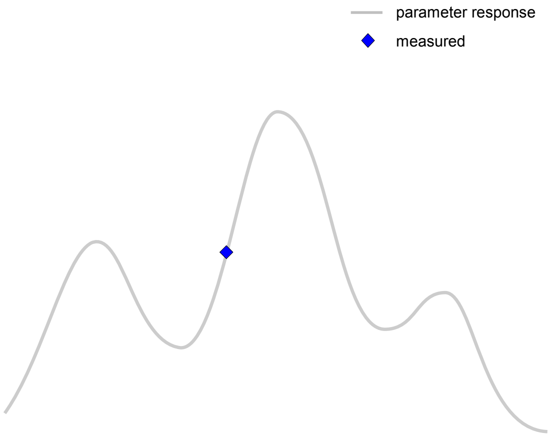
- Sequential model-based (aka Bayesian) optimisation
  (*e.g.*, SPO – Bartz-Beielstein 06; SMAC – Hutter, HH, Leyton-Brown 11–12)

Sequential model-based optimisation

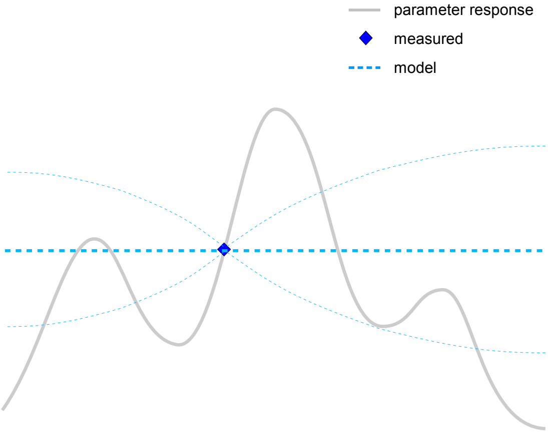
e.g., Jones (1998), Bartz-Beielstein (2006)

- **Key idea:**
  use predictive performance model (response surface model) to find good configurations
- perform runs for selected configurations (initial design) and fit model (*e.g.*, noise-free Gaussian process model)
- iteratively select promising configuration, perform run and update model

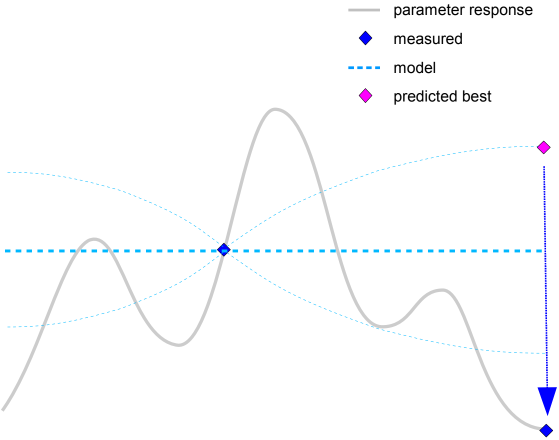
Sequential Model-based Optimisation



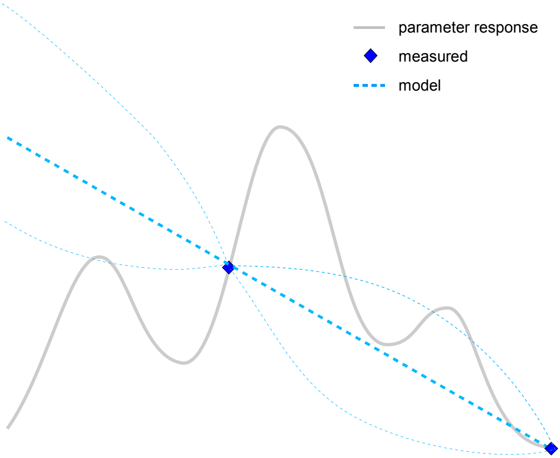
Sequential Model-based Optimisation



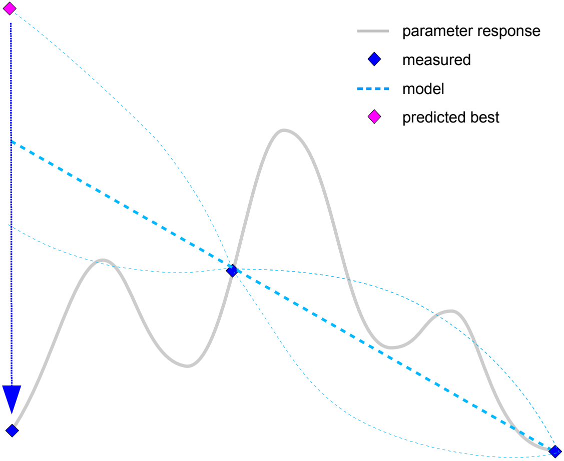
Sequential Model-based Optimisation



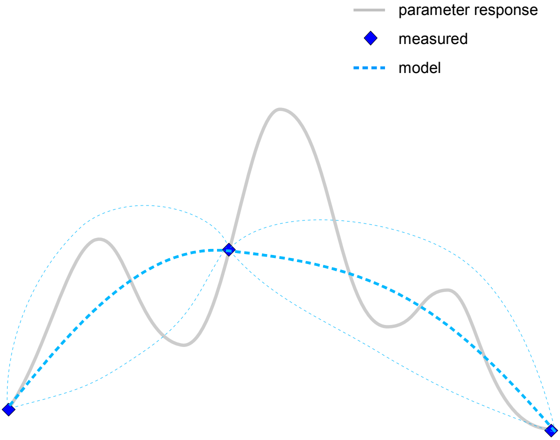
Sequential Model-based Optimisation



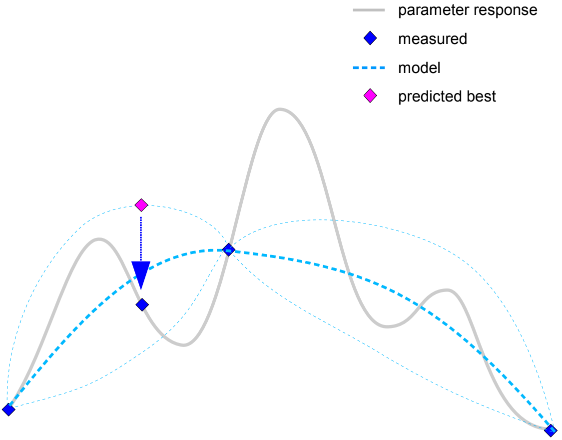
Sequential Model-based Optimisation



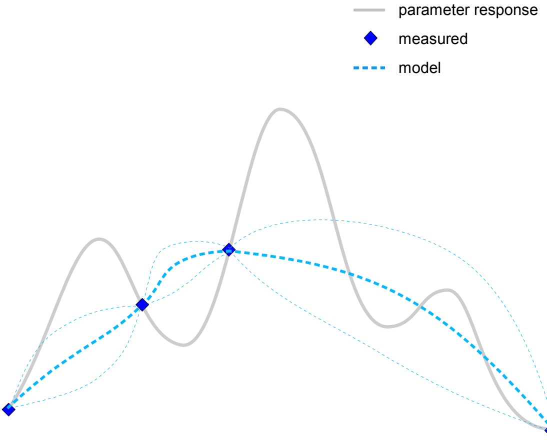
Sequential Model-based Optimisation



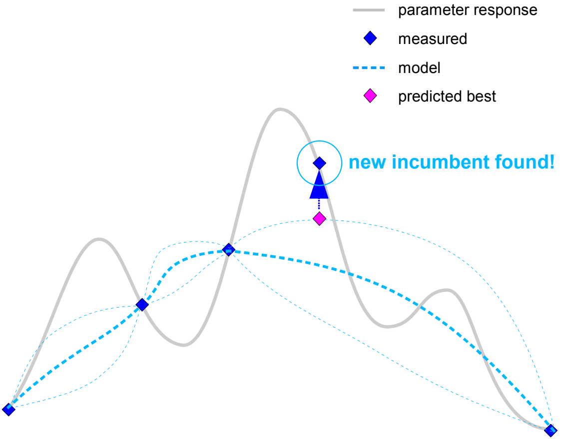
Sequential Model-based Optimisation



Sequential Model-based Optimisation



Sequential Model-based Optimisation

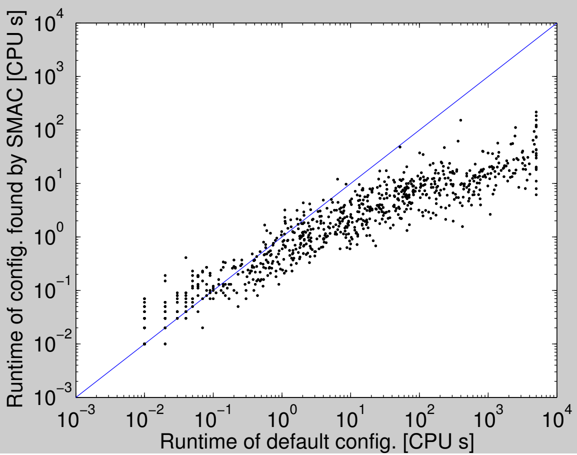


Sequential Model-based Algorithm Configuration (SMAC)

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
- impose time-limit for algorithm based on performance observed so far (adaptive capping)
- initialisation with single configuration (algorithm default or randomly chosen)

CPLEX 11 on Wildlife Corridor Design



$\rightsquigarrow$ 191 $\times$ speedup on average!

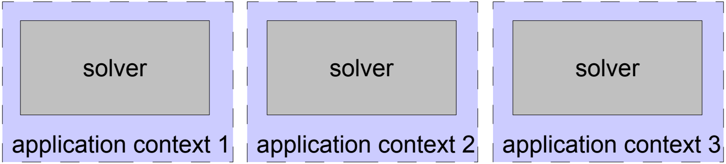
3

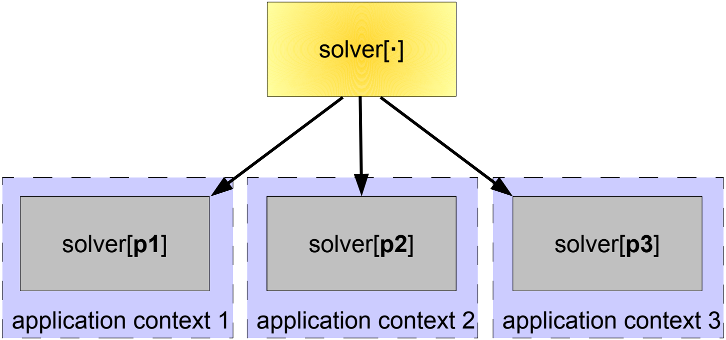
Programming by Optimisation (PbO)

HH (2010–12)

Key idea:

- program $\rightsquigarrow$ (large) space of programs
- encourage software developers to
  - avoid premature commitment to design choices
  - seek & maintain design alternatives
- automatically find performance-optimising designs for given use context(s)


solver
solver
solver
application context 1
application context 2
application context 3


solver[·]
solver[**p1**]
solver[**p2**]
solver[**p3**]
application context 1
application context 2
application context 3

Levels of PbO:

Level 4: Make no design choice prematurely that cannot be justified compellingly.

Level 3: Strive to provide design choices and alternatives.

Level 2: Keep and expose design choices considered during software development.

Level 1: Expose design choices hardwired into existing code (magic constants, hidden parameters, abandoned design alternatives).

Level 0: Optimise settings of parameters exposed by existing software.

Success in optimising speed:

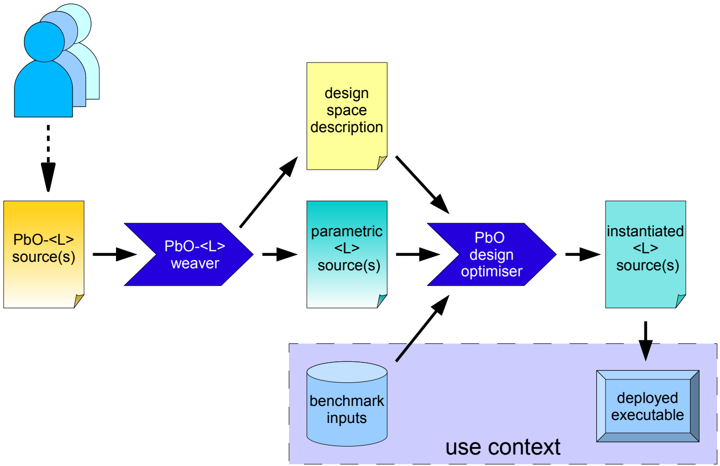
Application, Design choices	Speedup	PbO level
SAT-based software verification (SPEAR), 41 Hutter, Babić, HH, Hu (2007)	4.5–500 ×	2–3
AI Planning (LPG), 62 Vallati, Fawcett, Gerevini, HH, Saetti (2011)	3–118 ×	1
Mixed integer programming (CPLEX), 76 Hutter, HH, Leyton-Brown (2010)	2–52 ×	0

... and solution quality:

University timetabling, 18 design choices, PbO level 2–3
$\rightsquigarrow$ new state of the art; UBC exam scheduling
Fawcett, Chiarandini, HH (2009)

Machine learning / Classification, 786 design choices, PbO level 0–1
$\rightsquigarrow$ outperforms specialised model selection & hyper-parameter optimisation methods from machine learning
Thornton, Hutter, HH, Leyton-Brown (2012–13)

Software development in the PbO paradigm



PbO enables ...

- performance optimisation for different use contexts
  (as shown for many problems)

- adaptation to changing use contexts
  (see, *e.g.*, life-long learning – Thrun 1996)

- self-adaptation while solving given problem instance
  (*e.g.*, Battiti *et al.* 2008; Carchrae & Beck 2005; Da Costa *et al.* 2008; Wessing *et al.* 2011)

- automated generation of instance-based solver selectors
  (*e.g.*, SATzilla – Leyton-Brown *et al.* 2003, Xu *et al.* 2008; Hydra – Xu *et al.* 2010; ISAC – Kadioglu *et al.* 2010)

- automated generation of parallel solver portfolios
  (*e.g.*, Huberman *et al.* 1997; Gomes & Selman 2001; Schneider *et al.* 2012)

contributed articles

DOI:10.1145/2076450.2076469

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.

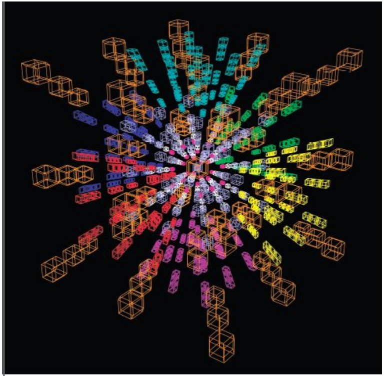
The potential of PbO is evident from recent empirical results (see the table here). In the first two use cases—mixed integer programming and planning—existing software exposing many design choices in the form of parameters was automatically optimized for speed. This resulted in, for example, up to 52-fold 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 algo-

» 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.

rithms 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

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

www.prog-by-opt.net

Take-home message:

- state-of-the-art algorithm configuration procedures enable effective selection and hyper-parameter optimisation of machine learning algorithms
  $\rightsquigarrow$ Auto-WEKA

- … as well as an algorithm design approach that avoids premature commitment to design choices and leverages human creativity
  $\rightsquigarrow$ PbO

- … this is just the beginning, lots of work further work to be done (methodology, tools, applications)

Auto-WEKA paper, code: www.cs.ubc.ca/labs/beta/Projects/autoweka