Automatically Improving Empirical Performance: Algorithm Configuration & Selection

Frank Hutter, Lars Kotthoff, Yuri Malitsky, Barry O’Sullivan, Lin Xu

15 July 2013
Design of Heuristic Algorithms

Problem instances \[\rightarrow\] Empirical performance

Domain expert(s) \[\rightarrow\] Solution strategies

Example: tree search SAT solvers
- Decision 1: heuristic to select next branching variable
- Decision 2: heuristic to select which value to try first
- Decision 3: preprocessing
- Decision 4: restart schedule
- Decision 5: clause learning strategy
- Decision 6: clause sharing strategy in parallel SAT solving
- …

SAT: 26 parameters, \(10^{18}\) configurations (Spear)
TSP: 24 parameters, \(10^{15}\) configs (LK-H)
MIP: 76 parameters, \(10^{47}\) configs (CPLEX)
AI planning: 62 parameters, \(10^{17}\) configs (LPG)
Supervised machine learning (classification):

768 params, \(10^{47}\) configs (WEKA)

Manual tuning
Hard to understand
Computer-Aided Algorithm Design and Analysis

Automated Procedures to model & optimize empirical performance

Algorithm configuration
Find best for

Algorithm portfolios
Select best for new

Performance prediction
Quantify importance of algorithm components & instance characteristics

Saves time

Improves Performance

Improves Understanding
The Algorithm Configuration Process

Parameter domains & starting values

Configurator

Calls with different parameter settings

Configuration scenario

Target algorithm

Solves

Problem instances

Returns solution cost
The Algorithm Configuration Problem

Definition

– Given:
  • Runnable algorithm $\mathcal{A}$ with configuration space $\Theta = \Theta_1 \times \cdots \times \Theta_n$
  • Distribution $D$ over problem instances $\Pi$
  • Performance metric $m : \Theta \times \Pi \to \mathbb{R}$

– Find:

$$\theta^* \in \arg \min_{\theta \in \Theta} \mathbb{E}_{\pi \sim D}[m(\theta, \pi)]$$

Motivation

Customize versatile algorithms for different application domains

– Fully automated improvements
– Optimize speed, accuracy, memory, energy consumption, ...

Very large space of configurations
Algorithm parameters

Parameter types

- Continuous, integer, ordinal
- **Categorical**: finite domain, unordered, e.g. \{A,B,C\}

Parameter space has **structure**

- E.g. parameter C of heuristic A is only active if A is used
- In this case, we say C is **conditional parameter** with parent A

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”)


A Concrete Example

New SAT solver for formal verification (Spear)
  – 26 user-specifiable parameters
  – 7 categorical, 3 Boolean, 12 continuous, 4 integer

Objective: minimize runtime on software verification instance set

Issues:
  – Many possible settings \((8.34 \times 10^{17} \text{ after discretization})\)
  – Evaluating performance of a configuration is expensive
  – Instances vary in hardness
    • Some take milliseconds, other days (for the default)
    • Thus, improvement on a few instances might not mean much
Configurators have two key components

• Component 1: which configuration to evaluate next?

• Component 2: how to evaluate that configuration?
Automated Algorithm Configuration: Outline

Methods (components of algorithm configuration)
- Systems (that instantiate these components)
- Demo & Practical Issues
- Case Studies
Component 1: Which configuration to evaluate?

• For this part, let’s restrict the problem: **Blackbox function optimization**

• Optimize a function $f$ over a domain $\Theta$:

\[
\min_{\theta \in \Theta} f(\theta)
\]

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

  \[
  \theta \rightarrow f(\theta)
  \]

  – $\Theta$ is still a structured space
    • Mixed continuous/discrete
    • Conditional parameters
The Simplest Search Strategy: Random Search

• Select configurations uniformly at random
  – Global search, won’t get stuck in a local region
  – But completely uninformed
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")
Stochastic Local Search

• **Balance intensification and diversification**
  – Intensification: gradient descent
  – Diversification: restarts, random steps, perturbations, ...

• Prominent general methods
  – Taboo search
  – Simulated annealing
  – Iterated local search
Population-based Methods

• Population of configurations
  – Global + local search via population
  – Maintain population fitness & diversity

• Examples
  – Genetic algorithms
  – Evolutionary strategies
  – Ant colony optimization
  – Particle swarm optimization
Sequential Model-Based Optimization

New data point
Sequential Model-Based Optimization

- Popular approach in statistics to minimize expensive blackbox functions [Mockus, '78]

- Recent progress in the machine learning literature: **global convergence rates** for continuous optimization
  - [Srinivas et al, ICML'10]
  - [Bull, JMLR'11]
  - [Bubeck et al, JMLR'11]
  - [de Freitas, Smola, Zoghi, ICML'12]
Summary 1: Which configuration to evaluate?

• Need to balance diversification and intensification
• The extremes
  – Random search
  – Hillclimbing
• Stochastic local search (SLS)
• Population-based methods
• Sequential Model-Based Optimization
Component 2: How to evaluate a configuration?

Back to general algorithm configuration

– Given:
  • Runnable algorithm $\mathcal{A}$ with configuration space $\Theta = \Theta_1 \times \cdots \times \Theta_n$
  • Distribution $D$ over problem instances $\Pi$
  • Performance metric $m : \Theta \times \Pi \rightarrow \mathbb{R}$

– Find:

$$\theta^* \in \arg\min_{\theta \in \Theta} \mathbb{E}_{\pi \sim D}[m(\theta, \pi)]$$

Recall the Spear example

– Instances vary in hardness
  • Some take milliseconds, other days (for the default)
  • Thus, improvement on a few instances might not mean much
Simplest Solution: Use Fixed N Instances

• Effectively treat the problem as a blackbox function optimization problem

• **Issue: how large to choose N?**
  – Too small: overtuning
  – Too large: every function evaluation is slow

• **General principle**
  – Don’t waste time on bad configurations
  – Evaluate good configurations more thoroughly
Racing Algorithms

• 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

• 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

(only when minimizing algorithm runtime)

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

– Is $\theta'$ better than $\theta$?

• Example:

```
<table>
<thead>
<tr>
<th>RT($\theta$)=20</th>
<th>RT($\theta'$)&gt;20</th>
</tr>
</thead>
</table>
```

• Can terminate evaluation of $\theta'$ once guaranteed to be worse than $\theta$
Summary 2: How to evaluate a configuration?

- Simplest: fixed set of N instances
- General principle
  - Don’t waste time on bad configurations
  - Evaluate good configurations more thoroughly
- Instantiations of principle
  - Racing
  - Aggressive racing
  - Adaptive capping
Automated Algorithm Configuration: Outline

• Methods (components of algorithm configuration)
• Systems (that instantiate these components)
• Demo & Practical Issues
• Case Studies
Iterated Local Search in parameter configuration space:

→ Performs biased random walk over local optima
Instantiations of ParamILS Framework

How to evaluate each configuration?

– BasicILS(N): use a fixed number of N runs

– FocusedILS:
  • aggressive racing, focus on good configurations

Theorem

As FocusedILS's overall time budget $\rightarrow \infty$, it converges to the optimal configuration
Standard adaptive capping

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

  - Example:

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

  - $RT(\theta) = 20$ and $RT(\theta') > 20$

Theorem

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

- Often yields substantial speedups
Gender-based Genetic Algorithm (GGA) [Ansotegui et al, ’09]

• Genetic algorithm
  – Genome = parameter configuration
  – Combine genomes of 2 parents to form an offspring

• Two genders in the population
  – Selection pressure only on one gender
  – Preserves diversity of the population
• Use N instances to evaluate configurations
  – Increase N in each generation
  – **Linear increase from** $N_{start}$ **to** $N_{end}$
    • User specifies #generations ahead of time

• **Can exploit parallel resources**
  – Evaluate population members in parallel
  – Adaptive capping: can stop when the first k succeed
F-Race and Iterated F-Race

[Birattari et al, ’02 and ‘10]

• 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
F-Race and Iterated F-Race

[Birattari et al, ’02 and ’10]

• **Can use parallel resources**
  – Simply do the \( k \) runs of each iteration in parallel
  – But does not support adaptive capping

• **Expected performance**
  – Strong when the key challenge are reliable comparisons between configurations
  – Less good when the search component is the challenge
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: models

• Model algorithm performance based on random forests

• Each run we did for $\theta$ only uses a single instance $\pi$
  – Fit a model $m : \Theta \times \Pi \rightarrow \mathbb{R}$

• Aggregate over instances by marginalization

\[ f(\theta) := \mathbb{E}_{\pi \sim D} [m(\theta, \pi)] \]
  – Intuition: predict for each instance and take the average
  – More efficient implementation in random forests

[Hutter et al, ’11]
Distributed SMAC

• **Distribute target algorithm runs across workers**
  – Maintain queue of promising configurations
  – Compare these to $\theta^*$ on distributed worker cores

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

[Hutter et al, ’12]
Summary: Algorithm Configuration Systems

- ParamILS
- Gender-based Genetic Algorithm (GGA)
- Iterated F-Race
- Sequential Model-based Algorithm Configuration (SMAC)
- Distributed SMAC

- Which one is best?
  - First configurator competition to come in 2014
Automated Algorithm Configuration: Outline

• Methods  (components of algorithm configuration)
• Systems   (that instantiate these components)

Demo & Practical Issues

• Case Studies
The Algorithm Configuration Process

Parameter domains & starting values

Configurator

Calls with different parameter settings

Configuration scenario

Solves

Problem instances

Target algorithm

Returns solution cost

What the user has to provide

Parameter space declaration file

preproc \{none, simple, expensive\} [simple]
apha [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-v2.04.01-master-447.tar.gz

tar xzvf smac-v2.04.01-master-447.tar.gz

cd smac-v2.04.01-master-447

./smac --seed 0 --scenarioFile example_spear/scenario-Spear-QCP-sat-small-train-small-test-mixed.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*****
   Iteration: 12
   Incumbent ID: 11 (0x27CA0)
   Number of Runs for Incumbent: 26
   Number of Instances for Incumbent: 5
   Number of Configurations Run: 25
   Performance of the Incumbent: 0.05399999999999999
   Total Number of runs performed: 101
   Configuration time budget used: 30.020000000000034 s
[INFO ] **********************************************

[INFO ] Total Objective of Final Incumbent 13 (0x30977) on training set:
   0.05399999999999999; on test set: 0.055

[INFO ] Sample Call for Final Incumbent 13 (0x30977)
cd /global/home/hutter/ac/smac-v2.04.01-master-447/example_spear; ruby spear_wrapper.rb example_data/QCP-instances/qcplin2006.10422.cnf 0 5.0 2147483647 2897346 -sp-clause-activity-inc '1.3162094350513607' -sp-clause-decay '1.739666995554204' -sp-clause-del-heur '1' -sp-first-restart '846' -sp-learned-clause-sort-heur '10' -sp-learned-clauses-inc '1.395279056466624' -sp-learned-size-factor '0.6071142792450034' -sp-orig-clause-sort-heur '7' -sp-phase-dec-heur '5' -sp-rand-phase-dec-freq '0.005' -sp-rand-phase-scaling '0.8863796134762909' -sp-rand-var-dec-freq '0.01' -sp-rand-var-dec-scaling '0.6433957166060014' -sp-resolution '0' -sp-restart-inc '1.7639087832223321' -sp-update-dec-queue '1' -sp-use-pure-literal-rule '0' -sp-var-activity-inc '0.7825881046949665' -sp-var-dec-heur '3' -sp-variable-decay '1.0374907487192533'
Decision #1: Configuration budget & max. captime

• **Configuration budget**
  – Dictated by your resources & needs
    • E.g., start the configurator before leaving work on Friday
  – The longer the better (but diminishing returns)
    • Rough rule of thumb: at least enough time for 1000 target runs

• **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 300s
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
  – Prefer $\geq 300$ instances, better $\geq 1000$ instances
Decision #2: Choosing the training instances

- Split instance set into training and test sets
  - Configure on the training instances $\rightarrow$ configuration $\theta^*$
  - Run $\theta^*$ 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, second half of 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
  - 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
Automated Algorithm Configuration: Outline

• Methods (components of algorithm configuration)
• Systems (that instantiate these components)
• Demo & Practical Issues

Case Studies
Configuration of a SAT Solver for Verification

Spear [Babic, 2007]
- 26 parameters
- $8.34 \times 10^{17}$ configurations

Ran ParamILS, 2 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

4.5-fold speedup
500-fold speedup $\Rightarrow$ won QF_BV category in 2007 SMT competition

[Log-log scale]

[Hutter, Babic, Hu & Hoos, FMCAD'07]
• Annual SAT competition
  – Scores SAT solvers by their performance across instances
  – Medals for best average performance with solver defaults

• CSSC 2013
  – Motivated by application context: homogeneous instances
    → can automatically optimize parameters
  – Medals for best performance after being optimization
  – Result: automated configuration affected rankings a lot
Configuration of a Commercial MIP solver

Mixed Integer Programming (MIP)

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

Commercial MIP solver: IBM ILOG CPLEX

– Leading solver for the last 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× to 50×
– Later work: speedups up to 10,000×

Minimizing optimality gap reached

– Gap reduction factor: 1.3× to 8.6×

[Hutter, Hoos & Leyton-Brown, CPAIOR’10]
Comparison to CPLEX Tuning Tool

**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

![Comparison Graphs](image)

2-fold speedup (our worst result)

50-fold speedup (our best result)
**WEKA**: most widely used off-the-shelf machine learning package (>18,000 citations on Google scholar)

**Different methods work best on different data sets**
- 30 base classifiers (with up to 8 parameters each)
- 14 meta-methods
- 3 ensemble methods
- 3 feature search methods & 8 feature evaluators
- Want a **true off-the-shelf solution**: Learn
Machine Learning Application: Auto-WEKA

• Combined model selection & hyperparameter optimization
  – All hyperparameters are conditional on their model being used
  – WEKA’s configuration space: 786 parameters
  – Optimize cross-validation (CV) performance

• Results
  – SMAC yielded best CV performance on 19/21 data sets
  – Best test performance for most sets; especially in 8 largest

• Auto-WEKA is online:
  http://www.cs.ubc.ca/labs/beta/Projects/autoweka/
Applications of Algorithm Configuration

**Mixed integer programming**

**Scheduling and Resource Allocation**

**Exam Timetabling since 2010**

**Spam filters**

**Helped win Competitions**
- SAT: since 2009
- IPC: since 2011
- Time-tableing: 2007
- SMT: 2007

**Other Academic Applications**
- Protein Folding
- Game Theory: Kidney Exchange
- Computer GO
- Linear algebra subroutines
- Evolutionary Algorithms
- Machine Learning: Classification
Looking for great students & postdocs

Just started a new research group in Freiburg, Germany: 
**Learning, Optimization, and Automated Algorithm Design**

- Algorithm configuration
- Algorithm portfolios
- Automated methods to gain insights into algorithms and instances
- Based on machine learning and optimization

Freiburg

- One of Germany’s top universities
- “Most livable place in Germany”
- Positions open anytime for outstanding MSc/PhD students, postdocs
- Please spread the word & contact me for details:
  [fh@informatik.uni-freiburg.de](mailto:fh@informatik.uni-freiburg.de) (or just google Frank Hutter)