

# Programming by Optimisation: A Practical Paradigm for Computer-Aided Algorithm Design

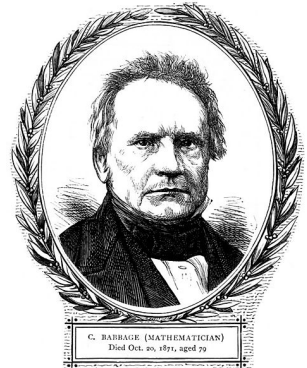
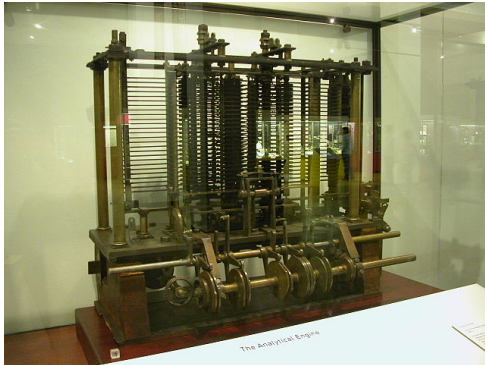
Holger H. Hoos & Frank Hutter

Department of Computer Science  
University of British Columbia  
Canada

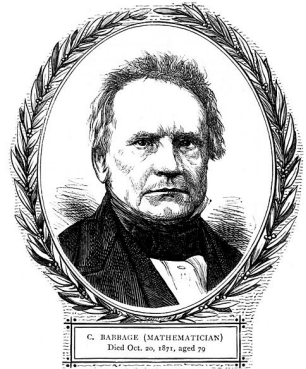
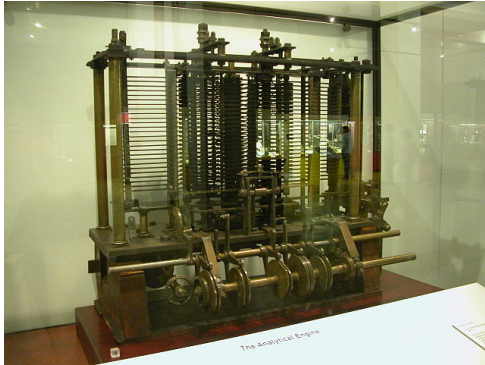
Department of Computer Science  
University of Freiburg  
Germany

IJCAI 2016  
New York City (NY), USA, 2016/07/10

## The age of machines

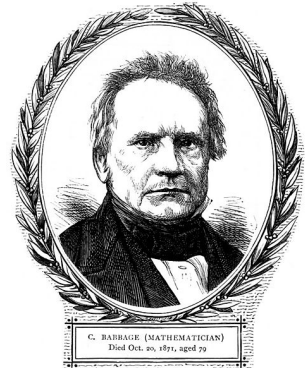
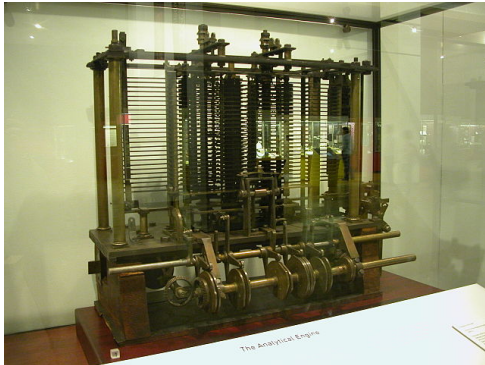


## The age of machines



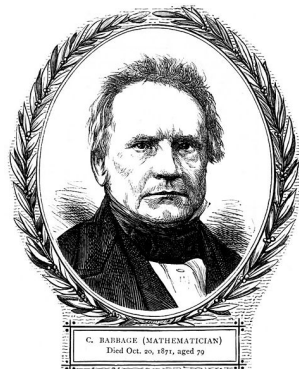
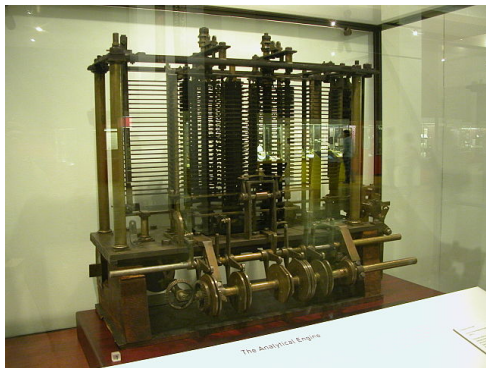
“As soon as an Analytical Engine exists, it will necessarily guide the future course of the science.

## 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 –

## 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)

22 August 2011 Last updated at 20:42 ET

1.4K     

## 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.

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.



Algorithms are spreading their influence around the globe

### Related Stories

[Are search engines skewing objectivity?](#)

[Robot reads minds to train itself](#)

# The age of computation



**BBC** mobile News Sport Weather Travel Food

## 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

Rate News Share Facebook Twitter YouTube

### 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.

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.



Algorithms are spreading their influence around the globe

#### Related Stories

Are search engines skewing objectivity?  
Robot made minds to train itself

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

# The age of computation



**BBC** mobile News Sport Weather Travel Food  
**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

## 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.

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.



Algorithms are spreading their influence around the globe

### Related Stories

- Are search engines skewing objectivity?
- Robot made minds to train itself

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

- ▶ financial markets
- ▶ social interactions



# The age of computation



**BBC** mobile News Sport Weather Travel Food

## 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

Rate News Share Facebook Twitter

### 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.

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.



Algorithms are spreading their influence around the globe

#### Related Stories

- Are search engines skewing objectivity?
- Robot made 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!)

## Performance matters ...

- ▶ computation speed (time is money!)
- ▶ energy consumption (battery life, ...)

## Performance matters ...

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

## Performance matters ...

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

... increasingly:

- ▶ globalised markets

## 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

## 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

## Example: Resource allocation

- ▶ resources  $>$  demands  $\rightsquigarrow$  many solutions, easy to find



## Example: Resource allocation

- ▶ resources  $>$  demands  $\rightsquigarrow$  many solutions, easy to find
  
- ▶ resources  $<$  demands  $\rightsquigarrow$  no solution, easy to demonstrate

## Example: Resource allocation

- ▶ resources  $>$  demands  $\rightsquigarrow$  many solutions, easy to find
  
- ▶ resources  $<$  demands  $\rightsquigarrow$  no solution, easy to demonstrate
  
- ▶ resources  $\approx$  demands  
 $\rightsquigarrow$  difficult to find solution / show infeasibility

## Example: Resource allocation

- ▶ resources  $>$  demands  $\rightsquigarrow$  many solutions, easy to find  
economically wasteful  
 $\rightsquigarrow$  reduction of resources / increase of demand
- ▶ resources  $<$  demands  $\rightsquigarrow$  no solution, easy to demonstrate
- ▶ resources  $\approx$  demands  
 $\rightsquigarrow$  difficult to find solution / show infeasibility

## Example: Resource allocation

- ▶ resources  $>$  demands  $\rightsquigarrow$  many solutions, easy to find  
economically wasteful  
 $\rightsquigarrow$  reduction of resources / increase of demand
- ▶ resources  $<$  demands  $\rightsquigarrow$  no solution, easy to demonstrate
- ▶ resources  $\approx$  demands  
 $\rightsquigarrow$  difficult to find solution / show infeasibility

## Example: Resource allocation

- ▶ resources  $>$  demands  $\rightsquigarrow$  many solutions, easy to find  
economically wasteful  
 $\rightsquigarrow$  reduction of resources / increase of demand
- ▶ resources  $<$  demands  $\rightsquigarrow$  no solution, easy to demonstrate  
lost market opportunity, strain within organisation  
 $\rightsquigarrow$  increase of resources / reduction of demand
- ▶ resources  $\approx$  demands  
 $\rightsquigarrow$  difficult to find solution / show infeasibility

## This tutorial:

new approach to software development, leveraging . . .

- ▶ human creativity

## This tutorial:

new approach to software development, leveraging . . .

- ▶ human creativity
- ▶ optimisation & machine learning

## This tutorial:

new approach to software development, leveraging . . .

- ▶ human creativity
- ▶ optimisation & machine learning
- ▶ large amounts of computation / data



## Key idea:

- ▶ program  $\rightsquigarrow$  (large) space of programs

## Key idea:

- ▶ program  $\rightsquigarrow$  (large) space of programs
- ▶ encourage software developers to
  - ▶ avoid premature commitment to design choices
  - ▶ seek & maintain design alternatives

## Key idea:

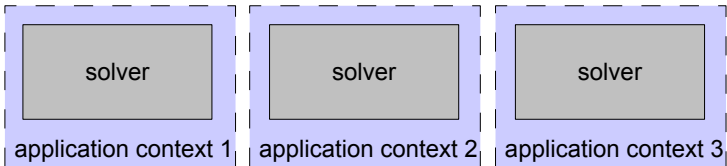
- ▶ program  $\rightsquigarrow$  (large) space of programs
- ▶ encourage software developers to
  - ▶ avoid premature commitment to design choices
  - ▶ seek & maintain design alternatives
- ▶ automatically find performance-optimising designs for given use context(s)

## 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)

⇒ Programming by Optimization (PbO)



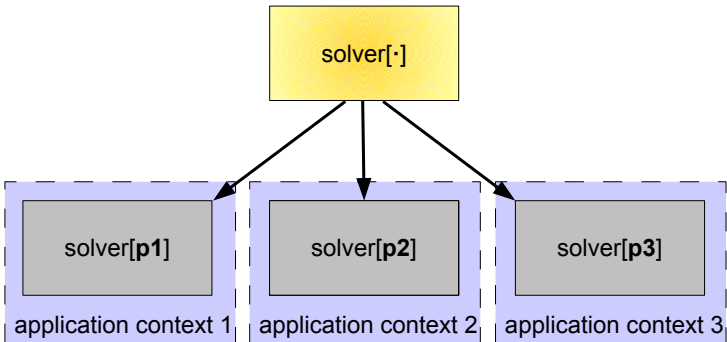


solver[.]

application context 1

application context 2

application context 3





## Outline

1. Programming by Optimization: Motivation & Introduction
2. Algorithm Configuration (incl. Coffee Break)
3. Portfolio-based Algorithm Selection
4. Software Development Support & Further Directions

# Programming by Optimization: Motivation & Introduction

## Example: SAT-based software verification

Hutter, Babić, Hoos, Hu (2007)

- ▶ **Goal:** Solve SAT-encoded software verification problems as fast as possible

## Example: SAT-based software verification

Hutter, Babić, Hoos, Hu (2007)

- ▶ **Goal:** Solve SAT-encoded software verification problems as fast as possible
- ▶ new DPLL-style SAT solver `SPEAR` (by Domagoj Babić)  
= highly parameterised heuristic algorithm  
(26 parameters,  $\approx 8.3 \times 10^{17}$  configurations)

## Example: SAT-based software verification

Hutter, Babić, Hoos, Hu (2007)

- ▶ **Goal:** Solve SAT-encoded software verification problems as fast as possible
- ▶ new DPLL-style SAT solver `SPEAR` (by Domagoj Babić)  
= highly parameterised heuristic algorithm  
(26 parameters,  $\approx 8.3 \times 10^{17}$  configurations)
- ▶ manual configuration by algorithm designer

## Example: SAT-based software verification

Hutter, Babić, Hoos, Hu (2007)

- ▶ **Goal:** Solve SAT-encoded software verification problems as fast as possible
- ▶ new DPLL-style SAT solver `SPEAR` (by Domagoj Babić)  
= highly parameterised heuristic algorithm  
(26 parameters,  $\approx 8.3 \times 10^{17}$  configurations)
- ▶ manual configuration by algorithm designer
- ▶ automated configuration using `ParamLLS`, a generic algorithm configuration procedure

Hutter, Hoos, Stützle (2007)

## SPEAR: Performance on software verification benchmarks

solver	num. solved	mean run-time
MiniSAT 2.0	302/302	161.3 CPU sec

## SPEAR: Performance on software verification benchmarks

solver	num. solved	mean run-time
MiniSAT 2.0	302/302	161.3 CPU sec
SPEAR original	298/302	787.1 CPU sec



## SPEAR: Performance on software verification benchmarks

solver	num. solved	mean run-time
MiniSAT 2.0	302/302	161.3 CPU sec
SPEAR original	298/302	787.1 CPU sec
SPEAR generic. opt. config.	302/302	35.9 CPU sec

## SPEAR: Performance on software verification benchmarks

solver	num. solved	mean run-time
MiniSAT 2.0	302/302	161.3 CPU sec
SPEAR original	298/302	787.1 CPU sec
SPEAR generic. opt. config.	302/302	35.9 CPU sec
SPEAR specific. opt. config.	302/302	1.5 CPU sec

## SPEAR: Performance on software verification benchmarks

solver	num. solved	mean run-time
MiniSAT 2.0	302/302	161.3 CPU sec
SPEAR original	298/302	787.1 CPU sec
SPEAR generic. opt. config.	302/302	35.9 CPU sec
SPEAR specific. opt. config.	302/302	1.5 CPU sec

- ▶  $\approx$  500-fold speedup through use automated algorithm configuration procedure (ParamLLS)
- ▶ new state of the art  
(winner of 2007 SMT Competition, QF\_BV category)

## Levels of PbO:

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



## Levels of PbO:

**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.



## Levels of PbO:

**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.



## Levels of PbO:

**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.



## Levels of PbO:

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

**Level 3:** Strive to provide design choices and alternatives.

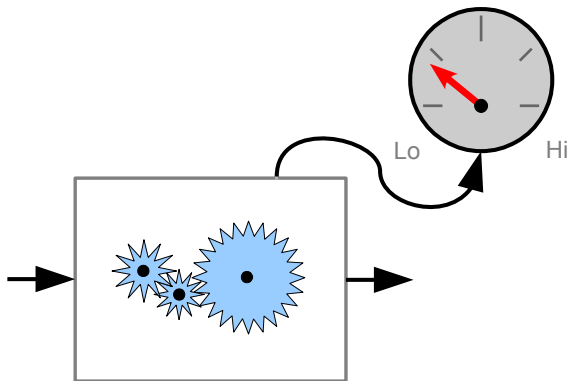
**Level 2:** Keep and expose design choices considered during software development.

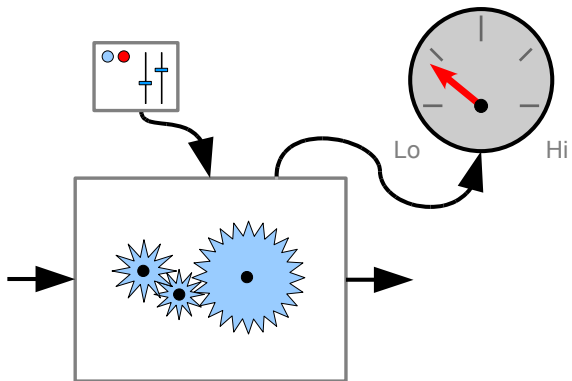
**Level 1:** Expose design choices hardwired into existing code (magic constants, hidden parameters, abandoned design alternatives).

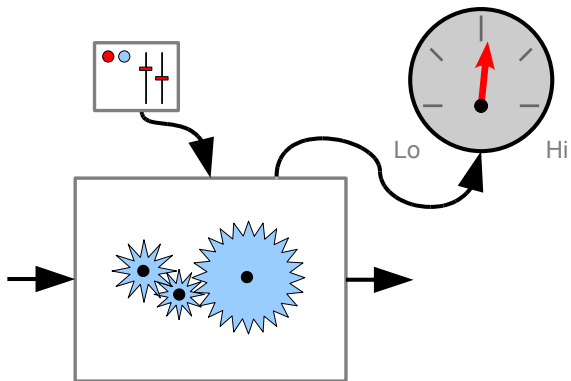
**Level 0:** Optimise settings of parameters exposed by existing software.

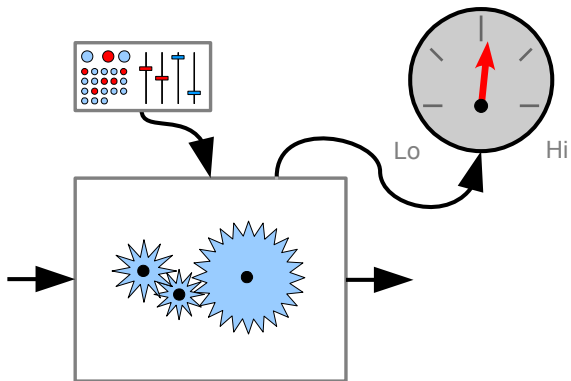


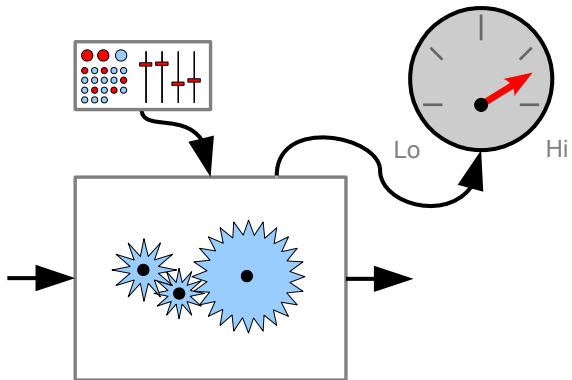












## Success in optimising speed:

Application, Design choices	Speedup	PbO level
SAT-based software verification (SPEAR), 41 Hutter, Babić, Hoos, Hu (2007)	4.5–500 ×	2–3

## Success in optimising speed:

Application, Design choices	Speedup	PbO level
<b>SAT-based software verification (SPEAR), 41</b> Hutter, Babić, Hoos, Hu (2007)	4.5–500 ×	2–3
<b>AI Planning (LPG), 62</b> Vallati, Fawcett, Gerevini, Hoos, Saetti (2011)	3–118 ×	1

## Success in optimising speed:

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



## Success in optimising speed:

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

## ... and solution quality:

University timetabling, 18 design choices, PbO level 2–3

↪ new state of the art; UBC exam scheduling

Fawcett, Chiarandini, Hoos (2009)

## Success in optimising speed:

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

## ... and solution quality:

University timetabling, 18 design choices, PbO level 2–3

↪ new state of the art; UBC exam scheduling

Fawcett, Chiarandini, Hoos (2009)

Machine learning / Classification, 786 design choices, PbO level 0–1

↪ outperforms specialised model selection & hyper-parameter optimisation methods from machine learning

Thornton, Hutter, Hoos, Leyton-Brown (2012–13)

PbO enables . . .

- ▶ **performance optimisation for different use contexts**  
(some details later)

## PbO enables . . .

- ▶ **performance optimisation for different use contexts**  
(some details later)
- ▶ **adaptation to changing use contexts**  
(see, e.g., life-long learning – Thrun 1996)

## PbO enables . . .

- ▶ **performance optimisation for different use contexts**  
(some details later)
- ▶ **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)

## PbO enables . . .

- ▶ **performance optimisation for different use contexts**  
(some details later)
- ▶ **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)
- ▶ **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)

## PbO enables . . .

- ▶ **performance optimisation for different use contexts**  
(some details later)
- ▶ **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)
- ▶ **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;  
Hoos *et al.* 2012)

# Cost & concerns

But what about ...



## Cost & concerns

But what about ...

- ▶ Computational complexity?

# Cost & concerns

But what about ...

- ▶ Computational complexity?
- ▶ Cost of development?

## Cost & concerns

But what about ...

- ▶ Computational complexity?
- ▶ Cost of development?
- ▶ Limitations of scope?

# Computationally too expensive?

## Computationally too expensive?

SPEAR revisited:

- ▶ total configuration time on software verification benchmarks:  
≈ 20 CPU days

## Computationally too expensive?

### SPEAR revisited:

- ▶ total configuration time on software verification benchmarks:  
≈ 20 CPU days
- ▶ wall-clock time on 10 CPU cluster:  
≈ 2 days

## Computationally too expensive?

### SPEAR revisited:

- ▶ total configuration time on software verification benchmarks:  
≈ 20 CPU days
- ▶ wall-clock time on 10 CPU cluster:  
≈ 2 days
- ▶ cost on Amazon Elastic Compute Cloud (EC2):  
50.40 USD

## Computationally too expensive?

### SPEAR revisited:

- ▶ total configuration time on software verification benchmarks:  
≈ 20 CPU days
- ▶ wall-clock time on 10 CPU cluster:  
≈ 2 days
- ▶ cost on Amazon Elastic Compute Cloud (EC2):  
50.40 USD
- ▶ 50.40 USD pays for ...
  - ▶ 1:14 hours of typical software engineer in the USA



## Computationally too expensive?

### SPEAR revisited:

- ▶ total configuration time on software verification benchmarks:  
≈ 20 CPU days
- ▶ wall-clock time on 10 CPU cluster:  
≈ 2 days
- ▶ cost on Amazon Elastic Compute Cloud (EC2):  
50.40 USD
- ▶ 50.40 USD pays for ...
  - ▶ 1:14 hours of typical software engineer in the USA
  - ▶ 5:36 hours at minimum wage in New York

Too expensive in terms of development?

## Too expensive in terms of development?

Design and coding:

- ▶ tradeoff between performance/flexibility and overhead

## Too expensive in terms of development?

Design and coding:

- ▶ tradeoff between performance/flexibility and overhead
- ▶ overhead depends on level of PbO

## Too expensive in terms of development?

### Design and coding:

- ▶ tradeoff between performance/flexibility and overhead
- ▶ overhead depends on level of PbO
- ▶ traditional approach: cost from manual exploration of design choices!

## Too expensive in terms of development?

### Design and coding:

- ▶ tradeoff between performance/flexibility and overhead
- ▶ overhead depends on level of PbO
- ▶ traditional approach: cost from manual exploration of design choices!

### Testing and debugging:

- ▶ design alternatives for individual mechanisms and components can be tested separately

## Too expensive in terms of development?

### Design and coding:

- ▶ tradeoff between performance/flexibility and overhead
- ▶ overhead depends on level of PbO
- ▶ traditional approach: cost from manual exploration of design choices!

### Testing and debugging:

- ▶ design alternatives for individual mechanisms and components can be tested separately
- ↪ effort linear (rather than exponential) in the number of design choices

Limited to the “niche” of NP-hard problem solving?



## Limited to the “niche” of NP-hard problem solving?

Some PbO-flavoured work in the literature:

- ▶ computing-platform-specific performance optimisation of linear algebra routines

(Whaley *et al.* 2001)

## Limited to the “niche” of NP-hard problem solving?

Some PbO-flavoured work in the literature:

- ▶ computing-platform-specific performance optimisation of linear algebra routines

(Whaley *et al.* 2001)

- ▶ optimisation of sorting algorithms using genetic programming

(Li *et al.* 2005)

## Limited to the “niche” of NP-hard problem solving?

Some PbO-flavoured work in the literature:

- ▶ computing-platform-specific performance optimisation of linear algebra routines

(Whaley *et al.* 2001)

- ▶ optimisation of sorting algorithms using genetic programming

(Li *et al.* 2005)

- ▶ compiler optimisation

(Pan & Eigenmann 2006; Cavazos *et al.* 2007)

## Limited to the “niche” of NP-hard problem solving?

Some PbO-flavoured work in the literature:

- ▶ computing-platform-specific performance optimisation of linear algebra routines

(Whaley *et al.* 2001)

- ▶ optimisation of sorting algorithms using genetic programming

(Li *et al.* 2005)

- ▶ compiler optimisation

(Pan & Eigenmann 2006; Cavazos *et al.* 2007)

- ▶ database server configuration

(Diao *et al.* 2003)

# Overview

- Programming by Optimization (PbO):  
Motivation and Introduction
- Algorithm Configuration
  - Methods (components of algorithm configuration)
  - Systems (that instantiate these components)
  - Demo & practical issues  
[coffee]
  - Case Studies
- Portfolio-Based Algorithm Selection
- Software Development Support & Further Directions

# Algorithm Configuration

- In a nutshell: **optimization of free parameters**
  - Which parameters? The ones you'd otherwise tune manually & more
- Examples of free parameters in various subfields of AI
  - **Tree search** (in particular for SAT): pre-processing, branching heuristics, clause learning & deletion, restarts, data structures, ...
  - **Local search**: neighbourhoods, perturbations, tabu length, annealing...
  - **Genetic algorithms**: population size, mating scheme, crossover operators, mutation rate, local improvement stages, hybridizations, ...
  - **Machine Learning**: pre-processing, regularization (type & strength), minibatch size, learning rate schedules, optimizer & its parameters, ...
  - **Deep learning** (in addition): #layers (& layer types), #units/layer, dropout constants, weight initialization and decay, non-linearities, ...

# Algorithm Parameters

## Parameter types

- Continuous, integer, ordinal
- **Categorical**: finite domain, unordered, e.g. {a,b,c}

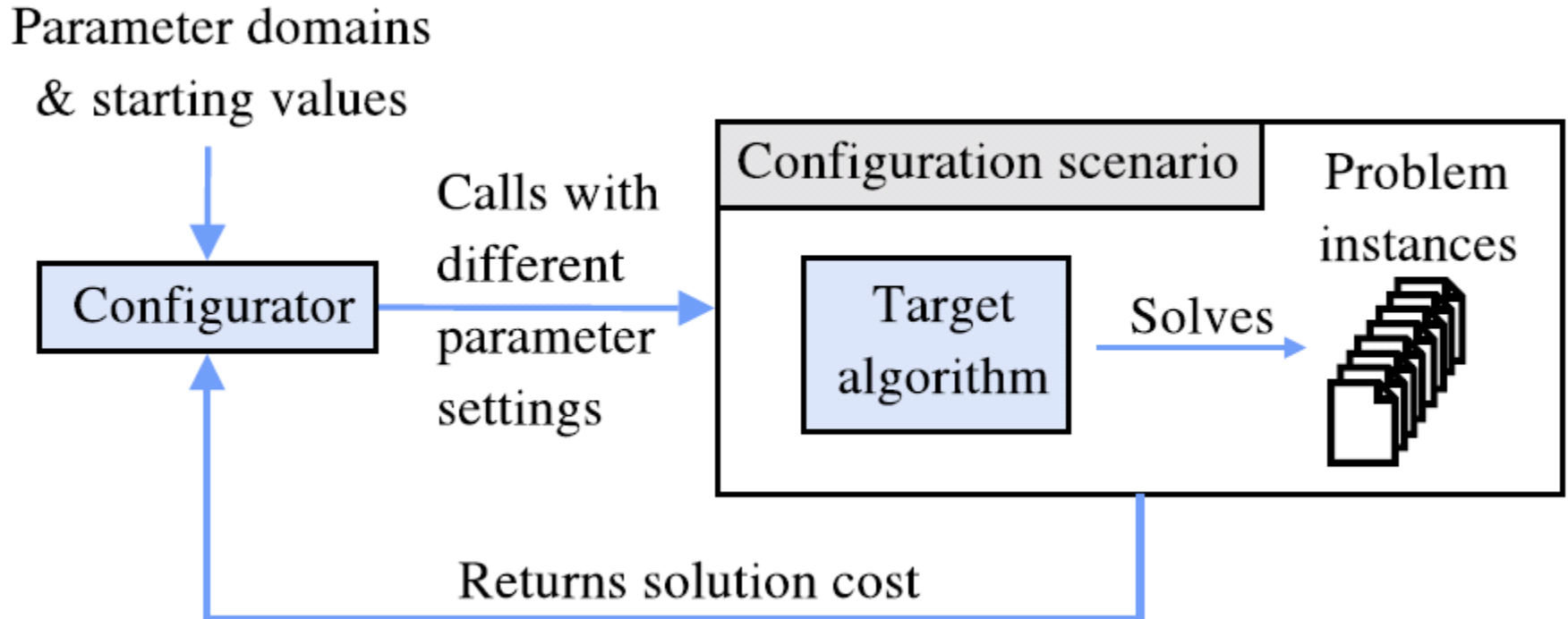
## Parameter space has **structure**

- E.g. parameter C is only active if heuristic H=h is used
- In this case, we say C is a **conditional parameter** with parent H

## Parameters give rise to a **structured space of algorithms**

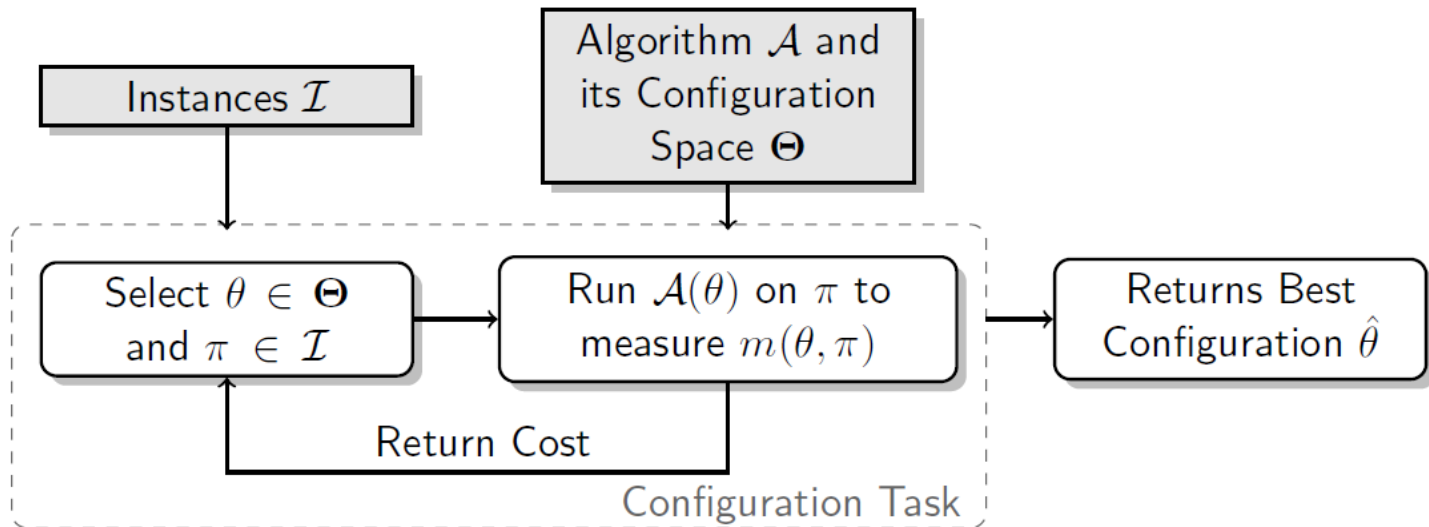
- Many **configurations** (e.g.  $10^{47}$ )
- Configurations often yield qualitatively different behaviour
- **Algorithm configuration** (as opposed to “parameter tuning”)

# The Algorithm Configuration Process





# Algorithm Configuration – in More Detail



## Definition: algorithm configuration

Given:

- a parameterized algorithm  $\mathcal{A}$  with possible parameter settings  $\Theta$ ;
- a distribution  $\mathcal{D}$  over problem instances with domain  $\mathcal{I}$ ; and
- a cost metric  $m : \Theta \times \mathcal{I} \rightarrow \mathbb{R}$ ,

Find:  $\theta^* \in \arg \min_{\theta \in \Theta} \mathbb{E}_{\pi \sim \mathcal{D}}(m(\theta, \pi))$ .

# Algorithm Configuration – Full Formal Definition

## Definition: algorithm configuration

An instance of the algorithm configuration problem is a 5-tuple  $(\mathcal{A}, \Theta, \mathcal{D}, \kappa, m)$  where:

- $\mathcal{A}$  is a parameterized algorithm;
- $\Theta$  is the parameter configuration space of  $\mathcal{A}$ ;
- $\mathcal{D}$  is a distribution over problem instances with domain  $\mathcal{I}$ ;
- $\kappa < \infty$  is a **cutoff time**, after which each run of  $\mathcal{A}$  will be terminated if still running
- $m : \Theta \times \mathcal{I} \rightarrow \mathbb{R}$  is a function that measures the observed cost of running  $\mathcal{A}(\theta)$  on an instance  $\pi \in \mathcal{I}$  with cutoff time  $\kappa$

The cost of a candidate solution  $\theta \in \Theta$  is  $c(\theta) = \mathbb{E}_{\pi \sim \mathcal{D}}(m(\theta, \pi))$ .

The goal is to find  $\theta^* \in \arg \min_{\theta \in \Theta} c(\theta)$ .

# Distribution vs. Set of Instances

Find:  $\theta^* \in \arg \min_{\theta \in \Theta} \mathbb{E}_{\pi \sim \mathcal{D}}(m(\theta, \pi))$ .

## Special case: distribution with finite support

- We often only have  $N$  instances from a given application
- In that case: split  $N$  instances into **training** and **test set**
- Find  $\theta^* \in \arg \min_{\theta \in \Theta} \frac{1}{N_{train}} \sum_{i=1}^{N_{train}} (m(\theta, \pi_i))$ .

## Evaluation on new test instances

Same approach as in machine learning

- We configure algorithms on the training instances
- We only use test instances to assess generalization performance
  - unbiased estimate of generalization performance for unseen instances

# Algorithm Configuration is a Useful Abstraction

Two quite different instantiations:

Minimize the runtime of a SAT solver for a benchmark set

- Optimize on training set:

$$\theta^* \in \arg \min_{\theta \in \Theta} \frac{1}{N_{train}} \sum_{i=1}^{N_{train}} (m(\theta, \pi_i))$$

Minimize  $K$ -fold cross-validation error of a machine learning algorithm

- A cross-validation fold  $k$  plays the role of an instance

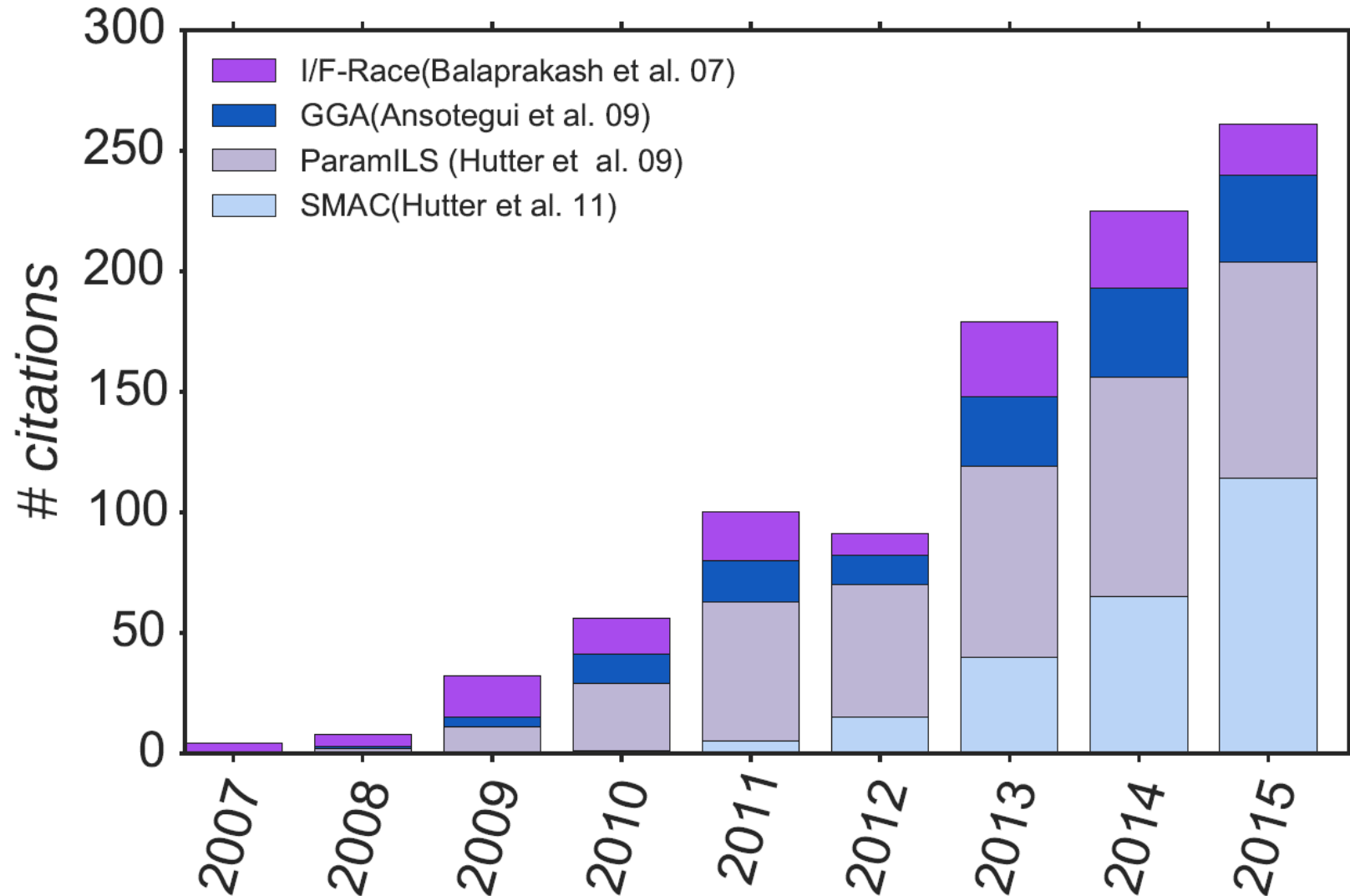
$$\theta^* \in \arg \min_{\theta \in \Theta} \frac{1}{K} \sum_{k=1}^K (m(\theta, k))$$

**Large improvements** to solvers for  
**many hard combinatorial problems**

SAT, Max-SAT, MIP, SMT, TSP, ASP, time-tabling, AI planning, ...  
Competition winners for all of these rely on configuration tools

# Algorithm Configuration is a Useful Abstraction

Increasingly popular (citation numbers from Google scholar)



# Overview

- Programming by Optimization (PbO):  
Motivation and Introduction
- Algorithm Configuration
  - ➔ Methods (components of algorithm configuration)
    - Systems (that instantiate these components)
    - Demo & practical issues
    - Case studies
- Portfolio-Based Algorithm Selection
- Software Development Support & Further Directions

# Configurators have Two Key Components

- Component 1: **which configuration to evaluate** next?
  - Out of a large combinatorial search space
  - E.g., CPLEX: 76 parameters,  $10^{47}$  configurations
- Component 2: **how to evaluate that configuration?**
  - Evaluating performance of a configuration is expensive
  - E.g., CPLEX: budget of 10000s per instance
  - Instances vary in hardness
    - Some take milliseconds, other days (for the default)
    - Improvement on a few instances might not mean much

# Component 1: Which Configuration to Choose?

- For this component, we can consider a simpler problem:

**Blackbox function optimization**

$$\min_{\theta \in \Theta} f(\theta)$$

- **Only mode of interaction: query  $f(\theta)$  at arbitrary  $\theta \in \Theta$**



- Abstracts away the complexity of multiple instances
- $\Theta$  is still a structured space
  - Mixed continuous/discrete
  - Conditional parameters
  - Still more general than “standard” continuous BBO [e.g., Hansen et al.]

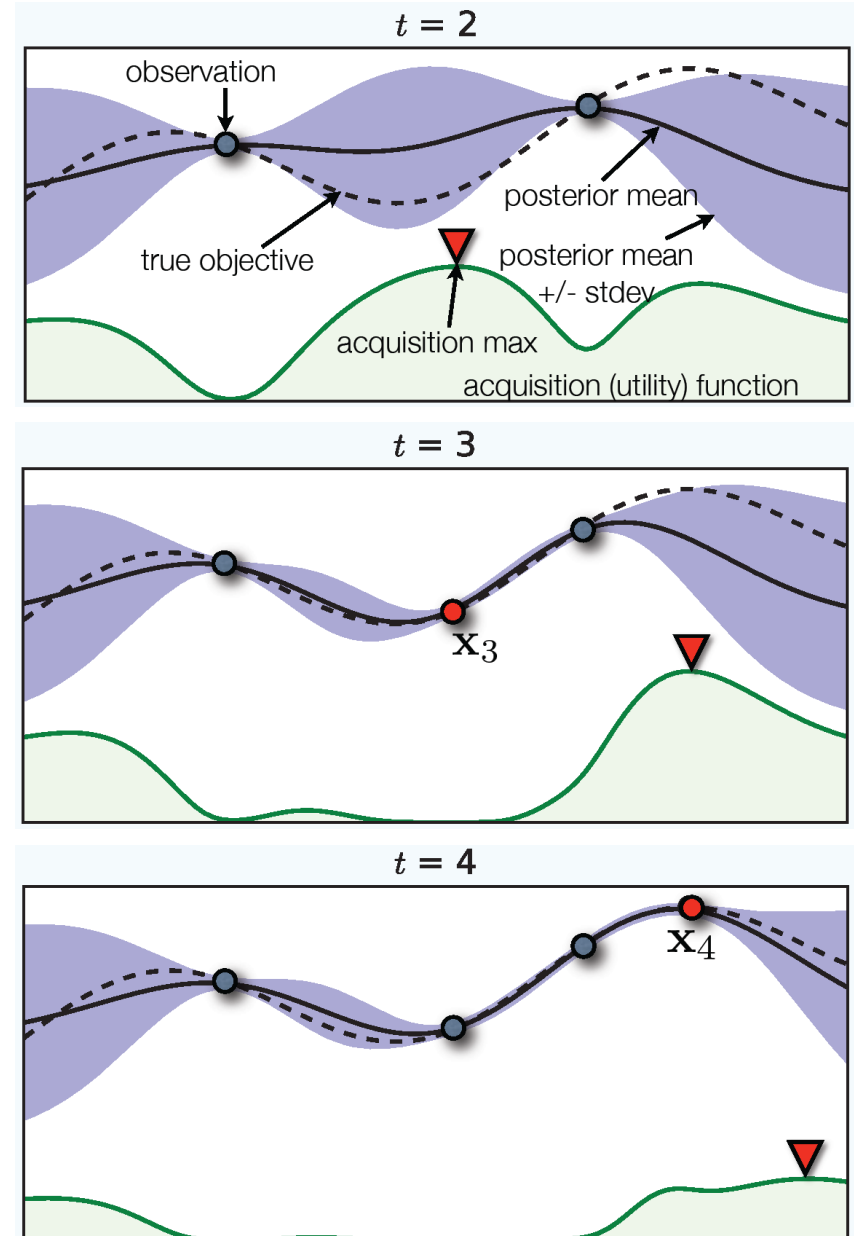


# Component 1: Which Configuration to Choose?

- Need to balance diversification and intensification
- The extremes
  - Random search
  - Hillclimbing
- Stochastic local search (SLS)
- Population-based methods
- Model-Based Optimization

# Sequential Model-Based Optimization

- Fit a (probabilistic) model of the function  $f$
- Use that model to trade off exploitation vs exploration
- In the machine learning literature known as **Bayesian Optimization**



# Component 2: How to Evaluate a Configuration?

Back to general algorithm configuration

## Definition: algorithm configuration

Given:

- a parameterized algorithm  $\mathcal{A}$  with possible parameter settings  $\Theta$ ;
- a distribution  $\mathcal{D}$  over problem instances with domain  $\mathcal{I}$ ; and
- a cost metric  $m : \Theta \times \mathcal{I} \rightarrow \mathbb{R}$ ,

Find:  $\theta^* \in \arg \min_{\theta \in \Theta} \mathbb{E}_{\pi \sim \mathcal{D}}(m(\theta, \pi))$ .

- **General principle**

- Don't waste too much time on bad configurations
- Evaluate good configurations more thoroughly

# Simplest Solution: Use Fixed N Instances

- Effectively treats the problem as a blackbox function optimization problem
- **Issue: how large to choose N?**
  - Too small: overtuning to those instances
  - Too large: every function evaluation is slow

# Racing Algorithms

[Maron & Moore, NIPS 1994]

[Birattari, Stützle, Paquete & Varrentrapp, GECCO 2002]

- Compare two or more algorithms against each other
  - Perform one run for each configuration at a time
  - **Discard configurations when dominated**

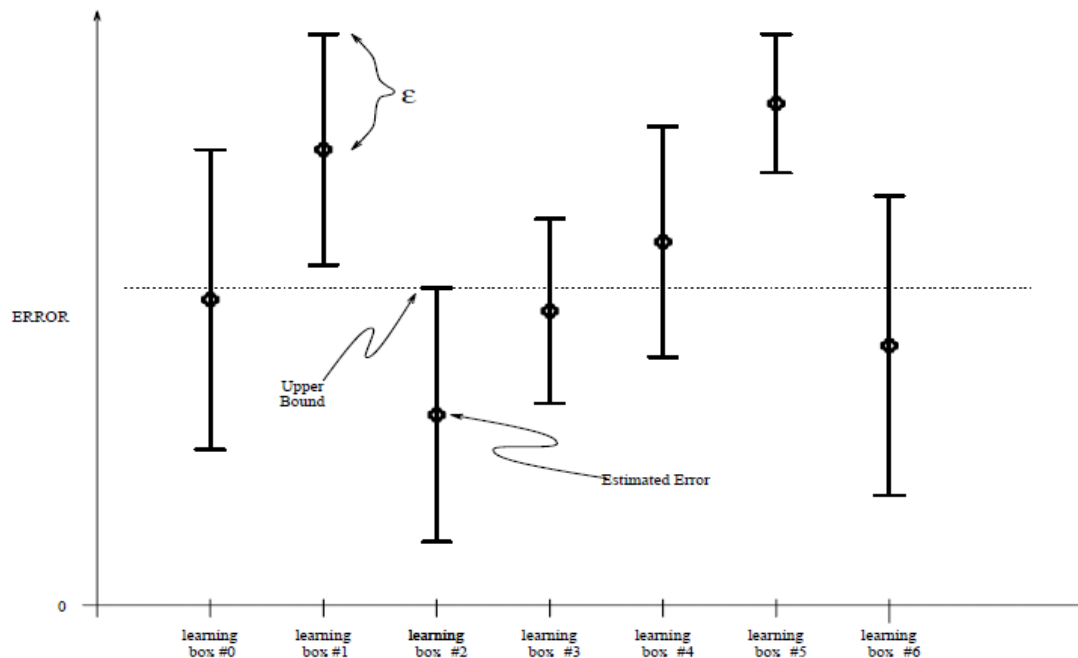


Image source: Maron & Moore, Hoeffding Races, NIPS 1994

# Saving Time: Aggressive Racing

[Hutter, Hoos & Stützle, AAAI 2007]

- Race new configurations against the best known
  - Discard poor new configurations quickly
  - **No requirement for statistical domination**
- Search component should allow to return to configurations discarded because they were “unlucky”

# Saving More Time: Adaptive Capping

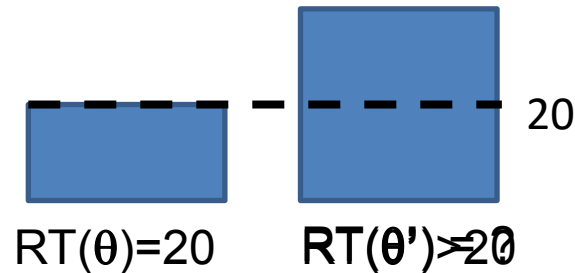
[Hutter, Hoos, Leyton-Brown & Stützle, JAIR 2009]

(only when minimizing algorithm runtime)

Can terminate runs for poor configurations  $\theta'$  early:

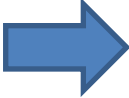
– Is  $\theta'$  better than  $\theta$ ?

• Example:



• Can terminate evaluation of  $\theta'$  once guaranteed to be worse than  $\theta$

# Overview

- Programming by Optimization (PbO):  
Motivation and Introduction
- Algorithm Configuration
  - Methods (components of algorithm configuration)
  -  Systems (that instantiate these components)
    - Demo & practical issues
    - Case studies
- Portfolio-Based Algorithm Selection
- Software Development Support & Further Directions



# Overview: Algorithm Configuration Systems

- Continuous parameters, single instances (**blackbox opt**)
  - Covariance adaptation evolutionary strategy (CMA-ES)  
[Hansen et al, since '06]
  - Sequential Parameter Optimization (SPO) [Bartz-Beielstein et al, '06]
- **General algorithm configuration** methods
  - ParamILS [Hutter et al, '07 and '09]
  - Gender-based Genetic Algorithm (GGA) [Ansotegui et al, '09]
  - Iterated F-Race [Birattari et al, '02 and '10]
  - Sequential Model-based Algorithm Configuration (SMAC)  
[Hutter et al, since '11]
  - Distributed SMAC [Hutter et al, since '12]

# The Baseline: Graduate Student Descent

*Start with some configuration*

**repeat**

**Modify a single parameter**

**if** *performance on a benchmark set degrades* **then**

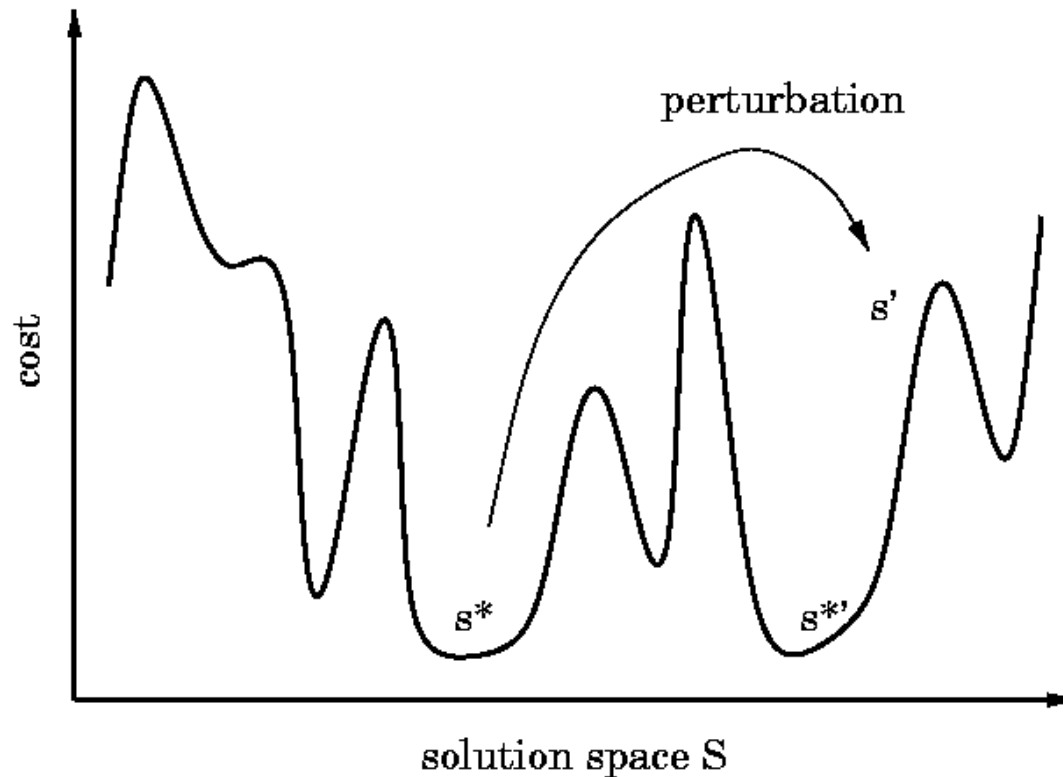
└ *undo modification*

**until** *no more improvement possible*  
*(or “good enough”)*

# The ParamLS Framework

[Hutter, Hoos, Leyton-Brown & Stützle, AAAI 2007 & JAIR 2009]

Iterated Local Search in parameter configuration space:

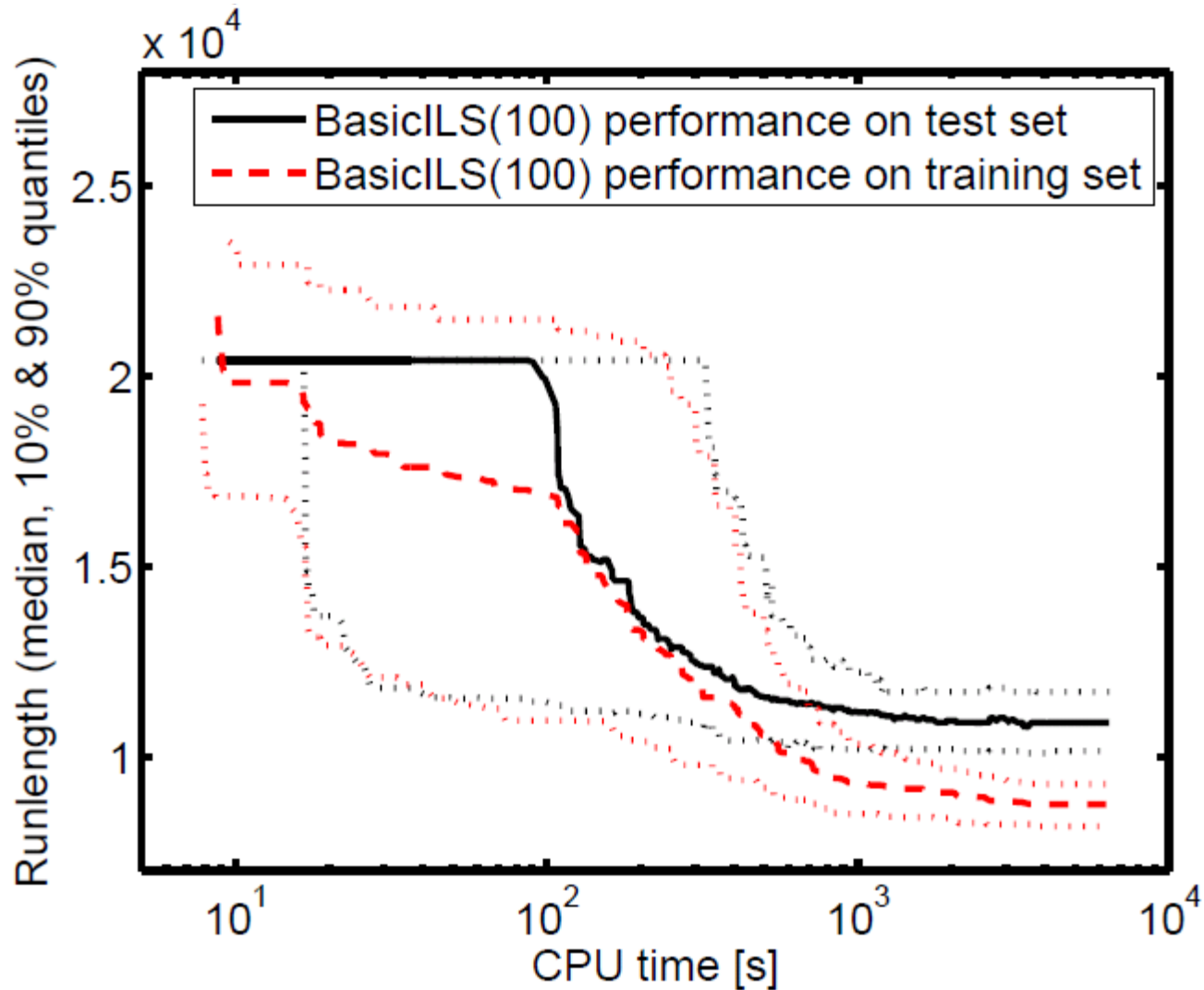


→ Performs **biased random walk over local optima**

# The BasicLS(N) algorithm

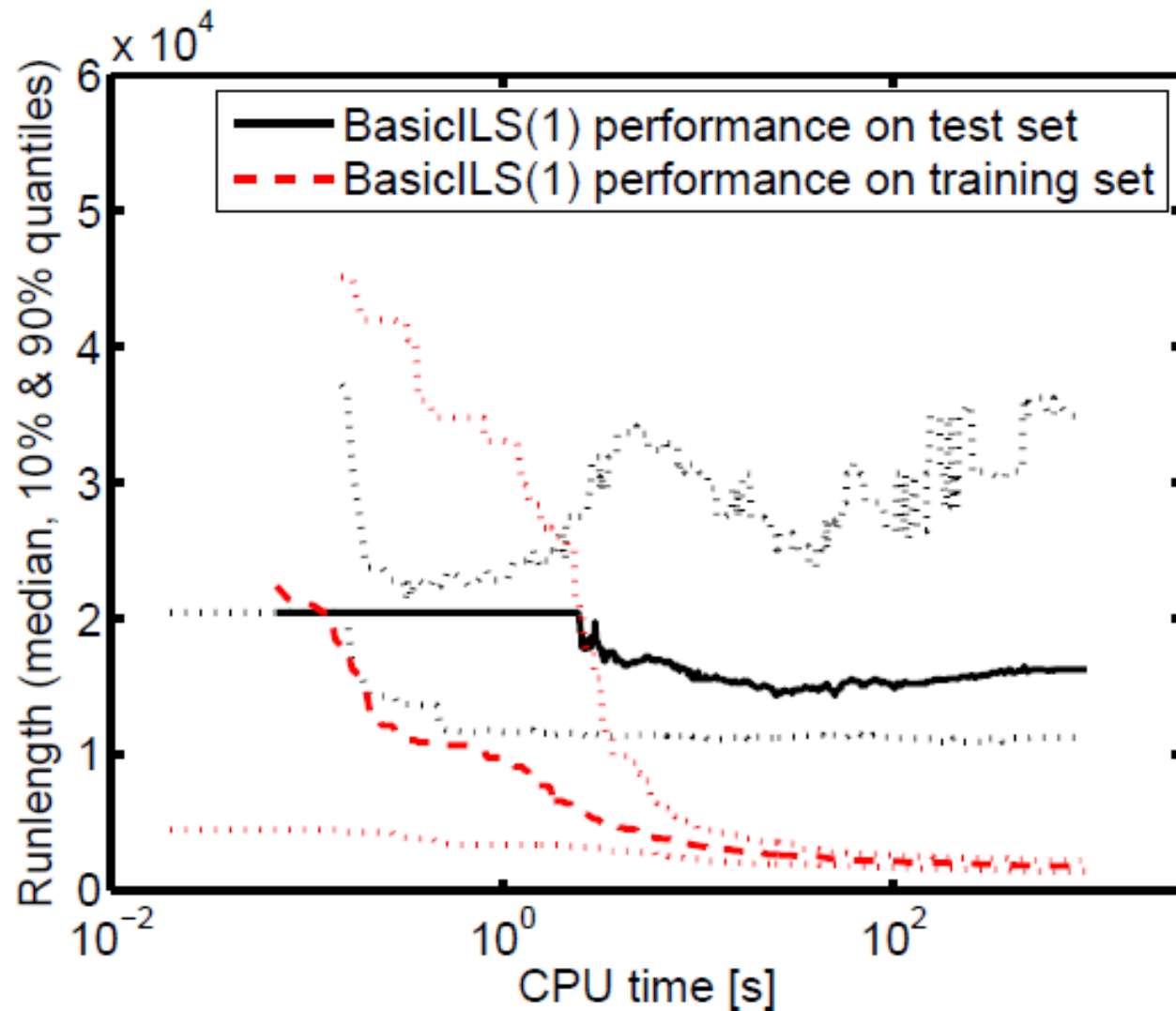
- Instantiates the ParamLS framework
- **Uses a fixed number of N runs for each evaluation**
  - Sample N instance from given set (with repetitions)
  - Same instances (and seeds) for evaluating all configurations
  - Essentially treats the problem as blackbox optimization
- **How to choose N?**
  - Too high: evaluating a configuration is expensive
    - Optimization process is slow
  - Too low: noisy approximations of true cost
    - Poor generalization to test instances / seeds

# Generalization to Test set, Large N (N=100)



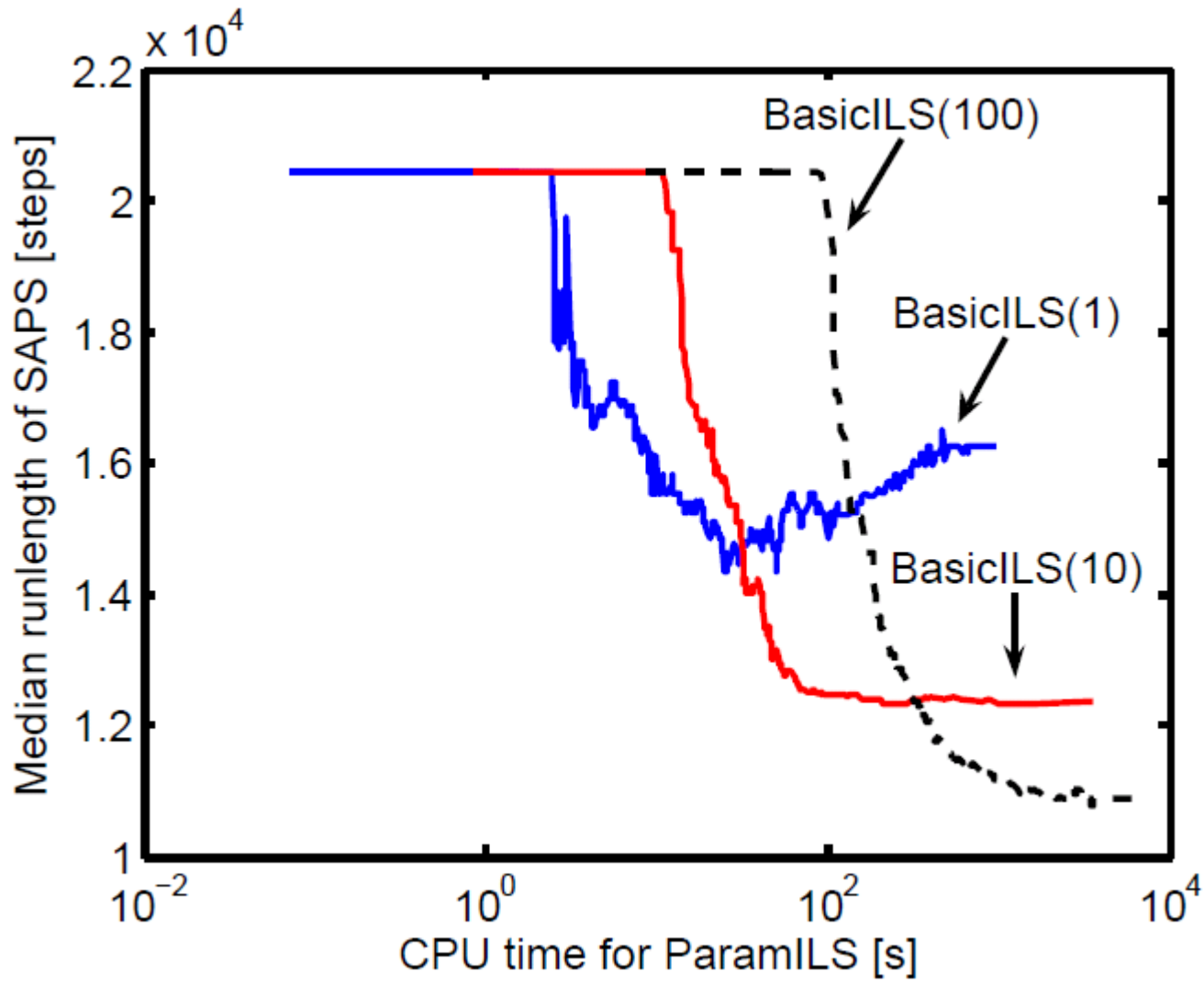
SAPS on a single QWH instance  
(same instance for training & test; only difference: seeds)

# Generalization to Test Set, Small N (N=1)



SAPS on a single QWH instance  
(same instance for training & test; only difference: seeds)

# BasicLS: Speed/Generalization Tradeoff



Test performance of SAPS on a single QWH instance

# The FocusedILS Algorithm

## Aggressive racing: more runs for good configurations

- Start with  $N(\theta) = 0$  for all configurations
- Increment  $N(\theta)$  whenever the search visits  $\theta$
- “Bonus” runs for configurations that win many comparisons

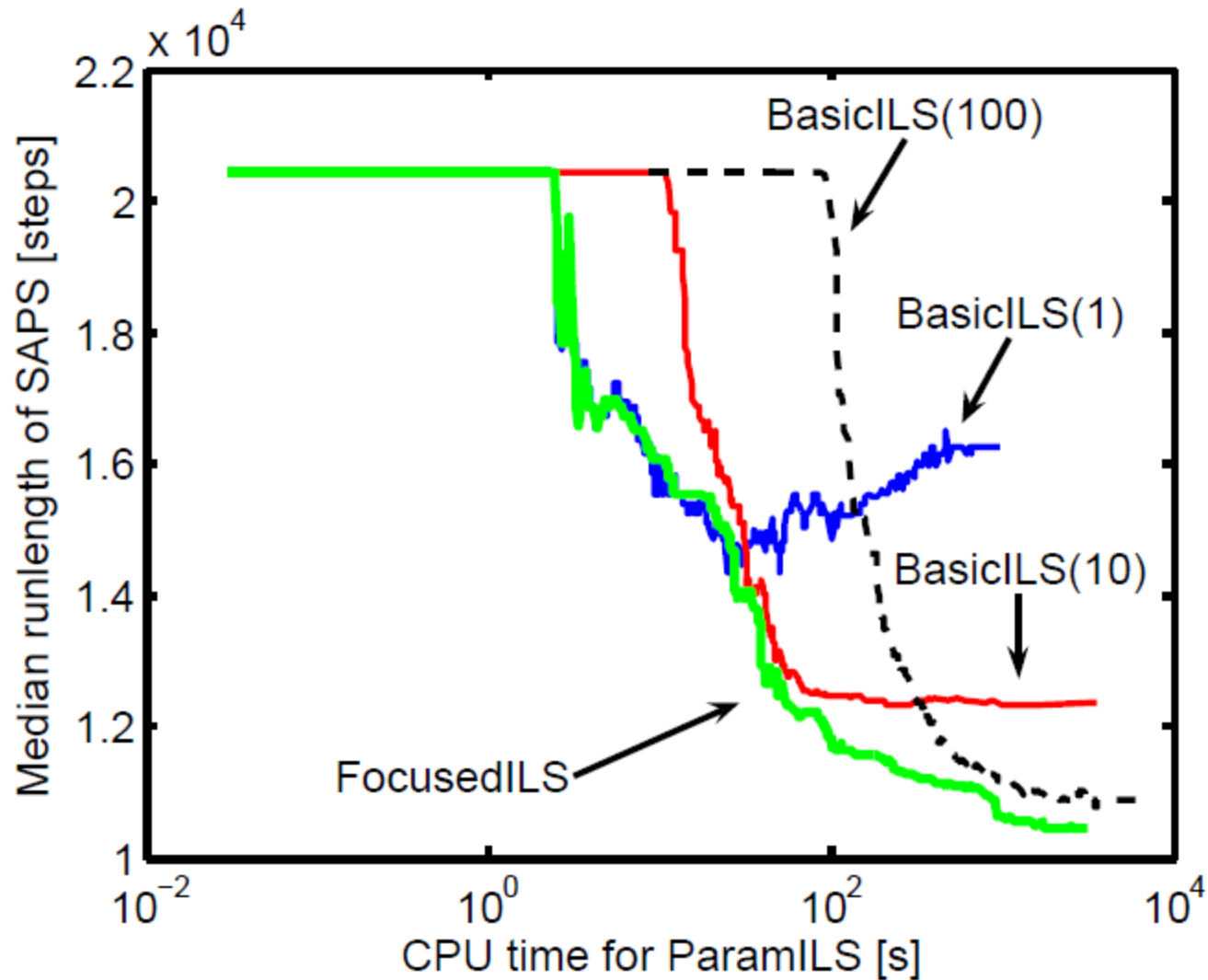
## Theorem

As the number of FocusedILS iterations  $\rightarrow \infty$ ,  
it converges to the true optimal configuration

- Key ideas in proof:
  1. The underlying ILS eventually reaches any configuration
  2. For  $N(\theta) \rightarrow \infty$ , the error in cost approximations vanishes



# FocusedILS: Speed/Generalization Tradeoff



Test performance of SAPS on a single QWH instance

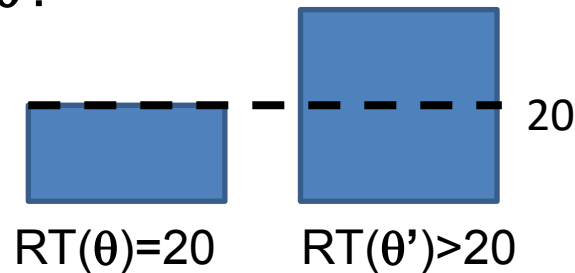
# Speeding up ParamILS

[Hutter , Hoos, Leyton-Brown, and Stützle, JAIR 2009]

## Standard adaptive capping

– Is  $\theta'$  better than  $\theta$ ?

• Example:



• Can terminate evaluation of  $\theta'$  once guaranteed to be worse than  $\theta$

## Theorem

Early termination of poor configurations does not change ParamILS's trajectory

- Often yields substantial speedups
- Especially when best configuration is much faster than worst

# Gender-based Genetic Algorithm (GGA)

[Ansotegui, Sellmann & Tierney, CP 2009]

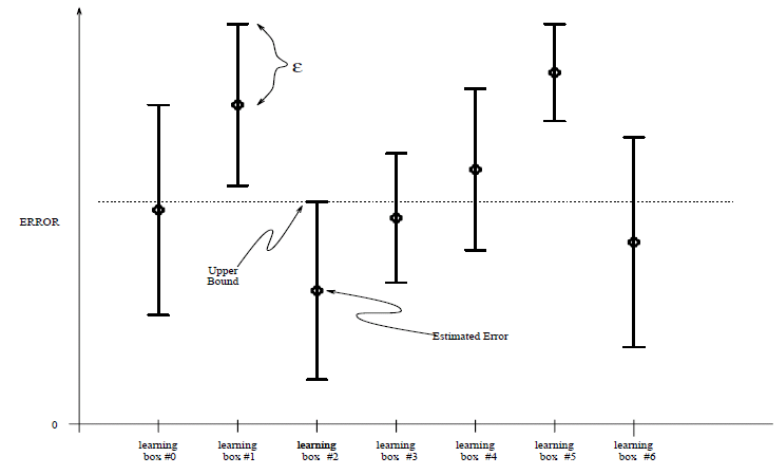
- Genetic algorithm
  - **Genome = parameter configuration**
  - Combine genomes of 2 parents to form an offspring
- **Supports adaptive capping**
  - Evaluate population members in parallel
  - Adaptive capping: can stop when the first k succeed
- Use N instances to evaluate configurations
  - Increase N in each generation
  - **Linear increase from  $N_{\text{start}}$  to  $N_{\text{end}}$** 
    - Not recommended for small budgets
    - User has to specify #generations ahead of time

# F-Race and Iterated F-Race

[Birattari et al, GECCO 2002 and book chapter 2010]

- **F-Race**

- Standard racing framework
- F-test to establish that some configuration is dominated
- Followed by pairwise t tests if F-test succeeds



- **Iterated F-Race**

- Maintain a probability distribution over which configurations are good
- Sample k configurations from that distribution & race them
- Update distributions with the results of the race

- **Well-supported software package in R**

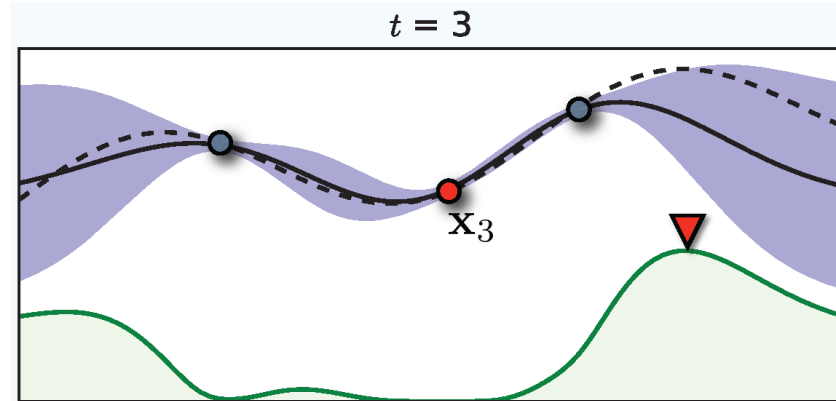
- Does not support adaptive capping → do not use to minimize runtime

# Model-Based Algorithm Configuration

[Hutter, Hoos & Leyton-Brown, LION 2011]

## SMAC: Sequential Model-Based Algorithm Configuration

- Sequential Model-Based Optimization  
& aggressive racing



**repeat**

- construct a model to predict performance
- use that model to select promising configurations
- compare each selected configuration against the best known




**until** time budget exhausted

# SMAC: Aggressive Racing

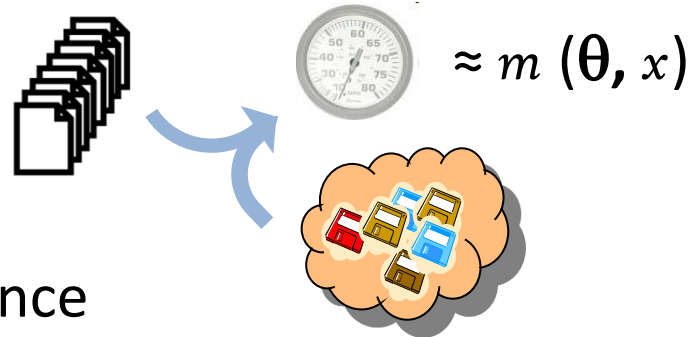
- Similar racing component as FocusedILS
  - More runs for good configurations
  - Increase #runs for incumbent over time
- **Theorem** for discrete configuration spaces:  
As SMAC's overall time budget  $\rightarrow \infty$ ,  
it converges to the optimal configuration

# Powering SMAC: Empirical Performance Models

Given:

- Configuration space  $\Theta = \Theta_1 \times \dots \times \Theta_n$  
- For each problem instance  $i$ :  $\mathbf{x}_i$ , a vector of feature values 
- Observed algorithm runtime data:  $(\theta_1, \mathbf{x}_1, y_1), \dots, (\theta_n, \mathbf{x}_n, y_n)$  

Find: a **mapping**  $m: [\theta, x] \mapsto y$  predicting A's performance



- Rich literature on such performance prediction problems  
[see, e.g, Hutter, Xu, Hoos, Leyton-Brown, AIJ 2014, for an overview]
- Here: use a model  $m$  based on **random forests**

# Regression Trees: Fitting to Data

- In each internal node
- In each leaf: store

param 1	feature 2	param 3	runtime
false	2	red	3.7
false	2.5	blue	20
true	5.5	red	2.1
false	5.5	blue	25
false	5	red	1.2
true	4.5	green	19
true	4	blue	12
true	3.5	green	17

criterion used

$param_3 \in \{red\}$

$param_3 \in \{blue, green\}$

param 1	feature 2	param 3	runtime
false	2	red	3.7
true	5.5	red	2.1
false	5	red	1.2

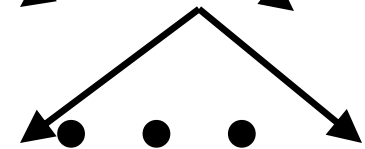
param 1	feature 2	param 3	runtime
false	2.5	blue	20
false	5.5	blue	25
true	4.5	green	19
true	4	blue	12
true	3.5	green	17

$feature_2 \leq 3.5$

$feature_2 > 3.5$

param 1	feature 2	param 3	runtime
false	2	red	3.7

param 1	feature 2	param 3	runtime
true	5.5	red	2.1
false	5	red	1.2

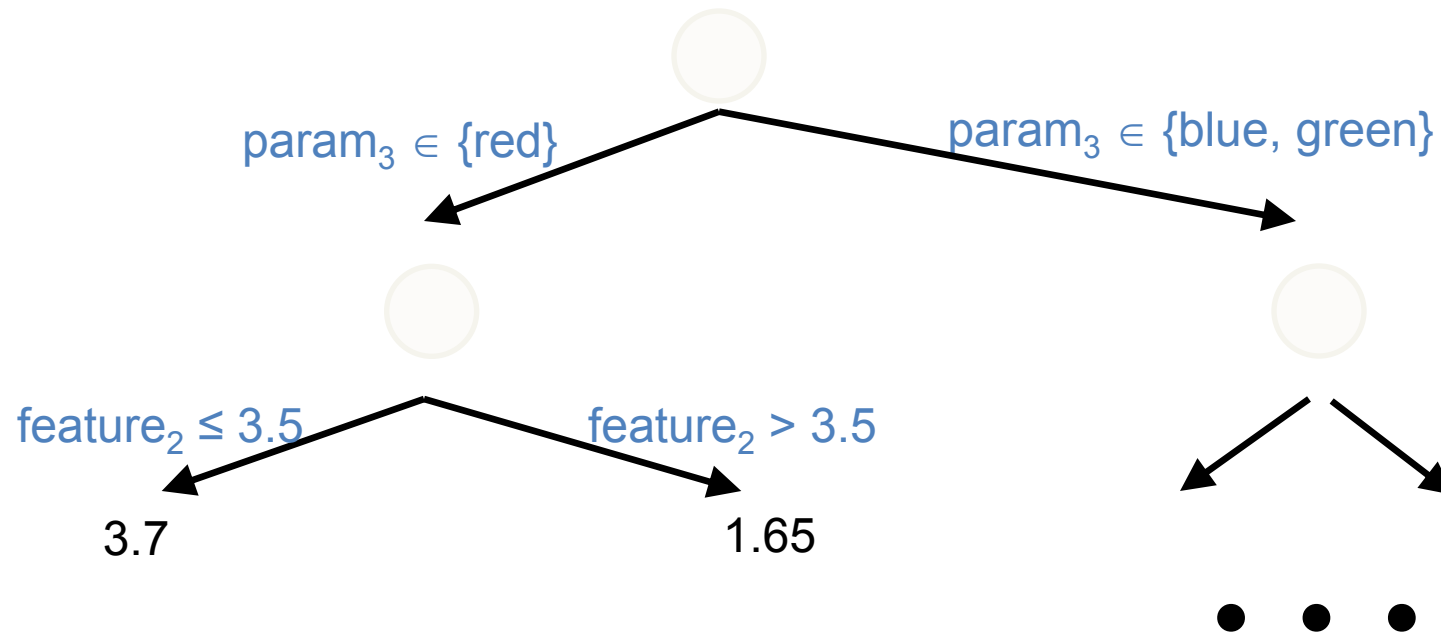




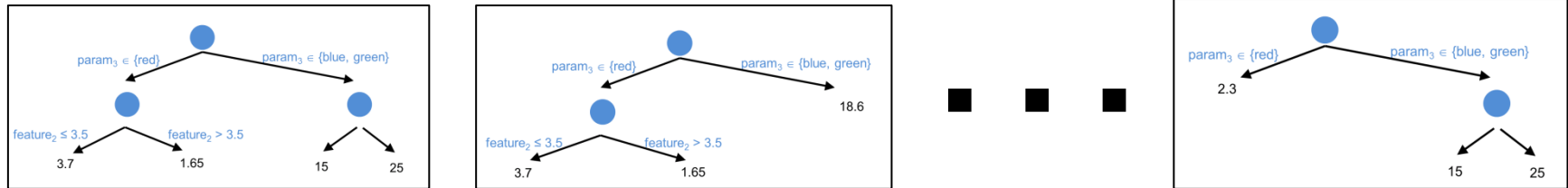
# Regression Trees: Predictions for New Inputs

E.g.  $x_{n+1} = (\text{true}, 4.7, \text{red})$

- Walk down tree, return mean runtime stored in leaf  $\Rightarrow 1.65$



# Random Forests: Sets of Regression Trees



## Training

- Draw  $T$  **bootstrap samples** of the data
- For each bootstrap sample, fit a **randomized** regression tree

## Prediction

- Predict with each of the  $T$  trees
- Return empirical **mean and variance** across these  $T$  predictions

## Complexity for $N$ data points

- Training:  $O(TN \log^2 N)$
- Prediction:  $O(T \log N)$

# Advantages of Random Forests

## Automated selection of important input dimensions

- Continuous, integer, and categorical inputs
- Up to 138 features, 76 parameters
- Can identify important feature and parameter subsets
  - Sometimes 1 feature and 2 parameters are enough

[Hutter, Hoos, Leyton-Brown, LION 2013]

## Robustness

- No need to optimize hyperparameters
- Already good predictions with few training data points

# SMAC: Averaging Across Multiple Instances

- Fit a random forest model  $m : \Theta \times \Pi \rightarrow \mathbb{R}$
- Aggregate over instances by marginalization

$$f(\boldsymbol{\theta}) := \mathbb{E}_{\pi \sim D}[m(\boldsymbol{\theta}, \pi)]$$

- **Intuition: predict for each instance and then average**
- More efficient implementation in random forests

# SMAC: Putting it all Together

Initialize with a single run for the default

**repeat**

1. Learn a RF model from data so far:

$$m : \Theta \times \Pi \rightarrow \mathbb{R}$$

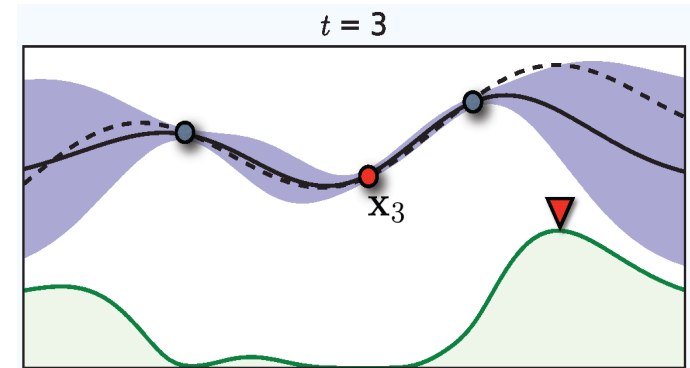
2. Aggregate over instances:

$$f(\theta) := \mathbb{E}_{\pi \sim D}[m(\theta, \pi)]$$

3. Use model  $f$  to select promising configurations

4. Race each selected configuration against the best known

**until** time budget exhausted



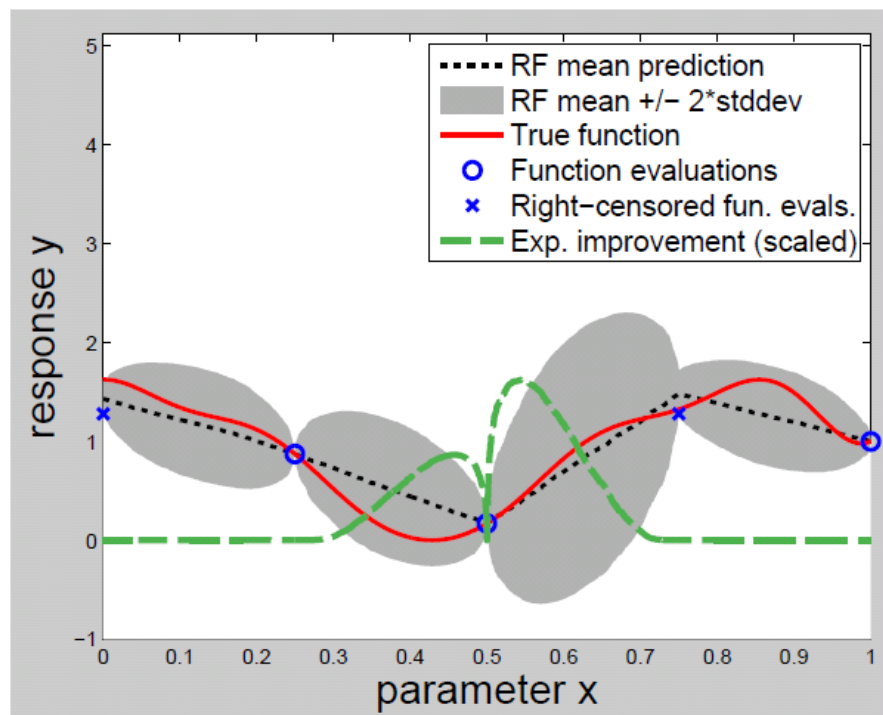
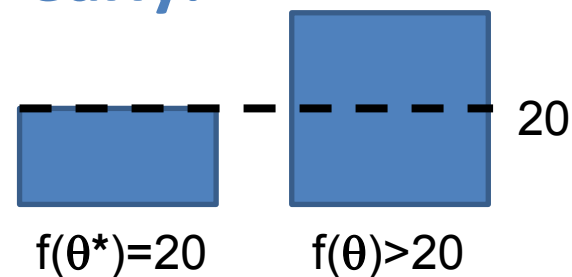
- **Distributed SMAC** [Hutter, Hoos & Leyton-Brown, 2012]
  - Maintain queue of promising configurations
  - Race these against best known on distributed worker cores

# SMAC: Adaptive Capping

[Hutter, Hoos & Leyton-Brown, BayesOpt 2011]

Terminate runs for poor configurations  $\theta$  early:

- Lower bound on runtime  
→ right-censored data point



# Experimental Evaluation

[Hutter, Hoos & Leyton-Brown, LION 2011]

## Compared SMAC vs. ParamILS and GGA

- On 17 SAT and MIP configuration scenarios, same time budget



## SMAC performed best

- Improvements in test performance of configurations returned
  - vs ParamILS: 0.93× – 2.25× (11/17 cases significantly better)
  - vs. GGA: 1.01× – 2.76× (13/17 cases significantly better)

## Wall-clock speedups in distributed SMAC

- Almost perfect with up to 16 parallel workers
- Up to 50-fold with 64 workers
  - Reductions in wall clock time: 5h → 6 min -15 min  
2 days → 40min - 2h

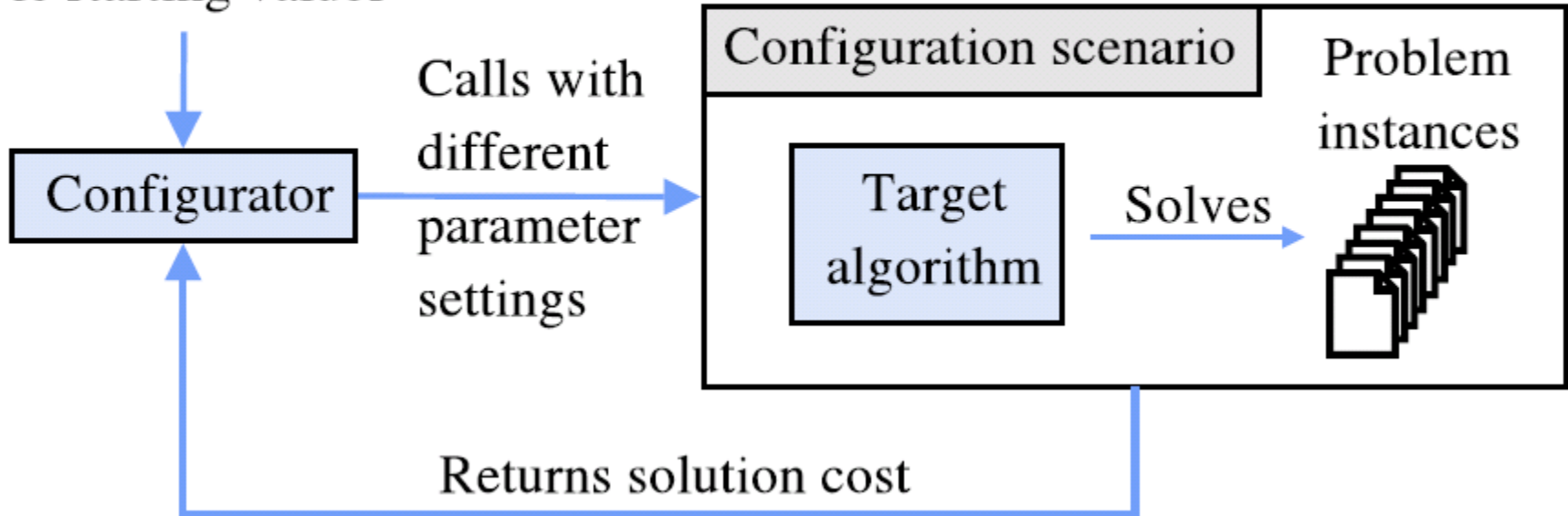
# Overview

- Programming by Optimization (PbO):  
Motivation and Introduction
- Algorithm Configuration
  - Methods (components of algorithm configuration)
  - Systems (that instantiate these components)
  - ➔ Demo & practical issues
    - Case studies
- Portfolio-Based Algorithm Selection
- Software Development Support & Further Directions



# The Algorithm Configuration Process

Parameter domains  
& starting values



What the user has to provide

Parameter space declaration file

```
preproc {none, simple, expensive} [simple]
alpha [1,5] [2]
beta [0.1,1] [0.5]
```

Wrapper for command line call

```
./wrapper -inst X -timeout 30
-preproc none -alpha 3 -beta 0.7
→ e.g. "successful after 3.4 seconds"
```

# Example: Running SMAC

```
wget http://www.cs.ubc.ca/labs/beta/Projects/SMAC/smac-v2.06.00-master-615.tar.gz
```

```
tar xzvf smac-v2.06.00-master-615.tar.gz
```

```
cd smac-v2.06.00-master-615
```

```
./smac
```

For a usage screen

```
./smac --seed 0 --scenarioFile example_scenarios/spear/spear-scenario.txt
```

Scenario file holds:

- Location of parameter file, wrapper & instances
- Objective function (here: minimize avg. runtime)
- Configuration budget (here: 30s)
- Maximal captime per target run (here: 5s)

# Output of a SMAC run

[...]

[INFO ] \*\*\*\*\*Runtime Statistics\*\*\*\*\*

Incumbent ID: 12 (0x22BB8)

Number of Runs for Incumbent: 43

Number of Instances for Incumbent: 5

Number of Configurations Run: 42

Performance of the Incumbent: 0.012555555555555556

Configuration Time Budget used: 30.589647351000067 s (101%)

Sum of Target Algorithm Execution Times (treating minimum value as 0.1): 24.70000s

CPU time of Configurator: 5.889042742 s

[INFO ] \*\*\*\*\*

[INFO ] Total Objective of Final Incumbent 12 (0x22BB8) on training set:  
0.012555555555555556; on test set: 0.014499999999999999

[INFO ] Sample Call for Final Incumbent 12 (0x22BB8)

```
cd /ubc/cs/home/h/hutter/tmp/smac-v2.06.00-master-615/example_scenarios/spear; ruby spear_wrapper.rb
instances/qcplin2006.10408.cnf 0 5.0 2147483647 3282095 -sp-update-dec-queue '0' -sp-rand-var-dec-scaling
'0.3528466348383826' -sp-clause-decay '1.713857938112484' -sp-variable-decay '1.461422623379798' -sp-orig-
clause-sort-heur '7' -sp-rand-phase-dec-freq '0.05' -sp-clause-del-heur '0' -sp-learned-clauses-inc
'1.452683835620401' -sp-restart-inc '1.6481745669620091' -sp-resolution '0' -sp-clause-activity-inc
'0.7121640599232154' -sp-learned-clause-sort-heur '12' -sp-var-activity-inc '0.9358501810374242' -sp-rand-var-dec-
freq '0.0001' -sp-use-pure-literal-rule '1' -sp-learned-size-factor '0.27995062371127827' -sp-var-dec-heur '16' -sp-
phase-dec-heur '6' -sp-rand-phase-scaling '1.0424648235977578' -sp-first-restart '31'
```

# Decision #1: Configuration Budget & Captime

- **Configuration budget**
  - Dictated by your resources & needs
    - E.g., start configuration before leaving work on Friday
  - The longer the better (but diminishing returns)
    - Rough rule of thumb: typically at least enough time for 1000 target runs
    - But have also achieved good results with 50 target runs in some cases
- **Maximal captime per target run**
  - Dictated by your needs (typical instance hardness, etc)
  - Too high: slow progress
  - Too low: possible overtuning to easy instances
  - For SAT etc, often use 300 CPU seconds

# Decision #2: Choosing the Training Instances

- **Representative instances, moderately hard**
  - Too hard: won't solve many instances, no traction
  - Too easy: will results generalize to harder instances?
  - Rule of thumb: mix of hardness ranges
    - Roughly 75% instances solvable by default in maximal captime
- **Enough instances**
  - The more training instances the better
  - Very homogeneous instance sets: 50 instances might suffice
  - Preferably  $\geq 300$  instances, better even  $\geq 1000$  instances

# Decision #2: Choosing the Training Instances

- **Split instance set into training and test sets**
  - Configure on the training instances → configuration  $\theta^*$
  - Run (only)  $\theta^*$  on the test instances
    - Unbiased estimate of performance

**Pitfall: configuring on your test instances**

That's from the dark ages

**Fine practice: do multiple configuration runs  
and pick the  $\theta^*$  with best training performance**

Not (!! ) the best on the test set

# Decision #2: Choosing the Training Instances

- **Works much better on homogeneous benchmarks**
  - Instances that have something in common
    - E.g., come from the same problem domain
    - E.g., use the same encoding
  - One configuration likely to perform well on all instances

## **Pitfall: configuration on too heterogeneous sets**

There often is no single great overall configuration  
(but see algorithm selection etc, later in the tutorial)

# Decision #3: How Many Parameters to Expose?

- Suggestion: all parameters you don't know to be useless
  - More parameters → larger gains possible
  - More parameters → harder problem
  - Max. #parameters tackled so far: 768  
[Thornton, Hutter, Hoos & Leyton-Brown, KDD'13]
    - With more time you can search a larger space

**Pitfall: including parameters that change the problem**

E.g., optimality threshold in MIP solving

E.g., how much memory to allow the target algorithm



# Decision #4: How to Wrap the Target Algorithm

- Do not trust any target algorithm
  - Will it terminate in the time you specify?
  - Will it correctly report its time?
  - Will it never use more memory than specified?
  - Will it be correct with all parameter settings?

**Good practice: wrap target runs with tool controlling time and memory (e.g., runsolver [Roussel et al, '11])**

**Good practice: verify correctness of target runs**

Detect crashes & penalize them

**Pitfall: blindly minimizing target algorithm runtime**

Typically, you will minimize the time to crash

# Overview

- Programming by Optimization (PbO):  
Motivation and Introduction
- Algorithm Configuration
  - Methods (components of algorithm configuration)
  - Systems (that instantiate these components)
  - Demo & practical issues
- ➔ Case studies
- Portfolio-Based Algorithm Selection
- Software Development Support & Further Directions

# Applications of Algorithm Configuration

## Helped win Competitions

SAT: since 2009

ASP: since 2009

IPC: since 2011

Time-tabling: 2007

SMT: 2007

## Other Academic Applications

Mixed integer programming (MIP)

TSP & Quadratic Assignment Problem

Game Theory: Kidney Exchange

Linear algebra subroutines

Improving Java Garbage Collection

ML Hyperparameter Optimization

Deep learning

FCC spectrum auction

[Auction]omics

IBM



Mixed integer programming

Analytics & Optimization

ORTEC  
OPTIMIZE YOUR WORLD

Social gaming

zynga

Scheduling and Resource Allocation

actenum

# Back to the Spear Example

[Hutter, Babic, Hu & Hoos, FMCAD 2007]

Spear [Babic, 2007]

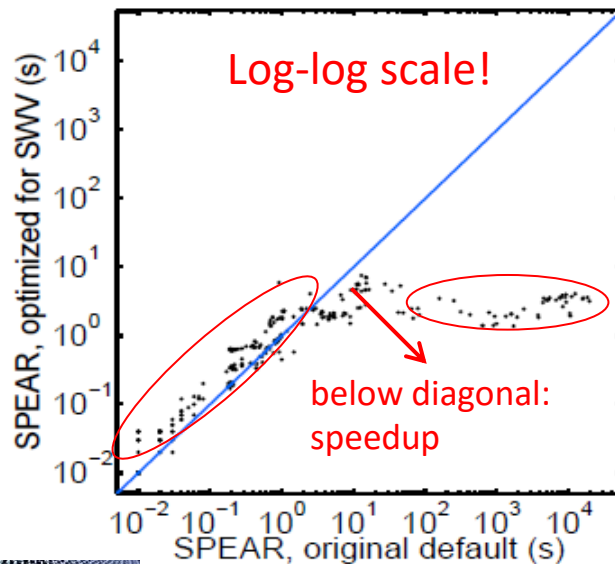
- 26 parameters
- $8.34 \times 10^{17}$  configurations

Ran ParamLS, 2 to 3 days  $\times$  10 machines

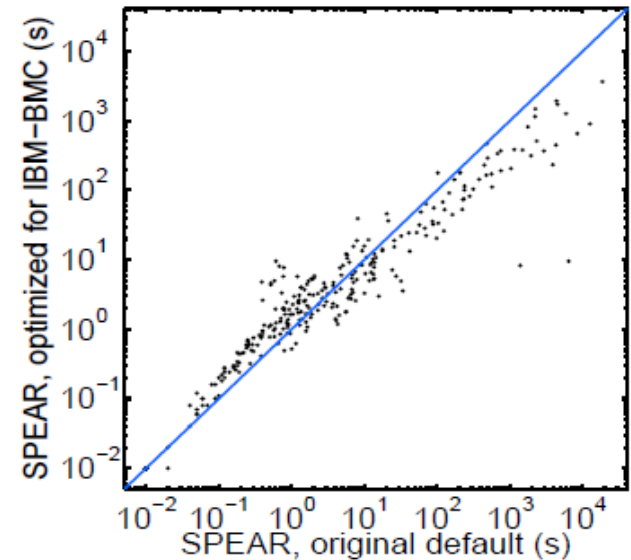
- On a training set from each of 2 distributions

Compared to default (1 week of manual tuning)

- On a disjoint test set from each distribution



500-fold speedup  $\Rightarrow$  won QF\_BV category in 2007 SMT competition



4.5-fold speedup

# Other Examples of PbO for SAT

- SATenstein [KhudaBukhsh, Xu, Hoos & Leyton-Brown, IJCAI 2009]
  - Combined ingredients from existing solvers
  - 54 parameters, over  $10^{12}$  configurations
  - Speedup factors: 1.6x to 218x
- Captain Jack [Tompkins & Hoos, SAT 2011]
  - Explored a completely new design space
  - 58 parameters, over  $10^{50}$  configurations
  - After configuration: best known solver for 3sat10k and IL50k

# Configurable SAT Solver Competition (CSSC)

[Hutter, Balint, Bayless, Hoos & Leyton-Brown 2013]

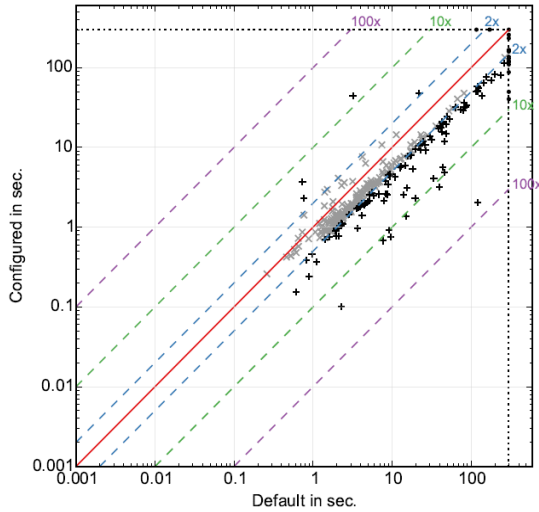
- Annual SAT competition
  - Scores SAT solvers by their performance across instances
  - Medals for best average performance with solver defaults
    - Misleading results: implicitly highlights solvers with good defaults
- CSSC 2013 & 2014
  - Better reflects an application setting: homogeneous instances
    - can automatically optimize parameters
  - Medals for best performance **after configuration**



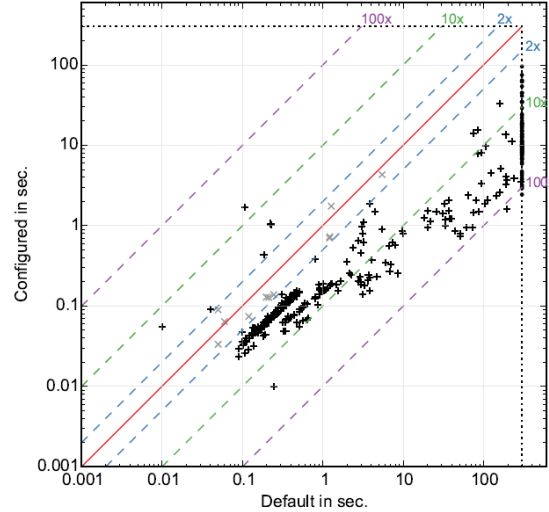
# CSSC Result #1

[Hutter, Lindauer, Balint, Bayless, Hoos & Leyton-Brown 2014]

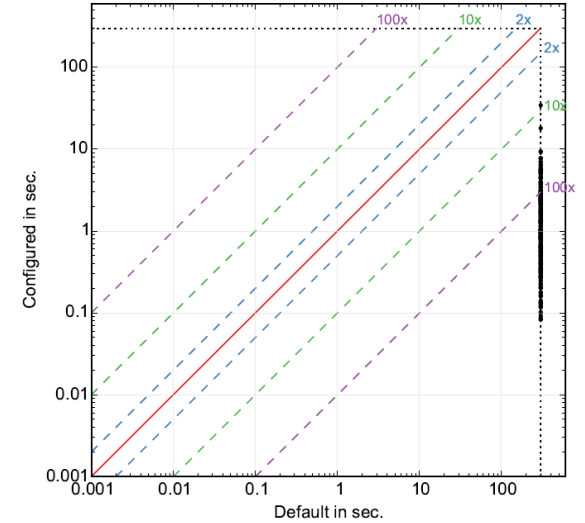
- Solver performance often improved a lot:



Lingeling on CircuitFuzz:  
Timeouts: 119 → 107



Clasp on n-queens:  
Timeouts: 211 → 102



probSAT on unif rnd 5-SAT:  
Timeouts: 250 → 0

# CSSC Result #2

[Hutter, Lindauer, Balint, Bayless, Hoos & Leyton-Brown 2014]

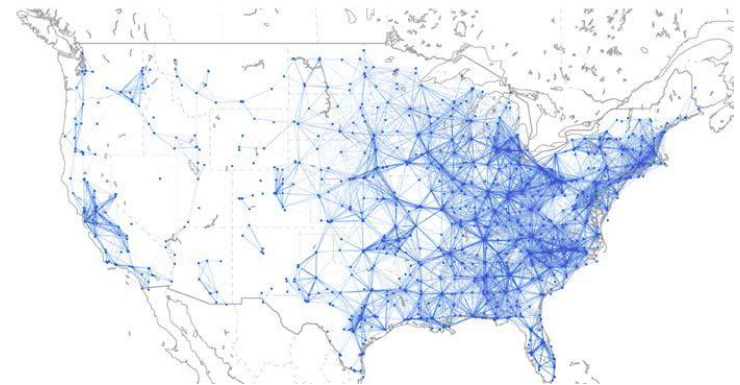
- Automated configuration changed algorithm rankings
  - Example: random SAT+UNSAT category in 2013

Solver	CSSC ranking	Default ranking
Clasp	1	6
Lingeling	2	4
Riss3g	3	5
Solver43	4	2
Simpsat	5	1
Sat4j	6	3
For1-nodrup	7	7
gNovelty+GCwa	8	8
gNovelty+Gca	9	9
gNovelty+PCL	10	10



# Real-World Application: FCC Spectrum Auction

- Wireless frequency spectra: demand increases
  - US Federal Communications Commission (FCC) is currently holding an auction
  - **Expected net revenue: \$10 billion to \$40 billion**
- **Key computational problem:**  
feasibility testing based on interference constraints
  - A hard **graph colouring problem**
  - 2991 stations (nodes) & 2.7 million interference constraints
  - Need to solve many different instances
  - More instances solved: higher revenue
- **Best solution: based on SAT solving & configuration with SMAC**
  - Improved #instances solved from 73% to 99.6%  
[Frechette, Newman & Leyton-Brown, AAAI'16]



# Configuration of a Commercial MIP solver

[Hutter, Hoos & Leyton-Brown, CPAIOR 2010]

## Mixed Integer Programming (MIP)

$$\begin{aligned} \min \quad & c^T x \\ \text{s. t.} \quad & Ax \leq b \\ & x_i \in \mathbb{Z} \text{ for } i \in I \end{aligned}$$

## Commercial MIP solver: IBM ILOG CPLEX

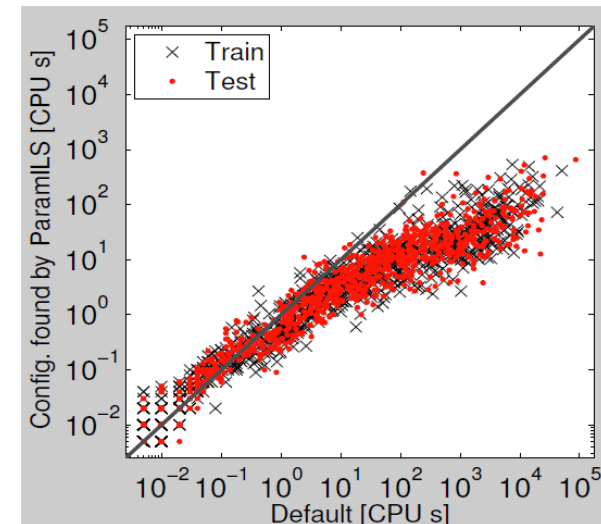
- Leading solver for 15 years
- Licensed by over 1 000 universities and 1 300 corporations
- 76 parameters,  $10^{47}$  configurations

## Minimizing runtime to optimal solution

- Speedup factor:  $2\times$  to  $50\times$
- Later work: speedups up to  $10,000\times$

## Minimizing optimality gap reached

- Gap reduction factor:  $1.3\times$  to  $8.6\times$



# Comparison to CPLEX Tuning Tool

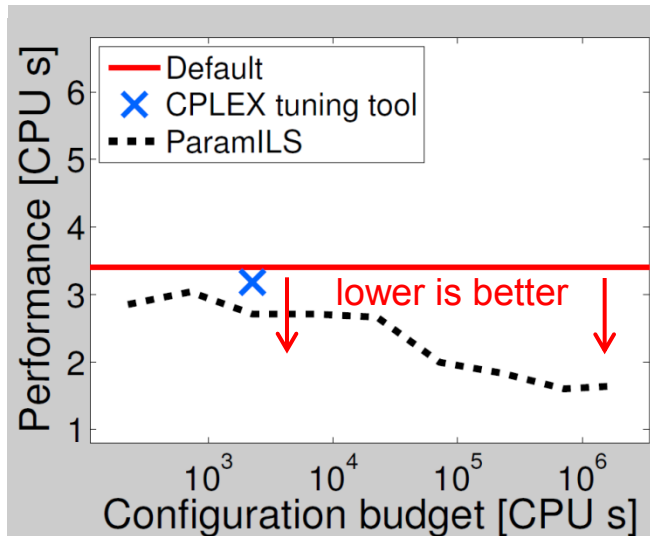
[Hutter, Hoos & Leyton-Brown, CPAIOR 2010]

## CPLEX tuning tool

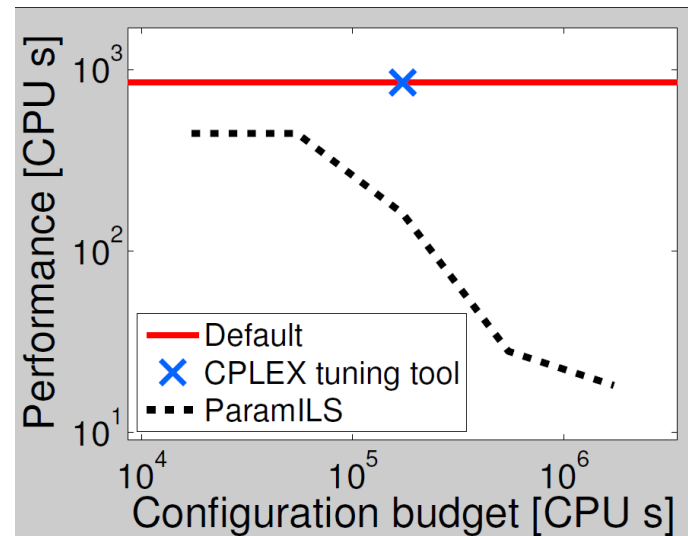
- Introduced in version 11 (late 2007, after ParamILS)
- Evaluates predefined good configurations, returns best one
- Required runtime varies (from < 1h to weeks)

## ParamILS: anytime algorithm

- At each time step, keeps track of its incumbent



2-fold speedup  
(our worst result)



50-fold speedup  
(our best result)

# Configuration of Machine Learning Algorithms

- Machine Learning has celebrated **substantial successes**
- But it requires **human machine learning experts** to
  - Preprocess the data
  - Perform feature selection
  - Select a model family
  - Optimize hyperparameters
  - ...
- **AutoML**: taking the human expert out of the inner loop
  - Yearly AutoML workshops at ICML since 2014
  - PbO applied to machine learning

# Auto-WEKA

[Thornton, Hutter, Hoos & Leyton-Brown, KDD'13]

**WEKA** [Witten et al, 1999-current]

- most widely used off-the-shelf machine learning package
- over 20,000 citations on Google scholar

Java implementation of a **broad range of methods**

- 27 base classifiers (with up to 10 parameters each)
- 10 meta-methods
- 2 ensemble methods
- 3 feature search methods & 8 feature evaluators

Different methods work best on different data sets

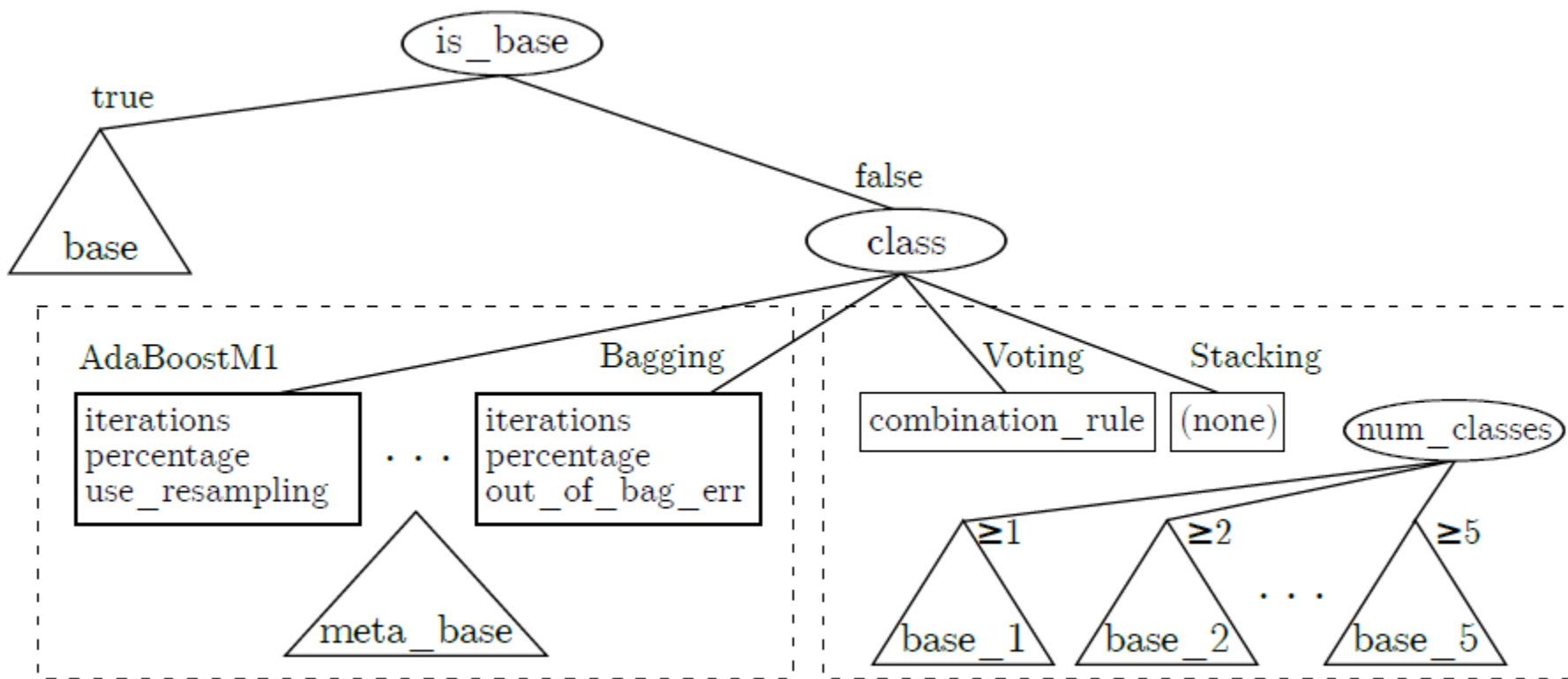
- Want a **true off-the-shelf solution**: **Learn**

# WEKA's configuration space

## Base classifiers

- 27 choices, each with up to 10 subparameters
- Coarse discretization: about  **$10^8$  instantiations**

## Hierarchical structure on top of base classifiers

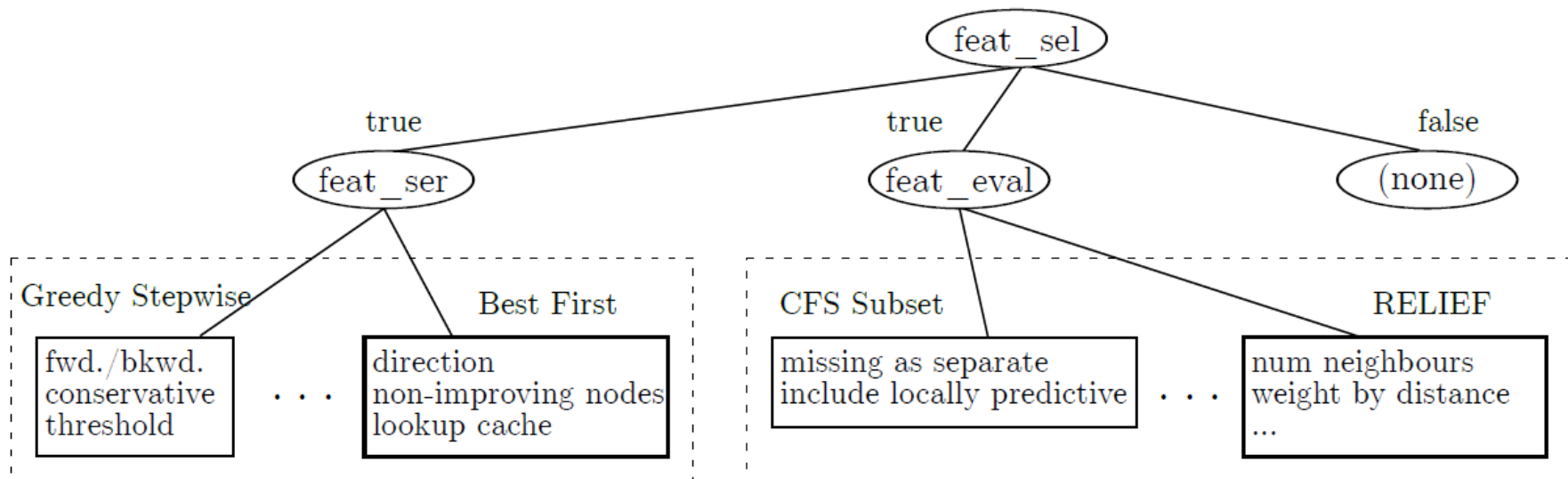


# WEKA's configuration space (cont'd)

## Feature selection

- **Search** method: which feature subsets to evaluate
- **Evaluation** method: how to evaluate feature subsets in search
- Both methods have **subparameters** → about  **$10^7$  instantiations**

In total: **768 parameters,  $10^{47}$  configurations**



# Auto-WEKA: Results

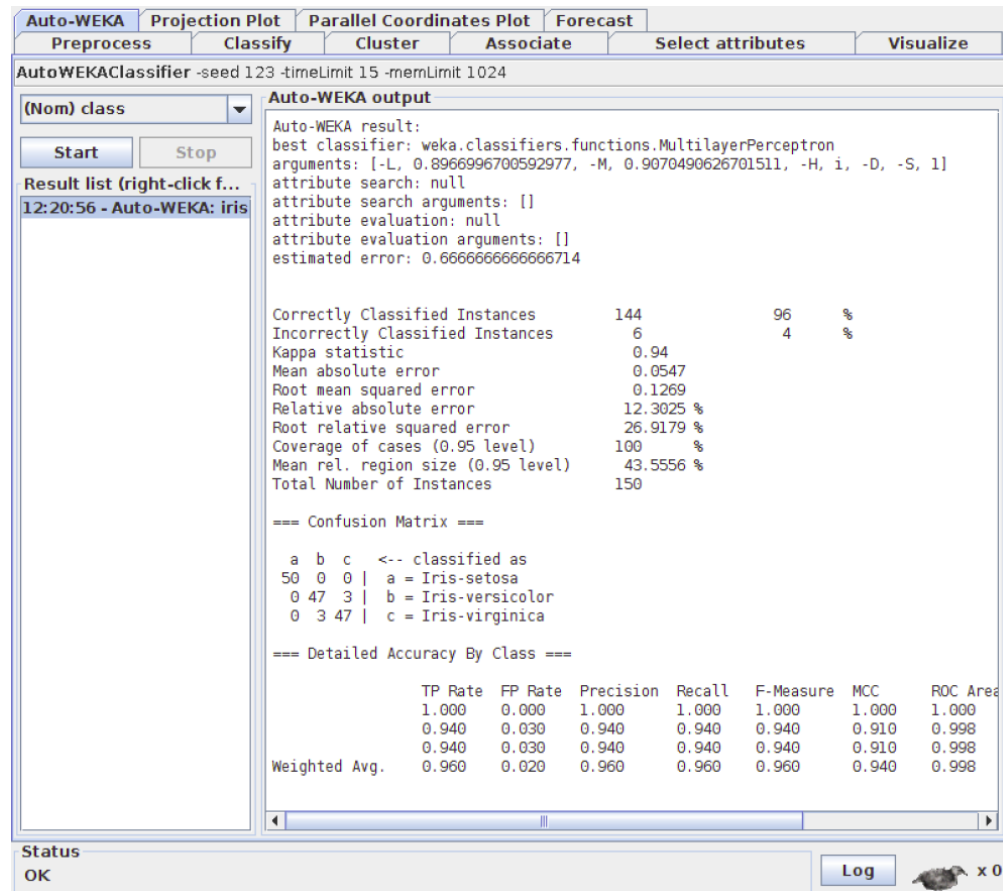
- **Auto-WEKA performed better than best base classifier**
  - Even when “best base classifier” determined by an oracle
  - In 6/21 datasets more than 10% reductions in relative error
- Comparison to full grid search
  - Union of grids over parameters of all 27 base classifiers
  - **Auto-WEKA was 100 times faster**
  - **Auto-WEKA had better test performance in 15/21 cases**
- Auto-WEKA based on SMAC vs. TPE [Bergstra et al, NIPS'11]
  - SMAC yielded **better CV performance in 19/21 cases**
  - SMAC yielded **better test performance in 14/21 cases**
  - Differences usually small, in 3 cases substantial (SMAC better)



# Auto-WEKA as a WEKA plugin

- Command line example:  
java -cp autoweka.jar weka.classifiers.meta.AutoWEKAClassifier  
-t iris.arff -timeLimit 5 -no-cv

- GUI example:



The screenshot shows the WEKA GUI with the Auto-WEKA classifier selected. The 'Auto-WEKA output' window displays the following results:

```
Auto-WEKA result:
best classifier: weka.classifiers.functions.MultilayerPerceptron
arguments: [-L, 0.8966996700592977, -M, 0.9070490626701511, -H, i, -D, -S, 1]
attribute search: null
attribute search arguments: []
attribute evaluation: null
attribute evaluation arguments: []
estimated error: 0.666666666666714
```

Correctly Classified Instances	144	96	%
Incorrectly Classified Instances	6	4	%
Kappa statistic	0.94		
Mean absolute error	0.0547		
Root mean squared error	0.1269		
Relative absolute error	12.3025 %		
Root relative squared error	26.9179 %		
Coverage of cases (0.95 level)	100 %		
Mean rel. region size (0.95 level)	43.5556 %		
Total Number of Instances	150		

=== Confusion Matrix ===

a	b	c	<- classified as
50	0	0	a = Iris-setosa
0	47	3	b = Iris-versicolor
0	3	47	c = Iris-virginica

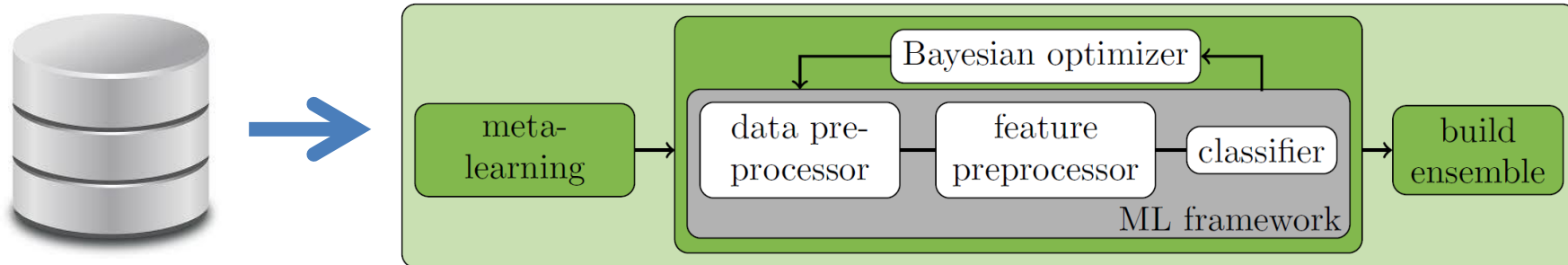
=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area
	1.000	0.000	1.000	1.000	1.000	1.000	1.000
	0.940	0.030	0.940	0.940	0.940	0.910	0.998
	0.940	0.030	0.940	0.940	0.940	0.910	0.998
Weighted Avg.	0.960	0.020	0.960	0.960	0.960	0.940	0.998

Status: OK

# Auto-sklearn

[Feurer, Klein, Eggenberger, Springenberg, Blum, Hutter; NIPS 2015]



- Followed and extended Auto-WEKA's AutoML approach
- Scikit-learn optimized by SMAC, plus
  - **Meta-learning** to warmstart Bayesian optimization
  - Automated **posthoc ensemble construction** to combine the models Bayesian optimization evaluated

# Auto-sklearn's configuration space

- Scikit-learn [Pedregosa et al, 2011-current] instead of WEKA

- 15 classifiers,  
(with a total of 59 hyperparameters)
- 13 feature preprocessors  
(42 hyperparams)
- 4 data preprocessors  
(5 hyperparams)

name	# $\lambda$
AdaBoost (AB)	4
Bernoulli naïve Bayes	2
decision tree (DT)	4
extreml. rand. trees	5
Gaussian naïve Bayes	-
gradient boosting (GB)	6
kNN	3
LDA	4
linear SVM	4
kernel SVM	7
multinomial naïve Bayes	2
passive aggressive	3
QDA	2
random forest (RF)	5
Linear Class. (SGD)	10

name	# $\lambda$
extreml. rand. trees prepr.	5
fast ICA	4
feature agglomeration	4
kernel PCA	5
rand. kitchen sinks	2
linear SVM prepr.	3
no preprocessing	-
nystroem sampler	5
PCA	2
polynomial	3
random trees embed.	4
select percentile	2
select rates	3
one-hot encoding	2
imputation	1
balancing	1
rescaling	1

- 110 hyperparameters  
vs. 768 in Auto-WEKA

# Auto-sklearn: Ready for Prime Time

- Winning approach in the 14-month AutoML challenge
  - **Best-performing approach in auto-track and human track**
  - Won both tracks in both final phases
  - **Vs. 150 teams of human experts**

- Trivial to use:

```
import autosklearn.classification as cls
automl=cls.AutoSklearnClassifier(include_
    estimators = ['lda', 'decision_tree'])
automl.fit(X_train, y_train)
y_hat = automl.predict(X_test)
```

- Available online:

<https://github.com/automl/auto-sklearn>

# Auto-sklearn Application (Sneak Peak)

- Collaboration with Andreas Eitel & Wolfram Burgard
- Binary classification task for object placement:  
**will the object fall over?**



Video credit: Andreas Eitel

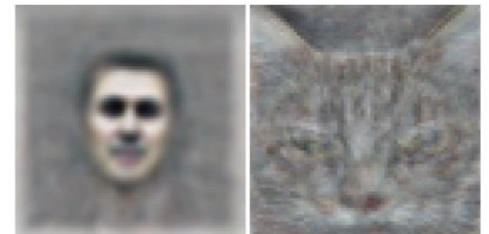
- Dataset
  - Based on BigBIRD and YCB Object and Model Set
  - 30000 data points
  - 50 features -- manually defined [BSc thesis, Hauff 2015]
- Performance
  - **Caffe deep learning framework: 2% error rate**
  - **Auto-sklearn: 0.6% error rate** (within 30 minutes)

# PbO for Deep Learning

- What is deep learning?
  - **Neural networks with many layers**
- Why is there so much excitement about it?
  - **Dramatically improved the state-of-the-art in many areas**, e.g.,
    - Speech recognition
    - Image recognition
  - Automatic learning of representations
    - **no more manual feature engineering**
- What changed?
  - Larger datasets
  - Better regularization methods, e.g., dropout [Hinton et al, 2012]
  - Fast GPU implementations [Krizhevsky et al, 2012]



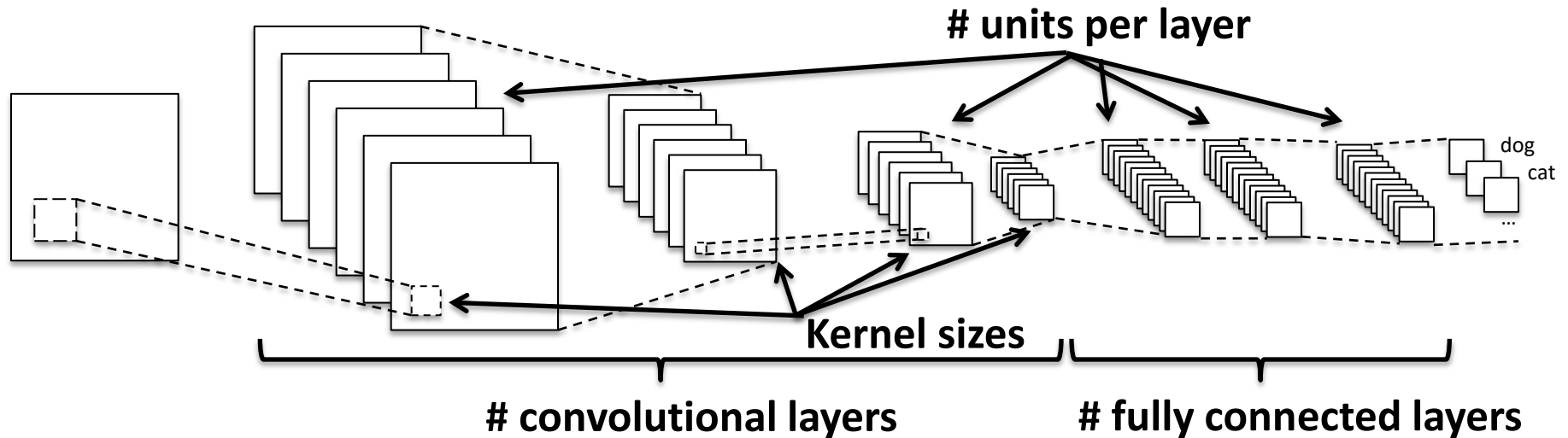
Source: Krizhevsky et al, 2012



Source: Le et al, 2012

# Deep Learning is Sensitive to Many Choices

## Choice of network architecture ...



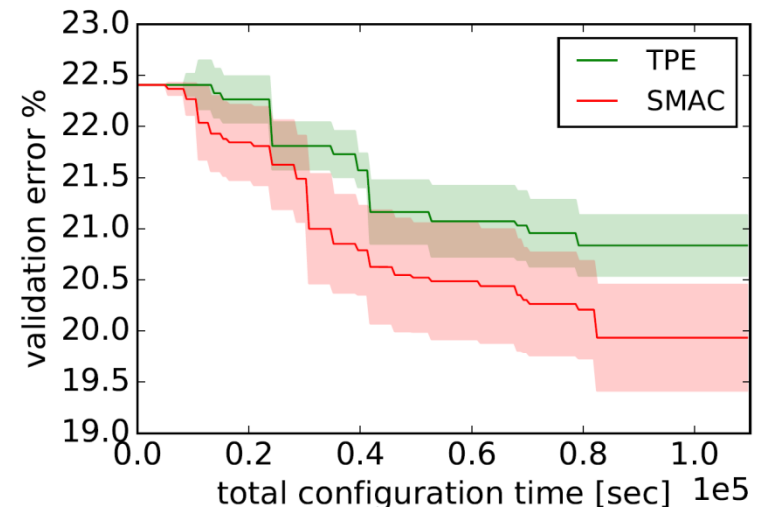
## ... and 20-30 other numerical choices

Learning rate schedule (initialization, decay, adaptation), momentum, batch normalization, batch size, #epochs, dropout rates, weight initializations, weight decay, ...

# Auto-Net for Computer Vision Data

[Domhan, Springenberg, Hutter, IJCAI 2015]

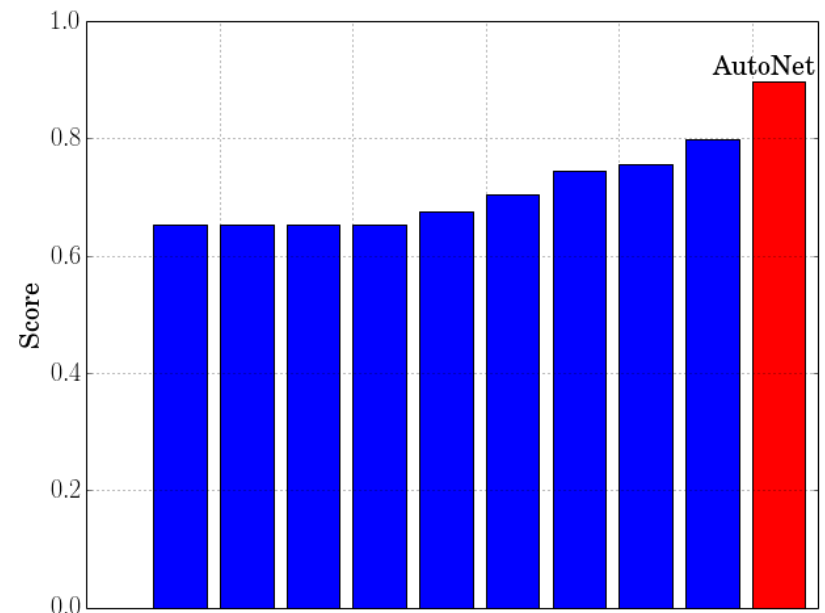
- Application: object recognition
- Parameterized the Caffe framework [Jia, 2013]
  - Convolutional neural network
  - 9 network hyperparameters
  - 12 hyperparameters per layer, up to 6 layers
  - In total **81 hyperparameters**
- Results for CIFAR-10
  - **New best result for CIFAR-10 without data augmentation**
  - SMAC outperformed TPE (only other applicable hyperparameter optimizer)





# Auto-Net in the AutoML Challenge

- **Clearly won a dataset of AutoML challenge**
  - 54491 data points, 5000 features, 18 classes
- Unstructured data → fully-connected network
  - Up to 5 layers (with 3 layer hyperparameters each)
  - 14 network hyperparameters, in total **29 hyperparameters**
  - Optimized for 18h on 5GPUs
- Result (on private test set)
  - AUC 90%
  - All other (manual) approaches < 80%

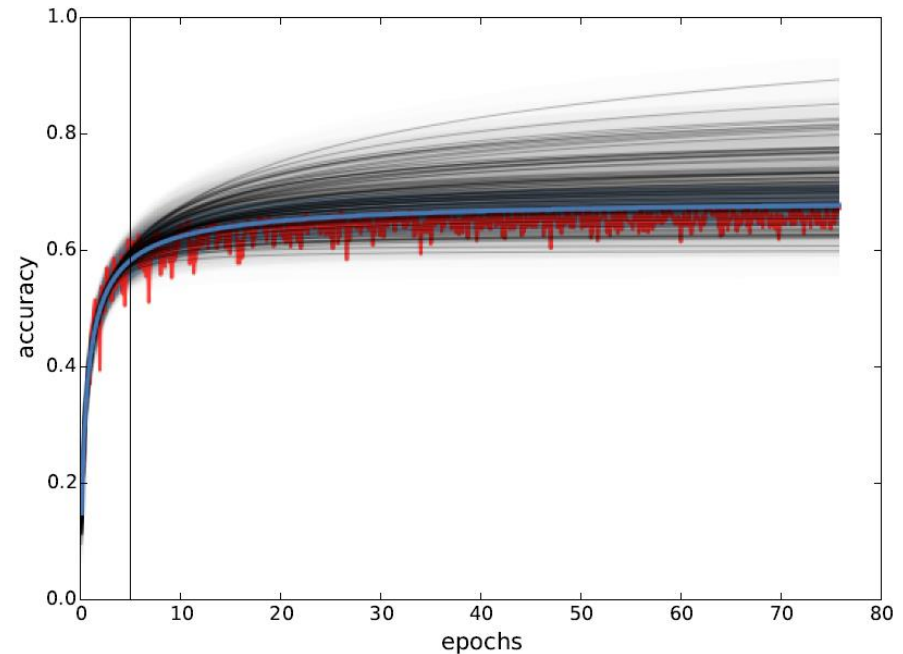


# Speedups by Prediction of Learning Curves

[Domhan, Springenberg & Hutter, IJCAI 2015]

- Humans can look inside the blackbox

- They **can predict the final performance of a target algorithm run early**
- After a few epochs of stochastic gradient descent
- **Stop if not promising**



- We automated that heuristic
  - Fitted linear combination of 22 parametric models
  - MCMC to preserve uncertainty over model parameters
  - Stopped poor runs early: overall 2-fold speedup

# Speedups by Reasoning over Data Subsets

[Klein, Bartels, Falkner, Hennig, Hutter, arXiv 2016]

- **Problem:** training is very slow for large datasets

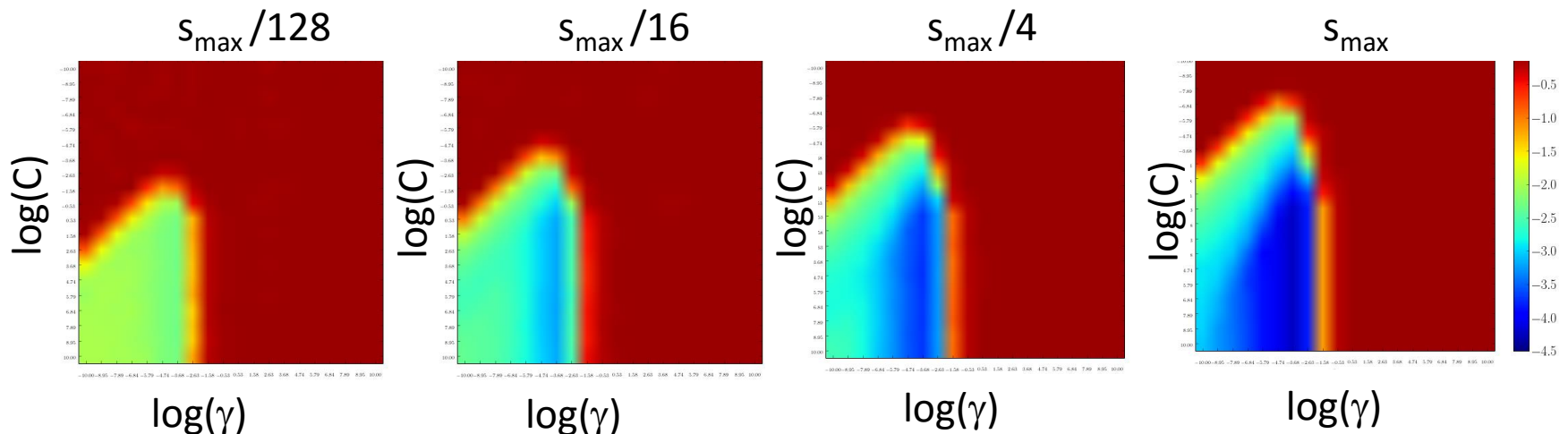
- **Solution approach:**

scaling up from subsets of the data



- **Example: SVM**

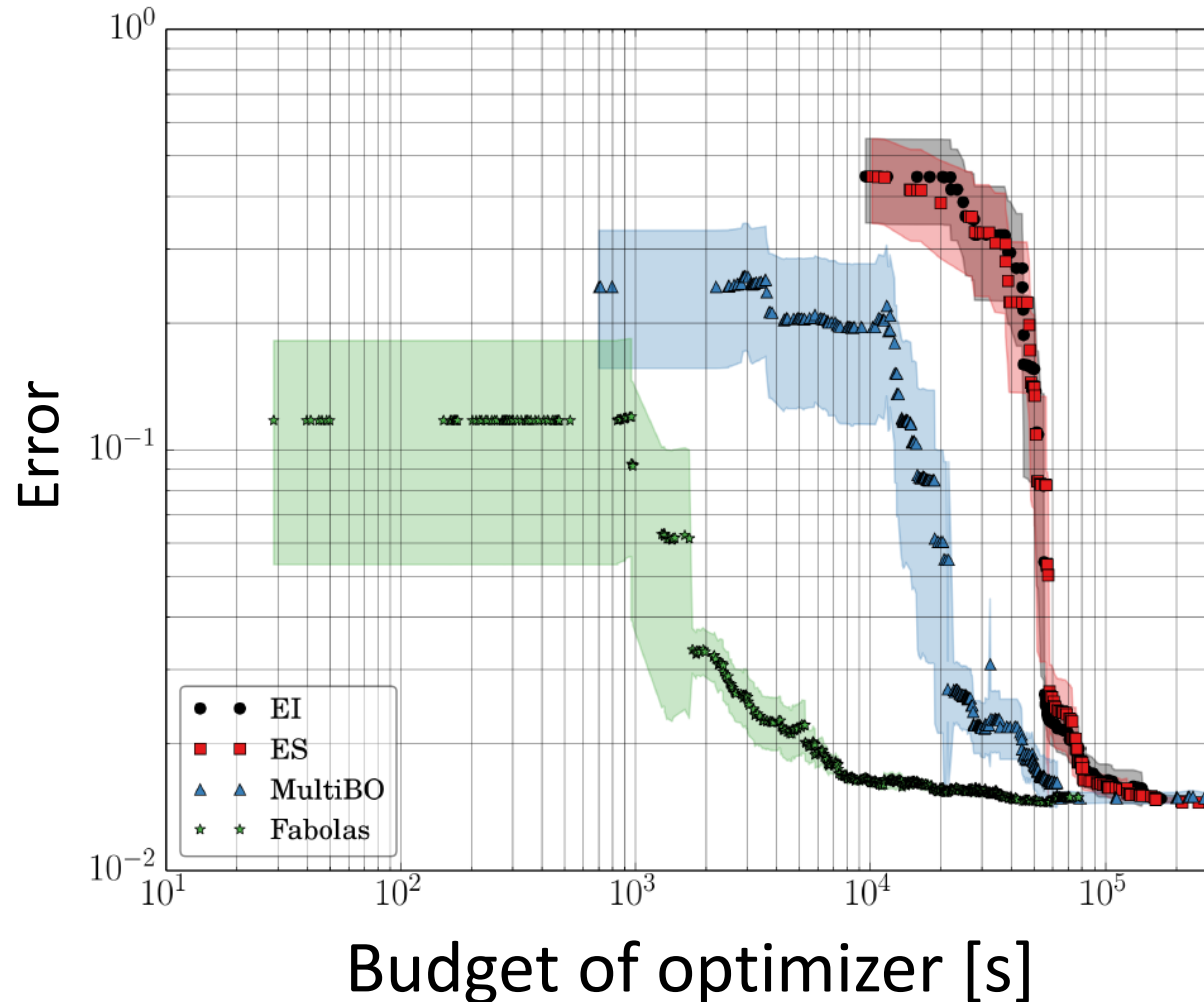
- Computational cost grows quadratically in dataset size  $s$
- **Error shrinks smoothly with  $s$**



# Speedups by Reasoning over Data Subsets

[Klein, Bartels, Falkner, Hennig, Hutter, arXiv 2016]

- **10-100x speedup** for optimizing SVM hyperparameters
- **5-10x speedup** for convolutional neural networks



# Summary of Algorithm Configuration

- Algorithm configuration
  - Methods (components of algorithm configuration)
  - Systems (that instantiate these components)
  - Demo & practical issues
  - Case studies
- Useful abstraction with many (!) applications
- Often better performance than human experts
- Much less human expert time required

Links to all our code: <http://ml4aad.org>

# Overview

- Programming by Optimization (PbO):  
Motivation and Introduction
- Algorithm Configuration

## Portfolio-Based Algorithm Selection

- **SATzilla: a framework for algorithm selection**
- **Hydra: automatic portfolio construction**
- Software Development Tools and Further Directions

# Motivation: no single great configuration exists

- Heterogeneous instance distributions
  - Even the best overall configuration is not great. E.g.:

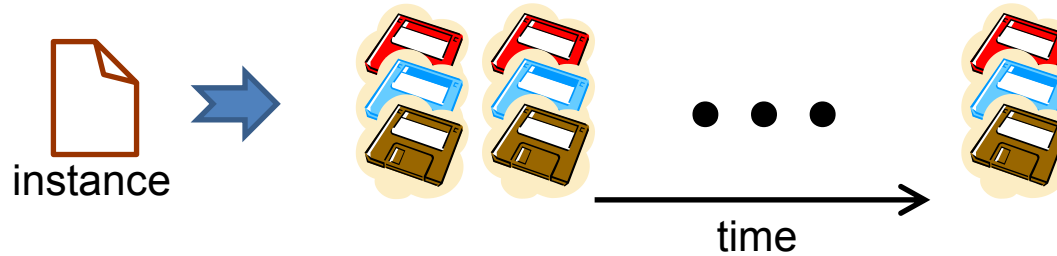
Configuration	Instance type 1	Instance type 2
#1	1s	1000s
#2	1000s	1s
#3	100s	100s

- Likewise, there is no single best solver
  - For example SAT solving:  
different solvers win different categories
  - **Virtual best solver (VBS) much better than single best solver (SBS)**

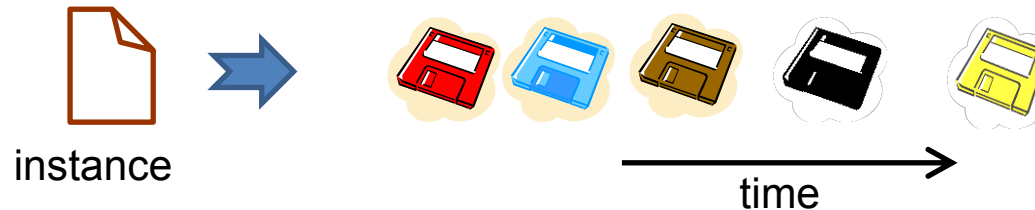
# Algorithm portfolios

Exploiting complementary strengths of different algorithms

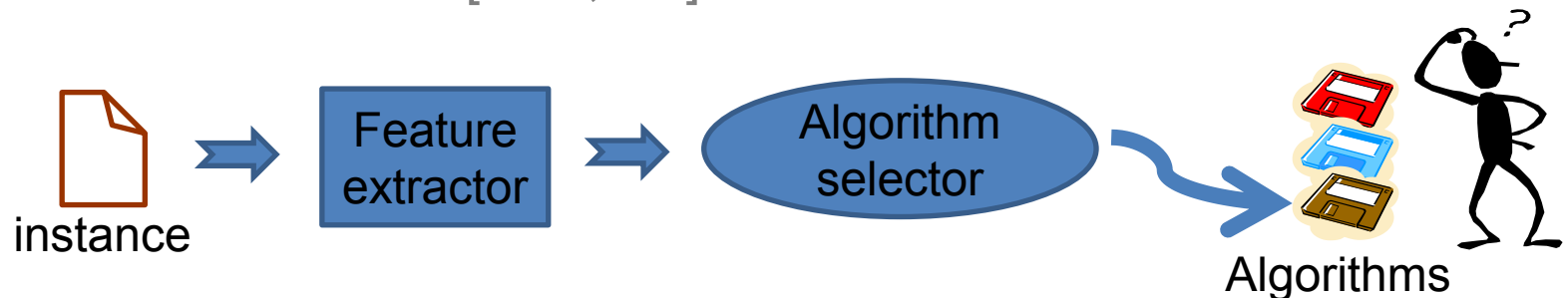
Parallel portfolios [Huberman et al, '97]



Algorithm schedules [Sayag et al, '06]



Algorithm selection [Rice, '76]





# Portfolios have been successful in many areas

*\*Algorithm Selection †Sequential Execution ‡Parallel Execution*

- **Satisfiability:**

- SATzilla<sup>\*†</sup> [various coauthors, cited in the following slides; 2003—ongoing]
- 3S<sup>\*†</sup> [Sellmann, 2011]
- ppfolio<sup>‡</sup> [Roussel, 2011]
- claspfolio<sup>\*</sup> [Gebser, Kaminski, Kaufmann, Schaub, Schneider, Ziller, 2011]
- aspeed<sup>†‡</sup> [Kaminski, Hoos, Schaub, Schneider, 2012]

- **Constraint Satisfaction:**

- CPHydra<sup>\*†</sup> [O'Mahony, Hebrard, Holland, Nugent, O'Sullivan, 2008]

# Portfolios have been successful in many areas

*\*Algorithm Selection †Sequential Execution ‡Parallel Execution*

- **Planning:**

- FD Stone Soup<sup>†</sup> [Helmert, Röger, Karpas, 2011]

- **Mixed Integer Programming:**

- ISAC<sup>\*</sup> [Kadioglu, Malitsky, Sellmann, Tierney, 2010]

- MIPzilla<sup>\*†</sup> [Xu, Hutter, Hoos, Leyton-Brown, 2011]

- **..and this is just the tip of the iceberg:**

- <http://dl.acm.org/citation.cfm?id=1456656> [Smith-Miles, 2008]

- <http://4c.ucc.ie/~larsko/assurvey> [Kotthoff, 2012]

# Overview

- Programming by Optimization (PbO):  
Motivation and Introduction
- Algorithm Configuration
- **Portfolio-Based Algorithm Selection**
  - ➔ **SATzilla: a framework for algorithm selection**
    - **Hydra: automatic portfolio construction**
- Software Development Tools and Further Directions

# SATzilla: the early core approach

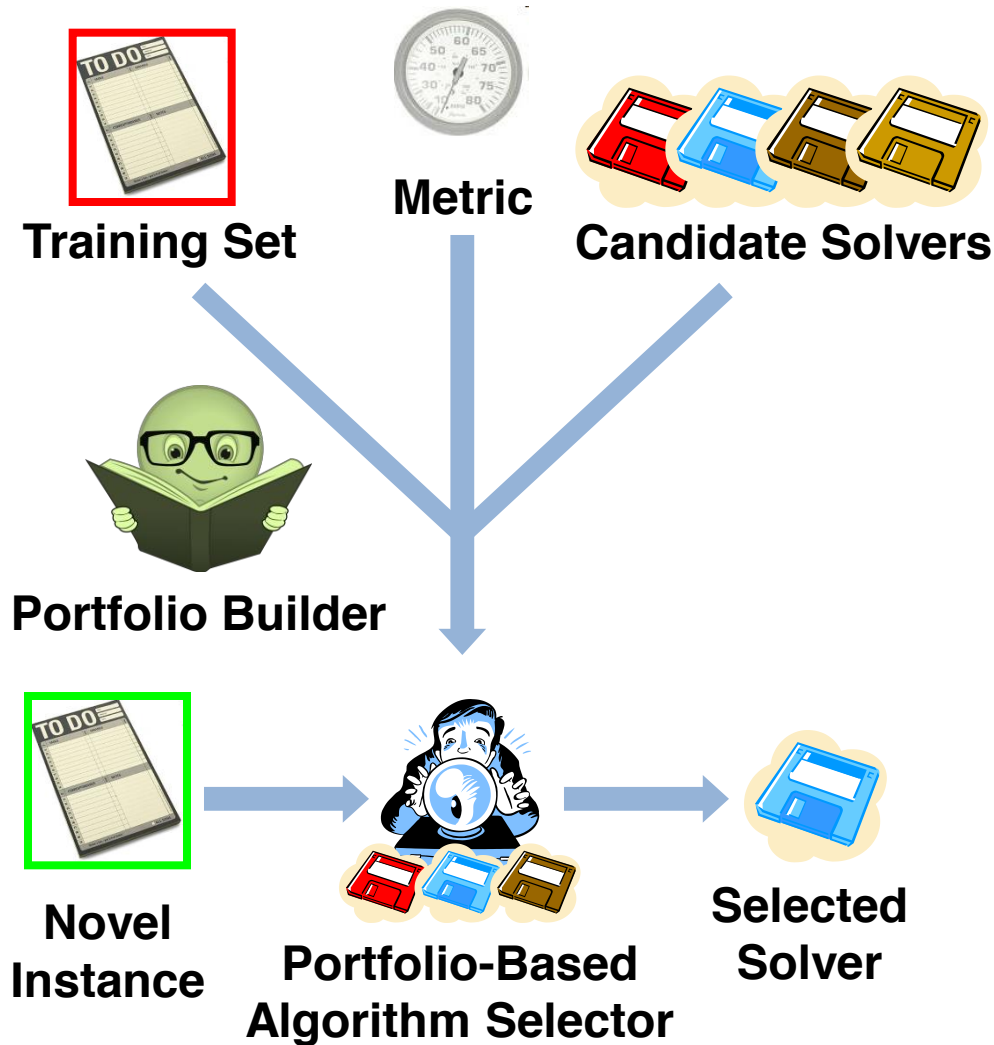
[Leyton-Brown, Nudelman, Andrew, J. McFadden, Shoham, '03]

[Nudelman, Leyton-Brown, Devkar, Shoham, Hoos; '04]

- **Training** (part of algorithm development)
  - Build a statistical model to predict runtime for each component algorithm
- **Test** (for each new instance)
  - Predict performance for each algorithm
  - Pick the algorithm predicted to be best
- **Good performance in SAT competitions**
  - 2003: 2 silver, 1 bronze medals
  - 2004: 2 bronze medals



# SATzilla (stylized version)



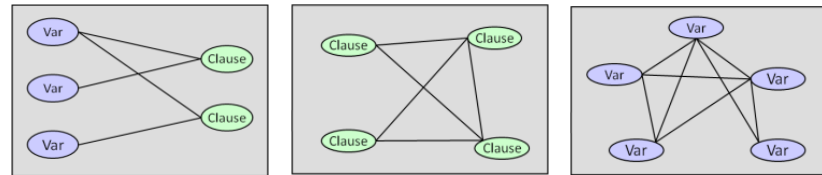
- **Given:**
  - training set of instances
  - performance metric
  - candidate solvers
  - portfolio builder (*incl. instance features*)
- **Training:**
  - collect performance data
  - learn a model for selecting among solvers
- **At Runtime:**
  - evaluate model
  - run selected solver

# SAT Instance Features (2003—2014)

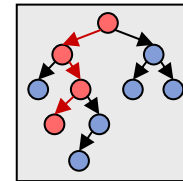
Over 100 features. Some illustrative examples:

- Instance **size** (clauses, variables, clauses/variables, ...)
- **Syntactic** properties (e.g., positive/negative clause ratio)
- Statistics of various **constraint graphs**

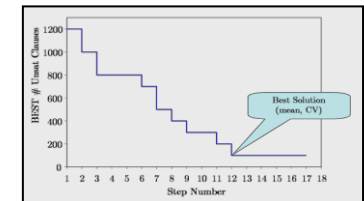
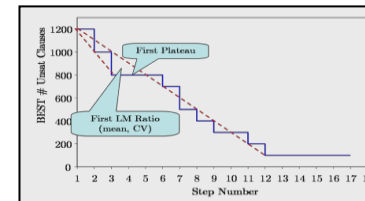
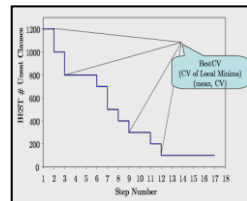
- factor graph
- clause–clause graph
- variable–variable graph



- Knuth's **search space size** estimate



- **Tree search probing**
- **Local search probing**
- **Linear programming** relaxation



$$\begin{aligned}
 &\text{maximize: } \sum_{k \in C} \left( \sum_{i \in L, i \in k} v_i + \sum_{j \in \bar{L}, i \in k} (1 - v_j) \right) \\
 &\text{subject to: } \sum_{i \in k, i \in L} v_i + \sum_{j \in k, j \in \bar{L}} (1 - v_j) \geq 1 \quad \forall k \in C \\
 &v_i \in \{0, 1\} \quad \forall i
 \end{aligned}$$

# SATzilla 2007

[Xu, Hutter, Hoos & Leyton-Brown, CP 2007; JAIR 2008]

- Substantially extended features
- Early algorithm schedule: identify a set of “**presolvers**” and a schedule for running them
  - For every choice of two presolvers + captimes, run the entire SATzilla pipeline and evaluate overall performance
  - Keep the choice that yields best performance
  - For later steps: Discard instances solved by this presolving schedule
- Identify a “**backup solver**”: SBS on the remaining data
  - Needed in case feature computation crashes
- 2007 SAT competition: **3 gold**, 1 silver, 1 bronze medals

# SATzilla 2009

[Xu, Hutter, Hoos & Leyton-Brown, CP 2007; JAIR 2008]

- Robustness: selection of best subset of component solvers
  - Consider every subset of the given solver set
    - omitting a weak solver prevents models from accidentally choosing it
    - conditioned on choice of presolvers
    - computationally cheap: models decompose across solvers
  - Keep the subset that achieves the best performance
- Fully automated procedure
  - optimizes loss on a validation set
- 2009 SAT competition: **3 gold**, 2 silver medals



# SATzilla 2011 and later: cost-sensitive DFs

[Xu, Hutter, Hoos & Leyton-Brown, SAT 2012]

- How it works:
  - Build classifier to determine which algorithm to prefer between each pair of algorithms in the portfolio
  - Loss function: **cost of misclassification**
- Both decision forests and support vector machines have **cost-sensitive variants**
- Classifiers vote for different algorithms; select algorithm with most votes
  - Advantage: selection is a classification problem
  - Advantage: big and small errors treated differently
- 2011 SAT competition: entered **Evaluation Track** (more later)

# 2012 SAT Challenge: Application

Rank	Solver	% solved	# solved
	<b>VBS</b>	<b>94.7</b>	<b>568</b>
1	SATzilla2012 APP	88.5	531
2	SATzilla2012 ALL	85.8	515
3	Industrial SAT Solver	83.2	499
4	interactSAT	80.0	480
5	glucose	79.2	475
6	SINN	78.7	472
7	ZENN	78.0	468
8	Lingeling	77.8	467

\* Interacting multi-engine solvers: like portfolios, but richer interaction between solvers

# 2012 SAT Challenge: Hard Combinatorial

Rank	Solver	% solved	# solved
	<b>VBS</b>	<b>88.2</b>	<b>529</b>
1	SATzilla2012 COMB	79.3	476
2	SATzilla2012 ALL	78.8	473
3	ppfolio2012	70.3	422
4	interactSAT_c	79.5	417
5	pfolioUZK	66.8	401
6	aspeed-crafted	61.7	370
7	clasp-crafted	61.2	367
8	claspfolio-crafted	58.7	352

# SAT Challenge 2012: Random

Rank	Solver	% solved	# solved
	<b>VBS</b>	<b>93.0</b>	<b>558</b>
1	CCASat	70.5	423
2	SATzilla2012 RAND	53.5	321
3	SATzilla2012 ALL	51.0	306
4	sattime2012	44.8	269
5	ppfolio2012	42.2	253
6	pfolioUZK	38.3	230
7	ssa	25.0	150
8	gNovelty+PCL	20.5	123

# 2012 SAT Challenge: Sequential Portfolio

Rank	Solver	% solved	# solved
	<b>VBS</b>	<b>80.7</b>	<b>484</b>
1	SATzilla2012 ALL	72.2	433
2	ppfolio2012	61.7	370
3	pfolioUZK	60.3	362

- 3S **deserves mentioning**, but didn't rank officially  
[Kadioglu, Malitsky, Sabharwal, Samulowitz, Sellmann, 2011]
  - Disqualified on a technicality
    - chose a buggy solver that returned an incorrect result
    - an occupational hazard for portfolios!
  - Overall performance **nearly as strong as SATzilla**

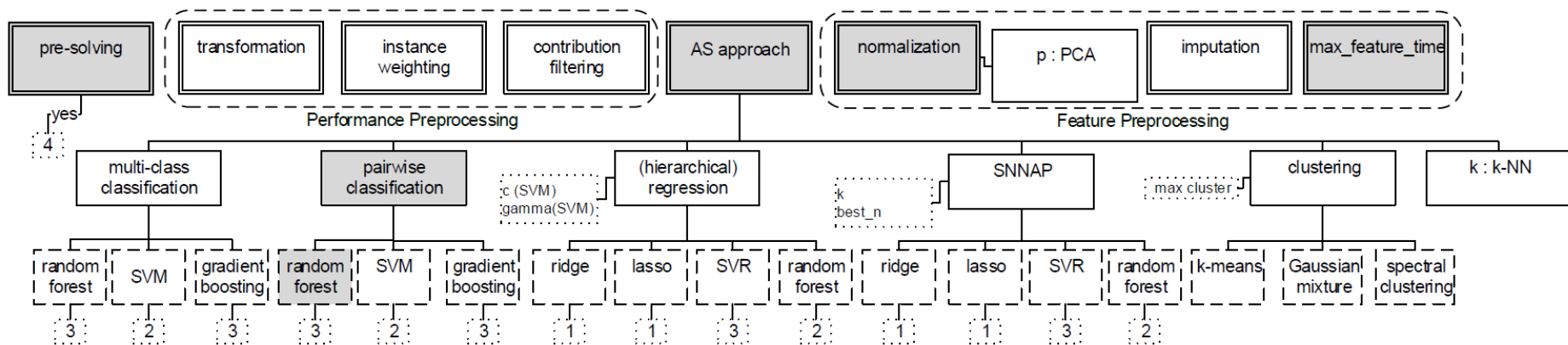
# 2013 onwards

- Algorithm selection a victim of its own success
- 2013: “The emphasis of SAT Competition 2013 is on evaluation of core solvers:”
  - Single-core portfolios of >2 solvers **not eligible**
  - One “open track” allowing parallel solvers, portfolios, etc
  - That open track was dominated by portfolios
- 2014
  - “SAT Competition 2014 **only allows submission of core solvers**”
  - Portfolio researchers started their own competition:  
the **ICON Algorithm Selection Challenge**

# AutoFolio: PbO for Algorithm Selection

[Lindauer, Hoos, Hutter & Schaub, JAIR 2015]

- **Define a general space** of algorithm selection methods
  - AutoFolio's configuration space: 54 choices



- Use algorithm configuration to **select best instantiation**
  - Partition the training benchmark instances into 10 folds
  - Use SMAC to find the algorithm selection approach & its hyperparameters that optimize CV performance

# ICON Algorithm Selection Challenge 2015

- Ingredients of an algorithm selection (AS) benchmark
  - A set of solvers
  - A set of benchmark instances (split into training & test)
  - Measured performance of all solvers on all instances
- Algorithm selection competition:
  - 13 AS benchmarks from SAT, MaxSAT, CSP, ASP, QBF, Premarshalling
  - 9 competitors using regression, classification, clustering, k-NN, etc
- Winning algorithms in the 3 tracks:
  - Penalized average runtime: **AutoFolio**
  - Number of instances solved: **AutoFolio**
  - Frequency of selecting the best algorithm: **SATzilla**
  - **Overall winner SATzilla** (2nd in first two tracks)



# Try it yourself!

- SATzilla and AutoFolio are **freely available online**

<http://www.cs.ubc.ca/labs/beta/Projects/SATzilla/>

<http://ml4aad.org/autofolio>

- You can try them for your problem
  - we have features for SAT, MIP and TSP
  - you need to provide features for other domains
    - in many cases, the general ideas behind our features apply
    - can also make features by reducing your problem to e.g. SAT and computing the SAT features

## Automatically Configuring Algorithms for Portfolio-Based Selection

Xu, Hoos, Leyton-Brown (2010); Kadioglu *et al.* (2010)

### **Note:**

- ▶ SATzilla builds algorithm selector based on given set of SAT solvers  
*but:* success entirely depends on quality of given solvers

## Automatically Configuring Algorithms for Portfolio-Based Selection

Xu, Hoos, Leyton-Brown (2010); Kadioglu *et al.* (2010)

### Note:

- ▶ SATzilla builds algorithm selector based on given set of SAT solvers  
*but:* success entirely depends on quality of given solvers
- ▶ Automated configuration produces solvers that work well *on average* on a given set of SAT instances  
(*e.g.*, SATenstein – KhudaBukhsh, Xu, Hoos, Leyton-Brown 2009)

## Automatically Configuring Algorithms for Portfolio-Based Selection

Xu, Hoos, Leyton-Brown (2010); Kadioglu *et al.* (2010)

### Note:

- ▶ SATzilla builds algorithm selector based on given set of SAT solvers  
*but:* success entirely depends on quality of given solvers
- ▶ Automated configuration produces solvers that work well *on average* on a given set of SAT instances  
(*e.g.*, SATenstein – KhudaBukhsh, Xu, Hoos, Leyton-Brown 2009)  
*but:* may have to settle for compromises  
for broad, heterogenous instance sets

## Automatically Configuring Algorithms for Portfolio-Based Selection

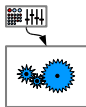
Xu, Hoos, Leyton-Brown (2010); Kadioglu *et al.* (2010)

### Note:

- ▶ SATzilla builds algorithm selector based on given set of SAT solvers  
*but:* success entirely depends on quality of given solvers
- ▶ Automated configuration produces solvers that work well *on average* on a given set of SAT instances  
(*e.g.*, SATenstein – KhudaBukhsh, Xu, Hoos, Leyton-Brown 2009)  
*but:* may have to settle for compromises  
for broad, heterogenous instance sets

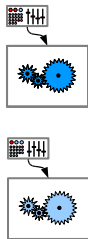
**Idea:** Combine the two approaches  $\rightsquigarrow$  portfolio-based selection from set of automatically constructed solvers

## Combined Configuration + Selection



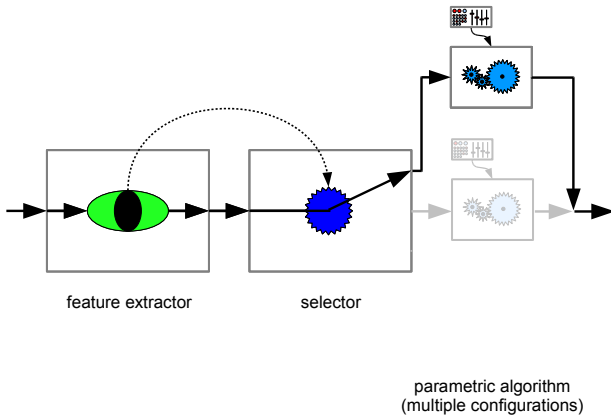
parametric algorithm

## Combined Configuration + Selection



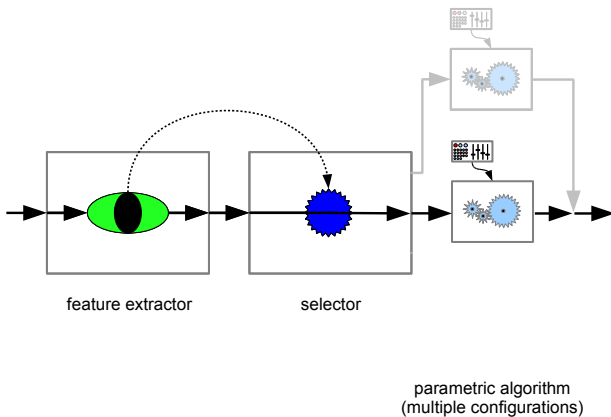
parametric algorithm  
(multiple configurations)

## Combined Configuration + Selection

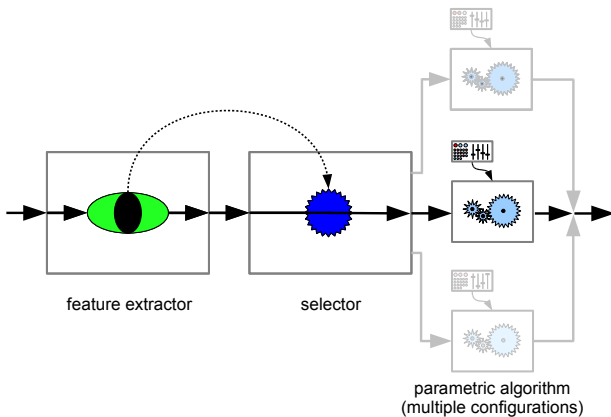




## Combined Configuration + Selection



## Combined Configuration + Selection



## Approach #1:

1. build solvers for various types of instances using automated algorithm configuration

## Approach #1:

1. build solvers for various types of instances using automated algorithm configuration
2. construct portfolio-based selector from these

## Approach #1:

1. build solvers for various types of instances using automated algorithm configuration
2. construct portfolio-based selector from these

**Problem:** requires suitably defined sets of instances

## Approach #1:

1. build solvers for various types of instances using automated algorithm configuration
2. construct portfolio-based selector from these

**Problem:** requires suitably defined sets of instances

**Solution:** automatically partition heterogeneous instance set

## Instance-specific algorithm configuration (ISAC)

Kadioglu, Malitsky, Sellmann, Tierney (2010); Malitky, Sellman (2012)

## Instance-specific algorithm configuration (ISAC)

Kadioglu, Malitsky, Sellmann, Tierney (2010); Malitky, Sellman (2012)

1. cluster training instances based on features  
(using G-means)



## Instance-specific algorithm configuration (ISAC)

Kadioglu, Malitsky, Sellmann, Tierney (2010); Malitky, Sellman (2012)

1. cluster training instances based on features  
(using G-means)
2. configure given parameterised algorithm independently  
for each cluster (using GGA)

## Instance-specific algorithm configuration (ISAC)

Kadioglu, Malitsky, Sellmann, Tierney (2010); Malitsky, Sellman (2012)

1. cluster training instances based on features  
(using G-means)
2. configure given parameterised algorithm independently  
for each cluster (using GGA)
3. construct portfolio-based selector from resulting configurations  
(using distance to cluster centroids)

## Instance-specific algorithm configuration (ISAC)

Kadioglu, Malitsky, Sellmann, Tierney (2010); Malitky, Sellman (2012)

1. cluster training instances based on features  
(using G-means)
2. configure given parameterised algorithm independently  
for each cluster (using GGA)
3. construct portfolio-based selector from resulting configurations  
(using distance to cluster centroids)

**Drawback:** Instance features may not correlate well  
with impact of algorithm parameters on performance  
(e.g., uninformative features)

## Approach #2:

**Key idea:** Augment existing selector *AS* by targetting instances on which *AS* performs poorly

(*cf.* Leyton-Brown *et al.* 2003; Leyton-Brown *et al.* 2009)

## Approach #2:

**Key idea:** Augment existing selector *AS* by targetting instances on which *AS* performs poorly

(*cf.* Leyton-Brown *et al.* 2003; Leyton-Brown *et al.* 2009)

- ▶ interleave configuration and selector construction

## Approach #2:

**Key idea:** Augment existing selector  $AS$  by targetting instances on which  $AS$  performs poorly

(*cf.* Leyton-Brown *et al.* 2003; Leyton-Brown *et al.* 2009)

- ▶ interleave configuration and selector construction
- ▶ in each iteration, determine configuration that complements current selector best

## Approach #2:

**Key idea:** Augment existing selector *AS* by targetting instances on which *AS* performs poorly

(*cf.* Leyton-Brown *et al.* 2003; Leyton-Brown *et al.* 2009)

- ▶ interleave configuration and selector construction
- ▶ in each iteration, determine configuration that complements current selector best

## Advantages:

- ▶ any-time behaviour: iteratively adds configurations

## Approach #2:

**Key idea:** Augment existing selector *AS* by targetting instances on which *AS* performs poorly

(*cf.* Leyton-Brown *et al.* 2003; Leyton-Brown *et al.* 2009)

- ▶ interleave configuration and selector construction
- ▶ in each iteration, determine configuration that complements current selector best

## Advantages:

- ▶ any-time behaviour: iteratively adds configurations
- ▶ desirable theoretical guarantees (under idealising assumptions)



# Hydra

Xu, Hoos, Leyton-Brown (2010); Xu, Hutter, Hoos, Leyton-Brown (2011)

## Hydra

Xu, Hoos, Leyton-Brown (2010); Xu, Hutter, Hoos, Leyton-Brown (2011)

1. configure given target algorithm  $A$  on complete instance set  $I$   
 $\rightsquigarrow$  configuration  $A_1 = \text{selector } AS_1$  (always selects  $A_1$ )

## Hydra

Xu, Hoos, Leyton-Brown (2010); Xu, Hutter, Hoos, Leyton-Brown (2011)

1. configure given target algorithm  $A$  on complete instance set  $I$   
 $\rightsquigarrow$  configuration  $A_1 = \text{selector } AS_1$  (always selects  $A_1$ )
2. configure a new copy of  $A$  on  $I$  such that performance of selector  $AS := AS_1 + A_{new}$  is optimised  
 $\rightsquigarrow$  configuration  $A_2$   
 $\rightsquigarrow$  selector  $AS_2 := AS_1 + A_2$  (selects from  $\{A_1, A_2\}$ )

## Hydra

Xu, Hoos, Leyton-Brown (2010); Xu, Hutter, Hoos, Leyton-Brown (2011)

1. configure given target algorithm  $A$  on complete instance set  $I$   
 $\rightsquigarrow$  configuration  $A_1 = \text{selector } AS_1$  (always selects  $A_1$ )
2. configure a new copy of  $A$  on  $I$  such that performance of selector  $AS := AS_1 + A_{new}$  is optimised  
 $\rightsquigarrow$  configuration  $A_2$   
 $\rightsquigarrow$  selector  $AS_2 := AS_1 + A_2$  (selects from  $\{A_1, A_2\}$ )
3. configure a new copy of  $A$  on  $I$  such that performance of selector  $AS := AS_2 + A_{new}$  is optimised  
 $\rightsquigarrow$  configuration  $A_3$   
 $\rightsquigarrow$  selector  $AS_3 := AS_2 + A_3$  (selects from  $\{A_1, A_2, A_3\}$ )
- ...

## Note:

- ▶ effectively adds  $A$  with maximal marginal contribution in each iteration

## Note:

- ▶ effectively adds  $A$  with maximal marginal contribution in each iteration
- ▶ estimate marginal contribution using perfect selector (oracle)  
     $\rightsquigarrow$  avoids costly construction of selectors during configuration

## Note:

- ▶ effectively adds  $A$  with maximal marginal contribution in each iteration
- ▶ estimate marginal contribution using perfect selector (oracle)  
     $\rightsquigarrow$  avoids costly construction of selectors during configuration
- ▶ works well using FocusedILS for configuration,  
    \*zilla for selection (but can use other configurators, selectors)

## Note:

- ▶ effectively adds  $A$  with maximal marginal contribution in each iteration
- ▶ estimate marginal contribution using perfect selector (oracle)  
~> avoids costly construction of selectors during configuration
- ▶ works well using FocusedILS for configuration,  
\*zilla for selection (but can use other configurators, selectors)
- ▶ can be further improved by adding multiple configurations per iteration; using performance estimates from configurator



## Results on SAT:

- ▶ target algorithm: SATenstein-LS (KhudaBukhsh *et al.* 2009)

## Results on SAT:

- ▶ target algorithm: SATenstein-LS (KhudaBukhsh *et al.* 2009)
- ▶ 6 well-known benchmark sets of SAT instances (application, crafted, random)

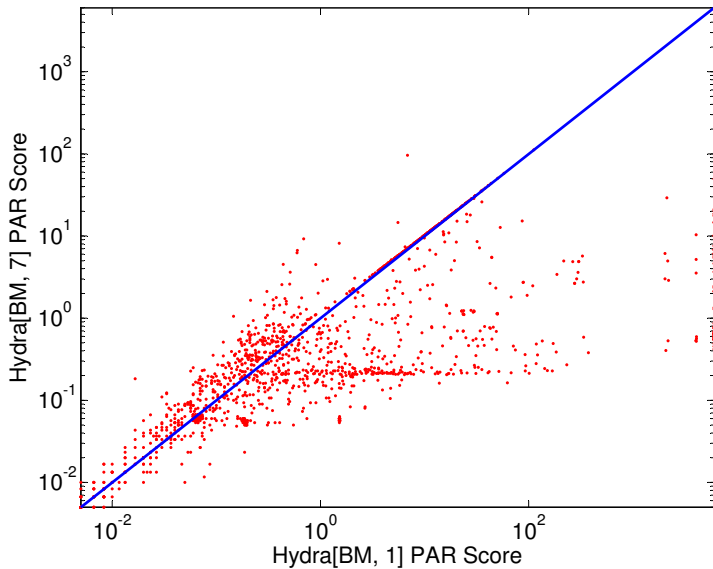
## Results on SAT:

- ▶ target algorithm: SATenstein-LS (KhudaBukhsh *et al.* 2009)
- ▶ 6 well-known benchmark sets of SAT instances (application, crafted, random)
- ▶ 7 iterations of Hydra

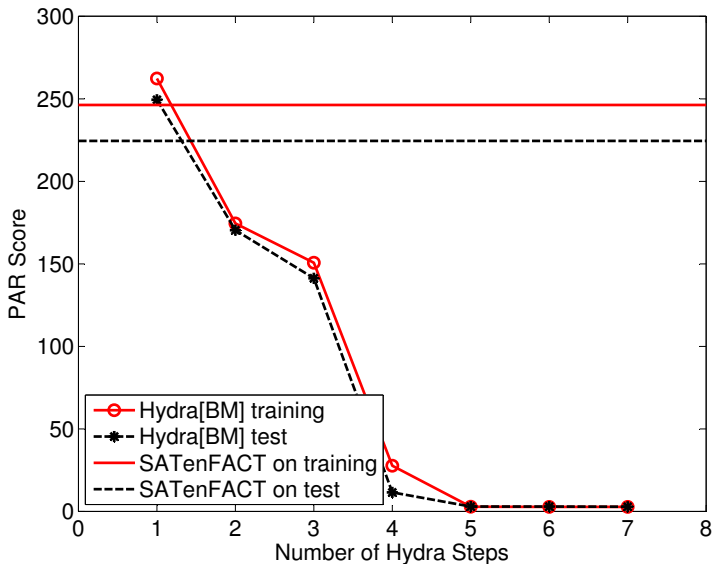
## Results on SAT:

- ▶ target algorithm: SATenstein-LS (KhudaBukhsh *et al.* 2009)
- ▶ 6 well-known benchmark sets of SAT instances (application, crafted, random)
- ▶ 7 iterations of Hydra
- ▶ 10 configurator runs per iteration, 1 CPU day each

## Results on mixture of 6 benchmark sets



## Results on mixture of 6 benchmark sets



## Note:

- ▶ good results also for MIP (CPLEX)  
(Xu, Hutter, Hoos, Leyton-Brown 2011)

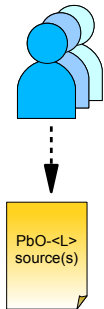
## Note:

- ▶ good results also for MIP (CPLEX)  
(Xu, Hutter, Hoos, Leyton-Brown 2011)
- ▶ idea underlying Hydra can also be applied to automatically construct *parallel algorithm portfolios* from single parameterised target algorithm  
(Hoos, Leyton-Brown, Schaub, Schneider 2012–14)

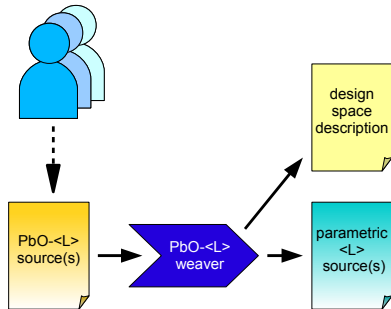


# Software Development Support and Further Directions

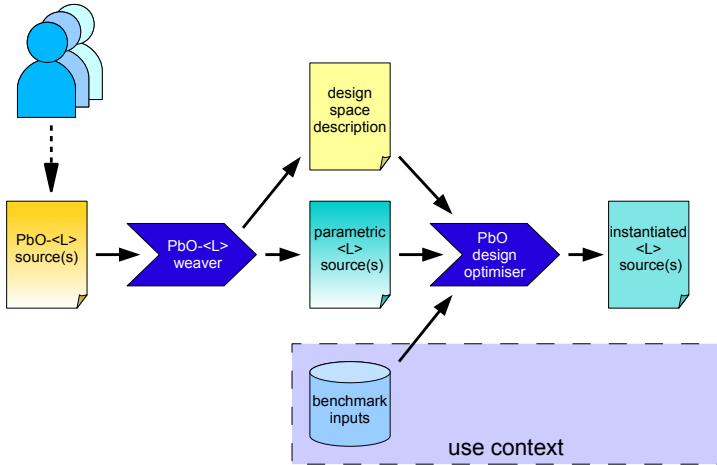
# Software development in the PbO paradigm



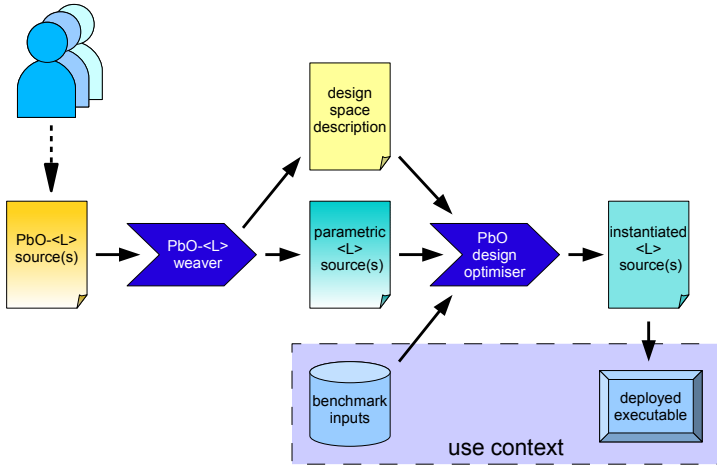
# Software development in the PbO paradigm



# Software development in the PbO paradigm



# Software development in the PbO paradigm



# Design space specification

Option 1: use language-specific mechanisms

# Design space specification

Option 1: use language-specific mechanisms

- ▶ command-line parameters
- ▶ conditional execution
- ▶ conditional compilation (`ifdef`)

# Design space specification

Option 1: use language-specific mechanisms

- ▶ command-line parameters
- ▶ conditional execution
- ▶ conditional compilation (`ifdef`)

Option 2: generic programming language extension



# Design space specification

## Option 1: use language-specific mechanisms

- ▶ command-line parameters
- ▶ conditional execution
- ▶ conditional compilation (`ifdef`)

## Option 2: generic programming language extension

Dedicated support for ...

- ▶ exposing parameters
- ▶ specifying alternative blocks of code

## Advantages of generic language extension:

- ▶ reduced overhead for programmer

## Advantages of generic language extension:

- ▶ reduced overhead for programmer
- ▶ clean separation of design choices from other code

## Advantages of generic language extension:

- ▶ reduced overhead for programmer
- ▶ clean separation of design choices from other code
- ▶ dedicated PbO support in software development environments

## Advantages of generic language extension:

- ▶ reduced overhead for programmer
- ▶ clean separation of design choices from other code
- ▶ dedicated PbO support in software development environments

## Key idea:

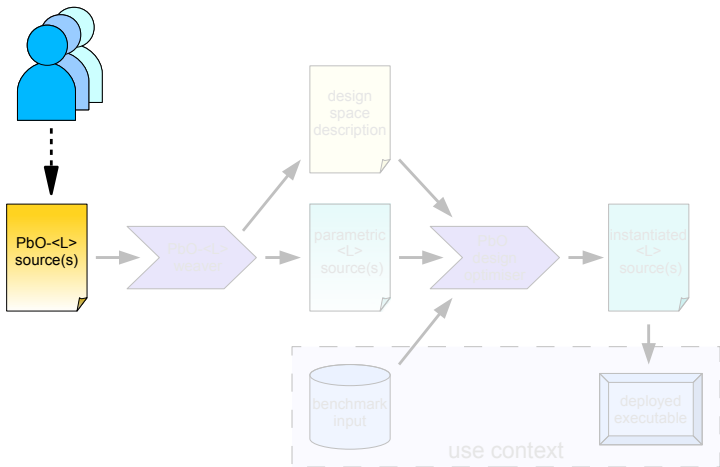
- ▶ augmented sources: *PbO-Java* = Java + PbO constructs, ...

## Advantages of generic language extension:

- ▶ reduced overhead for programmer
- ▶ clean separation of design choices from other code
- ▶ dedicated PbO support in software development environments

## Key idea:

- ▶ augmented sources: *PbO-Java* = Java + PbO constructs, ...
- ▶ tool to compile down into target language: *weaver*



## Exposing parameters

```
...  
numerator -= (int) (numerator / (adjfactor+1) * 1.4);  
...
```



## Exposing parameters

```
...  
##PARAM(float multiplier=1.4)  
numerator -= (int) (numerator / (adjfactor+1) * ##multiplier);  
...
```

## Exposing parameters

```
...  
##PARAM(float multiplier=1.4)  
numerator -= (int) (numerator / (adjfactor+1) * ##multiplier);  
...
```

- ▶ parameter declarations can appear at arbitrary places (before or after first use of parameter)

## Exposing parameters

```
...  
##PARAM(float multiplier=1.4)  
numerator -= (int) (numerator / (adjfactor+1) * ##multiplier);  
...
```

- ▶ parameter declarations can appear at arbitrary places (before or after first use of parameter)
- ▶ access to parameters is read-only (values can only be set/changed via command-line or config file)

## Specifying design alternatives

- ▶ **Choice:** set of interchangeable fragments of code that represent design alternatives (**instances of choice**)

## Specifying design alternatives

- ▶ **Choice:** set of interchangeable fragments of code that represent design alternatives (**instances of choice**)
- ▶ **Choice point:** location in a program at which a choice is available

## Specifying design alternatives

- ▶ **Choice:** set of interchangeable fragments of code that represent design alternatives (**instances of choice**)
- ▶ **Choice point:** location in a program at which a choice is available

```
##BEGIN CHOICE preProcessing  
<block 1>  
##END CHOICE preProcessing
```

## Specifying design alternatives

- ▶ **Choice:** set of interchangeable fragments of code that represent design alternatives (**instances of choice**)
- ▶ **Choice point:** location in a program at which a choice is available

```
##BEGIN CHOICE preProcessing=standard  
<block S>  
##END CHOICE preProcessing
```

```
##BEGIN CHOICE preProcessing=enhanced  
<block E>  
##END CHOICE preProcessing
```

## Specifying design alternatives

- ▶ **Choice:** set of interchangeable fragments of code that represent design alternatives (**instances of choice**)
- ▶ **Choice point:** location in a program at which a choice is available

```
##BEGIN CHOICE preProcessing  
<block 1>  
##END CHOICE preProcessing
```

...

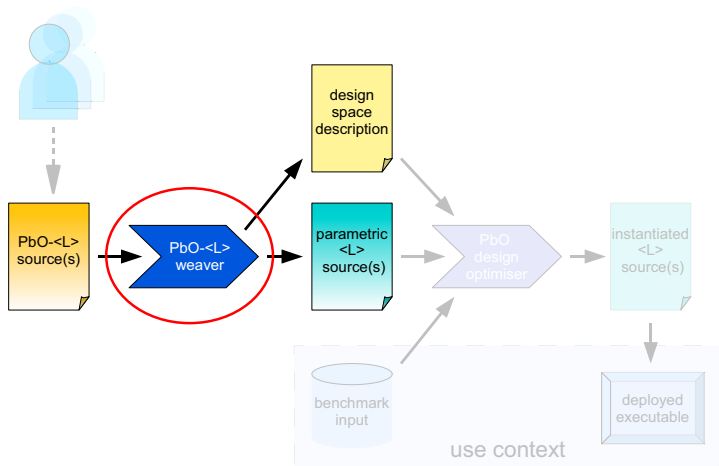
```
##BEGIN CHOICE preProcessing  
<block 2>  
##END CHOICE preProcessing
```



## Specifying design alternatives

- ▶ **Choice:** set of interchangeable fragments of code that represent design alternatives (**instances of choice**)
- ▶ **Choice point:** location in a program at which a choice is available

```
##BEGIN CHOICE preProcessing
<block 1a>
  ##BEGIN CHOICE extraPreProcessing
  <block 2>
  ##END CHOICE extraPreProcessing
<block 1b>
##END CHOICE preProcessing
```



## The Weaver

transforms PbO-<L> code into <L> code  
(<L> = Java, C++, ...)

## The Weaver

transforms PbO-<L> code into <L> code  
(<L> = Java, C++, ...)

- ▶ parametric mode:
  - ▶ expose parameters

## The Weaver

transforms PbO- $\langle L \rangle$  code into  $\langle L \rangle$  code  
( $\langle L \rangle = \text{Java, C++}, \dots$ )

- ▶ **parametric mode:**
  - ▶ expose parameters
  - ▶ make choices accessible via (conditional, categorical) parameters

## The Weaver

transforms PbO-<L> code into <L> code  
(<L> = Java, C++, ...)

- ▶ **parametric mode:**
  - ▶ expose parameters
  - ▶ make choices accessible via (conditional, categorical) parameters
- ▶ **(partial) instantiation mode:**
  - ▶ hardwire (some) parameters into code (expose others)

## The Weaver

transforms PbO- $\langle L \rangle$  code into  $\langle L \rangle$  code  
( $\langle L \rangle = \text{Java, C++}, \dots$ )

- ▶ **parametric mode:**
  - ▶ expose parameters
  - ▶ make choices accessible via (conditional, categorical) parameters
- ▶ **(partial) instantiation mode:**
  - ▶ hardwire (some) parameters into code (expose others)
  - ▶ hardwire (some) choices into code (make others accessible via parameters)

## The road ahead

- ▶ Support for PbO-based software development



## The road ahead

- ▶ Support for PbO-based software development
  - ▶ Weavers for PbO-C, PbO-C++, PbO-Java

## The road ahead

- ▶ Support for PbO-based software development
  - ▶ Weavers for PbO-C, PbO-C++, PbO-Java
  - ▶ PbO-aware development platforms

## The road ahead

- ▶ Support for PbO-based software development
  - ▶ Weavers for PbO-C, PbO-C++, PbO-Java
  - ▶ PbO-aware development platforms
  - ▶ Improved / integrated PbO design optimiser

## The road ahead

- ▶ Support for PbO-based software development
  - ▶ Weavers for PbO-C, PbO-C++, PbO-Java
  - ▶ PbO-aware development platforms
  - ▶ Improved / integrated PbO design optimiser
  - ▶ Debugging and performance analysis tools

# The road ahead

- ▶ Support for PbO-based software development
  - ▶ Weavers for PbO-C, PbO-C++, PbO-Java
  - ▶ PbO-aware development platforms
  - ▶ Improved / integrated PbO design optimiser
  - ▶ Debugging and performance analysis tools
  
- ▶ Best practices

# The road ahead

- ▶ Support for PbO-based software development
  - ▶ Weavers for PbO-C, PbO-C++, PbO-Java
  - ▶ PbO-aware development platforms
  - ▶ Improved / integrated PbO design optimiser
  - ▶ Debugging and performance analysis tools
- ▶ Best practices
- ▶ Many further applications

# The road ahead

- ▶ Support for PbO-based software development
  - ▶ Weavers for PbO-C, PbO-C++, PbO-Java
  - ▶ PbO-aware development platforms
  - ▶ Improved / integrated PbO design optimiser
  - ▶ Debugging and performance analysis tools
- ▶ Best practices
- ▶ Many further applications
- ▶ Scientific insights

## Which choices matter?

**Observation:** Some design choices matter more than others



## Which choices matter?

**Observation:** Some design choices matter more than others depending on . . .

- ▶ algorithm under consideration
- ▶ given use context

## Which choices matter?

**Observation:** Some design choices matter more than others depending on ...

- ▶ algorithm under consideration
- ▶ given use context

**Knowledge which choices / parameters matter may ...**

- ▶ guide algorithm development
- ▶ facilitate configuration

### 3 recent approaches:

- ▶ Forward selection based on empirical performance models  
Hutter, Hoos, Leyton-Brown (2013)

### 3 recent approaches:

- ▶ Forward selection based on empirical performance models

Hutter, Hoos, Leyton-Brown (2013)

- ▶ Functional ANOVA based on empirical performance models

Hutter, Hoos, Leyton-Brown (2014)

### 3 recent approaches:

- ▶ Forward selection based on empirical performance models

Hutter, Hoos, Leyton-Brown (2013)

- ▶ Functional ANOVA based on empirical performance models

Hutter, Hoos, Leyton-Brown (2014)

- ▶ Ablation analysis

Fawcett, Hoos (2013–14)

## Functional ANOVA based on empirical performance models

Hutter, Hoos, Leyton-Brown (2014)

### **Key idea:**

- ▶ build regression model of algorithm performance as a function of all input parameters (= design choices)

## Functional ANOVA based on empirical performance models

Hutter, Hoos, Leyton-Brown (2014)

### **Key idea:**

- ▶ build regression model of algorithm performance as a function of all input parameters (= design choices)  
     $\rightsquigarrow$  empirical performance models (EPMs)

## Functional ANOVA based on empirical performance models

Hutter, Hoos, Leyton-Brown (2014)

### Key idea:

- ▶ build regression model of algorithm performance as a function of all input parameters (= design choices)  
     $\rightsquigarrow$  empirical performance models (EPMs)
- ▶ analyse variance in model output (= predicted performance) due to each parameter, parameter interactions



## Functional ANOVA based on empirical performance models

Hutter, Hoos, Leyton-Brown (2014)

### Key idea:

- ▶ build regression model of algorithm performance as a function of all input parameters (= design choices)  
     $\rightsquigarrow$  empirical performance models (EPMs)
- ▶ analyse variance in model output (= predicted performance) due to each parameter, parameter interactions
- ▶ importance of parameter: fraction of performance variation over configuration space explained by it (main effect)

## Functional ANOVA based on empirical performance models

Hutter, Hoos, Leyton-Brown (2014)

### Key idea:

- ▶ build regression model of algorithm performance as a function of all input parameters (= design choices)  
     $\rightsquigarrow$  empirical performance models (EPMs)
- ▶ analyse variance in model output (= predicted performance) due to each parameter, parameter interactions
- ▶ importance of parameter: fraction of performance variation over configuration space explained by it (main effect)
- ▶ analogous for sets of parameters (interaction effects)

## Decomposition of variance in a nutshell

For parameters  $p_1, \dots, p_n$  and a function (performance model)  $y$ :

$$y(p_1, \dots, p_n) = \mu$$

## Decomposition of variance in a nutshell

For parameters  $p_1, \dots, p_n$  and a function (performance model)  $y$ :

$$y(p_1, \dots, p_n) = \mu + f_1(p_1) + f_2(p_2) + \dots + f_n(p_n)$$

## Decomposition of variance in a nutshell

For parameters  $p_1, \dots, p_n$  and a function (performance model)  $y$ :

$$\begin{aligned}y(p_1, \dots, p_n) &= \mu \\ &+ f_1(p_1) + f_2(p_2) + \dots + f_n(p_n) \\ &+ f_{1,2}(p_1, p_2) + f_{1,3}(p_1, p_3) + \dots + f_{n-1,n}(p_{n-1}, p_n)\end{aligned}$$

## Decomposition of variance in a nutshell

For parameters  $p_1, \dots, p_n$  and a function (performance model)  $y$ :

$$\begin{aligned}y(p_1, \dots, p_n) &= \mu \\ &+ f_1(p_1) + f_2(p_2) + \dots + f_n(p_n) \\ &+ f_{1,2}(p_1, p_2) + f_{1,3}(p_1, p_3) + \dots + f_{n-1,n}(p_{n-1}, p_n) \\ &+ f_{1,2,3}(p_1, p_2, p_3) + \dots\end{aligned}$$

## Decomposition of variance in a nutshell

For parameters  $p_1, \dots, p_n$  and a function (performance model)  $y$ :

$$\begin{aligned}y(p_1, \dots, p_n) &= \mu \\ &+ f_1(p_1) + f_2(p_2) + \dots + f_n(p_n) \\ &+ f_{1,2}(p_1, p_2) + f_{1,3}(p_1, p_3) + \dots + f_{n-1,n}(p_{n-1}, p_n) \\ &+ f_{1,2,3}(p_1, p_2, p_3) + \dots \\ &+ \dots\end{aligned}$$

## Note:

- ▶ Straightforward computation of main and interaction effects is **intractable**.  
(integration over combinatorial spaces of configurations)



## Note:

- ▶ Straightforward computation of main and interaction effects is **intractable**.  
(integration over combinatorial spaces of configurations)
- ▶ For random forest models, marginal performance predictions and variance decomposition (up to constant-sized interactions) can be computed **exactly** and **efficiently**.

## Empirical study:

- ▶ 8 high-performance solvers for SAT, ASP, MIP, TSP (4–85 parameters)
- ▶ 12 well-known sets of benchmark data (random + real-world structure)

## Empirical study:

- ▶ 8 high-performance solvers for SAT, ASP, MIP, TSP (4–85 parameters)
- ▶ 12 well-known sets of benchmark data (random + real-world structure)
- ▶ random forest models for performance prediction,

## Empirical study:

- ▶ 8 high-performance solvers for SAT, ASP, MIP, TSP (4–85 parameters)
- ▶ 12 well-known sets of benchmark data (random + real-world structure)
- ▶ random forest models for performance prediction, trained on 10 000 randomly sampled configurations per solver + data from 25+ runs of SMAC configuration procedure

Fraction of variance explained by main effects:

## Fraction of variance explained by main effects:

CPLEX on RCW (comp sust)	70.3%
CPLEX on CORLAT (comp sust)	35.0%

## Fraction of variance explained by main effects:

CPLEX on RCW (comp sust)	70.3%
CPLEX on CORLAT (comp sust)	35.0%
Clasp on software verification	78.9%
Clasp on DB query optimisation	62.5%

## Fraction of variance explained by main effects:

CPLEX on RCW (comp sust)	70.3%
CPLEX on CORLAT (comp sust)	35.0%
Clasp on software verification	78.9%
Clasp on DB query optimisation	62.5%
CryptoMiniSAT on bounded model checking	35.5%
CryptoMiniSAT on software verification	31.9%



## Fraction of variance explained by main + 2-interaction effects:

CPLEX on RCW (comp sust)	70.3% + 12.7%
CPLEX on CORLAT (comp sust)	35.0% + 8.3%
Clasp on software verification	78.9% + 14.3%
Clasp on DB query optimisation	62.5% + 11.7%
CryptoMiniSAT on bounded model checking	35.5% + 20.8%
CryptoMiniSAT on software verification	31.9% + 28.5%

## Note:

may pick up variation caused by poorly performing configurations

### Note:

may pick up variation caused by poorly performing configurations

### Simple solution:

cap at default performance or quantile from distribution of randomly sampled configurations; build model from capped data.

## Ablation analysis

Fawcett, Hoos (2013–14)

### Key idea:

- ▶ given two configurations,  $A$  and  $B$ , change one parameter at a time to get from  $A$  to  $B$

## Ablation analysis

Fawcett, Hoos (2013–14)

### Key idea:

- ▶ given two configurations,  $A$  and  $B$ , change one parameter at a time to get from  $A$  to  $B$   
     $\rightsquigarrow$  ablation path

## Ablation analysis

Fawcett, Hoos (2013–14)

### Key idea:

- ▶ given two configurations,  $A$  and  $B$ , change one parameter at a time to get from  $A$  to  $B$   
     $\rightsquigarrow$  ablation path
- ▶ in each step, change parameter to achieve maximal gain (or minimal loss) in performance

## Ablation analysis

Fawcett, Hoos (2013–14)

### Key idea:

- ▶ given two configurations,  $A$  and  $B$ , change one parameter at a time to get from  $A$  to  $B$   
     $\rightsquigarrow$  ablation path
- ▶ in each step, change parameter to achieve maximal gain (or minimal loss) in performance
- ▶ for computational efficiency, use racing (F-race) for evaluating parameters considered in each step

## Empirical study:

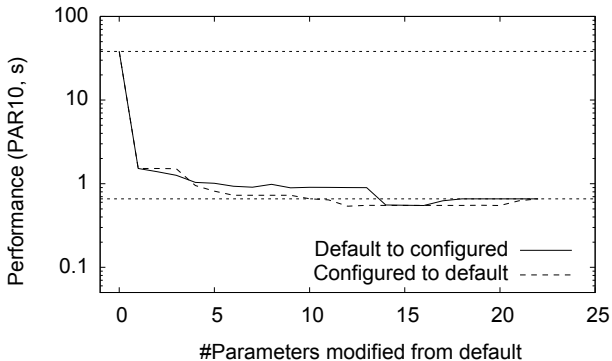
- ▶ high-performance solvers for SAT, MIP, **AI Planning** (26–76 parameters),  
well-known sets of benchmark data (real-world structure)



## Empirical study:

- ▶ high-performance solvers for SAT, MIP, [AI Planning](#) (26–76 parameters), well-known sets of benchmark data (real-world structure)
- ▶ optimised configurations obtained from ParamILS (minimisation of penalised average running time; 10 runs per scenario, 48 CPU hours each)

## Ablation between default and optimised configurations:



LPG on Depots planning domain

## Which parameters are important?

LPG on depots:

- ▶ `cri_intermediate_levels` (43% of overall gain!)
- ▶ `triomemory`
- ▶ `donot_try_suspected_actions`
- ▶ `walkplan`
- ▶ `weight_mutex_in_relaxed_plan`

## Which parameters are important?

LPG on depots:

- ▶ `cri_intermediate_levels` (43% of overall gain!)
- ▶ `triomemory`
- ▶ `donot_try_suspected_actions`
- ▶ `walkplan`
- ▶ `weight_mutex_in_relaxed_plan`

**Note:** Importance of parameters varies between planning domains

## Algorithm configuration: parameter importance

Algorithm configuration: parameter importance

↪ Algorithm selection: component contribution

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

Algorithm configuration: parameter importance

↔ Algorithm selection: component contribution

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

Consider:

portfolio-based algorithm selector  $AS$

with candidate algorithms  $A_1, A_2, \dots, A_k$

Algorithm configuration: parameter importance

↪ Algorithm selection: component contribution

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

Consider:

portfolio-based algorithm selector  $AS$   
with candidate algorithms  $A_1, A_2, \dots, A_k$

Question:

How much does each  $A_i$  contribute  
to overall performance of  $AS$ ?



## Marginal contribution of $A_i$ to portfolio-based selector $AS$

= difference in performance of  $AS$  with and without  $A_i$   
(trained separately)

## Marginal contribution of $A_i$ to portfolio-based selector $AS$

= difference in performance of  $AS$  with and without  $A_i$   
(trained separately)

$\neq$  frequency of selecting  $A_i$

## Marginal contribution of $A_i$ to portfolio-based selector $AS$

= difference in performance of  $AS$  with and without  $A_i$   
(trained separately)

$\neq$  frequency of selecting  $A_i$

$\neq$  fraction of instances solved by  $A_i$

## Marginal contribution of $A_i$ to portfolio-based selector $AS$

= difference in performance of  $AS$  with and without  $A_i$   
(trained separately)

≠ frequency of selecting  $A_i$

≠ fraction of instances solved by  $A_i$

≠ contribution of  $A_i$  to virtual best solver (VBS)

## Application to SATzilla:

- ▶ all instances from 2011 SAT Competition:  
300 Application; 300 Crafted; 300 Random

## Application to SATzilla:

- ▶ all instances from 2011 SAT Competition:  
300 Application; 300 Crafted; 300 Random
- ▶ candidate solvers from 2011 SAT Competition:
  - ▶ for determining virtual best solver (VBS)  
and single best solver (SBS):  
all solvers from Phase 2 of competition:  
31 Application; 25 Crafted; 17 Random

## Application to SATzilla:

- ▶ all instances from 2011 SAT Competition:  
300 Application; 300 Crafted; 300 Random
- ▶ candidate solvers from 2011 SAT Competition:
  - ▶ for determining virtual best solver (VBS)  
and single best solver (SBS):  
all solvers from Phase 2 of competition:  
31 Application; 25 Crafted; 17 Random
  - ▶ for building SATzilla:  
all sequential, non-portfolio solvers from Phase 2:  
18 Application; 15 Crafted; 9 Random

## Application to SATzilla:

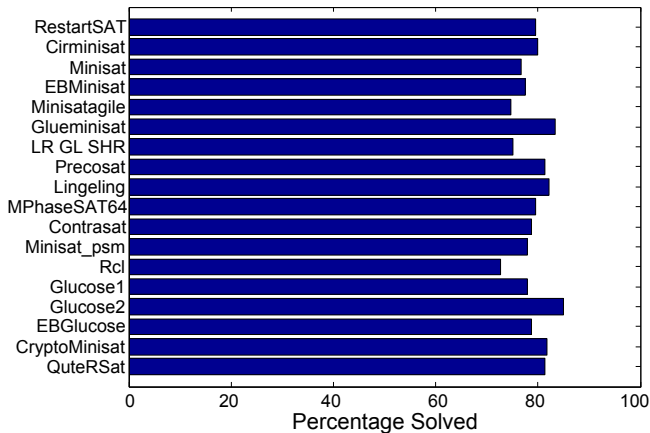
- ▶ all instances from 2011 SAT Competition:  
300 Application; 300 Crafted; 300 Random
- ▶ candidate solvers from 2011 SAT Competition:
  - ▶ for determining virtual best solver (VBS)  
and single best solver (SBS):  
all solvers from Phase 2 of competition:  
31 Application; 25 Crafted; 17 Random
  - ▶ for building SATzilla:  
all sequential, non-portfolio solvers from Phase 2:  
18 Application; 15 Crafted; 9 Random
- ▶ SATzilla assessed by 10-fold cross validation



## SATzilla 2011 Performance (Inst. Solved)

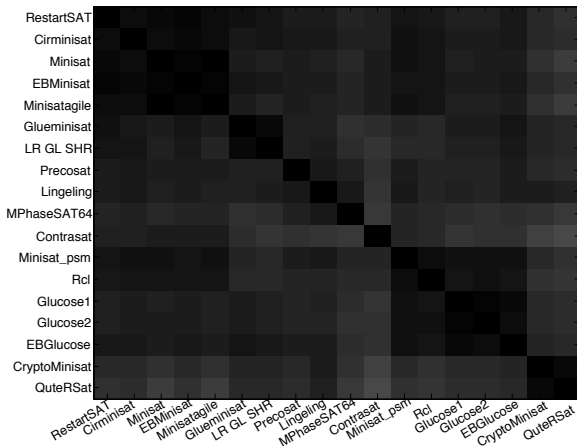
<i>Solver</i>	<i>Application</i>	<i>Crafted</i>	<i>Random</i>
VBS	84.7%	76.3%	82.2%
SATzilla 2011	75.3%	66.0%	80.8%
SATzilla 2009	70.3%	63.0%	80.3%
Gold medalist (SBS)	71.7%	54.3%	68.0%

## Performance of Individual Solvers Application



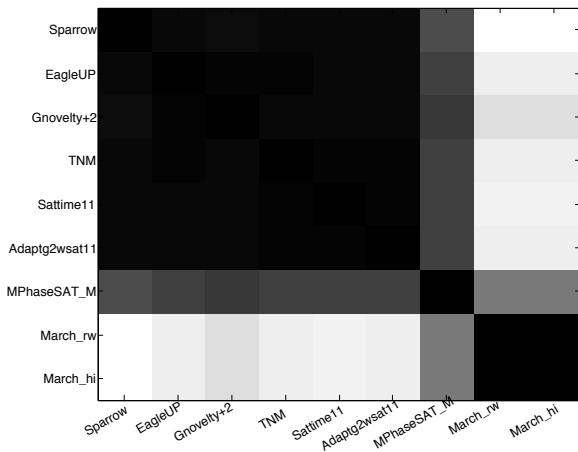
5000 CPU sec cutoff

## Correlation of Solver Performance Application



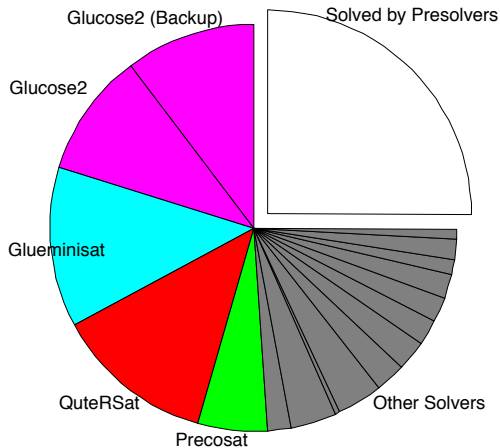
darker = higher Spearman correlation coefficient

## Correlation of Solver Performance Random

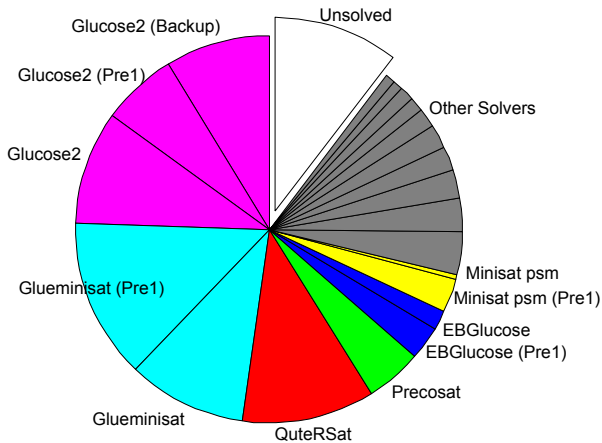


darker = higher Spearman correlation coefficient

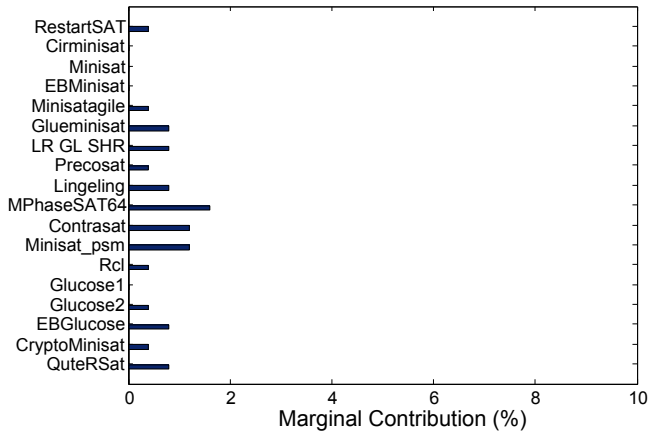
## Solver Selection Frequency in SATzilla 2011 Application



## Instances Solved by SATzilla 2011 Components Application



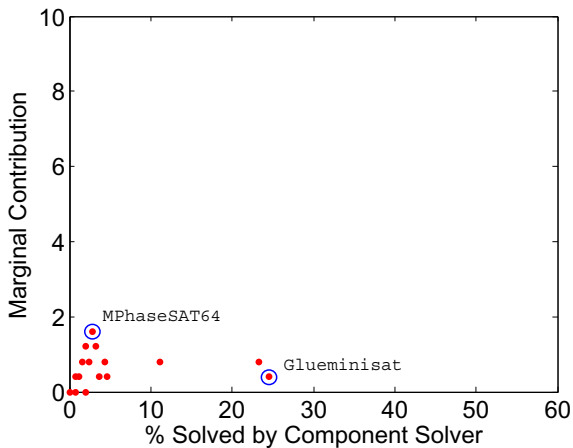
## Marginal Contribution of Components Application



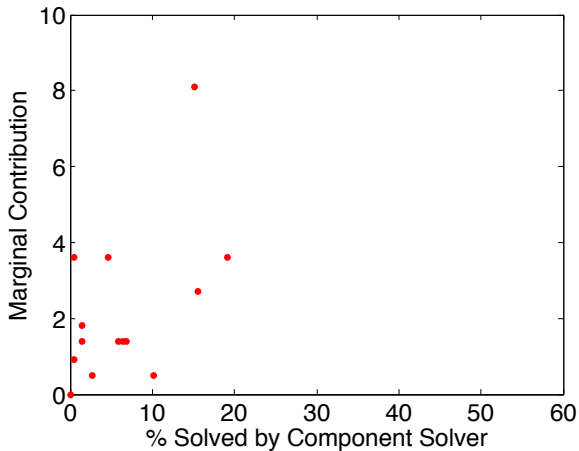




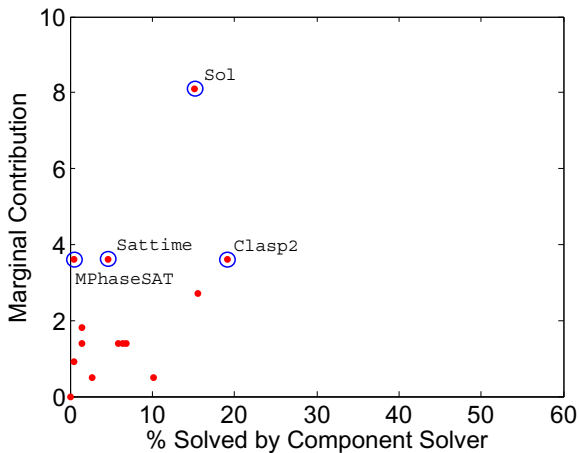
## Instances Solved vs Marginal Contribution of Components Application



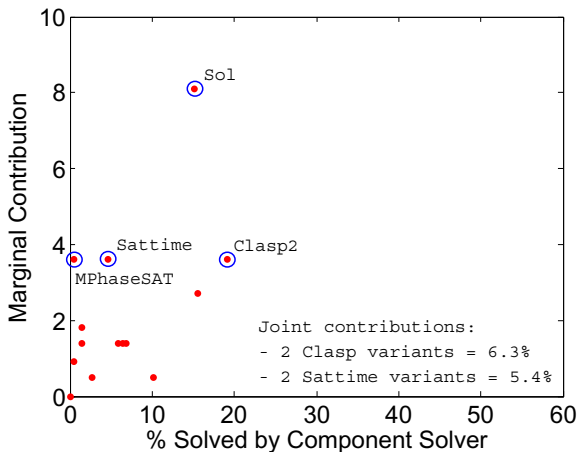
## Instances Solved vs Marginal Contribution of Components Crafted



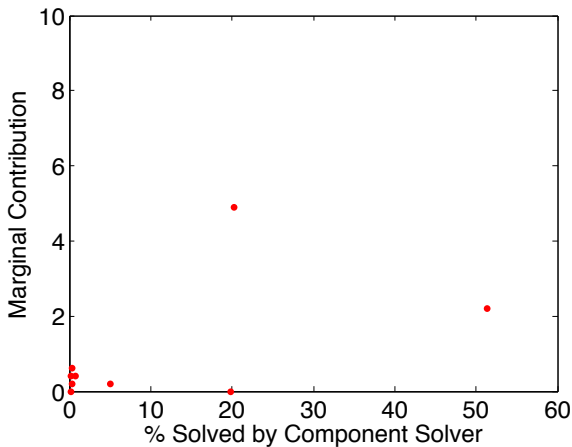
## Instances Solved vs Marginal Contribution of Components Crafted



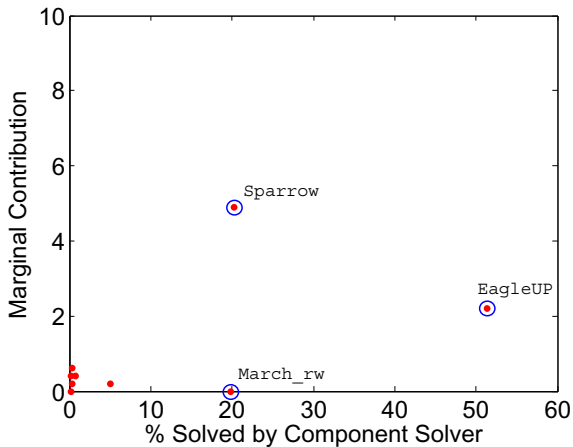
## Instances Solved vs Marginal Contribution of Components Crafted



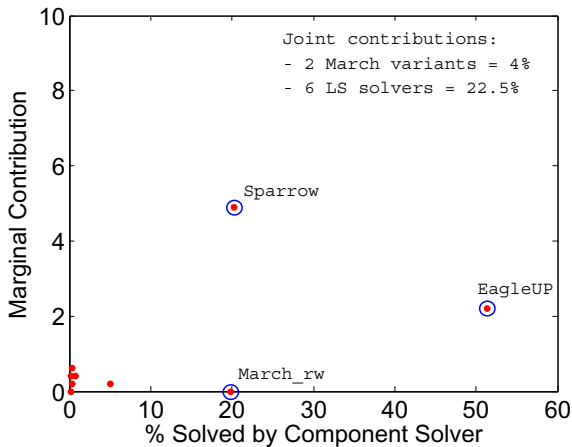
## Instances Solved vs Marginal Contribution of Components Random



## Instances Solved vs Marginal Contribution of Components Random



## Instances Solved vs Marginal Contribution of Components Random



## More nuanced analysis based on Shapley value:

↪ AAI-16 paper by Fréchet *et al.*



## **More nuanced analysis based on Shapley value:**

↪ AAI-16 paper by Fréchet *et al.*

## **Bias + correction in portfolio performance evaluation:**

↪ IJCAI-16 paper by Cameron, HH, Leyton-Brown  
Tue, 10:35, Gibson

## Leveraging parallelism

- ▶ design choices in parallel programs

(Hamadi, Jabhour, Sais 2009)

## Leveraging parallelism

- ▶ design choices in parallel programs  
(Hamadi, Jabhour, Sais 2009)
  
- ▶ deriving parallel programs from sequential sources  
~> concurrent execution of optimised designs  
(parallel portfolios)  
(Hoos, Leyton-Brown, Schaub, Schneider 2012)

## Leveraging parallelism

- ▶ design choices in parallel programs  
(Hamadi, Jabhour, Sais 2009)
- ▶ deriving parallel programs from sequential sources  
↔ concurrent execution of optimised designs  
(parallel portfolios)  
(Hoos, Leyton-Brown, Schaub, Schneider 2012)
- ▶ parallel design optimisers  
(e.g., Hutter, Hoos, Leyton-Brown 2012)

## Leveraging parallelism

- ▶ design choices in parallel programs  
(Hamadi, Jabhour, Sais 2009)
- ▶ deriving parallel programs from sequential sources  
↪ concurrent execution of optimised designs  
(parallel portfolios)  
(Hoos, Leyton-Brown, Schaub, Schneider 2012)
- ▶ parallel design optimisers  
(e.g., Hutter, Hoos, Leyton-Brown 2012)
- ▶ use of cloud resources (parallel runs of design optimisers, ...)  
(Geschwender, Hutter, Kotthoff, Malitsky, Hoos, Leyton-Brown 2014)

# Take-home Message

## Programming by Optimisation ...

- ▶ leverages computational power to construct better software

## Programming by Optimisation ...

- ▶ leverages computational power to construct better software
  
- ▶ enables creative thinking about design alternatives



## Programming by Optimisation ...

- ▶ leverages computational power to construct better software
- ▶ enables creative thinking about design alternatives
- ▶ produces better performing, more flexible software

## 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

## 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

## 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

## More Information:

- ▶ [www.prog-by-opt.net/Tutorials/IJCAI-16](http://www.prog-by-opt.net/Tutorials/IJCAI-16)

## More Information:

- ▶ [www.prog-by-opt.net/Tutorials/IJCAI-16](http://www.prog-by-opt.net/Tutorials/IJCAI-16)
- ▶ [www.prog-by-opt.net](http://www.prog-by-opt.net)

## More Information:

- ▶ [www.prog-by-opt.net/Tutorials/IJCAI-16](http://www.prog-by-opt.net/Tutorials/IJCAI-16)
- ▶ [www.prog-by-opt.net](http://www.prog-by-opt.net)
- ▶ PbO article in Communications of the ACM (Hoos 2012)

## More Information:

- ▶ [www.prog-by-opt.net/Tutorials/IJCAI-16](http://www.prog-by-opt.net/Tutorials/IJCAI-16)
- ▶ [www.prog-by-opt.net](http://www.prog-by-opt.net)
- ▶ PbO article in Communications of the ACM (Hoos 2012)
- ▶ Talk by Cameron *et al.*: Tue, 10:35, Gibson  
Invited talk by Leyton-Brown: Wed, 14:00, Ballroom East



## More Information:

- ▶ [www.prog-by-opt.net/Tutorials/IJCAI-16](http://www.prog-by-opt.net/Tutorials/IJCAI-16)
- ▶ [www.prog-by-opt.net](http://www.prog-by-opt.net)
- ▶ PbO article in Communications of the ACM (Hoos 2012)
- ▶ Talk by Cameron *et al.*: Tue, 10:35, Gibson  
Invited talk by Leyton-Brown: Wed, 14:00, Ballroom East
- ▶ Forthcoming book (Morgan & Claypool)

## More Information:

- ▶ [www.prog-by-opt.net/Tutorials/IJCAI-16](http://www.prog-by-opt.net/Tutorials/IJCAI-16)
- ▶ [www.prog-by-opt.net](http://www.prog-by-opt.net)
- ▶ PbO article in Communications of the ACM (Hoos 2012)
- ▶ Talk by Cameron *et al.*: Tue, 10:35, Gibson  
Invited talk by Leyton-Brown: Wed, 14:00, Ballroom East
- ▶ Forthcoming book (Morgan & Claypool)

## If PbO works for you:

- ▶ Make our day – let us know!

## More Information:

- ▶ [www.prog-by-opt.net/Tutorials/IJCAI-16](http://www.prog-by-opt.net/Tutorials/IJCAI-16)
- ▶ [www.prog-by-opt.net](http://www.prog-by-opt.net)
- ▶ PbO article in Communications of the ACM (Hoos 2012)
- ▶ Talk by Cameron *et al.*: Tue, 10:35, Gibson  
Invited talk by Leyton-Brown: Wed, 14:00, Ballroom East
- ▶ Forthcoming book (Morgan & Claypool)

## If PbO works for you:

- ▶ Make our day – let us know!
- ▶ Share the joy – tell everyone else!