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

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)
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 ($8.34 \times 10^{17}$ after discretization)
Standard algorithm configuration approach

- Choose a “representative” benchmark set for tuning
Choose a “representative” benchmark set for tuning

Perform iterative manual tuning:

1. **Start with some parameter configuration**
2. **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 $\sim$ local minimum only

Solution:

- Automate process
- Use more powerful search method
Problems of standard approach

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

Solution:
- Automate process
- Use more powerful search method
Problems of standard approach

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

Solution:

- Automate process
- Use more powerful search method
Related work

▶ Search approaches


⇝ orthogonal to the approach followed here
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**  

- **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**  
  ~ orthogonal to the approach followed here
1. Introduction

2. Iterated local search over parameter configurations

3. The BasicILS and FocusedILS algorithms

4. Sample applications and performance results

5. Conclusions and future work
The ParamILS framework

**ILS in parameter configuration space (ParamILS):**

- Choose initial parameter configuration $\theta$
- Perform *subsidiary local search* on $\theta$
- While tuning time left:
  - $\theta' := \theta$
  - Perform perturbation on $\theta$
  - Perform subsidiary local search on $\theta$
  - Based on acceptance criterion, keep $\theta$ or revert to $\theta' := \theta$
  - With probability $p$, restart randomly pick new $\theta$

Performs biased random walk over local optima.
**The ParamILS framework**

**ILS in parameter configuration space (ParamILS):**

Choose initial parameter configuration $\theta$

Perform *subsidiary local search* on $\theta$

While tuning time left:

$\theta' := \theta$

perform *perturbation* on $\theta$

perform *subsidiary local search* on $\theta$
The ParamILS framework

**ILS in parameter configuration space (ParamILS):**

Choose initial parameter configuration \( \theta \)
Perform *subsidiary local search* on \( \theta \)
While tuning time left:

\[ \theta' := \theta \]
perform *perturbation* on \( \theta \)
perform *subsidiary local search* on \( \theta \)

based on *acceptance criterion*,
keep \( \theta \) or revert to \( \theta := \theta' \)
The ParamILS framework

**ILS in parameter configuration space (ParamILS):**

Choose initial parameter configuration $\theta$
Perform *subsidiary local search* on $\theta$
While tuning time left:

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

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

with probability $p_{\text{restart}}$ randomly pick new $\theta$

$\rightsquigarrow$ Performs *biased random walk over local optima*
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
Details on ParamILS:

- Initialisation: pick \textit{best} of default & \( R \) random configurations
- Subsidiary local search: iterative first improvement, change one parameter in each step
Details on ParamILS:

- Initialisation: pick \textit{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 $\theta$ (based on $N$ runs)

- Sample $N$ instances from given set (with repetitions)
Evaluation of a parameter configuration $\theta$
(based on $N$ runs)

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

$\hat{c}_N(\theta)$ of the $N$ costs
(user-defined: e.g. empirical mean, median, ...)

Hutter, Hoos, Stützle: Automatic Algorithm Configuration based on Local Search
Evaluation of a parameter configuration $\theta$ (based on $N$ runs)

- Sample $N$ instances from given set (with repetitions)
- For each of the $N$ instances:
  - Execute algorithm with configuration $\theta$
  - 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 $\theta$
The BasicILS(N) algorithm

- Use a fixed number of $N$ runs to evaluate each configuration $\theta$

**Question:** How to choose number of runs $N$?

- Too many
  - $\Rightarrow$ evaluating a configuration is very expensive
  - $\Rightarrow$ optimisation process is very slow
The BasicILS(N) algorithm

- Use a fixed number of $N$ runs to evaluate each configuration $\theta$

**Question:** How to choose number of runs $N$?

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

- Too few
  - $\leadsto$ very noisy approximations $\hat{c}_N(\theta)$
  - $\leadsto$ poor generalisation to independent test runs
Generalisation to independent test set, large N (N=100)

(SAPS on quasigroups with holes)

![Graph showing runlength (median, 10% & 90% quantiles) vs CPU time [s]. The graph compares BasicILS(100) performance on training set. The x-axis represents CPU time in seconds, ranging from $10^1$ to $10^4$. The y-axis represents runlength (median, 10% & 90% quantiles), ranging from $10^4$ to 1.]
Generalisation to independent test set, large N \( (N=100) \)

\( \text{SAPS on quasigroups with holes} \)

![Graph showing BasicILS(100) performance on test set and training set](image)
Generalisation to independent test set, small N (N=1)

(SAPS on quasigroups with holes)

Runlength (median, 10\% & 90\% quantiles)

BasicILS(1) performance on test set

BasicILS(1) performance on training set
Test performance of BasicILS with different N

(SAPS on quasigroups with holes)

![Graph showing the performance of BasicILS(100)]
Test performance of BasicILS with different N

(SAPS on quasigroups with holes)
Test performance of BasicILS with different $N$.

$(SAPS$ on quasigroups with holes$)$

![Graph showing performance comparison between different BasicILS configurations](image-url)
The FocusedILS algorithm

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

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

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

Hutter, Hoos, Stützle: Automatic Algorithm Configuration based on Local Search 14
The FocusedILS algorithm

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

**Idea:** Use high $N(\theta)$ only for good $\theta$

- start with $N(\theta) = 0$ for all $\theta$
- increment $N(\theta)$ whenever $\theta$ is visited
- additional runs upon finding new, better configuration $\theta$

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

Key ideas in proof
1. For $N(\theta) \to \infty$, $\hat{c}_{N(\theta)} \to c(\theta)$
2. Underlying ILS eventually reaches any configuration $\theta$. 
The FocusedILS algorithm

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

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

**Theorem:**
As number of FocusedILS iterations $\to \infty$, it converges to true optimal configuration $\theta^*$
The FocusedILS algorithm

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

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

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

Hutter, Hoos, Stützle: Automatic Algorithm Configuration based on Local Search 14
Performance of FocusedILS vs BasicILS

(Test performance of SAPS on quasigroups with holes)

Median runlength of SAPS [steps]

CPU time for ParamILS [s]

BasicILS(100)
BasicILS(10)
BasicILS(1)
Performance of FocusedILS vs BasicILS

(Test performance of SAPS on quasigroups with holes)

- CPU time for ParamILS [s]
- Median runlength of SAPS [steps]

FocusedILS
BasicILS(100)
BasicILS(10)
BasicILS(1)
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

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Metric</th>
<th>Default</th>
<th>FocusedILS</th>
<th>BasicILS(100)</th>
<th>CALIBRA(100)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAPS on GC</td>
<td>Runtime</td>
<td>5.60 s</td>
<td>0.043 ± 0.005</td>
<td>0.046 ± 0.01</td>
<td>0.053 ± 0.010</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Metric</th>
<th>Default</th>
<th>FocusedILS</th>
<th>BasicILS(100)</th>
<th>CALIBRA(100)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAPS on GC</td>
<td>Runtime</td>
<td>5.60 s</td>
<td><strong>0.043 ± 0.005</strong></td>
<td>0.046 ± 0.01</td>
<td>0.053 ± 0.010</td>
</tr>
<tr>
<td>GLS+ for MPE</td>
<td>Approx. error</td>
<td>$\varepsilon = 1.81$</td>
<td><strong>0.949 ± 0.0001</strong></td>
<td>0.951 ± 0.004</td>
<td>1.234 ± 0.492</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Metric</th>
<th>Default</th>
<th>FocusedILS</th>
<th>BasicILS(100)</th>
<th>CALIBRA(100)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAPS on GC</td>
<td>Runtime</td>
<td>5.60 s</td>
<td><strong>0.043 ± 0.005</strong></td>
<td>0.046 ± 0.01</td>
<td>0.053 ± 0.010</td>
</tr>
<tr>
<td>GLS$^+$ for MPE</td>
<td>Approx. error</td>
<td>$\epsilon = 1.81$</td>
<td><strong>0.949 ± 0.0001</strong></td>
<td>0.951 ± 0.004</td>
<td>1.234 ± 0.492</td>
</tr>
<tr>
<td>SAT4J on GC</td>
<td>Runtime</td>
<td>7.02 s</td>
<td><strong>0.65 ± 0.2</strong></td>
<td>1.19 ± 0.58</td>
<td>(too many param.)</td>
</tr>
</tbody>
</table>
Speedup obtained by automated tuning

(SAPS default vs tuned on graph colouring, test set performance)
Two “real-world” applications

- New DPLL-type SAT solver \textit{SPEAR}
  - 26 parameters
  - Software verification: 500-fold speedup (won QB-FQ category in SMT’07 competition)
  - Hardware verification: 4.5-fold speedup
  \rightarrow New state of the art for those instances
  \rightarrow [Hutter, Babić, Hoos & Hu: FMCAD ’07 (to appear)]
Two “real-world” applications

▶ New DPLL-type SAT solver \textit{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)]

▶ 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: 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

- Converges provably towards optimal configuration
- Excellent performance in practice (outperforms BasicILS, CALIBRA)
- Huge speedups:
  - ≈ 100× for Saps (local search) on graph colouring
  - ≈ 500× for Spear (tree search) on software verification

Publically available at: http://www.cs.ubc.ca/labs/beta/Projects/ParamILS
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)

Publically available at: http://www.cs.ubc.ca/labs/beta/Projects/ParamILS
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 \textit{SAPS} (local search) on graph colouring
  - $\approx 500 \times$ for \textit{SPEAR} (tree search) on software verification

Publically available at: [http://www.cs.ubc.ca/labs/beta/Projects/ParamILS](http://www.cs.ubc.ca/labs/beta/Projects/ParamILS)
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