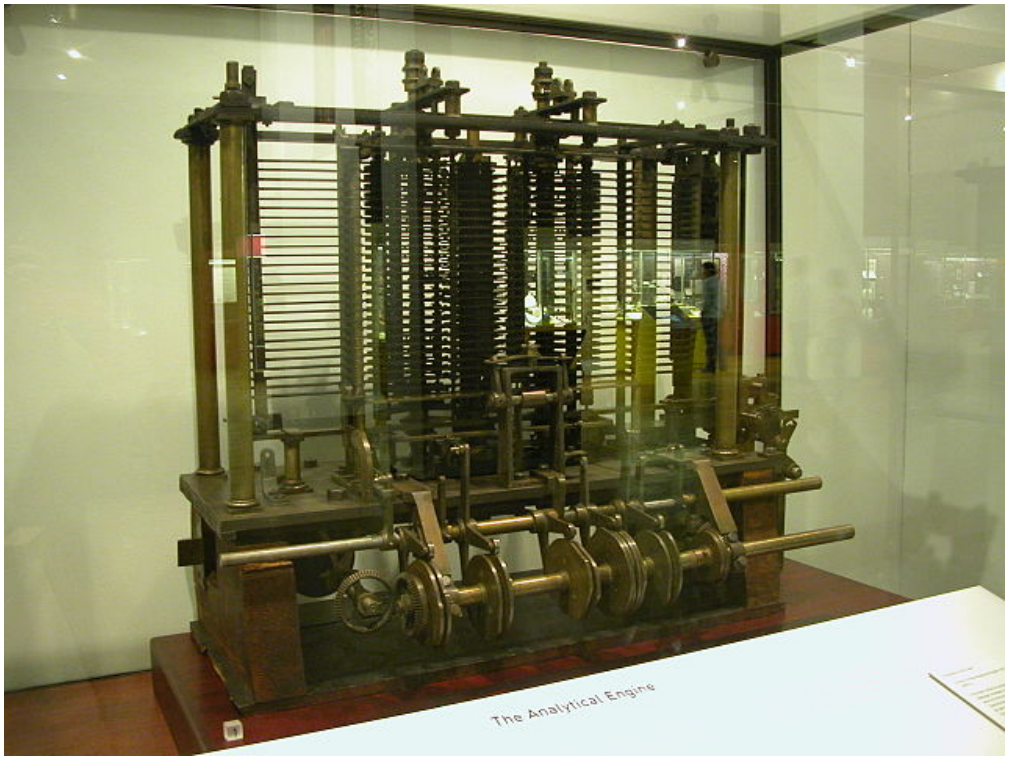
From Programs to Program Spaces: Programming by Optimisation

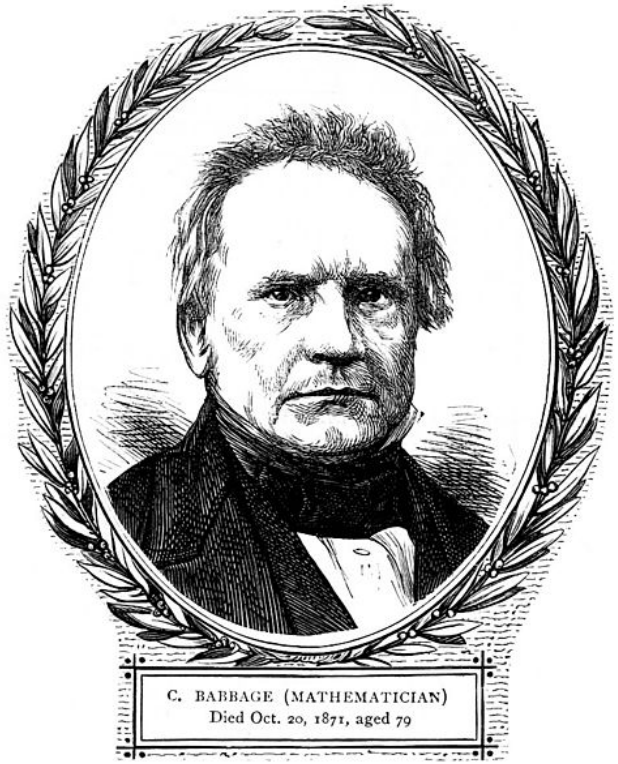
Holger H. Hoos

BETA Lab
Department of Computer Science
University of British Columbia
Canada

45th CREST Open Workshop on Genetic Improvement
London, UK, 2016/01/26

The age of machines





"As soon as an Analytical Engine exists, it will necessarily guide the future course of the science. Whenever any result is sought by its aid, the question will then arise – by what course of calculation can these results be arrived at by the machine in the shortest time?"

(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."

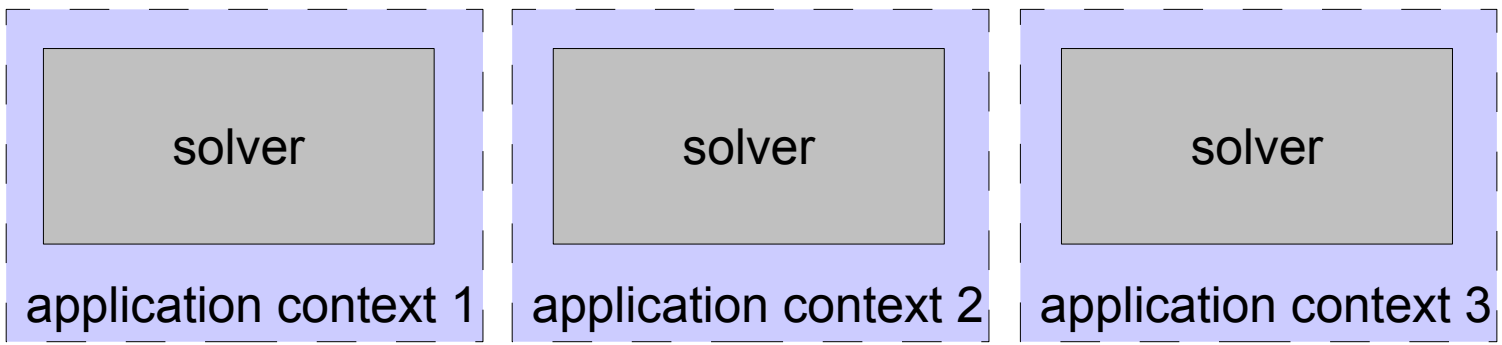
- financial markets
- social interactions
- cultural preferences
- artistic production
- ...

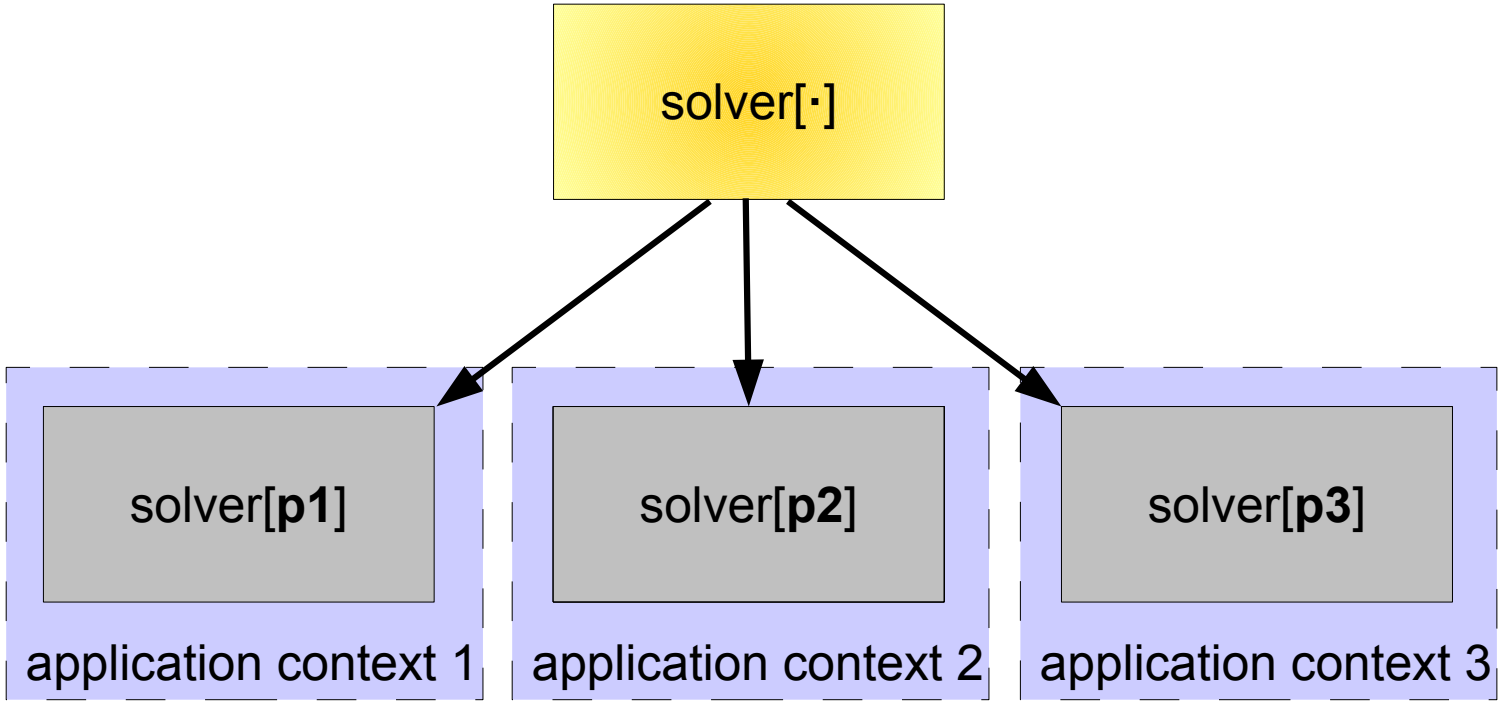
Performance matters ...

- computation speed (time is money!)
- energy consumption (battery life, ...)
- quality of results (cost, profit, weight, ...)

... increasingly:

- globalised markets
- just-in-time production & services
- tighter resource constraints


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


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

1

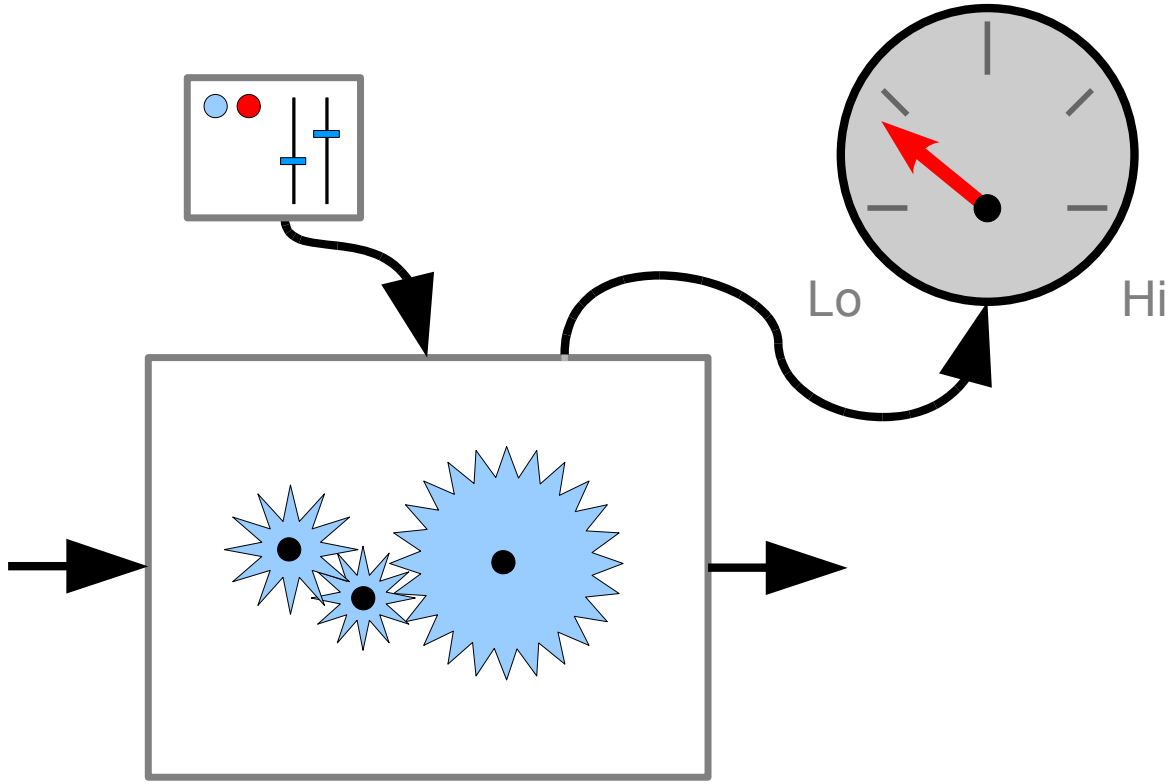
Algorithm configuration

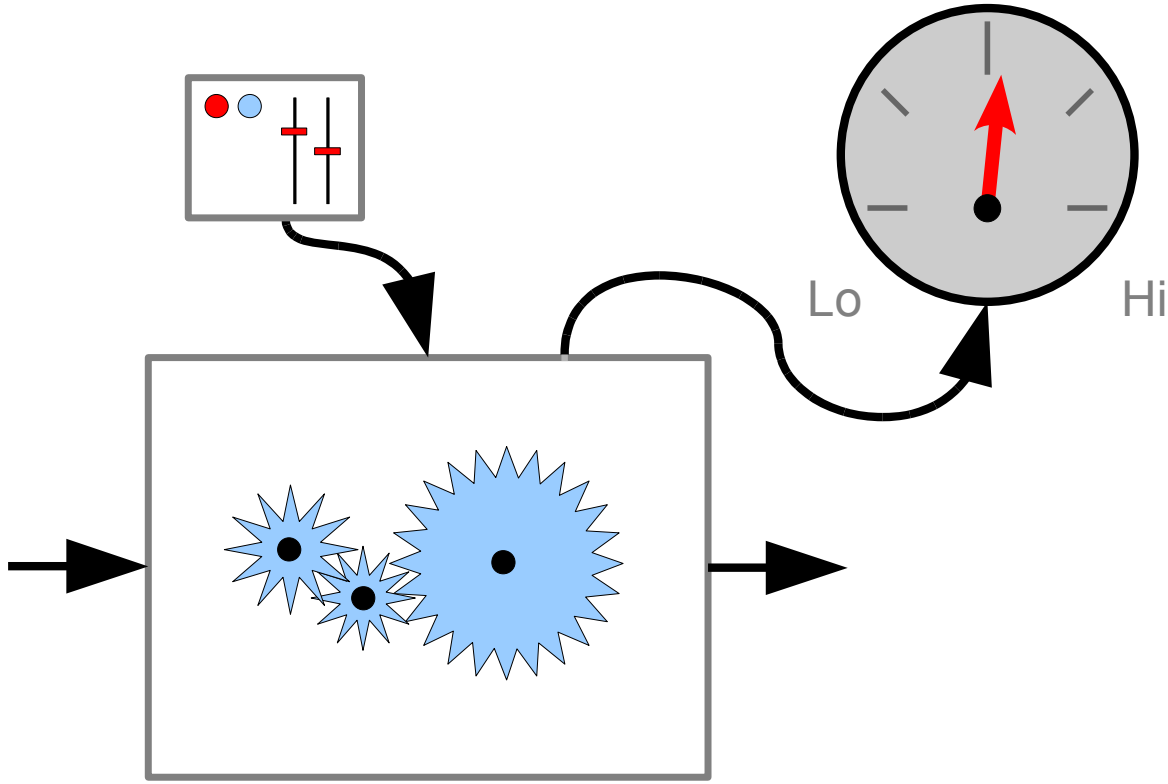
Observation: Many algorithms have parameters (sometimes hidden / hardwired) whose settings affect performance

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

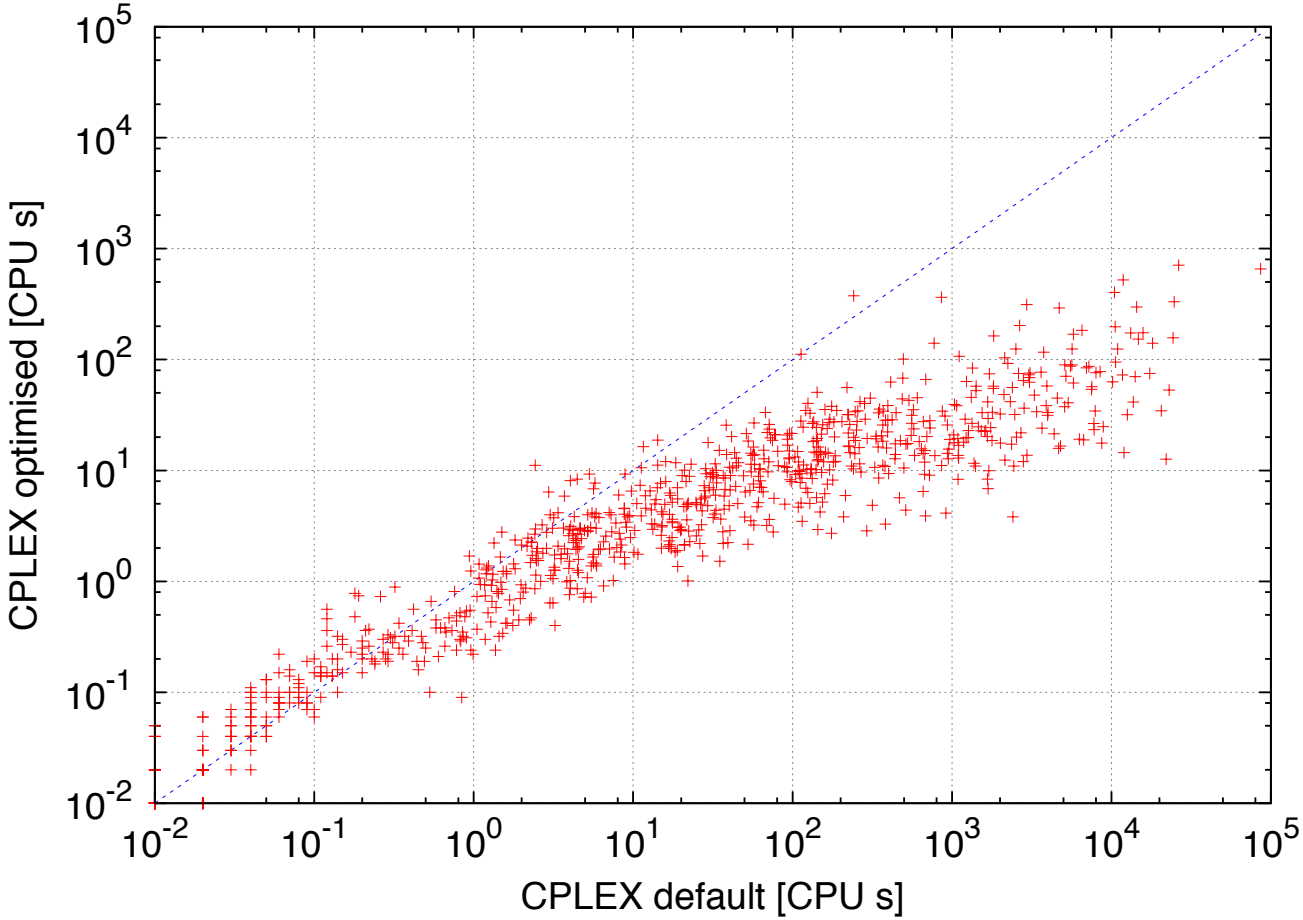
Example: IBM ILOG CPLEX

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


Lo
Hi


Lo
Hi

CPLEX on Wildlife Corridor Design



$\rightsquigarrow$ 52.3 $\times$ speedup on average!

The algorithm configuration problem (AC)

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

What if ... we could solve AC effectively?

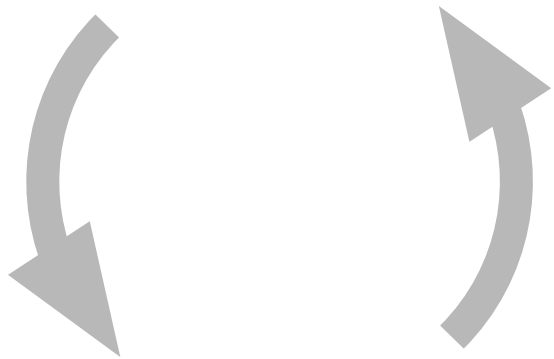
1. Less fiddling / hand-tuning.
2. Better performing algorithms.
3. More parameters, fewer ad-hoc choices.
4. More expert time spent on ideas, high-level design choices.
5. More broadly applicable software.
6. Automatically customised software.
7. Automatic parallelisation.
8. Partial automation of programming.
9. Fairer evaluation of algorithms and ideas.
10. New insights into efficacy of heuristic mechanisms, empirical complexity of problems.

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

Machine Learning



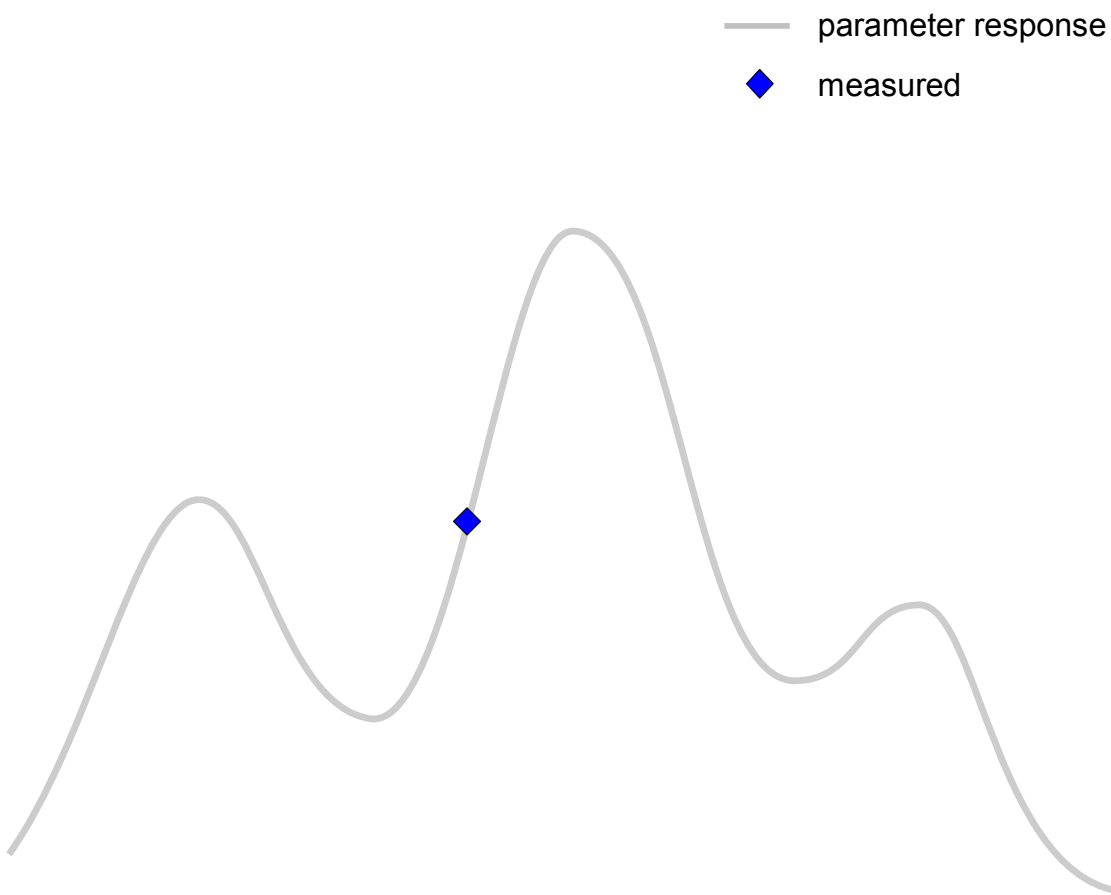
Optimisation

Sequential Model-based Optimisation

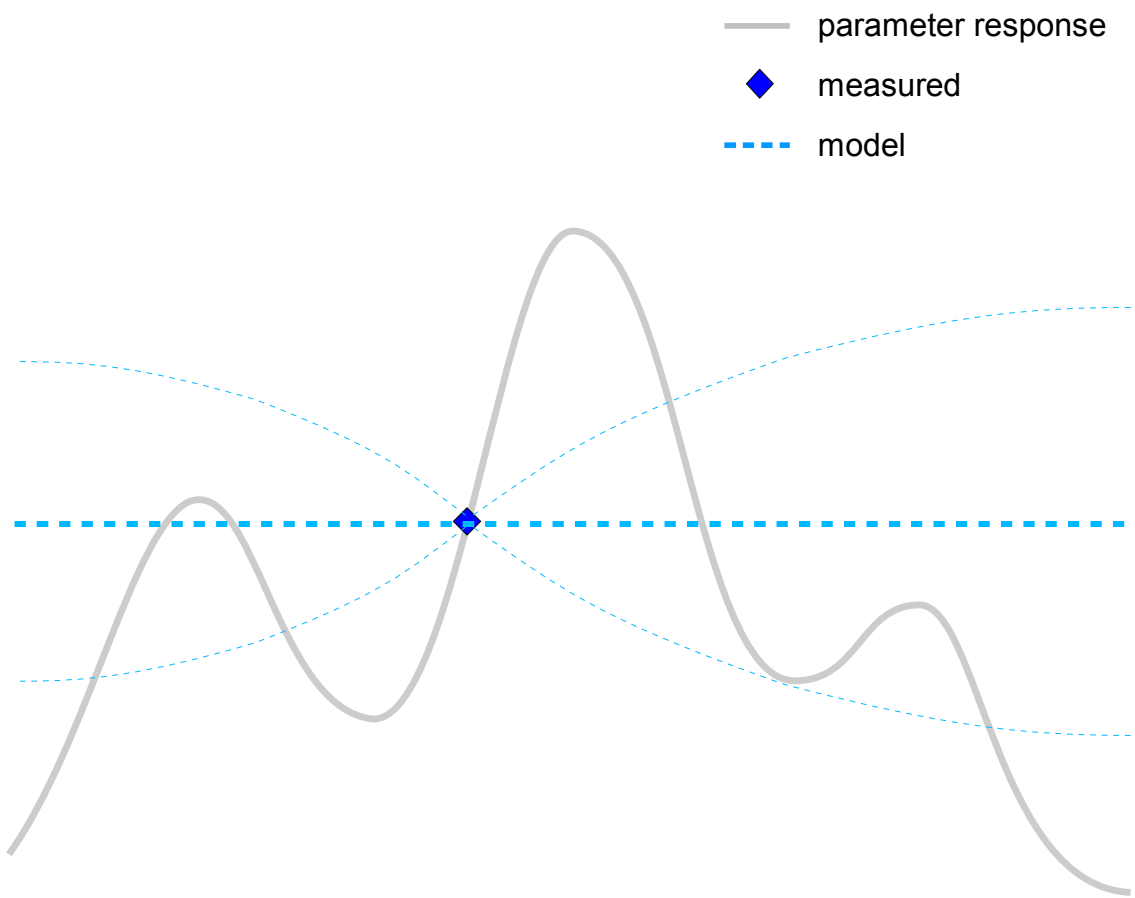
e.g., Jones (1998), Bartz-Beielstein (2006)

- **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

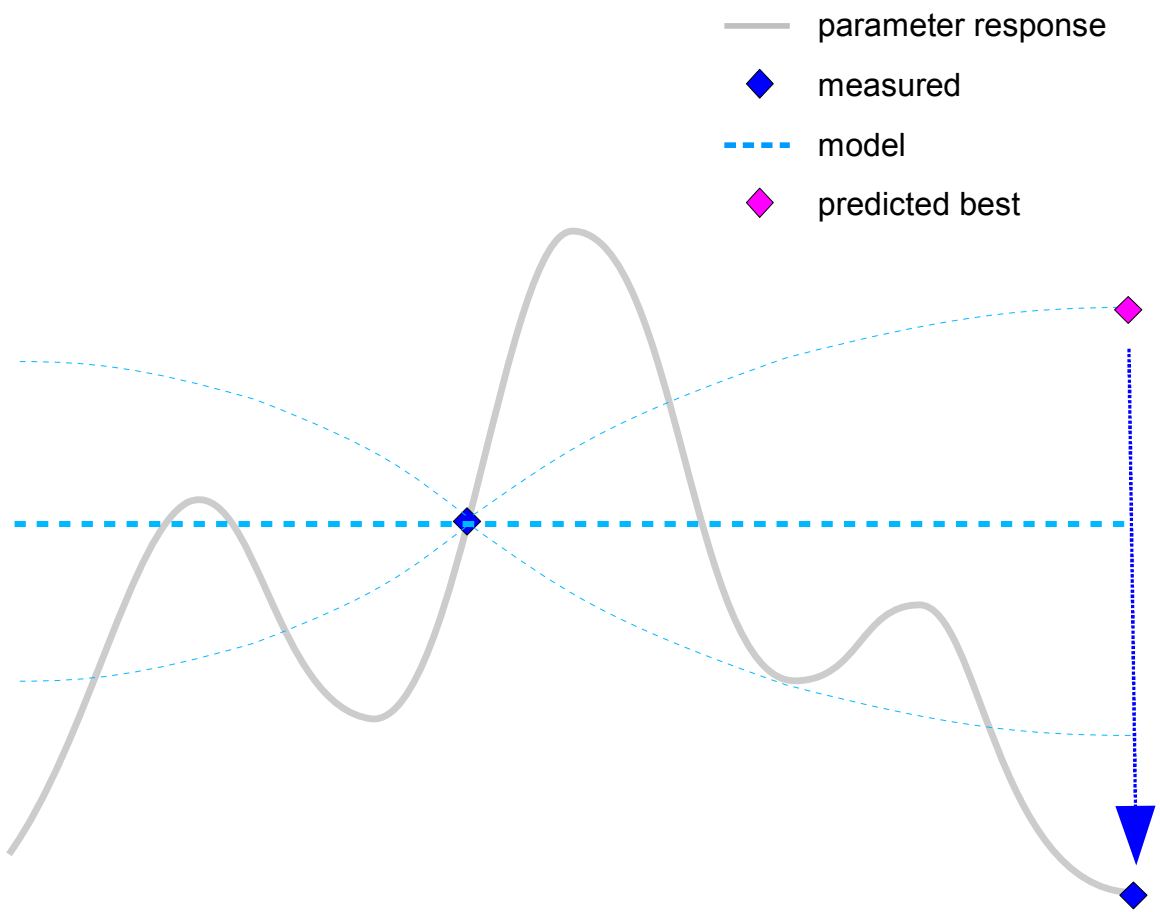
Sequential Model-based Optimisation



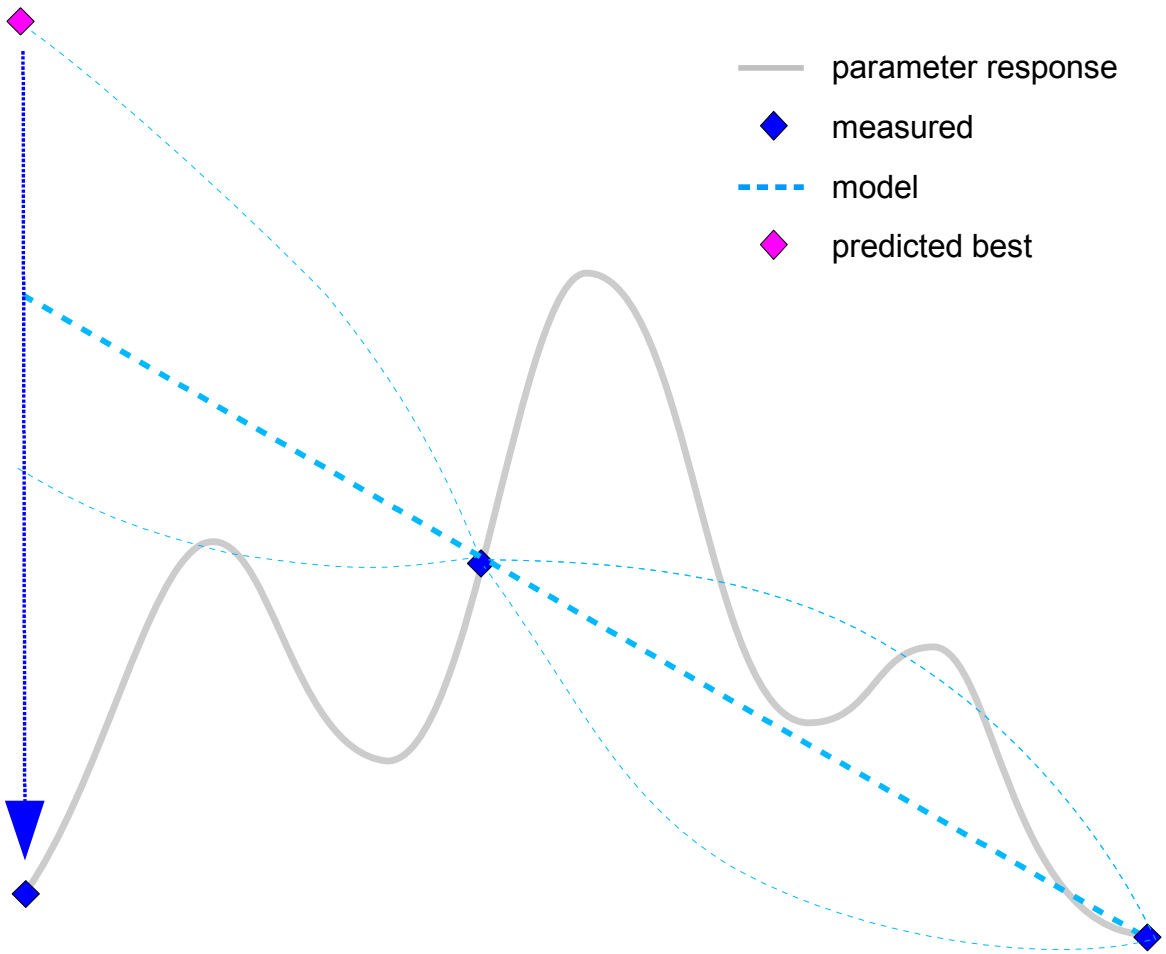
Sequential Model-based Optimisation



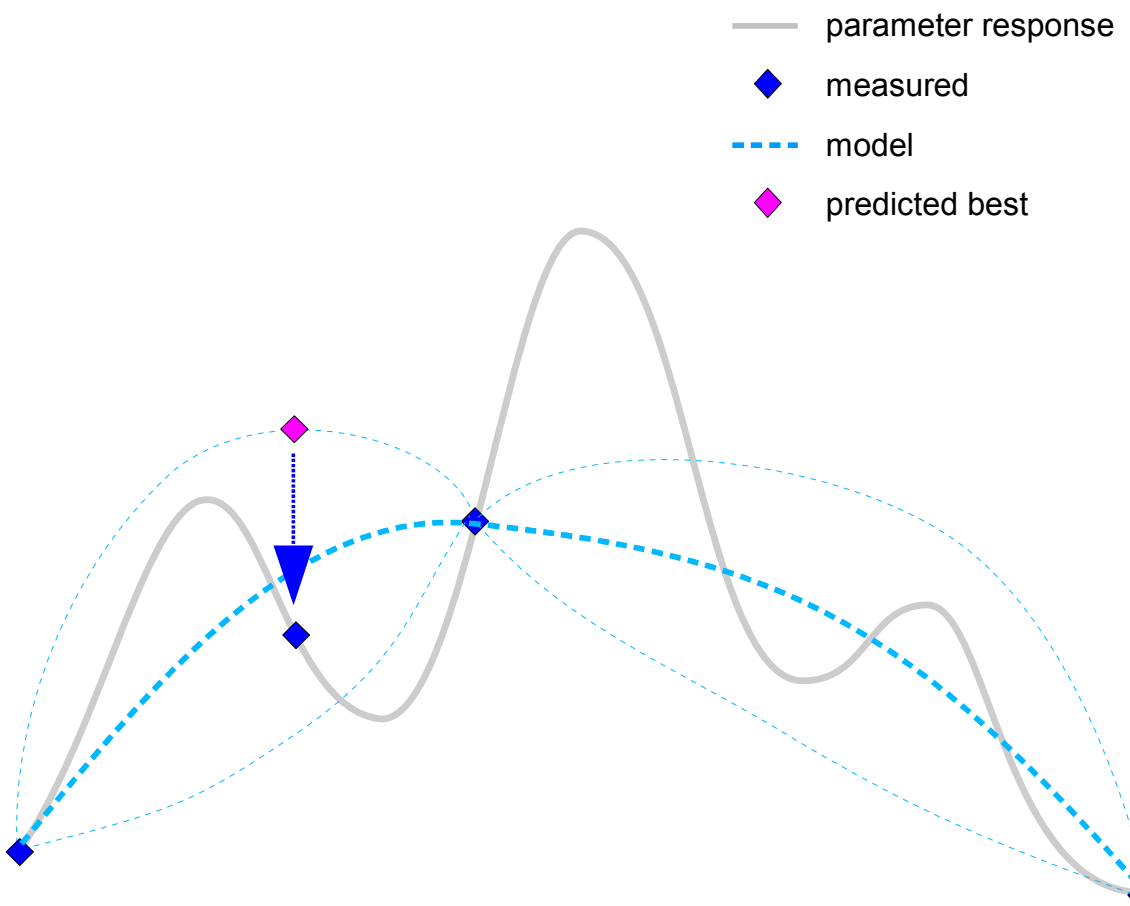
Sequential Model-based Optimisation



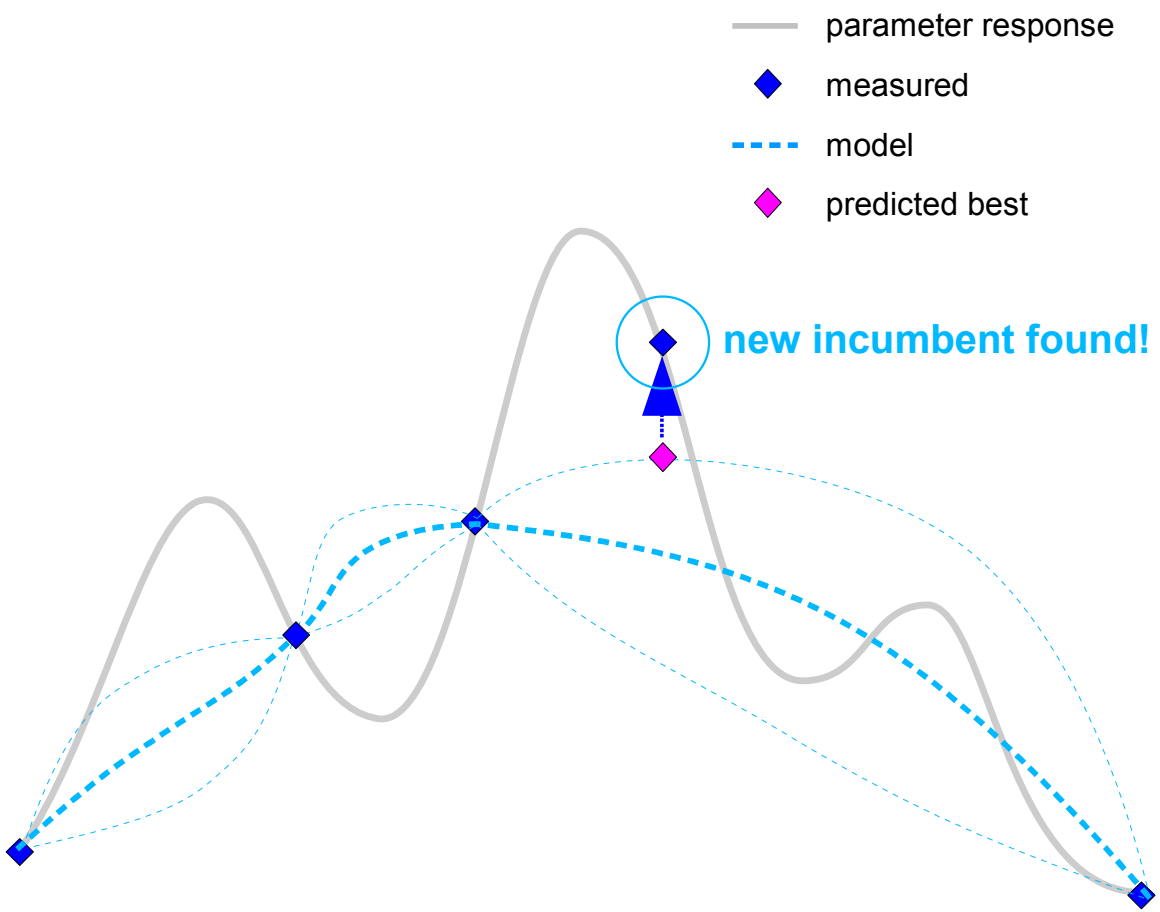
Sequential Model-based Optimisation



Sequential Model-based Optimisation



Sequential Model-based Optimisation

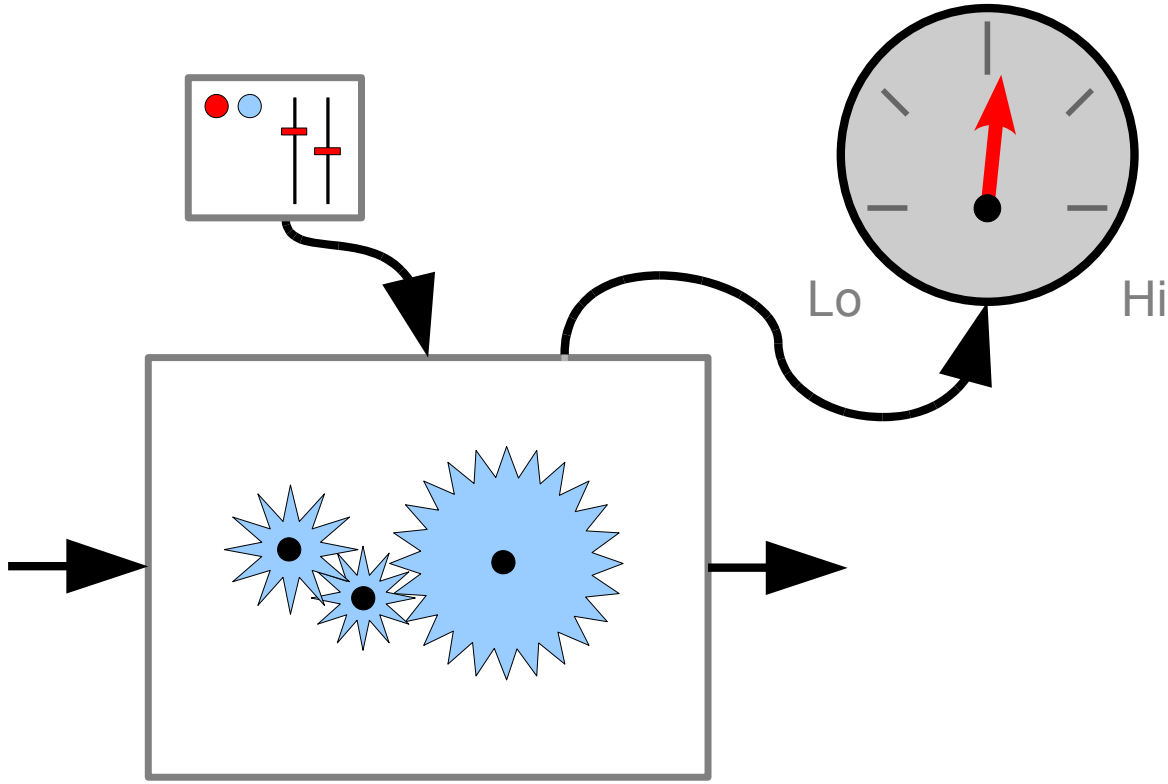


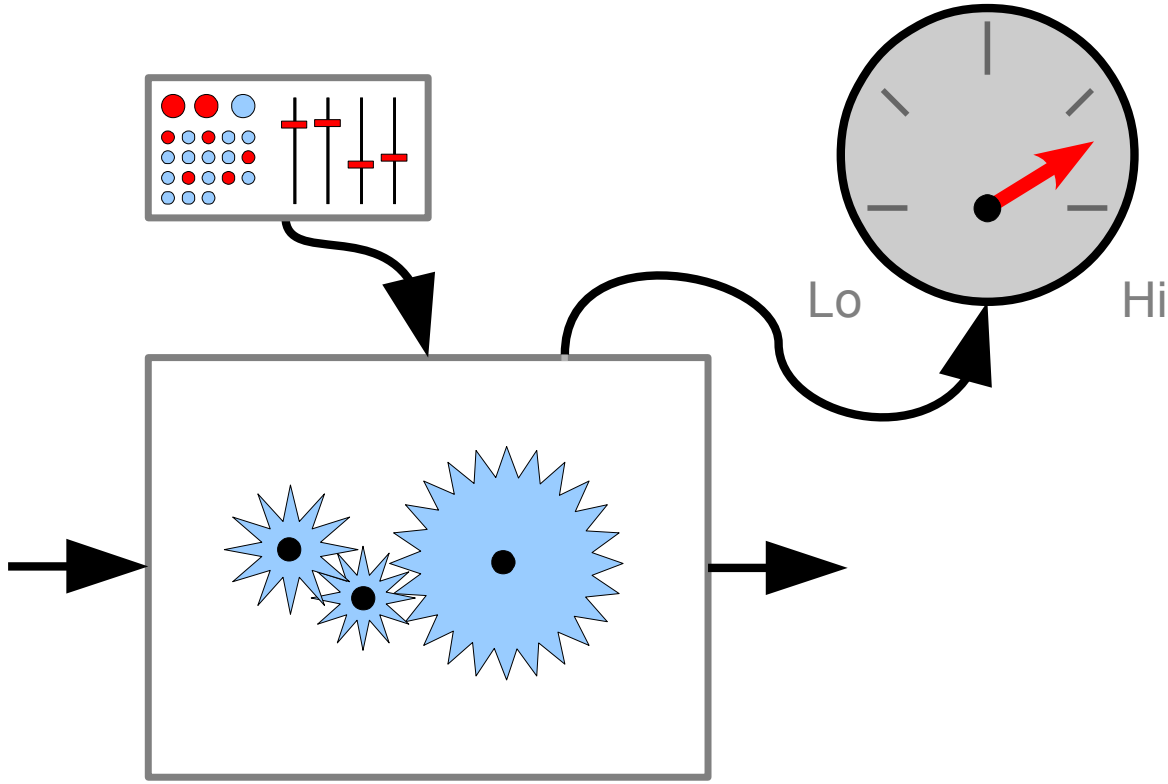
Sequential Model-based Algorithm Configuration (SMAC)

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)

2


Lo
Hi


Lo
Hi

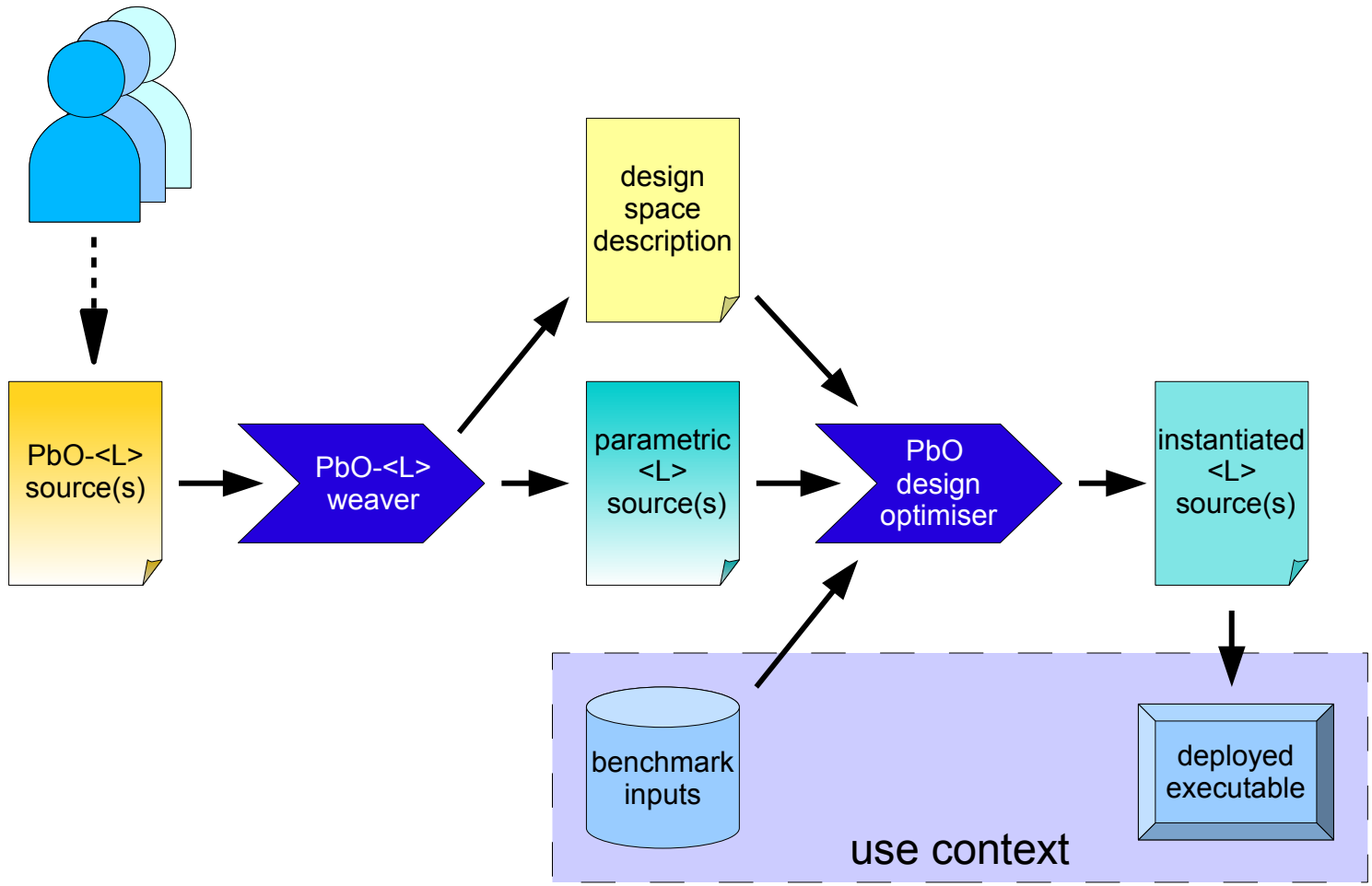
Programming by Optimisation (PbO)

HH (2012)

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)

Software development in the PbO paradigm



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), 26 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)

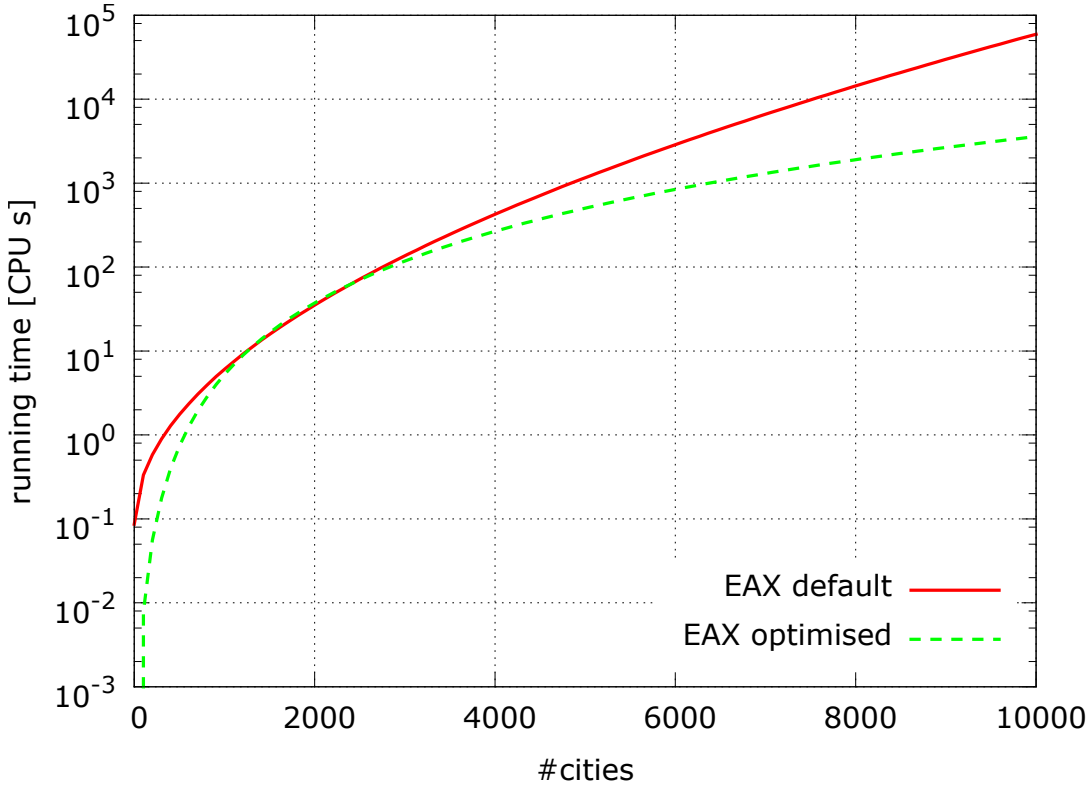
Machine learning / Classification, 786 design choices, PbO level 0–1
$\rightsquigarrow$ outperforms specialised model selection & hyper-parameter optimisation methods from machine learning
Thornton, Hutter, HH, Leyton-Brown (2012)

Median running time of EAX (state-of-the-art TSP solver)

Mu, Hoos, Stützle (under review)

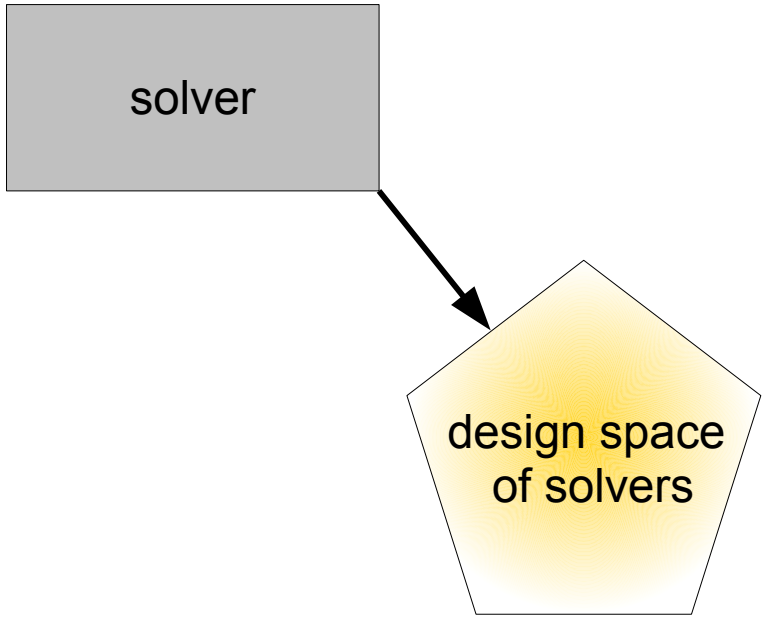
EAX (default): $0.086 \times 1.144^{\sqrt{n}}$

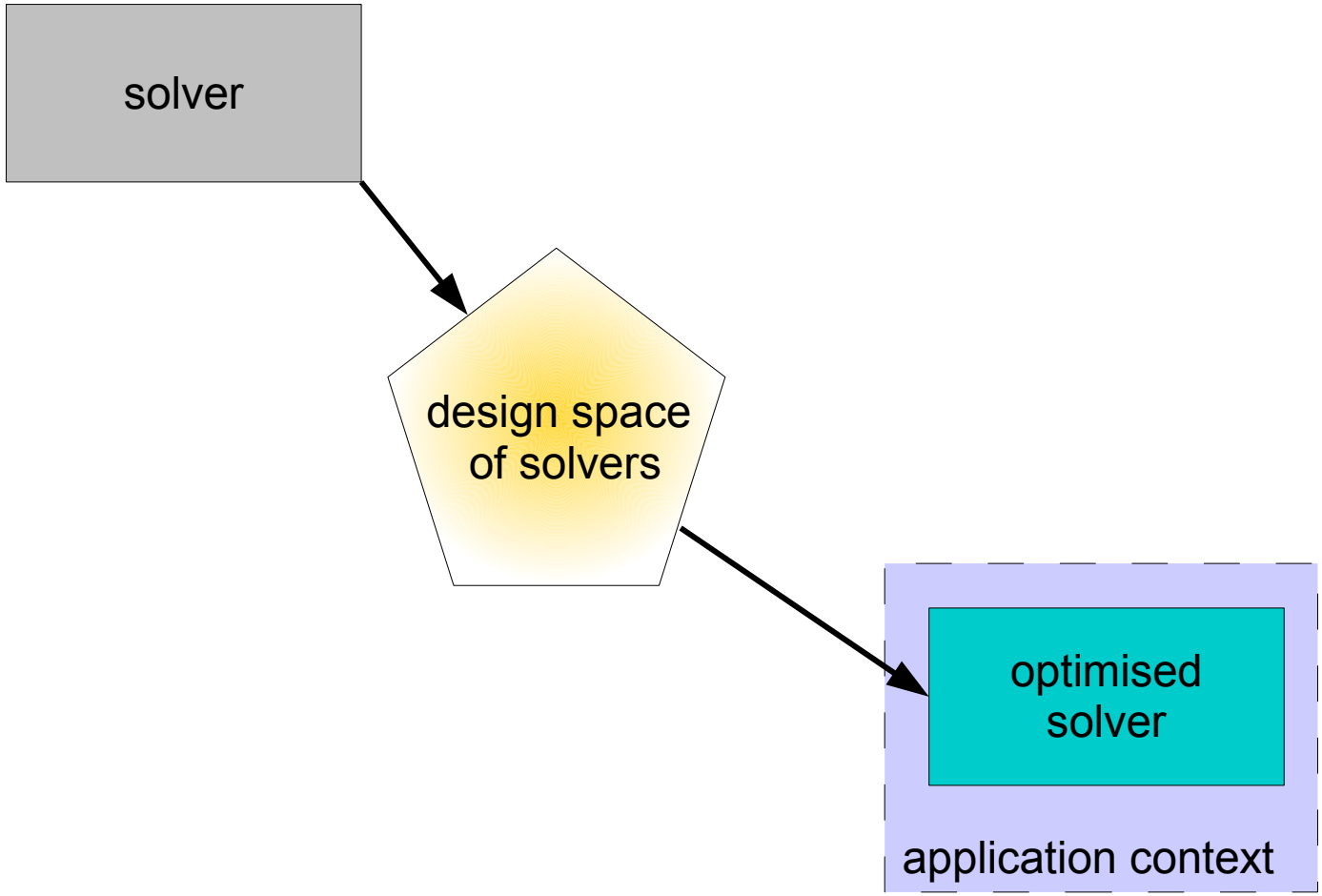
EAX (optimised): $1.62 \times 10^{-8} \times n^{2.836}$

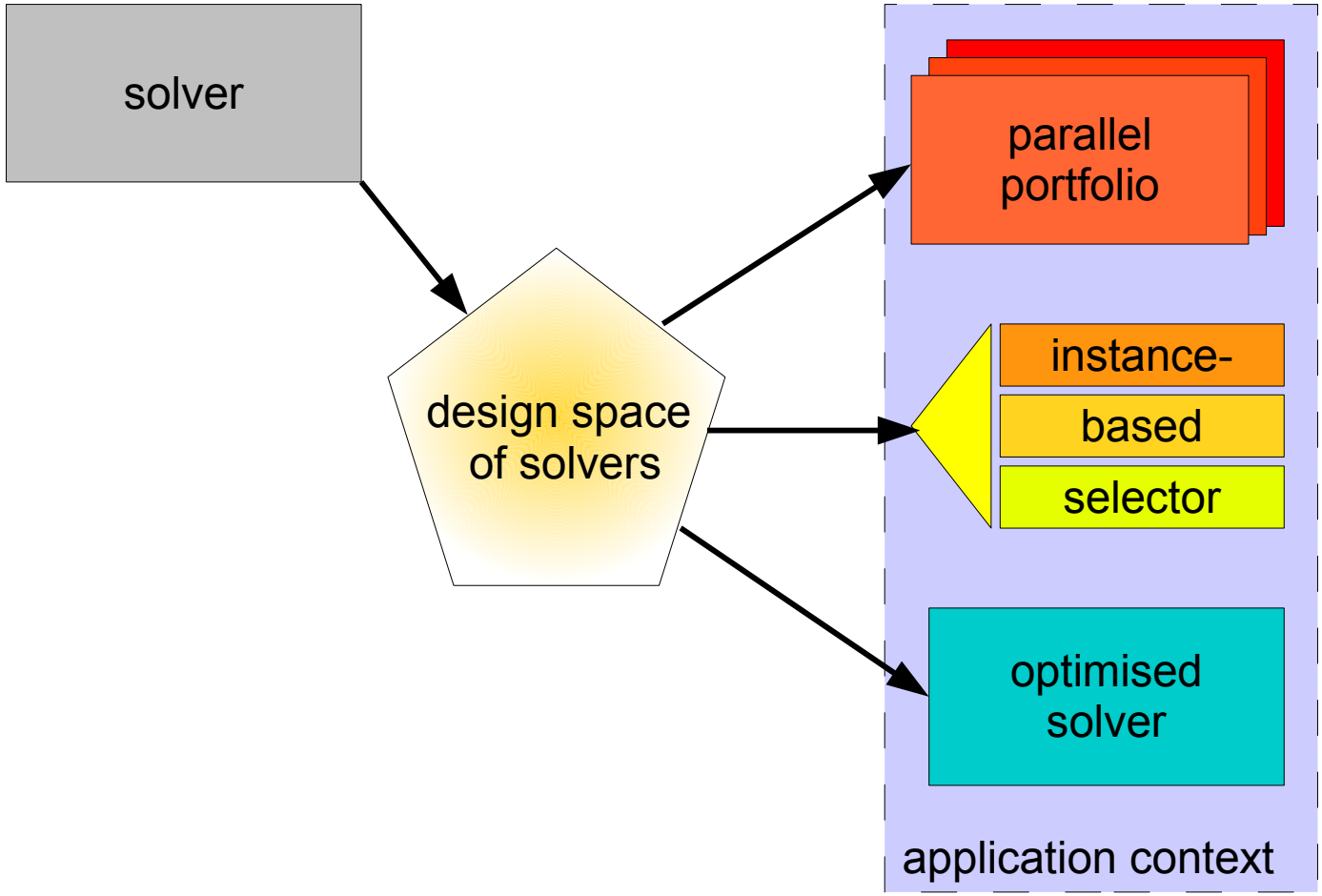


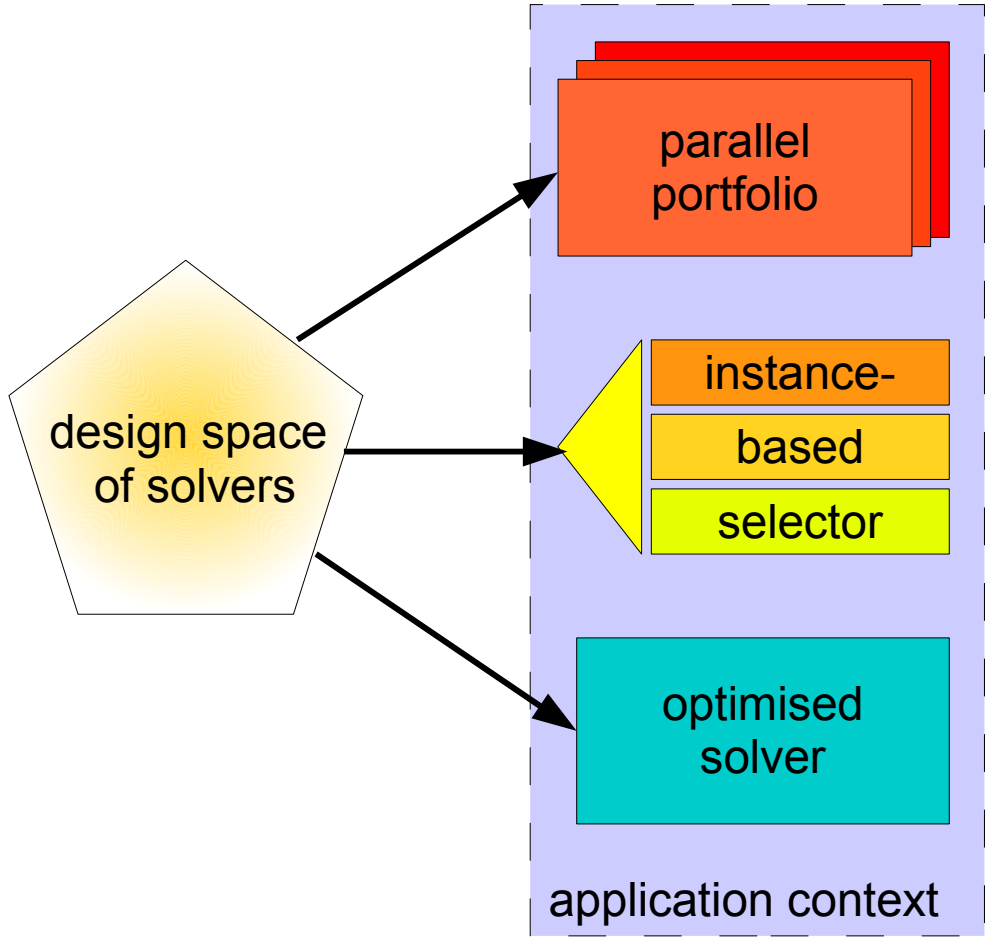
Further successful applications:

- macro learning in planning (Alhossaini & Beck 2012)
- garbage collection in Java (Lengauer & Mössenböck 2014)
- kidney exchange (Dickerson *et al.* 2012)


solver
design space
of solvers


solver
design space
of solvers
optimised
solver
application context


solver
design space
of solvers
parallel
portfolio
instance-
based
selector
optimised
solver
application context


parallel
portfolio
instance-
based
selector
design space
of solvers
optimised
solver
application context

Programming by Optimisation ...

- leverages computational power to construct better software
- enables creative thinking about design alternatives
- produces better performing, more flexible software
- facilitates scientific insights into
  - efficacy of algorithms and their components
  - empirical complexity of computational problems

... changes how we build and use high-performance software

Gli uomini hanno idee [...]
– Le idee, se sono allo stato puro, sono belle.
Ma sono un meraviglioso casino.
Sono apparizioni provvisorie di infinito.

People have ideas [...]
– Ideas, in their pure state, are beautiful.
But they are an amazing mess.
They are fleeting apparitions of the infinite.

(Prof. Mondrian Kilroy *in Alessandro Baricco: City*)