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

Motivation for automatic algorithm configuration

- Want to design 'best' algorithm to solve a problem
  - Many design choices need to be made
  - Some choices deferred to later: free parameters of algorithm
  - Set parameters to maximise empirical performance

Motivation for automatic algorithm configuration

- Want to design 'best' algorithm to solve a problem
  - Many design choices need to be made
  - Some choices deferred to later: free parameters of algorithm
  - Set parameters to maximise empirical performance
- Finding best parameter configuration is non-trivial
  - Many parameters, discrete & continuous
  - Dependencies between parameters
  - Many test instances needed to generalize
  - Many runs per instance needed for randomised algorithms

Motivation for automatic algorithm configuration

- Want to design 'best' algorithm to solve a problem
  - Many design choices need to be made
  - Some choices deferred to later: free parameters of algorithm
  - Set parameters to maximise empirical performance
- Finding best parameter configuration is non-trivial
  - Many parameters, discrete & continuous
  - Dependencies between parameters
  - Many test instances needed to generalize
  - Many runs per instance needed for randomised algorithms
- Algorithm configuration / tuning still often done manually, using ad-hoc methods
  - $\rightsquigarrow$ tedious and time-consuming, sub-optimal results
  - $\rightsquigarrow$ big incentive for automation

Real-world example:

- Application: Solving SAT-encoded software verification problems

Real-world example:

- Application: Solving SAT-encoded software verification problems
- Tune 26 parameters of new DPLL-type SAT solver (SPEAR)
  - 7 categorical, 3 boolean, 12 continuous, 4 integer parameters
  - Variable/value heuristics, clause learning, restarts, ...

Real-world example:

- Application: Solving SAT-encoded software verification problems
- Tune 26 parameters of new DPLL-type SAT solver (SPEAR)
  - 7 categorical, 3 boolean, 12 continuous, 4 integer parameters
  - Variable/value heuristics, clause learning, restarts, ...
- Minimize expected run-time

Real-world example:

- Application: Solving SAT-encoded software verification problems
- Tune 26 parameters of new DPLL-type SAT solver (SPEAR)
  - 7 categorical, 3 boolean, 12 continuous, 4 integer parameters
  - Variable/value heuristics, clause learning, restarts, ...
- Minimize expected run-time
- Problems:
  - default settings $\rightsquigarrow \approx$ 300 seconds / run
  - good performance on a few instances may not generalise

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:

- cost for evaluating configuration depends on number and hardness of problem instances
- constraints on tuning time (per iteration and overall)
  $\rightsquigarrow$ typically use few and fairly easy instances, few iterations

Problems:

- cost for evaluating configuration depends on number and hardness of problem instances
- constraints on tuning time (per iteration and overall)
  $\rightsquigarrow$ typically use few and fairly easy instances, few iterations
- manual search = iterative improvement (hill climbing)
  $\rightsquigarrow$ finds local optimum only

Problems:

- cost for evaluating configuration depends on number and hardness of problem instances
- constraints on tuning time (per iteration and overall)
  $\rightsquigarrow$ typically use few and fairly easy instances, few iterations
- manual search = iterative improvement (hill climbing)
  $\rightsquigarrow$ finds local optimum only
- slow and tedious, requires significant human time
  $\rightsquigarrow$ procedure often performed in ad-hoc way

Problems:

- cost for evaluating configuration depends on number and hardness of problem instances
- constraints on tuning time (per iteration and overall)
  $\rightsquigarrow$ typically use few and fairly easy instances, few iterations
- manual search = iterative improvement (hill climbing)
  $\rightsquigarrow$ finds local optimum only
- slow and tedious, requires significant human time
  $\rightsquigarrow$ procedure often performed in ad-hoc way

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, 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, 2006]
- 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, 2006]
- 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 / reactive tuning
  $\rightsquigarrow$ orthogonal to the approach followed here

Outline

1. Introduction
2. Iterated local search over parameter configurations
3. The FocusedILS algorithm
4. Sample applications and performance results
5. Conclusions and future work

ILS in parameter configuration space (ParamILS):

choose initial parameter configuration θ
perform *subsidiary local search* on s

ILS in parameter configuration space (ParamILS):

choose initial parameter configuration θ
perform *subsidiary local search* on s
While tuning time left:
  $\theta' := \theta$
  perform *perturbation* on θ
  perform *subsidiary local search* on θ

ILS in parameter configuration space (ParamILS):

choose initial parameter configuration θ
perform *subsidiary local search* on s
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 $\theta := \theta'$

ILS in parameter configuration space (ParamILS):

choose initial parameter configuration θ
perform *subsidiary local search* on s
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 minima**

Details on ParamILS:

- subsidiary local search: iterative first improvement, change on parameter in each step

Details on ParamILS:

- subsidiary local search: iterative first improvement, change on parameter in each step
- perturbation: change 3 randomly chosen parameters

Details on ParamILS:

- subsidiary local search: iterative first improvement, change on parameter in each step
- perturbation: change 3 randomly chosen parameters
- acceptance criterion: always select *better* configuration

Details on ParamILS:

- subsidiary local search: iterative first improvement, change on parameter in each step
- perturbation: change 3 randomly chosen parameters
- acceptance criterion: always select *better* configuration
- initialisation: pick *best* of default + R random configurations

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.

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 each of the N runs: $sc_1, \ldots, sc_n$ (run-time, solution quality, ...)
  $\rightsquigarrow$ empirical cost distribution $\widehat{CD}$

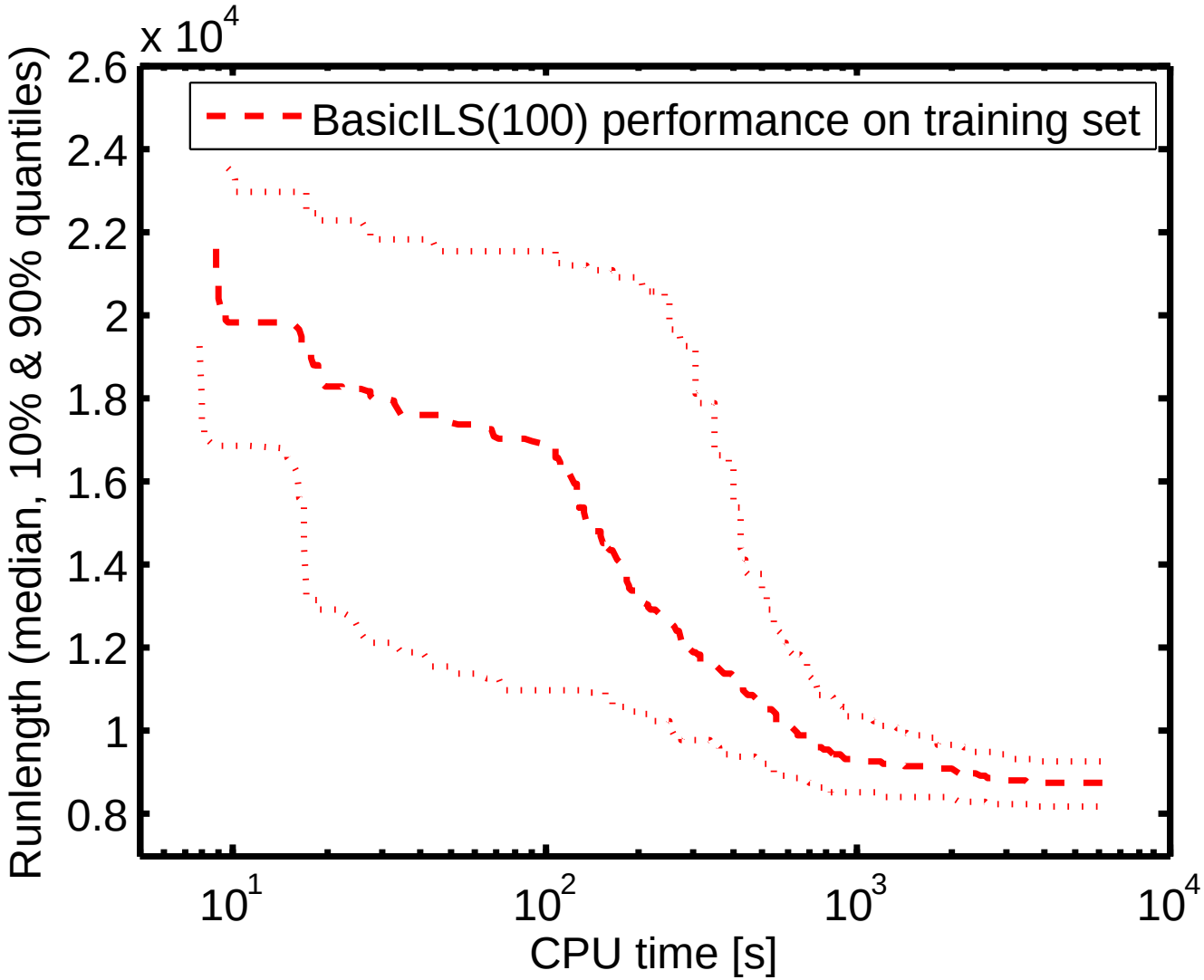
Evaluation of a parameter configuration θ (based on N runs)

- Sample N instances from given set (with repetitions)
- For each of the N instances, execute algorithm with configuration θ.
- Record scalar cost of each of the N runs: $sc_1, \ldots, sc_n$ (run-time, solution quality, ...)
  $\rightsquigarrow$ empirical cost distribution $\widehat{CD}$
- Compute scalar statistic $\hat{c}_N(\theta)$ of $\widehat{CD}$ (mean, median, ...)

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 each of the N runs: $sc_1, \ldots, sc_n$ (run-time, solution quality, ...)
  $\rightsquigarrow$ empirical cost distribution $\widehat{CD}$
- Compute scalar statistic $\hat{c}_N(\theta)$ of $\widehat{CD}$ (mean, median, ...)
- Note: For large N, $\hat{c}_N(\theta)$ approaches true cost $c(\theta)$

Solution quality over time achieved by ParamILS



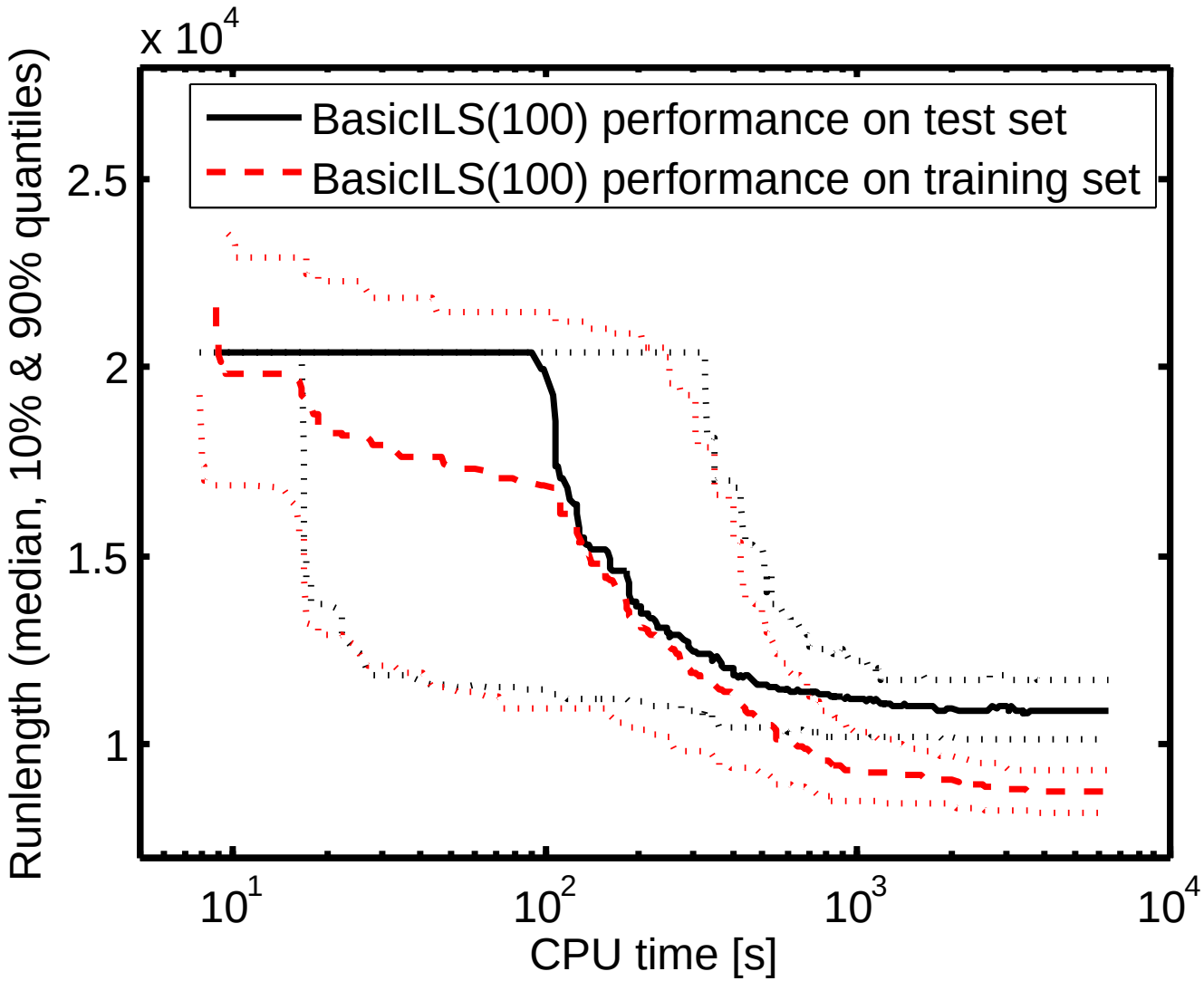
Question: How many runs/instances?

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

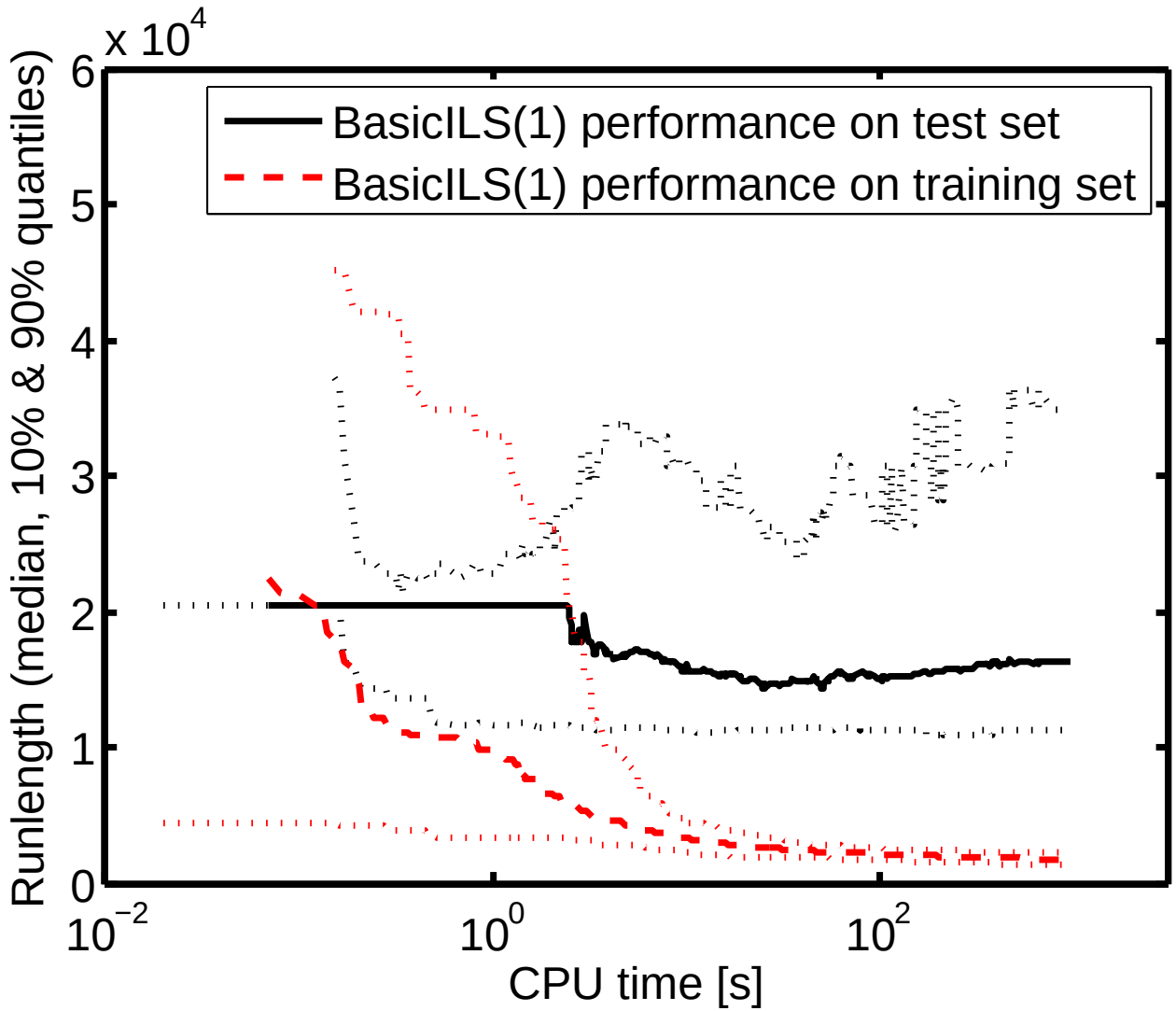
Question: How many runs/instances?

- 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 runs (1)



Generalisation to independent test runs (2)



The FocusedILS algorithm

- Given a budget of available CPU time for tuning, use different numbers of runs, $N(\theta)$, for each configuration θ

The FocusedILS algorithm

- Given a budget of available CPU time for tuning, 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

Theorem:

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

The FocusedILS algorithm

Theorem:

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

Key ideas in proof:

1. For $N(\theta), N(\theta') \rightarrow \infty$, comparisons between θ, θ' become precise.

The FocusedILS algorithm

Theorem:

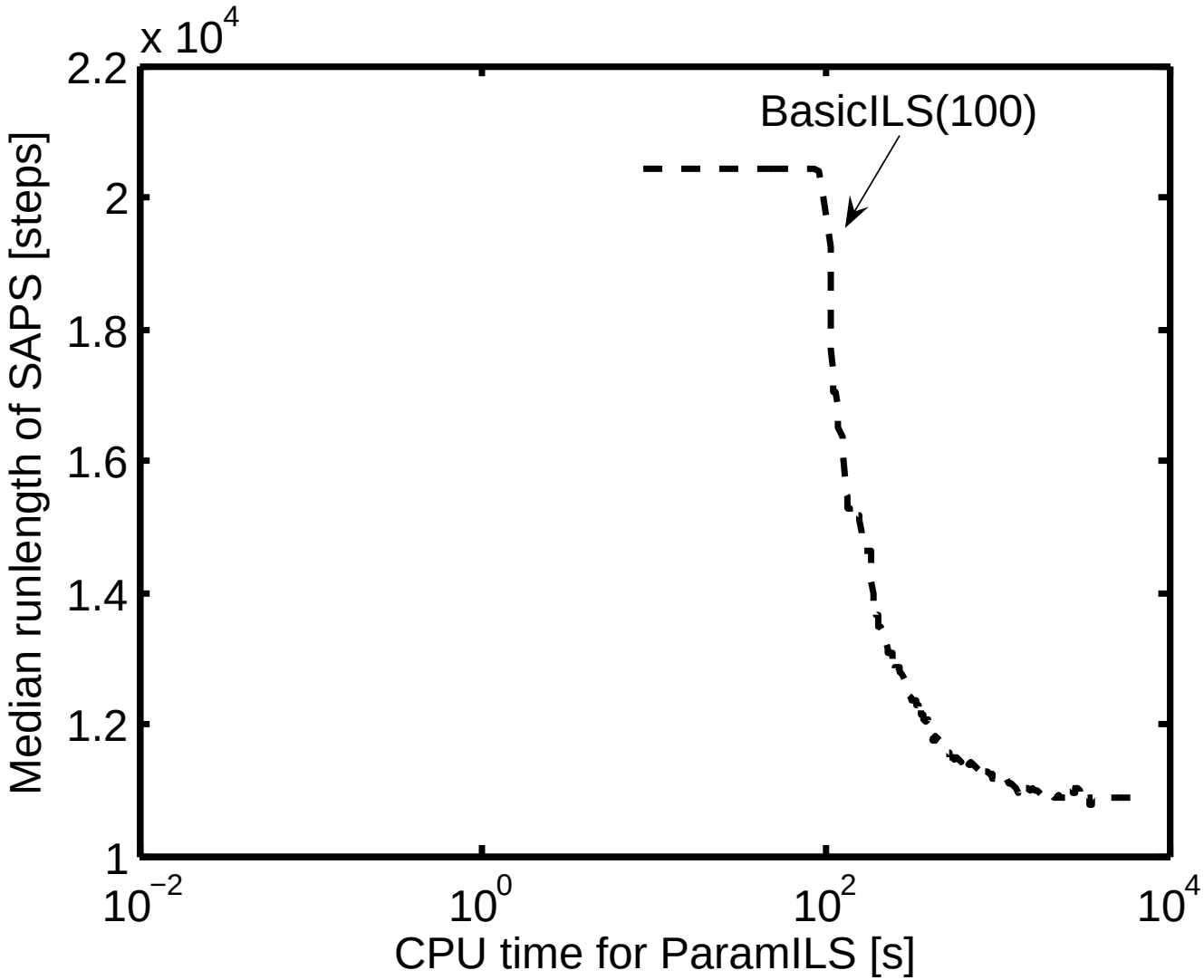
As number of FocusedILS iterations $\rightarrow \infty$,
it converges to true optimal configuration θ^*

Key ideas in proof:

1. For $N(\theta), N(\theta') \rightarrow \infty$, comparisons between θ, θ' become precise.
2. Underlying ILS eventually reaches any configuration θ.

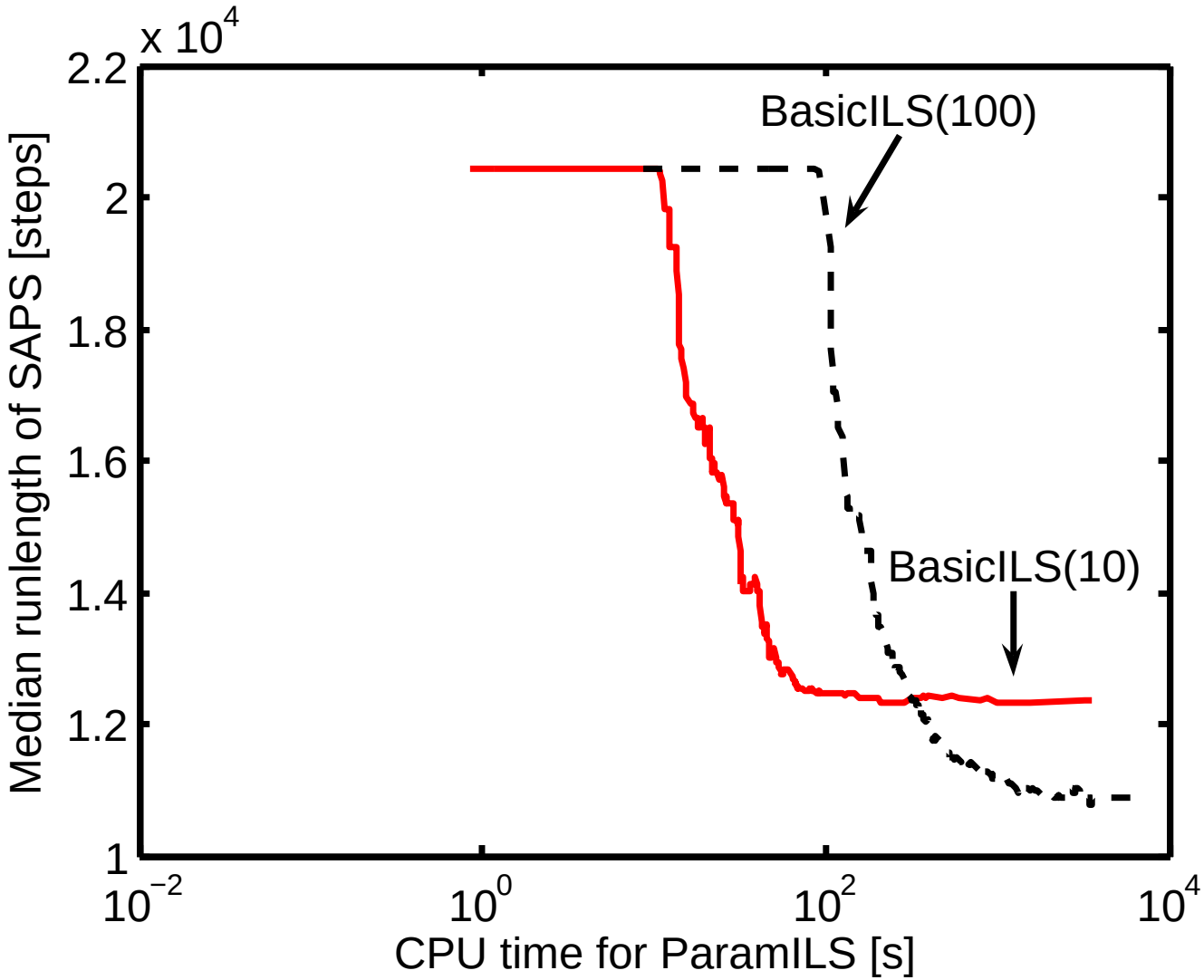
Performance of FocusedILS vs BasicILS

(Test performance on SAPS-QWH)



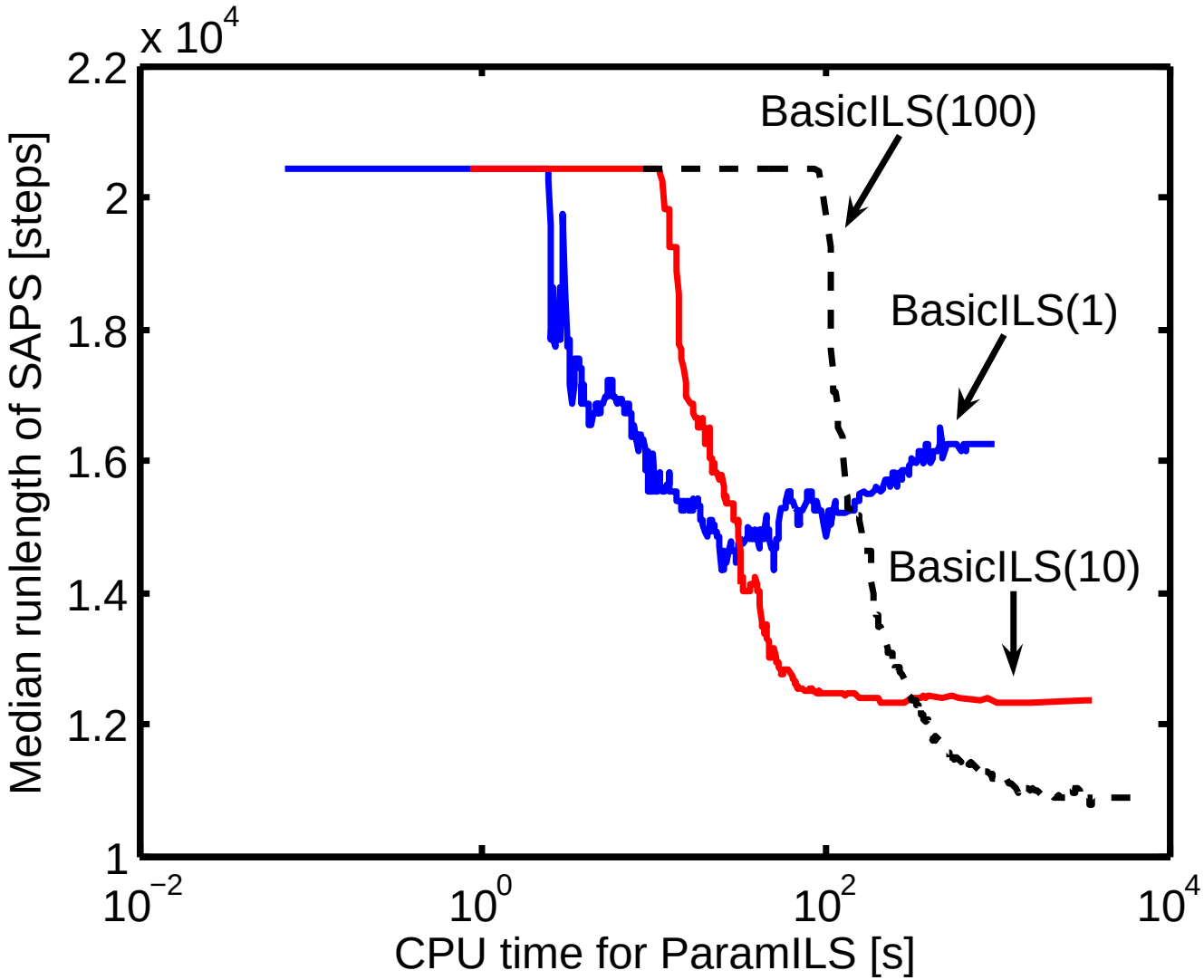
Performance of FocusedILS vs BasicILS

(Test performance on SAPS-QWH)



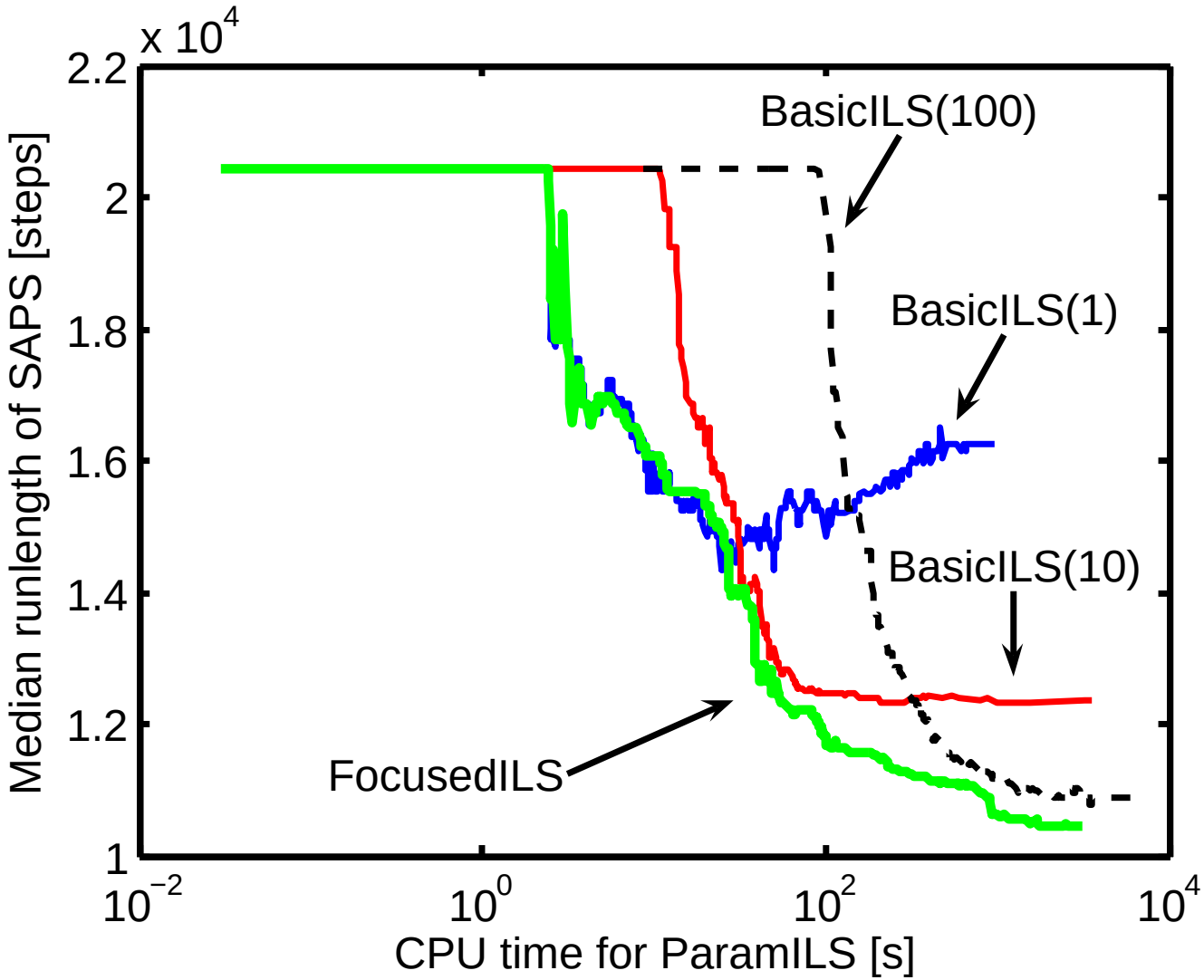
Performance of FocusedILS vs BasicILS

(Test performance on SAPS-QWH)



Performance of FocusedILS vs BasicILS

(Test performance on SAPS-QWH)



Sample applications and performance results

Experiment: Comparison to CALIBRA

Scenario	Default	CALIBRA(100)	BasicILS(100)	FocusedILS
GLS-GRID	$\epsilon = 1.81$	1.234 ± 0.492	0.951 ± 0.004	$\mathbf{0.949 \pm 0.0001}$

Sample applications and performance results

Experiment: Comparison to CALIBRA

Scenario	Default	CALIBRA(100)	BasicILS(100)	FocusedILS
GLS-GRID	$\epsilon = 1.81$	1.234 ± 0.492	0.951 ± 0.004	$\mathbf{0.949 \pm 0.0001}$
SAPS-QWH	$85.5K$ steps	$10.7K \pm 1.1K$	$10.9K \pm 0.6K$	$\mathbf{10.6K \pm 0.5K}$

Sample applications and performance results

Experiment: Comparison to CALIBRA

Scenario	Default	CALIBRA(100)	BasicILS(100)	FocusedILS
GLS-GRID	$\epsilon = 1.81$	1.234 ± 0.492	0.951 ± 0.004	$\mathbf{0.949 \pm 0.0001}$
SAPS-QWH	$85.5K$ steps	$10.7K \pm 1.1K$	$10.9K \pm 0.6K$	$\mathbf{10.6K \pm 0.5K}$
SAPS-SW	5.60 s	0.053 ± 0.010	0.046 ± 0.01	$\mathbf{0.043 \pm 0.005}$

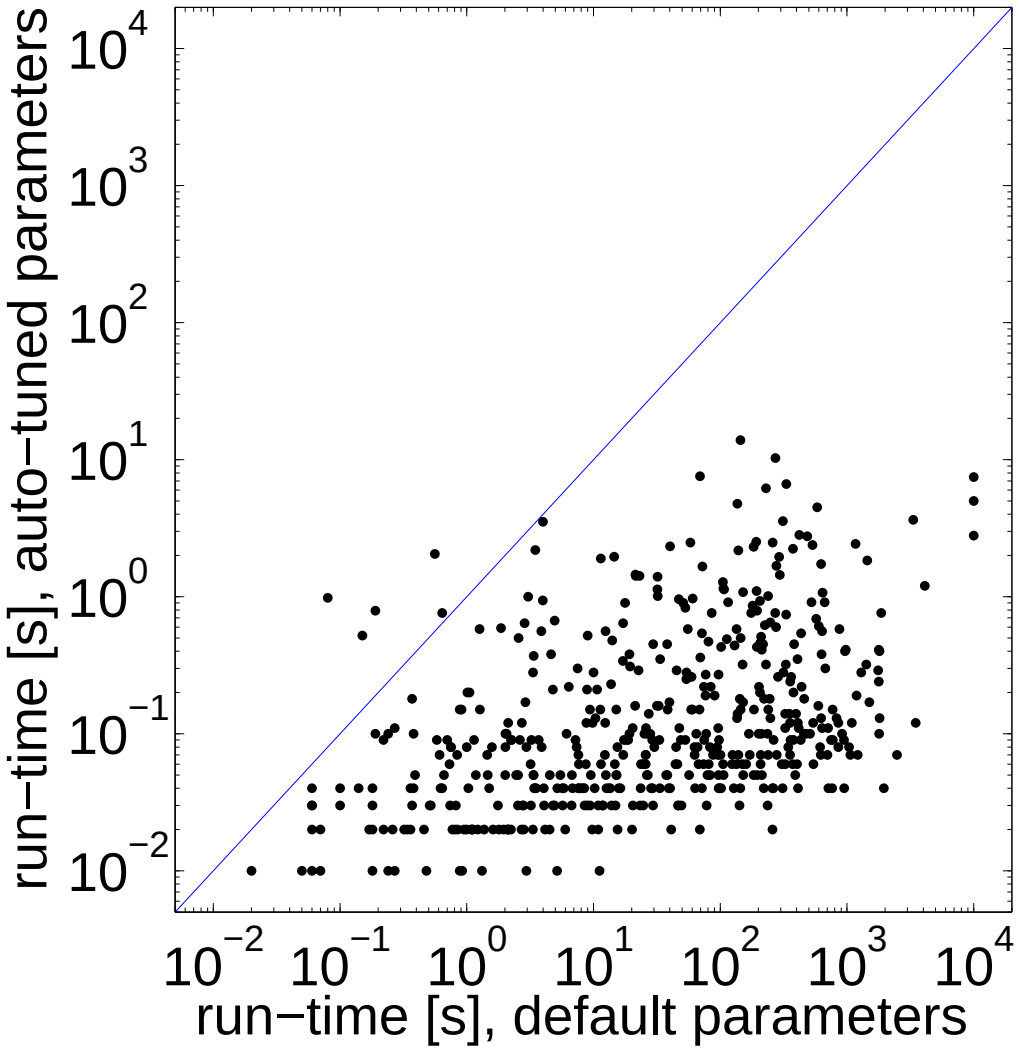
Sample applications and performance results

Experiment: Comparison to CALIBRA

Scenario	Default	CALIBRA(100)	BasicILS(100)	FocusedILS
GLS-GRID	$\epsilon = 1.81$	1.234 ± 0.492	0.951 ± 0.004	$\mathbf{0.949 \pm 0.0001}$
SAPS-QWH	$85.5K$ steps	$10.7K \pm 1.1K$	$10.9K \pm 0.6K$	$\mathbf{10.6K \pm 0.5K}$
SAPS-SW	5.60 s	0.053 ± 0.010	0.046 ± 0.01	$\mathbf{0.043 \pm 0.005}$
SAT4J-SW	7.02 s	(too many param.)	1.19 ± 0.58	$\mathbf{0.65 \pm 0.2}$

Speedup obtained by automated tuning

(SAPS default *vs* tuned on test set SW-GCP)



Two 'real-world" applications

- New DPLL-type SAT solver SPEAR
  - 26 parameters
  - Software verification: 500-fold speedup
  - 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
  - Hardware verification: 4.5-fold speedup
  - ⇝ New state of the art for those instances
  - ⇝ Hutter, Babić, Hoos & Hu: FMCAD '07 (to appear)
- New replica exchange Monte Carlo algorithm for protein structure prediction
  - 3 parameters
  - 2-fold improvement
  - ⇝ New state of the art for 2D/3D protein structure prediction
  - ⇝ Thachuk, Shmygelska & Hoos (under review)

Conclusions

- ParamILS: Simple and efficient framework for automatic parameter optimization

Conclusions

- ParamILS: Simple and efficient framework for automatic parameter optimization
- FocusedILS:
  - converges provably towards optimal configuration, no over-confidence
  - excellent performance in practice (outperforms BasicILS, CALIBRA)

Conclusions

- ParamILS: Simple and efficient framework for automatic parameter optimization
- FocusedILS:
  - converges provably towards optimal configuration, no over-confidence
  - 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
- FocusedILS:
  - converges provably towards optimal configuration, no over-confidence
  - 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)
- Per-instance tuning

Future work

- Continuous parameters (currently discretised)
- Statistical tests (cf. racing algorithms)
- Per-instance tuning
- Automatic algorithm design