Automatic Algorithm Configuration based on Local Search

Frank Hutter[1] Holger Hoos[1] Thomas Stützle[2]

[1]Department of Computer Science
University of British Columbia
Canada

[2]IRIDIA
Université Libre de Bruxelles
Belgium

Real-world example for algorithm configuration: Tree search for SAT-encoded software verification

- New DPLL-type SAT solver (Spear)
  - Variable/value heuristics, clause learning, restarts, ...

Real-world example for algorithm configuration: Tree search for SAT-encoded software verification

- New DPLL-type SAT solver (SPEAR)
  - Variable/value heuristics, clause learning, restarts, ...
  - 26 user-specifiable parameters:
    7 categorical, 3 boolean, 12 continuous, 4 integer parameters

Real-world example for algorithm configuration: Tree search for SAT-encoded software verification

- New DPLL-type SAT solver (SPEAR)
  - Variable/value heuristics, clause learning, restarts, ...
  - 26 user-specifiable parameters:
    7 categorical, 3 boolean, 12 continuous, 4 integer parameters

- Minimize expected run-time

Real-world example for algorithm configuration: Tree search for SAT-encoded software verification

- New DPLL-type SAT solver (SPEAR)
  - Variable/value heuristics, clause learning, restarts, ...
  - 26 user-specifiable parameters:
    7 categorical, 3 boolean, 12 continuous, 4 integer parameters

- Minimize expected run-time

- Problems:
  - Huge variation in runtime (with default setting):
    <1 second for some instances
    >1 day for others
  - Good performance on a few instances does not generalise well
  - Many possible configurations (8.34×10^{17} after discretization)

Standard algorithm configuration approach

- Choose a “representative” benchmark set for tuning

Standard algorithm configuration approach

- Choose a "representative" benchmark set for tuning
- Perform iterative manual tuning:

  start with some parameter configuration
  repeat
      modify a single parameter
      if *results on tuning set improve* **then**
          keep new configuration
  until *no more improvement possible (or "good enough")*

Problems of standard approach

- Slow and tedious, requires significant human time

Problems of standard approach

- Slow and tedious, requires significant human time
- Not guaranteed to find global optimum
  - Hill climbing $\rightsquigarrow$ local minimum only

Problems of standard approach

- Slow and tedious, requires significant human time
- Not guaranteed to find global optimum
  - Hill climbing $\rightsquigarrow$ local minimum only
- "Representative" benchmark set may not be representative
  - Constraints on tuning time
    $\rightsquigarrow$ typically only few instances
    $\rightsquigarrow$ typically fairly easy instances

Problems of standard approach

- Slow and tedious, requires significant human time
- Not guaranteed to find global optimum
  - Hill climbing $\rightsquigarrow$ local minimum only
- "Representative" benchmark set may not be representative
  - Constraints on tuning time
    $\rightsquigarrow$ typically only few instances
    $\rightsquigarrow$ typically fairly easy instances

Solution:

- Automate process
- Use more powerful search method

Related work

- Search approaches
  [Minton 1993, 1996], [Hutter 2004], [Cavazos & O'Boyle 2005], [Adenso-Diaz & Laguna 2006], [Audet & Orban 2006]

Related work

- Search approaches
  [Minton 1993, 1996], [Hutter 2004], [Cavazos & O'Boyle 2005], [Adenso-Diaz & Laguna 2006], [Audet & Orban 2006]

- Racing algorithms/Bandit solvers
  [Birattari et al. 2002], [Smith et al. 2004 – 2007]

Related work

- Search approaches
  [Minton 1993, 1996], [Hutter 2004], [Cavazos & O'Boyle 2005], [Adenso-Diaz & Laguna 2006], [Audet & Orban 2006]
- Racing algorithms/Bandit solvers
  [Birattari et al. 2002], [Smith et al. 2004 – 2007]
- Stochastic Optimisation [Kiefer & Wolfowitz 1952], [Spall 1987]

Related work

- Search approaches
  [Minton 1993, 1996], [Hutter 2004], [Cavazos & O'Boyle 2005], [Adenso-Diaz & Laguna 2006], [Audet & Orban 2006]
- Racing algorithms/Bandit solvers
  [Birattari et al. 2002], [Smith et al. 2004 – 2007]
- Stochastic Optimisation [Kiefer & Wolfowitz 1952], [Spall 1987]
- Learning approaches
  - Regression trees [Bartz-Beielstein et al. 2004]
  - Response surface models, DACE
    [Bartz-Beielstein et al. 2004–2006]

Related work

- Search approaches
  [Minton 1993, 1996], [Hutter 2004], [Cavazos & O'Boyle 2005], [Adenso-Diaz & Laguna 2006], [Audet & Orban 2006]

- Racing algorithms/Bandit solvers
  [Birattari et al. 2002], [Smith et al. 2004 – 2007]

- Stochastic Optimisation [Kiefer & Wolfowitz 1952], [Spall 1987]

- Learning approaches
  - Regression trees [Bartz-Beielstein et al. 2004]
  - Response surface models, DACE
    [Bartz-Beielstein et al. 2004–2006]

- Lots of work on per-instance tuning / reactive search
  $\rightsquigarrow$ orthogonal to the approach followed here

Outline

The ParamILS framework

ILS in parameter configuration space (ParamILS):

Choose initial parameter configuration θ

Perform *subsidiary local search* on θ

The ParamILS framework

ILS in parameter configuration space (ParamILS):

Choose initial parameter configuration θ
Perform *subsidiary local search* on θ
While tuning time left:

$\theta' := \theta$
perform *perturbation* on θ
perform *subsidiary local search* on θ

The ParamILS framework

ILS in parameter configuration space (ParamILS):

Choose initial parameter configuration θ
Perform *subsidiary local search* on θ
While tuning time left:

> $\theta' := \theta$
> perform *perturbation* on θ
> perform *subsidiary local search* on θ
>
> based on *acceptance criterion*,
> keep θ or revert to $\theta := \theta'$

The ParamILS framework

ILS in parameter configuration space (ParamILS):

Choose initial parameter configuration θ
Perform *subsidiary local search* on θ
While tuning time left:

$\theta' := \theta$
perform *perturbation* on θ
perform *subsidiary local search* on θ

based on *acceptance criterion*,
keep θ or revert to $\theta := \theta'$

with probability $p_{restart}$ randomly pick new θ

$\rightsquigarrow$ Performs **biased random walk over local optima**

Details on ParamILS:

- Initialisation: pick *best* of default & R random configurations

Details on ParamILS:

- Initialisation: pick *best* of default & R random configurations
- Subsidiary local search: iterative first improvement, change one parameter in each step

Details on ParamILS:

- Initialisation: pick *best* of default & R random configurations
- Subsidiary local search: iterative first improvement, change one parameter in each step
- Perturbation: change s randomly chosen parameters

Details on ParamILS:

- Initialisation: pick *best* of default & R random configurations
- Subsidiary local search: iterative first improvement, change one parameter in each step
- Perturbation: change s randomly chosen parameters
- Acceptance criterion: always select *better* local optimum

Evaluation of a parameter configuration θ (based on N runs)

- Sample N instances from given set (with repetitions)

Evaluation of a parameter configuration θ (based on N runs)

- Sample N instances from given set (with repetitions)
- For each of the N instances:
  - Execute algorithm with configuration θ
  - Record scalar cost of the run
    (user-defined: *e.g.* run-time, solution quality, ...)

Evaluation of a parameter configuration θ (based on N runs)

- Sample N instances from given set (with repetitions)
- For each of the N instances:
  - Execute algorithm with configuration θ
  - Record scalar cost of the run
    (user-defined: *e.g.* run-time, solution quality, ...)
- Compute *scalar statistic* $\hat{c}_N(\theta)$ of the N costs
  (user-defined: *e.g.* empirical mean, median, ...)

The BasicILS(N) algorithm

- Use a fixed number of N runs to evaluate each configuration θ

The BasicILS(N) algorithm

- Use a fixed number of N runs to evaluate each configuration θ

Question: How to choose number of runs N?

- Too many
  - $\rightsquigarrow$ evaluating a configuration is very expensive
  - $\rightsquigarrow$ optimisation process is very slow

The BasicILS(N) algorithm

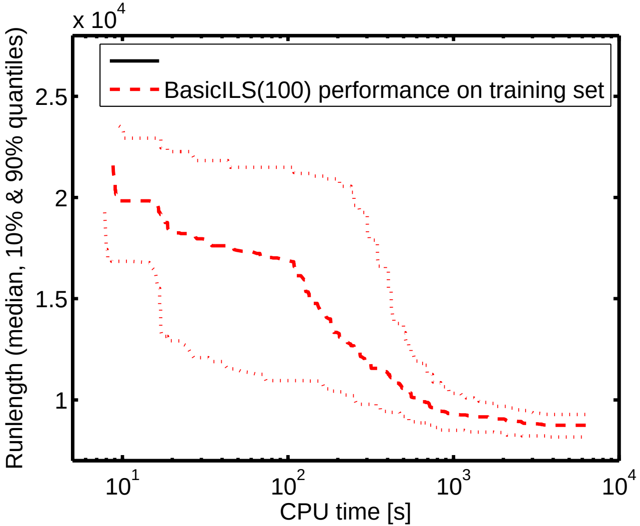
- Use a fixed number of N runs to evaluate each configuration θ

Question: How to choose number of runs N?

- Too many
  - $\rightsquigarrow$ evaluating a configuration is very expensive
  - $\rightsquigarrow$ optimisation process is very slow
- Too few
  - $\rightsquigarrow$ very noisy approximations $\hat{c}_N(\theta)$
  - $\rightsquigarrow$ poor generalisation to independent test runs

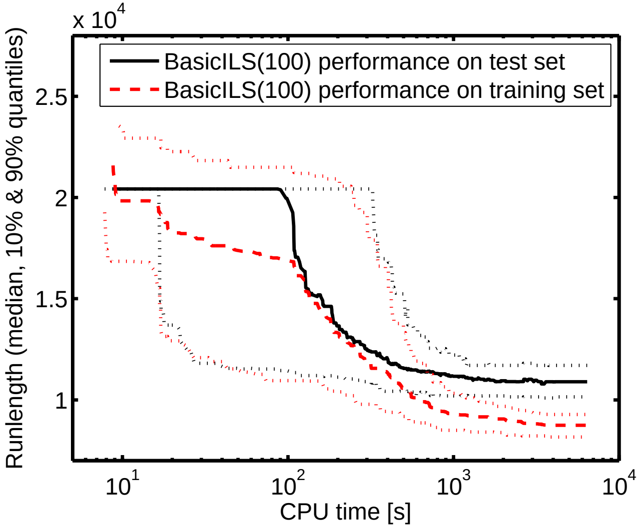
Generalisation to independent test set,large N (N=100)

(SAPS on quasigroups with holes)



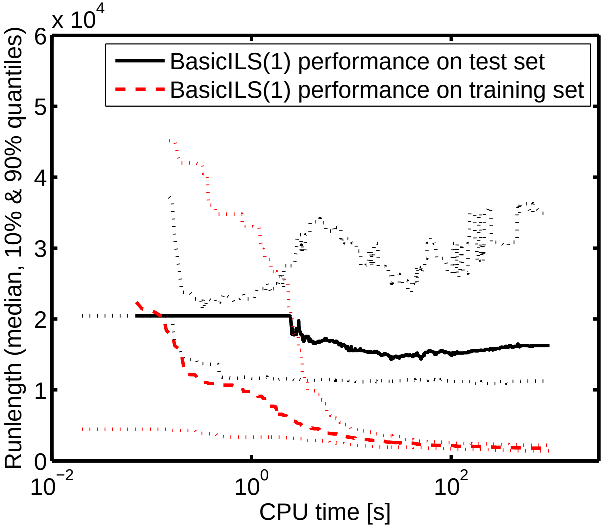
Generalisation to independent test set,large N (N=100)

(SAPS on quasigroups with holes)



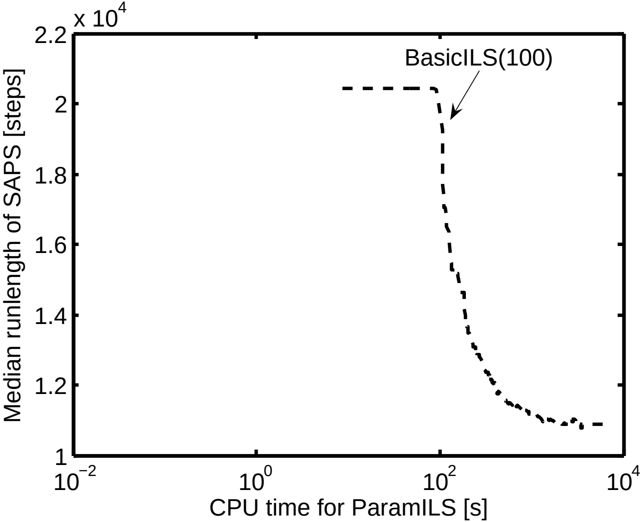
Generalisation to independent test set, small N (N=1)

(SAPS on quasigroups with holes)



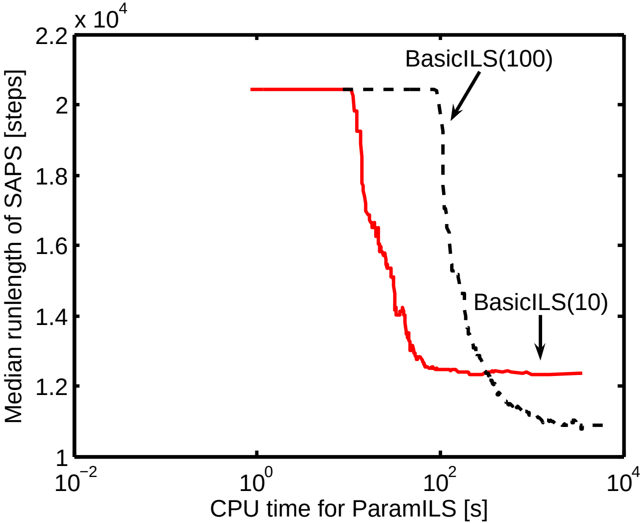
Test performance of BasicILS with different N

(SAPS on quasigroups with holes)



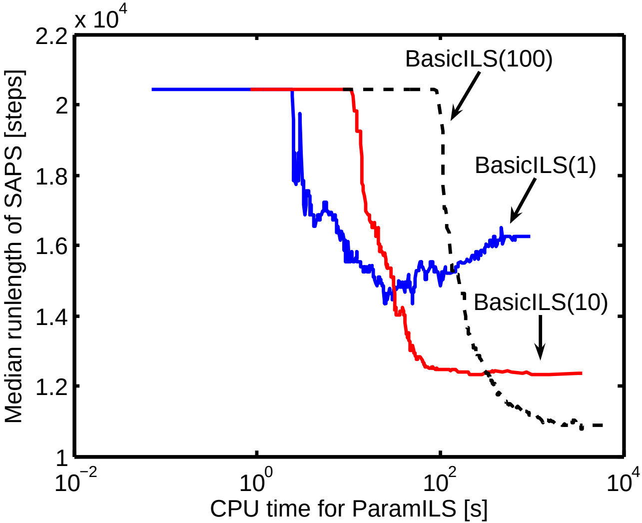
Test performance of BasicILS with different N

(SAPS on quasigroups with holes)



Test performance of BasicILS with different N

(SAPS on quasigroups with holes)



The FocusedILS algorithm

- Use different numbers of runs, $N(\theta)$, for each configuration θ

The FocusedILS algorithm

- Use different numbers of runs, $N(\theta)$, for each configuration θ
- **Idea:** Use high $N(\theta)$ only for good θ
  - start with $N(\theta) = 0$ for all θ
  - increment $N(\theta)$ whenever θ is visited
  - additional runs upon finding new, better configuration θ

The FocusedILS algorithm

- Use different numbers of runs, $N(\theta)$, for each configuration θ
- **Idea:** Use high $N(\theta)$ only for good θ
  - start with $N(\theta) = 0$ for all θ
  - increment $N(\theta)$ whenever θ is visited
  - additional runs upon finding new, better configuration θ

Theorem:
As number of FocusedILS iterations $\rightarrow \infty$,
it converges to true optimal configuration θ^*

The FocusedILS algorithm

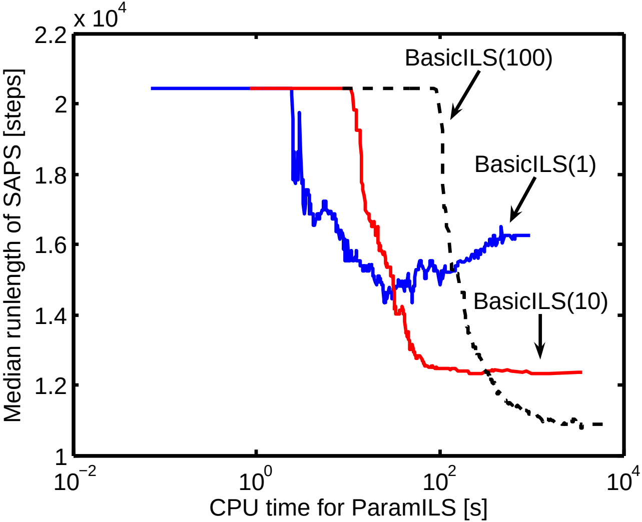
- Use different numbers of runs, $N(\theta)$, for each configuration θ
- **Idea:** Use high $N(\theta)$ only for good θ
  - start with $N(\theta) = 0$ for all θ
  - increment $N(\theta)$ whenever θ is visited
  - additional runs upon finding new, better configuration θ

Theorem:
As number of FocusedILS iterations $\rightarrow \infty$,
it converges to true optimal configuration θ^*

- Key ideas in proof
  1. For $N(\theta) \rightarrow \infty$, $\hat{c}_N(\theta) \rightarrow c(\theta)$
  2. Underlying ILS eventually reaches any configuration θ.

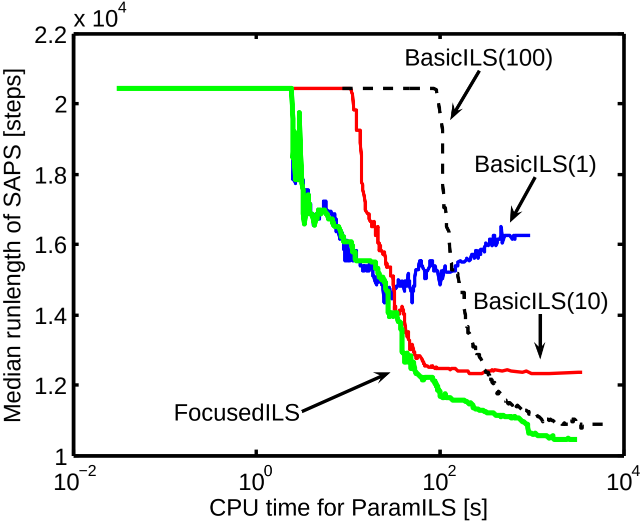
Performance of FocusedILS vs BasicILS

(Test performance of SAPS on quasigroups with holes)



Performance of FocusedILS vs BasicILS

(Test performance of SAPS on quasigroups with holes)



Sample applications and performance results

Comparison against CALIBRA [Adenso-Diaz & Laguna 2006]

- CALIBRA: limited to 5 continuous/integer parameters
- ParamILS better results with same tuning time

Sample applications and performance results

Comparison against CALIBRA [Adenso-Diaz & Laguna 2006]

- CALIBRA: limited to 5 continuous/integer parameters
- ParamILS better results with same tuning time

Scenario	Metric	Default	FocusedILS	BasicILS(100)	CALIBRA(100)
SAPS on GC	Runtime	5.60 s	**0.043 ± 0.005**	0.046 ± 0.01	0.053 ± 0.010

Sample applications and performance results

Comparison against CALIBRA [Adenso-Diaz & Laguna 2006]

- CALIBRA: limited to 5 continuous/integer parameters
- ParamILS better results with same tuning time

Scenario	Metric	Default	FocusedILS	BasicILS(100)	CALIBRA(100)
SAPS on GC	Runtime	5.60 s	$\mathbf{0.043 \pm 0.005}$	0.046 ± 0.01	0.053 ± 0.010
GLS$^+$ for MPE	Approx. error	$\epsilon = 1.81$	$\mathbf{0.949 \pm 0.0001}$	0.951 ± 0.004	1.234 ± 0.492

Sample applications and performance results

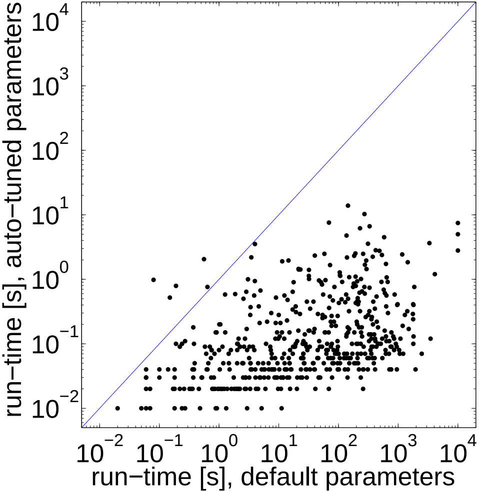
Comparison against CALIBRA [Adenso-Diaz & Laguna 2006]

- CALIBRA: limited to 5 continuous/integer parameters
- ParamILS better results with same tuning time

Scenario	Metric	Default	FocusedILS	BasicILS(100)	CALIBRA(100)
SAPS on GC	Runtime	5.60 s	$\mathbf{0.043 \pm 0.005}$	0.046 ± 0.01	0.053 ± 0.010
GLS$^+$ for MPE	Approx. error	$\epsilon = 1.81$	$\mathbf{0.949 \pm 0.0001}$	0.951 ± 0.004	1.234 ± 0.492
SAT4J on GC	Runtime	7.02 s	$\mathbf{0.65 \pm 0.2}$	1.19 ± 0.58	(too many param.)

Speedup obtained by automated tuning

(SAPS default *vs* tuned on graph colouring, test set performance)



Two "real-world" applications

- New DPLL-type SAT solver SPEAR
  - 26 parameters
  - Software verification: 500-fold speedup (won QB_FQ category in SMT'07 competition)
  - Hardware verification: 4.5-fold speedup
  - ⇝ New state of the art for those instances
  - ⇝ [Hutter, Babić, Hoos & Hu: FMCAD '07 (to appear)]

Two "real-world" applications

- New DPLL-type SAT solver SPEAR
  - 26 parameters
  - Software verification: 500-fold speedup (won QB_FQ category in SMT'07 competition)
  - Hardware verification: 4.5-fold speedup
  - $\rightsquigarrow$ New state of the art for those instances
  - $\rightsquigarrow$ [Hutter, Babić, Hoos & Hu: FMCAD '07 (to appear)]

- New replica exchange Monte Carlo algorithm for protein structure prediction
  - 3 parameters
  - 2-fold improvement
  - $\rightsquigarrow$ New state of the art for 2D/3D protein structure prediction
  - $\rightsquigarrow$ [Thachuk, Shmygelska & Hoos: BMC Bioinformatics '07 (to appear)]

Conclusions

- ParamILS: Simple and efficient framework for automatic parameter optimization
  - Arbitrary number and types of parameters
  - User-defined objective function

Conclusions

- ParamILS: Simple and efficient framework for automatic parameter optimization
  - Arbitrary number and types of parameters
  - User-defined objective function

- FocusedILS:
  - Converges provably towards optimal configuration
  - Excellent performance in practice (outperforms BasicILS, CALIBRA)

Conclusions

- ParamILS: Simple and efficient framework for automatic parameter optimization
  - Arbitrary number and types of parameters
  - User-defined objective function
- FocusedILS:
  - Converges provably towards optimal configuration
  - Excellent performance in practice (outperforms BasicILS, CALIBRA)
- Huge speedups:
  - $\approx 100\times$ for SAPS (local search) on graph colouring
  - $\approx 500\times$ for SPEAR (tree search) on software verification

Conclusions

- ParamILS: Simple and efficient framework for automatic parameter optimization
  - Arbitrary number and types of parameters
  - User-defined objective function
- FocusedILS:
  - Converges provably towards optimal configuration
  - Excellent performance in practice (outperforms BasicILS, CALIBRA)
- Huge speedups:
  - $\approx 100\times$ for SAPS (local search) on graph colouring
  - $\approx 500\times$ for SPEAR (tree search) on software verification
- Publically available at: `http://www.cs.ubc.ca/labs/beta/Projects/ParamILS`

Future work

- Continuous parameters (currently discretised)

Future work

- Continuous parameters (currently discretised)
- Statistical tests (cf. racing algorithms)

Future work

- Continuous parameters (currently discretised)
- Statistical tests (cf. racing algorithms)
- Learning approaches, sequential design of experiments

Future work

- Continuous parameters (currently discretised)
- Statistical tests (cf. racing algorithms)
- Learning approaches, sequential design of experiments
- Per-instance tuning

Future work

- Continuous parameters (currently discretised)
- Statistical tests (cf. racing algorithms)
- Learning approaches, sequential design of experiments
- Per-instance tuning
- Automatic algorithm design