Algorithm Configuration: How to boost performance of your SAT solver?

Marius Lindauer[1]

University of Freiburg

SAT Summer School 2016, Lisbon

[1] Thanks to Frank Hutter!

Ever looked into --help?

MiniSat (10 parameters)

```
CORE OPTIONS:

  -rnd-init, -no-rnd-init                    (default: off)
  -luby, -no-luby                            (default: on)

  -rnd-freq     = <double> [   0 ..    1] (default: 0)
  -rnd-seed     = <double> (   0 ..  inf) (default: 9.16483e+07)
  -var-decay    = <double> (   0 ..    1) (default: 0.95)
  -cla-decay    = <double> (   0 ..    1) (default: 0.999)
  -rinc         = <double> (   1 ..  inf) (default: 2)
  -gc-frac      = <double> (   0 ..  inf) (default: 0.2)

  -rfirst       = <int32>  [   1 .. imax] (default: 100)
  -ccmin-mode   = <int32>  [   0 ..    2] (default: 2)
  -phase-saving = <int32>  [   0 ..    2] (default: 2)

MAIN OPTIONS:

  -verb         = <int32>  [   0 ..    2] (default: 1)
  -cpu-lim      = <int32>  [   0 .. imax] (default: 2147483647)
  -mem-lim      = <int32>  [   0 .. imax] (default: 2147483647)

HELP OPTIONS:

  --help        Print help message.
  --help-verb   Print verbose help message.
```

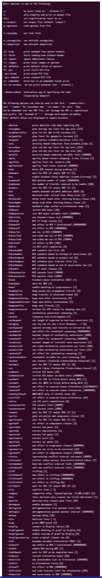
Ever looked into `--help`?

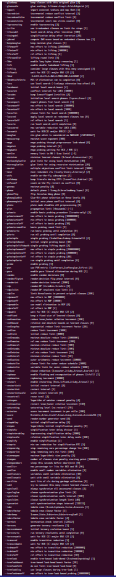
Glucose (20 parameters)

```
CORE OPTIONS:

  -rnd-init, -no-rnd-init                 (default: off)

  -gc-frac      = <double> (   0 ..  inf) (default: 0.2)
  -rnd-seed     = <double> (   0 ..  inf) (default: 9.16483e+07)
  -rnd-freq     = <double> [   0 ..    1] (default: 0)
  -cla-decay    = <double> (   0 ..    1) (default: 0.999)
  -max-var-decay = <double> (   0 ..    1) (default: 0.95)
  -var-decay    = <double> (   0 ..    1) (default: 0.8)

  -ccmin-mode   = <int32>  [   0 ..    2] (default: 2)
  -phase-saving = <int32>  [   0 ..    2] (default: 2)

CORE -- CERTIFIED UNSAT OPTIONS:

  -certified, -no-certified               (default: off)

  -certified-output = <string>

CORE -- MINIMIZE OPTIONS:

  -minLBDMinimizingClause = <int32>  [   3 .. imax] (default: 6)
  -minSizeMinimizingClause = <int32>  [   3 .. imax] (default: 30)

CORE -- REDUCE OPTIONS:

  -incReduceDB  = <int32>  [   0 .. imax] (default: 300)
  -specialIncReduceDB = <int32>  [   0 .. imax] (default: 1000)
  -minLBDFrozenClause = <int32>  [   0 .. imax] (default: 30)
  -firstReduceDB = <int32>  [   0 .. imax] (default: 2000)

CORE -- RESTART OPTIONS:

  -R            = <double> (   1 ..    5) (default: 1.4)
  -K            = <double> (   0 ..    1) (default: 0.8)

  -szLBDQueue   = <int32>  [  10 .. imax] (default: 50)
  -szTrailQueue = <int32>  [  10 .. imax] (default: 5000)

MAIN OPTIONS:

  -pre, -no-pre                           (default: on)
  -model, -no-model                       (default: off)

  -mem-lim      = <int32>  [   0 .. imax] (default: 2147483647)
  -vv           = <int32>  [   1 .. imax] (default: 10000)
  -verb         = <int32>  [   0 ..    2] (default: 1)
  -cpu-lim      = <int32>  [   0 .. imax] (default: 2147483647)

  -dimacs     = <string>

SIMP OPTIONS:

  -elim, -no-elim                         (default: on)
  -rcheck, -no-rcheck                     (default: off)
  -asymm, -no-asymm                       (default: off)

  -simp-gc-frac = <double> (   0 ..  inf) (default: 0.5)

  -grow         = <int32>  [imin .. imax] (default: 0)
  -sub-lim      = <int32>  [  -1 .. imax] (default: 1000)
  -cl-lim       = <int32>  [  -1 .. imax] (default: 20)

HELP OPTIONS:

  --help        Print help message.
  --help-verb   Print verbose help message.
```

Ever looked into `--help`?

lingeling (> 300 parameters)





Importance of Algorithm Configuration?

SAT Competition

- Submission of a solver
- Same parameter configuration on all instances

$\rightarrow$ Robust performance across instances

Importance of Algorithm Configuration?

SAT Competition

- Submission of a solver
- Same parameter configuration on all instances

→ Robust performance across instances

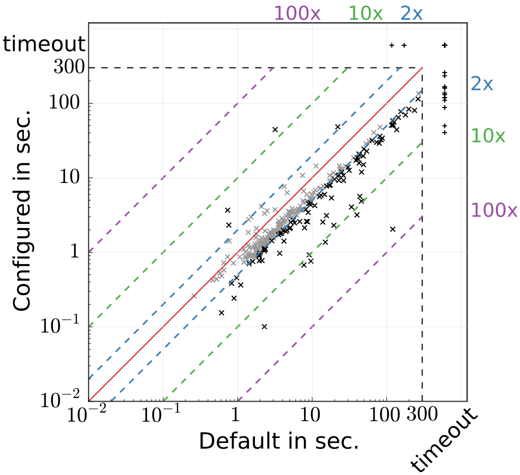
Configurable SAT Solver Challenge (CSSC)

- Submission of a solver
- We tuned the parameter configuration for each instance set

→ Peak performance on each set

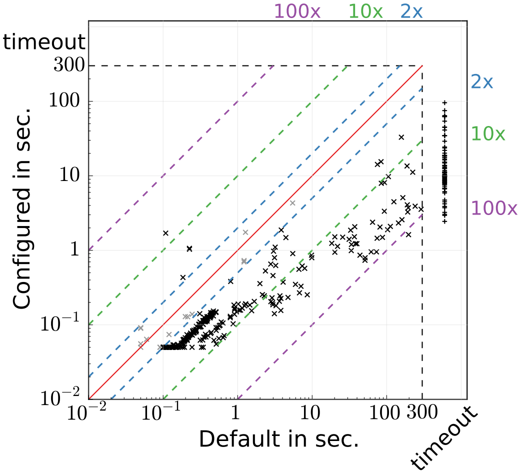
Importance of Algorithm Configuration? (Example from CSSC)

Lingeling on CircuitFuzz (#TOs: $30 \rightarrow 18$)



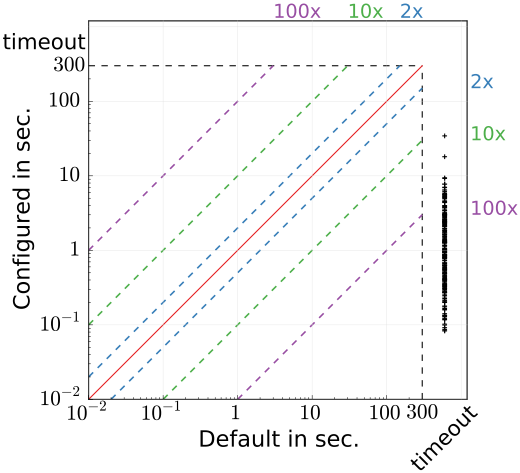
Importance of Algorithm Configuration? (Example from CSSC)

Clasp on Rooks (#TOs: $81 \rightarrow 0$)



Importance of Algorithm Configuration? (Example from CSSC)

ProbSAT on 5SAT500 (#TOs: $250 \rightarrow 0$)



What is this lecture about?

In a Nutshell: Algorithm Configuration

How to automatically determine a well-performing parameter configuration?

What is this lecture about?

In a Nutshell: Algorithm Configuration

How to automatically determine a well-performing parameter configuration?

Focus on basics

1. State-of-the-art in algorithm configuration
2. Parameter importance
3. Pitfalls and best practices in algorithm configuration

What is this lecture about?

In a Nutshell: Algorithm Configuration

How to automatically determine a well-performing parameter configuration?

Focus on basics

1. State-of-the-art in algorithm configuration
2. Parameter importance
3. Pitfalls and best practices in algorithm configuration

- Please ask questions
- No special background assumed
- All literature references are hyperlinks

Slides at: `www.ml4aad.org`

Outline

1. The Algorithm Configuration Problem
   - Problem Statement
   - Motivation: a Success Stories
   - Overview of Methods
2. Using AC Systems
3. Importance of Parameters
4. Pitfalls and Best Practices
5. Final Remarks

Outline

1. The Algorithm Configuration Problem
   - Problem Statement
   - Motivation: a Success Stories
   - Overview of Methods
2. Using AC Systems
3. Importance of Parameters
4. Pitfalls and Best Practices
5. Final Remarks

Algorithm Parameters

Parameter Types

- Continuous, integer, ordinal
- Categorical: finite domain, unordered, e.g., {apple, tomato, pepper}

Algorithm Parameters

Parameter Types

- Continuous, integer, ordinal
- Categorical: finite domain, unordered, e.g., $\{$apple, tomato, pepper$\}$

Parameter space has structure

- E.g., parameter θ_2 of heuristic H is only active if H is used $(\theta_1 = H)$
- In this case, we say θ_2 is a conditional parameter with parent θ_1
- Sometimes, some combinations of parameter settings are forbidden
  e.g., the combination of $\theta_3 = 1$ and $\theta_4 = 2$ is forbidden

Algorithm Parameters

Parameter Types

- Continuous, integer, ordinal
- Categorical: finite domain, unordered, e.g., {apple, tomato, pepper}

Parameter space has structure

- E.g., parameter θ_2 of heuristic H is only active if H is used $(\theta_1 = H)$
- In this case, we say θ_2 is a conditional parameter with parent θ_1
- Sometimes, some combinations of parameter settings are forbidden
  e.g., the combination of $\theta_3 = 1$ and $\theta_4 = 2$ is forbidden

Parameters give rise to a structured space of configurations

- Many configurations (e.g., SAT solver *lingeling* with 10^{947})
- Configurations often yield qualitatively different behaviour

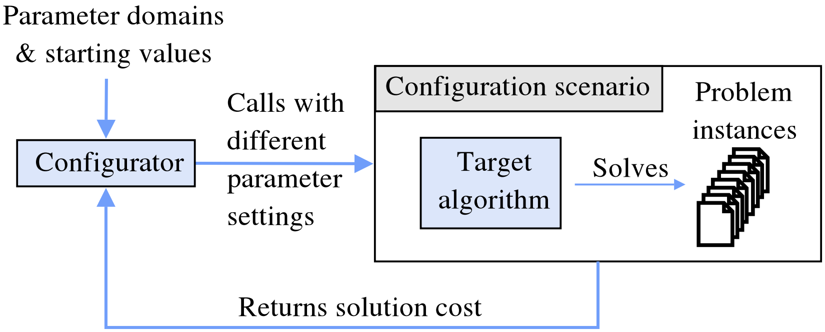
$\rightarrow$ Algorithm Configuration (as opposed to "parameter tuning")

Parameters of *MiniSAT*

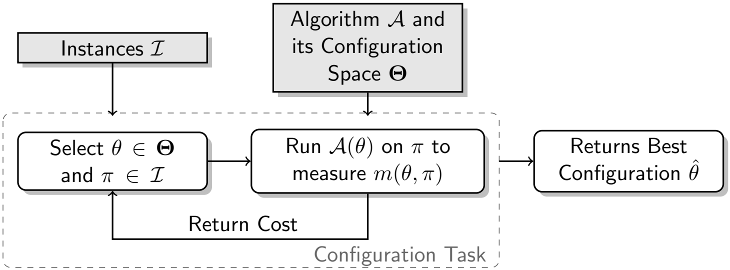
MiniSAT

```
CORE OPTIONS:

  -rnd-init, -no-rnd-init                 (default: off)
  -luby, -no-luby                         (default: on)

  -rnd-freq      = <double> [   0 ..    1] (default: 0)
  -rnd-seed      = <double> (   0 ..  inf) (default: 9.16483e+07)
  -var-decay     = <double> (   0 ..    1) (default: 0.95)
  -cla-decay     = <double> (   0 ..    1) (default: 0.999)
  -rinc          = <double> (   1 ..  inf) (default: 2)
  -gc-frac       = <double> (   0 ..  inf) (default: 0.2)

  -rfirst        = <int32>  [   1 .. imax] (default: 100)
  -ccmin-mode    = <int32>  [   0 ..    2] (default: 2)
  -phase-saving  = <int32>  [   0 ..    2] (default: 2)

MAIN OPTIONS:

  -verb          = <int32>  [   0 ..    2] (default: 1)
  -cpu-lim       = <int32>  [   0 .. imax] (default: 2147483647)
  -mem-lim       = <int32>  [   0 .. imax] (default: 2147483647)

HELP OPTIONS:

  --help         Print help message.
  --help-verb    Print verbose help message.
```

Algorithm Configuration Visualized



Algorithm Configuration – in More Detail

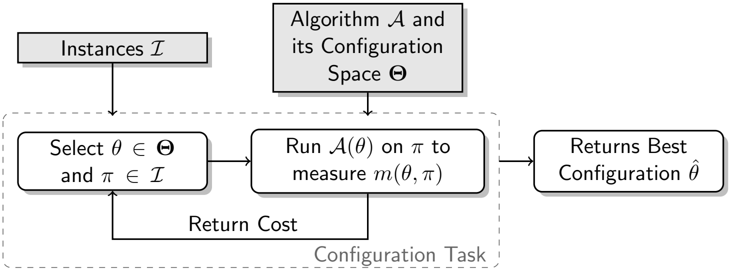


Definition: algorithm configuration

Given:

- a parameterized algorithm $\mathcal{A}$ with possible parameter settings $\boldsymbol{\Theta}$;
- a distribution $\mathcal{D}$ over problem instances with domain $\mathcal{I}$; and

Algorithm Configuration – in More Detail

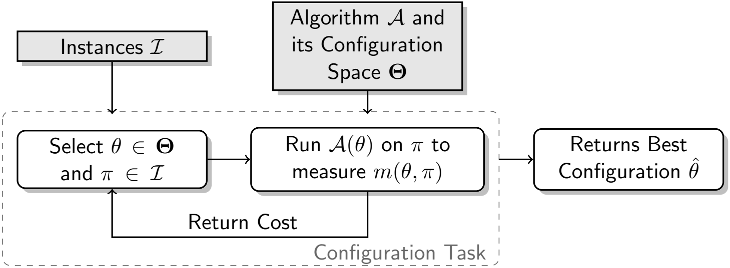


Definition: algorithm configuration

Given:

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

Algorithm Configuration – in More Detail



Definition: algorithm configuration

Given:

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

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

Outline

1. The Algorithm Configuration Problem
   - Problem Statement
   - Motivation: a Success Stories
   - Overview of Methods
2. Using AC Systems
3. Importance of Parameters
4. Pitfalls and Best Practices
5. Final Remarks

Configuration of a SAT Solver for Verification [Hutter et al, 2007]

Formal verification

- Software verification [Babić & Hu; CAV '07]
- Hardware verification (Bounded model checking) [Zarpas; SAT '05]

Configuration of a SAT Solver for Verification [Hutter et al, 2007]

Formal verification

- Software verification [Babić & Hu; CAV '07]
- Hardware verification (Bounded model checking) [Zarpas; SAT '05]

Tree search solver for SAT-based verification

- SPEAR, developed by Domagoj Babić at UBC
- 26 parameters, 8.34×10^{17} configurations

Configuration of a SAT Solver for Verification [Hutter et al, 2007]

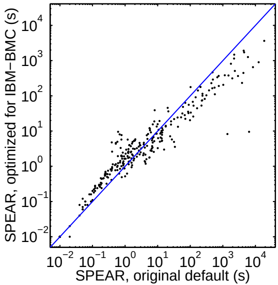
- Ran *ParamILS*, 2 days $\times$ 10 machines
  - On a training set from each benchmark

Configuration of a SAT Solver for Verification [Hutter et al, 2007]

- Ran *ParamILS*, 2 days $\times$ 10 machines
  - On a training set from each benchmark
- Compared to manually-engineered configuration
  - 1 week of performance tuning
  - Competitive with the state of the art
  - Comparison on unseen test instances

Configuration of a SAT Solver for Verification [Hutter et al, 2007]

- Ran *ParamILS*, 2 days $\times$ 10 machines
  - On a training set from each benchmark
- Compared to manually-engineered configuration
  - 1 week of performance tuning
  - Competitive with the state of the art
  - Comparison on unseen test instances

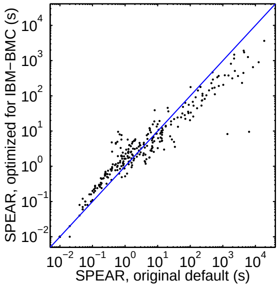


4.5-fold speedup

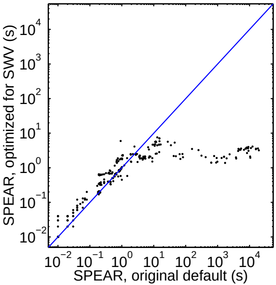
on hardware verification

Configuration of a SAT Solver for Verification [Hutter et al, 2007]

- Ran *ParamILS*, 2 days $\times$ 10 machines
  - On a training set from each benchmark
- Compared to manually-engineered configuration
  - 1 week of performance tuning
  - Competitive with the state of the art
  - Comparison on unseen test instances



4.5-fold speedup
on hardware verification



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

Algorithm Configuration is Widely Applicable

- Hard combinatorial problems
  - SAT, MIP, TSP, AI planning, ASP, Time-tabling, ...
  - UBC exam time-tabling since 2010
- Game Theory: Kidney Exchange
- Mobile Robotics
- Monte Carlo Localization
- Motion Capture
- Machine Learning
  - Automated Machine Learning
  - Deep Learning

Also popular in industry

- Better performance
- Increased productivity

FCC spectrum auction
Auctionomics

IBM ILOG
Mixed integer programming

Social gaming
zynga

Scheduling and Resource Allocation
actenum

Outline

1. The Algorithm Configuration Problem
   - Problem Statement
   - Motivation: a Success Stories
   - Overview of Methods
2. Using AC Systems
3. Importance of Parameters
4. Pitfalls and Best Practices
5. Final Remarks

Challenges of Algorithm Configuration

Expensive Algorithm Runs

- Evaluation of 1 configuration on 1 instance is already expensive (solving a $\mathcal{NP}$ problem)
- Evaluation of $n > 1000$ configurations on $m > 100$ instances can be infeasible in practice

Challenges of Algorithm Configuration

Expensive Algorithm Runs

- Evaluation of 1 configuration on 1 instance is already expensive (solving a $\mathcal{NP}$ problem)
- Evaluation of $n > 1000$ configurations on $m > 100$ instances can be infeasible in practice

Structured high-dimensional parameter space

- Categorical vs. continuous parameters
- Conditionals between parameters

Challenges of Algorithm Configuration

Expensive Algorithm Runs

- Evaluation of 1 configuration on 1 instance is already expensive (solving a $\mathcal{NP}$ problem)
- Evaluation of $n > 1000$ configurations on $m > 100$ instances can be infeasible in practice

Structured high-dimensional parameter space

- Categorical vs. continuous parameters
- Conditionals between parameters

Stochastic optimization

- Randomized algorithms: optimization across various seeds
- Distribution of benchmark instances (often wide range of hardness)
- Subsumes so-called *multi-armed bandit problem*

Algorithm Configuration: Components

1. Which configuration to choose?
2. How to evaluate a configuration?

Component 1: Which Configuration to Choose?

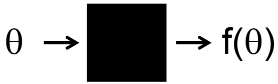
For this component, we can consider a simpler problem:
Blackbox function optimization: $\min_{\theta \in \boldsymbol{\Theta}} f(\theta)$

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

Component 1: Which Configuration to Choose?

For this component, we can consider a simpler problem:
Blackbox function optimization: $\min_{\theta \in \mathbf{\Theta}} f(\theta)$

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



- Abstracts away the complexity of evaluating multiple instances
- A query is expensive
- $\mathbf{\Theta}$ is still a structured space
  - Mixed continuous/discrete
  - Conditional parameters

Component 1: Which Configuration to Evaluate?

- Trade-off between diversification and intensification
- The extremes
  - Random search
  - Gradient Descent

Component 1: Which Configuration to Evaluate?

- Trade-off between diversification and intensification
- The extremes
  - Random search
  - Gradient Descent
- How would you solve this problem?

Component 1: Which Configuration to Evaluate?

- Trade-off between diversification and intensification
- The extremes
  - Random search
  - Gradient Descent
- How would you solve this problem? 
- Stochastic local search (SLS)
- Population-based methods
- Model-based Optimization (e.g. Bayesian Optimization)
- . . .

Component 2: How to Evaluate a Configuration?

Back to the general algorithm configuration problem

- Distribution over problem instances with domain $\mathcal{I}$;
- Performance metric $m : \mathbf{\Theta} \times \mathcal{I} \rightarrow \mathbb{R}$
- $c(\theta) = \mathbb{E}_{\pi \sim \mathcal{D}}(m(\theta, \pi))$

Component 2: How to Evaluate a Configuration?

Back to the general algorithm configuration problem

- Distribution over problem instances with domain $\mathcal{I}$;
- Performance metric $m : \mathbf{\Theta} \times \mathcal{I} \rightarrow \mathbb{R}$
- $c(\theta) = \mathbb{E}_{\pi \sim \mathcal{D}}(m(\theta, \pi))$

Simplest, suboptimal solution: use N runs for each evaluation

- Treats the problem as a blackbox function optimization problem
- Issue: how large to choose N?
  - too small: overtuning
  - too large: every function evaluation is slow

Component 2: How to Evaluate a Configuration?

Back to the general algorithm configuration problem

- Distribution over problem instances with domain $\mathcal{I}$;
- Performance metric $m : \mathbf{\Theta} \times \mathcal{I} \to \mathbb{R}$
- $c(\theta) = \mathbb{E}_{\pi \sim \mathcal{D}}(m(\theta, \pi))$

Simplest, suboptimal solution: use N runs for each evaluation

- Treats 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 to strive for

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

Racing algorithms: the general approach

Problem: which one of N candidate algorithms is best?

- Start with empty set of runs for each algorithm
- Iteratively:
  - Perform one run each
  - Discard inferior candidates
  - E.g., as judged by a statistical test (e.g., F-race uses an F-test)
- Stop when a single candidate remains or configuration budget expires

Saving Time: Aggressive Racing

- Race new configurations against the best known
  - Discard poor new configurations quickly
  - No requirement for statistical domination
  - Evaluate best configurations with many runs

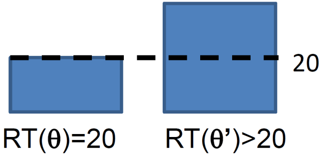
Saving Time: Aggressive Racing

- Race new configurations against the best known
  - Discard poor new configurations quickly
  - No requirement for statistical domination
  - Evaluate best configurations with many runs
- Search component should allow to return to configurations discarded because they were "unlucky"

Saving More Time: Adaptive Capping

When minimizing algorithm runtime,
we can terminate runs for poor configurations θ' early:

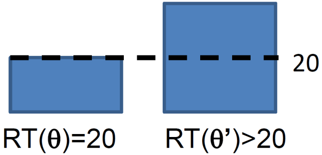
- Is θ' better than θ?
  - Example:



Saving More Time: Adaptive Capping

When minimizing algorithm runtime,
we can terminate runs for poor configurations θ' early:

- Is θ' better than θ?
  - Example:



- Can terminate evaluation of θ' once guaranteed to be worse than θ

General algorithm configuration systems

- ParamILS [Hutter et al, 2007 & 2009]
- Gender-based Genetic Algorithm (GGA) [Ansotegui et al, 2009]
- Iterated F-Race [López-Ibáñez et al, 2011]
- Sequential Model-based Algorithm Configuration (SMAC) [Hutter et al, since 2011]

The Baseline: Graduate Student Descent

Algorithm 1: Manual Greedy Algorithm Configuration

Start with some configuration θ

The Baseline: Graduate Student Descent

Algorithm 1: Manual Greedy Algorithm Configuration

Start with some configuration θ

    Modify a single parameter

The Baseline: Graduate Student Descent

Algorithm 1: Manual Greedy Algorithm Configuration

Start with some configuration θ

Modify a single parameter
if *results on benchmark set improve* **then**
keep new configuration

The Baseline: Graduate Student Descent

Algorithm 1: Manual Greedy Algorithm Configuration

Start with some configuration θ
repeat
  Modify a single parameter
  if *results on benchmark set improve* **then**
    keep new configuration
until *no more improvement possible (or "good enough")*

The Baseline: Graduate Student Descent

Algorithm 1: Manual Greedy Algorithm Configuration

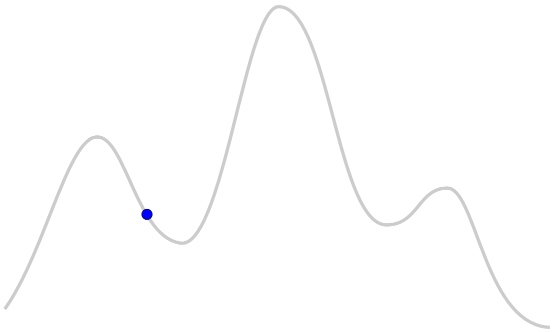
Start with some configuration θ
repeat
  Modify a single parameter
  if *results on benchmark set improve* **then**
    keep new configuration
until *no more improvement possible (or "good enough")*

$\rightsquigarrow$ Manually-executed first-improvement local search

Going Beyond Local Optima: Iterated Local Search

Animation credit: Holger Hoos

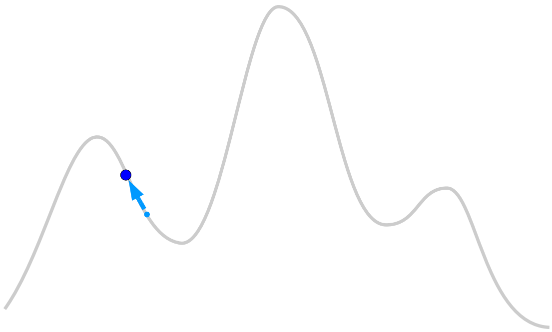
Going Beyond Local Optima: Iterated Local Search



Initialization

Animation credit: Holger Hoos

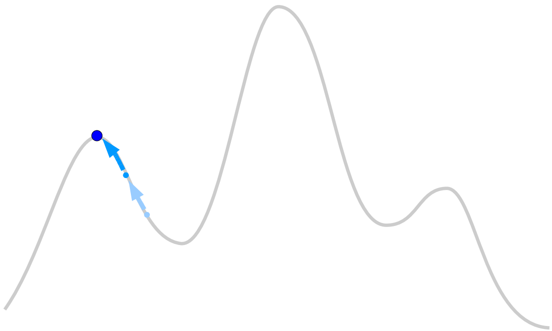
Going Beyond Local Optima: Iterated Local Search



Local Search

Animation credit: Holger Hoos

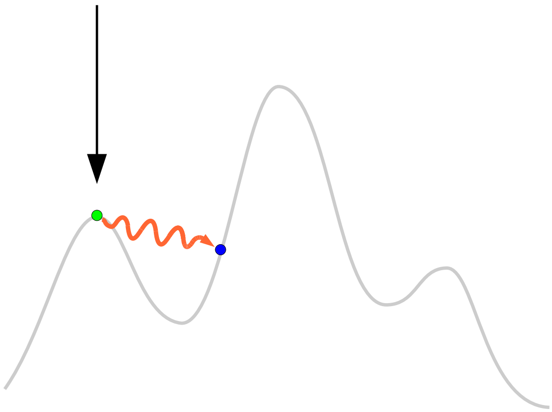
Going Beyond Local Optima: Iterated Local Search



Local Search

Animation credit: Holger Hoos

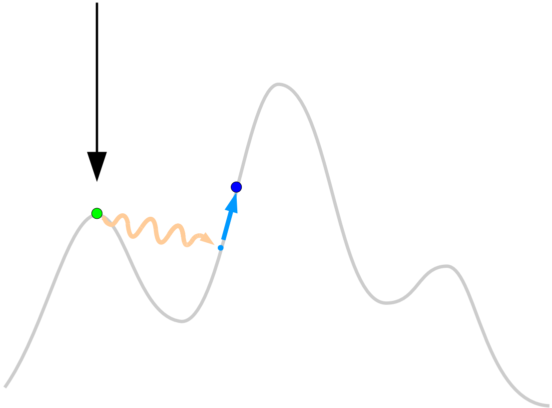
Going Beyond Local Optima: Iterated Local Search



Perturbation

Animation credit: Holger Hoos

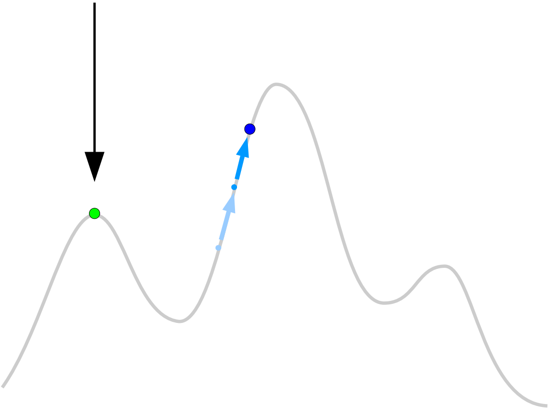
Going Beyond Local Optima: Iterated Local Search



Local Search

Animation credit: Holger Hoos

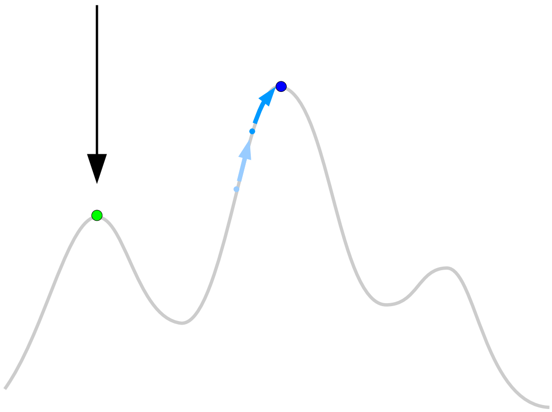
Going Beyond Local Optima: Iterated Local Search



Local Search

Animation credit: Holger Hoos

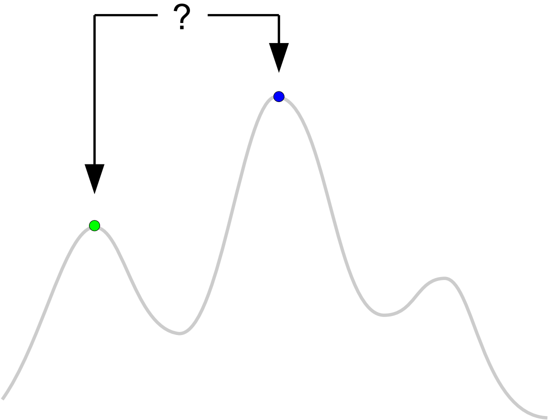
Going Beyond Local Optima: Iterated Local Search



Local Search

Animation credit: Holger Hoos

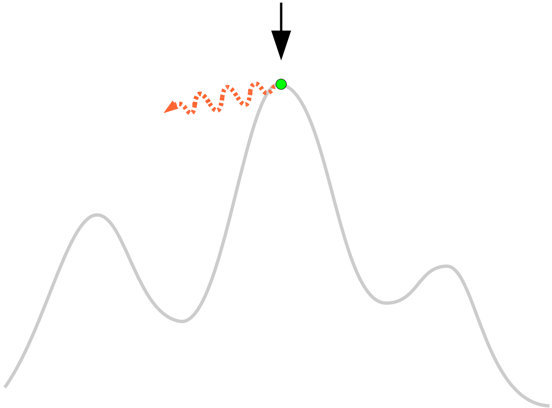
Going Beyond Local Optima: Iterated Local Search



Selection (using Acceptance Criterion)

Animation credit: Holger Hoos

Going Beyond Local Optima: Iterated Local Search



Perturbation

Animation credit: Holger Hoos

The *ParamILS* Framework [Hutter et al, 2007 & 2009]

ParamILS = Iterated Local Search in parameter configuration space

$\rightsquigarrow$ Performs biased random walk over local optima

The *ParamILS* Framework [Hutter et al, 2007 & 2009]

ParamILS = Iterated Local Search in parameter configuration space

$\rightsquigarrow$ Performs biased random walk over local optima

How to evaluate a configuration's quality?

- BasicILS(N): use N fixed instances
- FocusedILS: increase #instances for good configurations over time

The *ParamILS* Framework [Hutter et al, 2007 & 2009]

Advantages

- Theoretically shown to converge
- Often quickly finds local improvements over default (can exploit a good default)
- Very randomized $\rightarrow$ almost k-fold speedup for k parallel runs

The *ParamILS* Framework [Hutter et al, 2007 & 2009]

Advantages

- Theoretically shown to converge
- Often quickly finds local improvements over default (can exploit a good default)
- Very randomized $\rightarrow$ almost k-fold speedup for k parallel runs

Disadvantages

- Very randomized $\rightarrow$ unreliable when only run once for a short time
- Can be slow to find the global optimum

GGA [Ansotegui et al, 2009]

Genetic algorithm for algorithm configuration

- Genes = parameter values
- Population: trades of exploration and exploitation

GGA [Ansotegui et al, 2009]

Genetic algorithm for algorithm configuration

- Genes = parameter values
- Population: trades of exploration and exploitation
- Use N instances to evaluate configurations in each generation
  - Increase N in each generation: linearly from N_{start} to N_{end}

GGA [Ansotegui et al, 2009]

Genetic algorithm for algorithm configuration

- Genes = parameter values
- Population: trades of exploration and exploitation
- Use N instances to evaluate configurations in each generation
  - Increase N in each generation: linearly from N_{start} to N_{end}

Advantages

- Easy to use parallel resources: evaluate several population members in parallel

GGA [Ansotegui et al, 2009]

Genetic algorithm for algorithm configuration

- Genes = parameter values
- Population: trades of exploration and exploitation
- Use N instances to evaluate configurations in each generation
  - Increase N in each generation: linearly from N_{start} to N_{end}

Advantages

- Easy to use parallel resources: evaluate several population members in parallel

Disadvantages

- User has to specify #generations ahead of time
- Not recommended for small budgets and categorical parameters

Iterated F-race [López-Ibáñez et al, 2011]

Basic Idea

- Use F-Race as a building block
- Iteratively sample configurations to race

Iterated F-race [López-Ibáñez et al, 2011]

Basic Idea

- Use F-Race as a building block
- Iteratively sample configurations to race

Advantages

- Can parallelize easily: runs of each racing iteration are independent
- Well-supported software package (for the community that uses R)

Iterated F-race [López-Ibáñez et al, 2011]

Basic Idea

- Use F-Race as a building block
- Iteratively sample configurations to race

Advantages

- Can parallelize easily: runs of each racing iteration are independent
- Well-supported software package (for the community that uses R)

Disadvantages

- Does not support adaptive capping → Don't use for runtime
- The sampling of new configurations is not very strong for complex search spaces

SMAC in a Nutshell [Hutter et al, since 2011]

SMAC = Sequential Model-based Algorithm Configuration

- Use a predictive model of algorithm performance to guide the search
- Combine this search strategy with aggressive racing & adaptive capping

SMAC in a Nutshell [Hutter et al, since 2011]

SMAC = Sequential Model-based Algorithm Configuration

- Use a predictive model of algorithm performance to guide the search
- Combine this search strategy with aggressive racing & adaptive capping

One SMAC iteration

- Construct a model to predict performance
- Use that model to select promising configurations
- Compare each of the selected configurations against the best known
  - Using a similar procedure as FocusedILS

Bayesian Optimization – Detour into Machine Learning

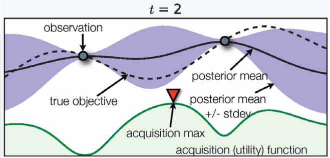
General approach

- Fit a probabilistic model to the collected function samples $\langle \theta, f(\theta) \rangle$
- Use the model to guide optimization, trading off exploration *vs* exploitation

Bayesian Optimization – Detour into Machine Learning

General approach

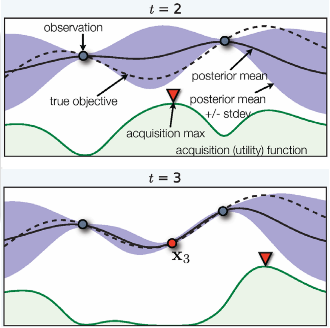
- Fit a probabilistic model to the collected function samples $\langle \theta, f(\theta) \rangle$
- Use the model to guide optimization, trading off exploration *vs* exploitation



Bayesian Optimization – Detour into Machine Learning

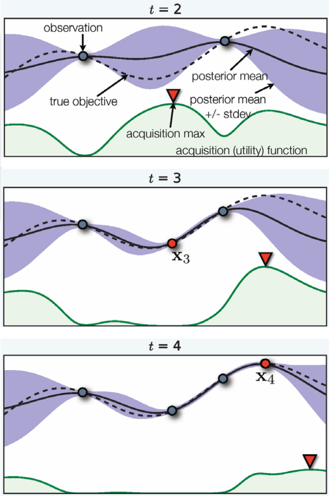
General approach

- Fit a probabilistic model to the collected function samples $\langle \theta, f(\theta) \rangle$
- Use the model to guide optimization, trading off exploration *vs* exploitation



General approach

- Fit a probabilistic model to the collected function samples $\langle\theta, f(\theta)\rangle$
- Use the model to guide optimization, trading off exploration *vs* exploitation



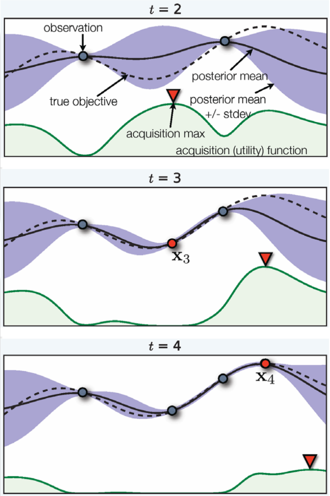
Bayesian Optimization – Detour into Machine Learning

General approach

- Fit a probabilistic model to the collected function samples $\langle \theta, f(\theta) \rangle$
- Use the model to guide optimization, trading off exploration *vs* exploitation

Popular approach in the statistics literature since [Mockus, 1978]

- Efficient in # function evaluations
- Works when objective is nonconvex, noisy, has unknown derivatives, etc
- Recent convergence results
  [Srinivas et al, 2010; Bull 2011; de Freitas et al, 2012; Kawaguchi et al, 2015]



Powering SMAC: Empirical Performance Models

Empirical Performance Models

Given:

- Configuration space $\mathbf{\Theta} = \Theta_1 \times \cdots \times \Theta_n$
- For each problem instance π_i: $\mathbf{f_i}$, a vector of feature values
- Observed algorithm runtime data: $\langle(\theta_i, \mathbf{f}_i, y_i)\rangle_{i=1}^{N}$

Find: a mapping $\hat{m} : [\theta, \mathbf{f}] \mapsto y$ predicting performance

Powering SMAC: Empirical Performance Models

Empirical Performance Models

Given:

- Configuration space $\mathbf{\Theta} = \Theta_1 \times \cdots \times \Theta_n$
- For each problem instance π_i: $\mathbf{f_i}$, a vector of feature values
- Observed algorithm runtime data: $\langle(\theta_i, \mathbf{f}_i, y_i)\rangle_{i=1}^{N}$

Find: a mapping $\hat{m} : [\theta, \mathbf{f}] \mapsto y$ predicting performance

Which type of regression model?

- Rich literature on performance prediction (overview: [Hutter et al, AIJ 2014])
- Here: we use a model $\hat{m}$ based on random forests

Instance Features for SAT [Hutter et al, 2014]

Instance Features

Instance features are numerical representations of instances.

Instance Features for SAT [Hutter et al, 2014]

Instance Features

Instance features are numerical representations of instances.

What could be instance features for CNFs?

Instance Features for SAT [Hutter et al, 2014]

Instance Features

Instance features are numerical representations of instances.

What could be instance features for CNFs? 

Static Features

- Problem size features
- Variable-Clause graph features
- Variable graph features
- Clause graph features
- Balance features

Probing Features

- DPLL probing
- LP-based Probing
- SLS Probing
- CDCL Probing
- Survey Propagation

SMAC: Overview

Algorithm 2: SMAC

Initialize with a single run for the default

SMAC: Overview

Algorithm 2: SMAC

Initialize with a single run for the default

Learn a RF model from data so far: $\hat{m} : \mathbf{\Theta} \times \mathcal{I} \rightarrow \mathbb{R}$

SMAC: Overview

Algorithm 2: SMAC

Initialize with a single run for the default

Learn a RF model from data so far: $\hat{m} : \mathbf{\Theta} \times \mathcal{I} \rightarrow \mathbb{R}$

Aggregate over instances: $\hat{f}(\theta) = \mathbb{E}_{\pi \sim \mathcal{D}}(\hat{m}(\theta, \pi))$

Algorithm 2: SMAC

Initialize with a single run for the default

Learn a RF model from data so far: $\hat{m} : \mathbf{\Theta} \times \mathcal{I} \rightarrow \mathbb{R}$

Aggregate over instances: $\hat{f}(\theta) = \mathbb{E}_{\pi \sim \mathcal{D}}(\hat{m}(\theta, \pi))$

Use model $\hat{f}$ to select promising configurations

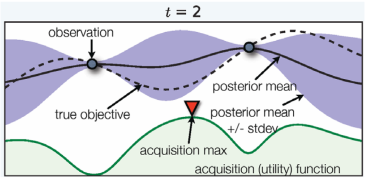
Algorithm 2: SMAC

Initialize with a single run for the default

Learn a RF model from data so far: $\hat{m} : \mathbf{\Theta} \times \mathcal{I} \to \mathbb{R}$

Aggregate over instances: $\hat{f}(\theta) = \mathbb{E}_{\pi\sim\mathcal{D}}(\hat{m}(\theta, \pi))$

Use model $\hat{f}$ to select promising configurations



SMAC: Overview

Algorithm 2: SMAC

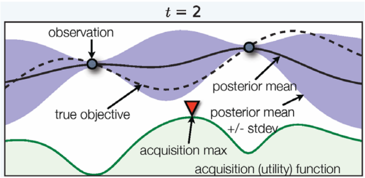
Initialize with a single run for the default

Learn a RF model from data so far: $\hat{m} : \mathbf{\Theta} \times \mathcal{I} \to \mathbb{R}$

Aggregate over instances: $\hat{f}(\theta) = \mathbb{E}_{\pi \sim \mathcal{D}}(\hat{m}(\theta, \pi))$

Use model $\hat{f}$ to select promising configurations

Race selected configurations against best known



SMAC: Overview

Algorithm 2: SMAC

Initialize with a single run for the default

repeat

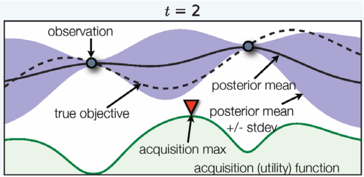
Learn a RF model from data so far: $\hat{m} : \mathbf{\Theta} \times \mathcal{I} \rightarrow \mathbb{R}$

Aggregate over instances: $\hat{f}(\theta) = \mathbb{E}_{\pi \sim \mathcal{D}}(\hat{m}(\theta, \pi))$

Use model $\hat{f}$ to select promising configurations

Race selected configurations against best known

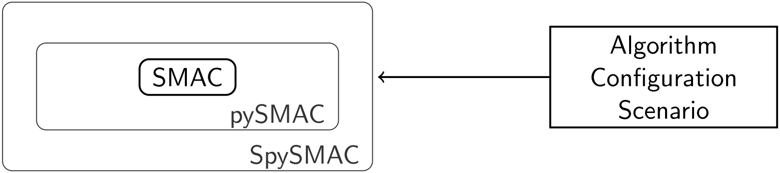
until *time budget exhausted*



Outline

1. The Algorithm Configuration Problem
2. Using AC Systems
3. Importance of Parameters
4. Pitfalls and Best Practices
5. Final Remarks

SMAC, pySMAC, SpySMAC

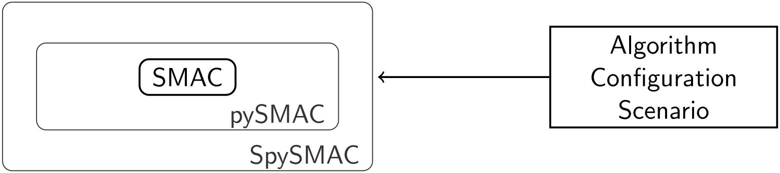


SMAC Configurator implemented in JAVA

pySMAC Python Interface to *SMAC*

SpySMAC SAT-pySMAC: an easy-to-use AC framework for SAT-solvers

SMAC, pySMAC, SpySMAC



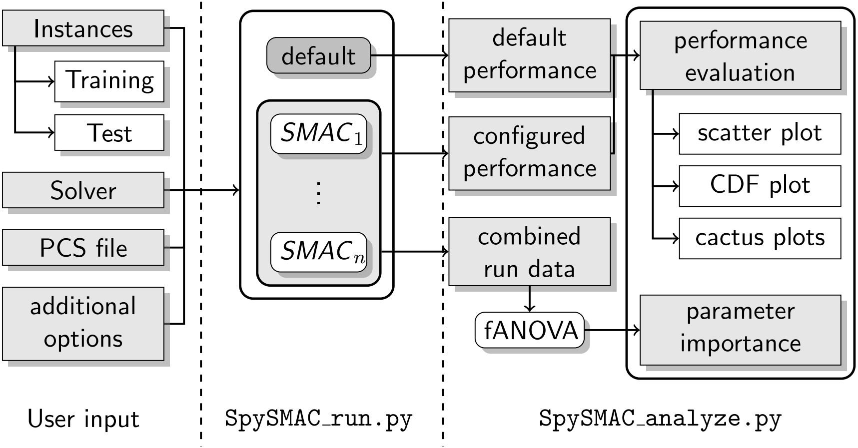
SMAC Configurator implemented in JAVA

pySMAC Python Interface to *SMAC*

SpySMAC SAT-pySMAC: an easy-to-use AC framework for SAT-solvers

Future: One tool in Python.

SpySMAC [Falkner et al, 2015]



Example: *MiniSAT* [Een et al, '03-'07]

MiniSAT (`http://minisat.se/`) is a SAT solver that is

- minimalistic,
- open-source,
- and developed to help researchers and developers alike to get started on SAT

Example: *MiniSAT* [Een et al, '03-'07]

MiniSAT (`http://minisat.se/`) is a SAT solver that is

- minimalistic,
- open-source,
- and developed to help researchers and developers alike to get started on SAT
- *MiniSAT* has 8 (performance-relevant) parameters

```
CORE OPTIONS:

  -rnd-init, -no-rnd-init                        (default: off)
  -luby, -no-luby                                (default: on)

  -rnd-freq      = <double> [   0 ..    1] (default: 0)
  -rnd-seed      = <double> (   0 ..  inf) (default: 9.16483e+07)
  -var-decay     = <double> (   0 ..    1) (default: 0.95)
  -cla-decay     = <double> (   0 ..    1) (default: 0.999)
  -rinc          = <double> (   1 ..  inf) (default: 2)
  -gc-frac       = <double> (   0 ..  inf) (default: 0.2)

  -rfirst        = <int32>  [   1 .. imax] (default: 100)
  -ccmin-mode    = <int32>  [   0 ..    2] (default: 2)
  -phase-saving  = <int32>  [   0 ..    2] (default: 2)

MAIN OPTIONS:
```

Hands-on: *SpySMAC*

Determine optimized configuration

```
$ python SpySMAC_run.py      ← Call
-i swv-inst/SWV-GZIP/        ← Instances
-b minisat/core/minisat      ← Binary
-p minisat/pcs.txt           ← Configuration Space
-o minisat-logs              ← log-files
--prefix "-"                 ← parameter prefix
-c 2                         ← cutoff
-B 60                        ← budget [sec]
```

Specifying Parameter Configuration Spaces (PCS)

There are many different types of parameter

- As for other combinatorial problems, there is a standard representation that different configuration procedures can read

Specifying Parameter Configuration Spaces (PCS)

There are many different types of parameter

- As for other combinatorial problems, there is a standard representation that different configuration procedures can read

The simple standard format: PCS

- PCS (short for "parameter configuration space")
- human readable/writable
- allows to express a wide range of parameter types

PCS Example: *MiniSAT*

```
rnd-freq [0,1][0]
var-decay [0.001,1][0.95]l
cla-decay [0.001,1][0.999]l
rinc [1.00001,1024][2]l
gc-frac [0,1][0.2]
rfirst [1,10000000][100]il
ccmin-mode {0,1,2}[2]
phase-saving {0,1,2}[2]
```

```
CORE OPTIONS:

  -rnd-init, -no-rnd-init                (default: off)
  -luby, -no-luby                        (default: on)

  -rnd-freq     = <double> [   0 ..    1] (default: 0)
  -rnd-seed     = <double> (   0 ..  inf) (default: 9.16483e+07)
  -var-decay    = <double> (   0 ..    1) (default: 0.95)
  -cla-decay    = <double> (   0 ..    1) (default: 0.999)
  -rinc         = <double> (   1 ..  inf) (default: 2)
  -gc-frac      = <double> (   0 ..  inf) (default: 0.2)

  -rfirst       = <int32>  [   1 .. imax] (default: 100)
  -ccmin-mode   = <int32>  [   0 ..    2] (default: 2)
  -phase-saving = <int32>  [   0 ..    2] (default: 2)

MAIN OPTIONS:

  -verb         = <int32>  [   0 ..    2] (default: 1)
  -cpu-lim      = <int32>  [   0 .. imax] (default: 2147483647)
  -mem-lim      = <int32>  [   0 .. imax] (default: 2147483647)

HELP OPTIONS:

  --help        Print help message.
  --help-verb   Print verbose help message.
```

Decision: Configuration Budget and Cutoff

Configuration Budget

- Dictated by your resources and 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

Decision: Configuration Budget and Cutoff

Configuration Budget

- Dictated by your resources and 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 cutoff time 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 at least 300 CPU seconds

Live Demo of a *SpySMAC* Report

Outline

1. The Algorithm Configuration Problem
2. Using AC Systems
3. Importance of Parameters
   - Ablation
   - fANOVA
4. Pitfalls and Best Practices
5. Final Remarks

Parameter Importance

Recommendations & Observation

- Configure all parameters that could influence performance
- Dependent on the instance set, different parameters matter
- How to determine the important parameters?

Parameter Importance

Recommendations & Observation

- Configure all parameters that could influence performance
- Dependent on the instance set, different parameters matter
- How to determine the important parameters?

Example

- SAT-solver *lingeling* has more than 300 parameters
- Often, less than 10 are important to optimize performance

Outline

1. The Algorithm Configuration Problem
2. Using AC Systems
3. Importance of Parameters
   - Ablation
   - fANOVA
4. Pitfalls and Best Practices
5. Final Remarks

Ablation [Fawcett et al. 2013]

Idea

- Starting from the default configuration, we change the value of the parameters
- Which of these changes were important?

$\rightarrow$ Ablation compares parameter flips between default and incumbent configuration

Ablation [Fawcett et al. 2013]

Idea

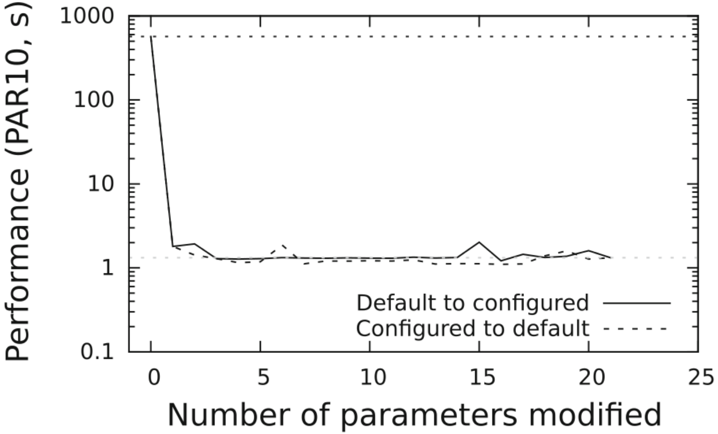
- Starting from the default configuration, we change the value of the parameters
- Which of these changes were important?

→ Ablation compares parameter flips between default and incumbent configuration

Basic Approach

- Iterate over all non-flipped parameters
- Flip the parameter with the largest influence on the performance in each iteration

Ablation Example: *Spear* on SWV



Source: [Fawcett et al. 2013]

Outline

1. The Algorithm Configuration Problem
2. Using AC Systems
3. Importance of Parameters
   - Ablation
   - fANOVA
4. Pitfalls and Best Practices
5. Final Remarks

Reminder: Empirical Performance Model (EPM)

Using an EPM $\hat{m} : \mathbf{\Theta} \rightarrow \mathbb{R}$, predict the performance of configurations θ.

fANOVA [Hutter et al. 2014]

Reminder: Empirical Performance Model (EPM)

Using an EPM $\hat{m} : \mathbf{\Theta} \rightarrow \mathbb{R}$, predict the performance of configurations θ.

fANOVA [Sobol 1993]

Using *fANOVA*, write performance predictions $\hat{y}$ as a sum of components:

$$\hat{y}(\theta_1, \ldots, \theta_n) = \quad \hat{m}_0 + \sum_{i=1}^{n} \hat{m}_i(\theta_i) + \sum_{i \neq j} \hat{m}_{ij}(\theta_i, \theta_j) + \ldots$$

fANOVA [Hutter et al. 2014]

Reminder: Empirical Performance Model (EPM)

Using an EPM $\hat{m} : \mathbf{\Theta} \to \mathbb{R}$, predict the performance of configurations θ.

fANOVA [Sobol 1993]

Using *fANOVA*, write performance predictions $\hat{y}$ as a sum of components:

$$\hat{y}(\theta_1, \dots, \theta_n) = \hat{m}_0 + \sum_{i=1}^{n} \hat{m}_i(\theta_i) + \sum_{i \neq j} \hat{m}_{ij}(\theta_i, \theta_j) + \dots$$

With variance decomposition, compute the performance variance explained by a single parameter (or combinations of them)

Reminder: Empirical Performance Model (EPM)

Using an EPM $\hat{m} : \mathbf{\Theta} \to \mathbb{R}$, predict the performance of configurations θ.

fANOVA [Sobol 1993]

Using *fANOVA*, write performance predictions $\hat{y}$ as a sum of components:

$$\hat{y}(\theta_1, \ldots, \theta_n) = \hat{m}_0 + \sum_{i=1}^{n} \hat{m}_i(\theta_i) + \sum_{i \neq j} \hat{m}_{ij}(\theta_i, \theta_j) + \ldots$$

With variance decomposition, compute the performance variance explained by a single parameter (or combinations of them)

Application to Parameter Importance

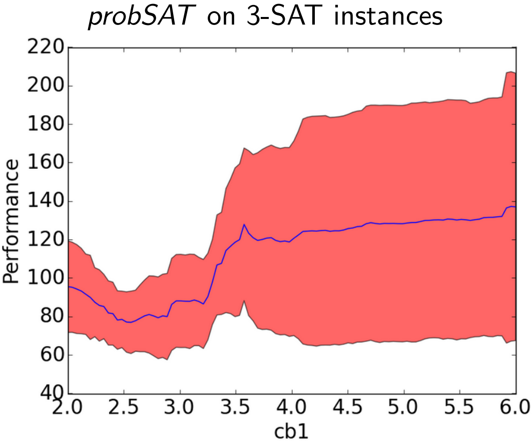
How much of the variance can be explained by a parameter (or combinations of parameters) marginalized over all other parameters?

fANOVA Example

lingeling on circuit fuzz

Parameter	Importance
score	24.95
minlocalgluelim	6.52
blkclslim	0.85
gaussreleff	0.85
blksuccesslim	0.79
seed	0.70
unhdlnpr	0.51
gluekeep	0.47
trnrmaxeff	0.47
blkboostvlim	0.47

fANOVA Example



Comparison Parameter Importance Procedures

Ablation

+ Only method to compare two configurations
- Needs a lot of algorithm runs $\rightarrow$ slow

fANOVA

+ EPM can be trained by the performance data collected during configuration
+ Considers the complete configuration space or only "interesting" areas
- Importance of interactions between parameters can be expensive

Outline

1. The Algorithm Configuration Problem
2. Using AC Systems
3. Importance of Parameters
4. Pitfalls and Best Practices
   - Overtuning
   - General Advice
5. Final Remarks

Outline

1. The Algorithm Configuration Problem

2. Using AC Systems

3. Importance of Parameters

4. Pitfalls and Best Practices
   - Overtuning
   - General Advice

5. Final Remarks

Generalization of Performance

The dark ages

1. Student tweaks the parameters manually on 10 problems until it works
2. Supervisor may not even know about the tuning
3. Results get published without acknowledging the tuning
4. Of course, the approach does not generalize

Generalization of Performance

The dark ages

1. Student tweaks the parameters manually on 10 problems until it works
2. Supervisor may not even know about the tuning
3. Results get published without acknowledging the tuning
4. Of course, the approach does not generalize

How to do better? 

Generalization of Performance

The dark ages

1. Student tweaks the parameters manually on 10 problems until it works
2. Supervisor may not even know about the tuning
3. Results get published without acknowledging the tuning
4. Of course, the approach does not generalize

A step further

- Optimize parameters on a training set
- Evaluate generalization on a test set

Generalization of Performance

The dark ages

1. Student tweaks the parameters manually on 10 problems until it works
2. Supervisor may not even know about the tuning
3. Results get published without acknowledging the tuning
4. Of course, the approach does not generalize

A step further

- Optimize parameters on a training set
- Evaluate generalization on a test set

Even better: avoid "peeking" at the test set

- Put test set into a vault (i.e., never look at it)
- Split training set again into training and validation set
- Only use test set in the end to generate results for publication

The concept of overtuning

Very related to overfitting in machine learning

- Performance improves on the training set
- Performance does not improve on the test set, and may even degrade

The concept of overtuning

Very related to overfitting in machine learning

- Performance improves on the training set
- Performance does not improve on the test set, and may even degrade

More pronounced for more heterogeneous benchmark sets

- But it even happens for very homogeneous sets
- Indeed, one can even overfit on a single instance, to the seeds used for training

Overtuning Visualized

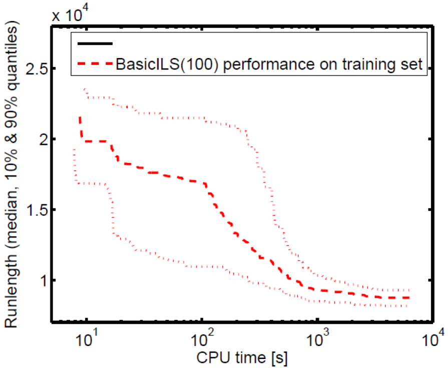
Example: minimizing SLS solver runlengths for a single SAT instance

- Training cost, here based on N=100 runs with different seeds
- Test cost of $\hat{\theta}$ here based on 1000 new seeds

Overtuning Visualized

Example: minimizing SLS solver runlengths for a single SAT instance

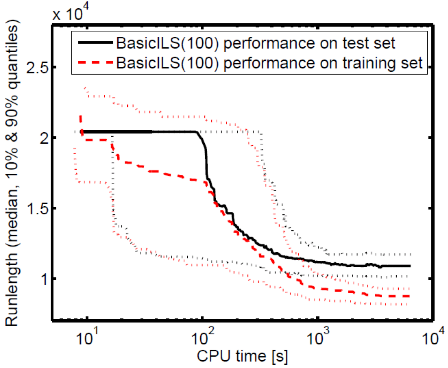
- Training cost, here based on N=100 runs with different seeds
- Test cost of $\hat{\theta}$ here based on 1000 new seeds



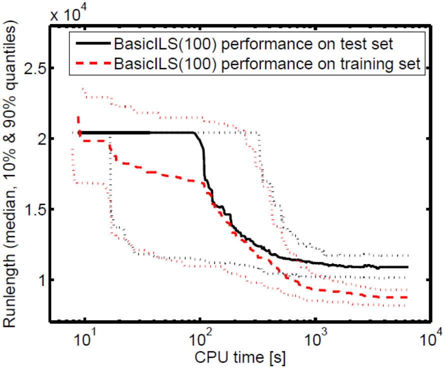
Overtuning Visualized

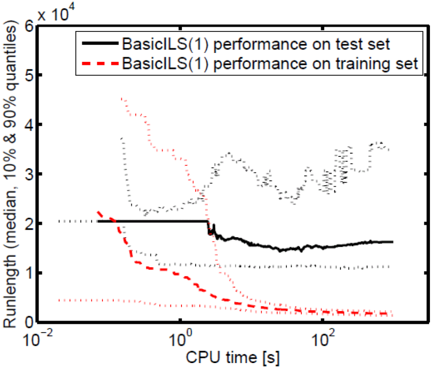
Example: minimizing SLS solver runlengths for a single SAT instance

- Training cost, here based on N=100 runs with different seeds
- Test cost of $\hat{\theta}$ here based on 1000 new seeds

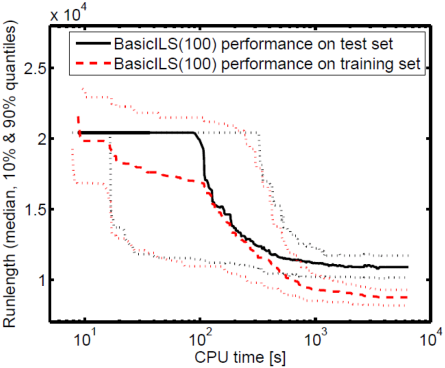


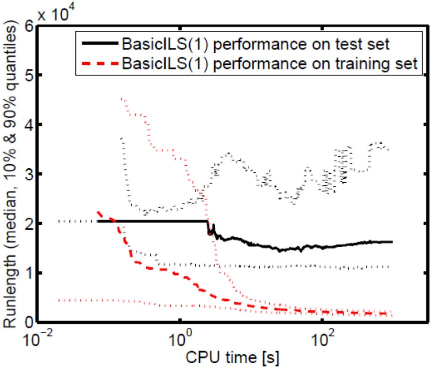
Overtuning is Stronger For Smaller Training Sets





Overtuning is Stronger For Smaller Training Sets





Best Practice

Provide as many instances as possible, and we will take care to run only as many as necessary.

General advice: make solver's randomness explicit

Several communities dislike randomness

Key arguments: reproducibility, tracking down bugs

- I agree these are important
- But you can achieve them by keeping track of your seeds
- In fact: your tests will cover more cases when randomized

General advice: make solver's randomness explicit

Several communities dislike randomness

Key arguments: reproducibility, tracking down bugs

- I agree these are important
- But you can achieve them by keeping track of your seeds
- In fact: your tests will cover more cases when randomized

It's much easier to get more seeds than more instances

- Performance should generalize to new seeds
- Otherwise, it's less likely to generalize to new instances

Different Types of Overtuning

One can overtune to various specifics of the training setup

- To the specific instances used in the training set
- To the specific seeds used in the training set

Different Types of Overtuning

One can overtune to various specifics of the training setup

- To the specific instances used in the training set
- To the specific seeds used in the training set
- To the (small) runtime cutoff used during training
- To a particular machine type

Different Types of Overtuning

One can overtune to various specifics of the training setup

- To the specific instances used in the training set
- To the specific seeds used in the training set
- To the (small) runtime cutoff used during training
- To a particular machine type
- To the type of instances in the training set
  - These should just be drawn according to the distribution of interest
  - But in practice, the distribution might change over time

Outline

1. The Algorithm Configuration Problem
2. Using AC Systems
3. Importance of Parameters
4. Pitfalls and Best Practices
   - Overtuning
   - General Advice
5. Final Remarks

Choosing the Training Instances #1

Split instance set into training and test sets

- Configure on the training instances $\rightarrow$ configuration $\hat{\theta}$
- Run (only) $\hat{\theta}$ on the test instances $\rightarrow$ unbiased performance estimate

Choosing the Training Instances #1

Split instance set into training and test sets

- Configure on the training instances $\rightarrow$ configuration $\hat{\theta}$
- Run (only) $\hat{\theta}$ on the test instances $\rightarrow$ unbiased performance estimate

Pitfall

Configuring on your test instances

$\rightarrow$ overtuning effects – no unbiased performance estimate

Choosing the Training Instances #1

Split instance set into training and test sets

- Configure on the training instances $\rightarrow$ configuration $\hat{\theta}$
- Run (only) $\hat{\theta}$ on the test instances $\rightarrow$ unbiased performance estimate

Pitfall

Configuring on your test instances

$\rightarrow$ overtuning effects – no unbiased performance estimate

Fine practice

Do multiple configuration runs and pick the $\hat{\theta}$ with the best training performance

Choosing the Training Instances #2

AC works much better on homogeneous instance sets

- Instances 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

Choosing the Training Instances #2

AC works much better on homogeneous instance sets

- Instances 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 (e.g., SAT Competition)

$\rightarrow$ There often is no single great overall configuration

Choosing the Training Instances: Recommendation

Representative instances

- Representative of the instances you want to solve later

Choosing the Training Instances: Recommendation

Representative instances

- Representative of the instances you want to solve later

Moderately hard instances

- Too hard: will not 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 cutoff time

Choosing the Training Instances: Recommendation

Representative instances

- Representative of the instances you want to solve later

Moderately hard instances

- Too hard: will not 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 cutoff time

Enough instances

- The more training instances the better
- Very homogeneous instance sets: 50 instances might suffice
- Preferably ≥ 300 instances, better even ≥ 1000 instances

Using parallel computation

Simplest method: use multiple independent configurator runs

This can work very well [Hutter et al, LION 2012]

- FocusedILS: basically linear speedups with up to 16 runs
- SMAC: about 8-fold speedup with 16 runs

Using parallel computation

Simplest method: use multiple independent configurator runs

This can work very well [Hutter et al, LION 2012]

- FocusedILS: basically linear speedups with up to 16 runs
- SMAC: about 8-fold speedup with 16 runs

Distributed SMAC (d-SMAC) [Hutter et al, LION 2012]

Up to 50-fold speedups with 64 workers

- But so far synchronous parallelization
- Not applicable for runtime optimization

Using parallel computation

Simplest method: use multiple independent configurator runs

This can work very well [Hutter et al, LION 2012]

- FocusedILS: basically linear speedups with up to 16 runs
- SMAC: about 8-fold speedup with 16 runs

Distributed SMAC (d-SMAC) [Hutter et al, LION 2012]

Up to 50-fold speedups with 64 workers

- But so far synchronous parallelization
- Not applicable for runtime optimization

Parallel SMAC (p-SMAC) [unpublished]

Simple asynchronous scheme

- Simply exectue k different SMAC runs with different seeds
- Add `--shared-model-mode true`

Outline

1. The Algorithm Configuration Problem
2. Using AC Systems
3. Importance of Parameters
4. Pitfalls and Best Practices
5. Final Remarks

Advanced Topics

Automatic Construction of Parallel Portfolios [Lindauer et al, AIJ 2016]
parallel portfolio of complementary parameter configurations

Advanced Topics

Automatic Construction of Parallel Portfolios [Lindauer et al, AIJ 2016]

parallel portfolio of complementary parameter configurations

Robust Benchmark Sets [Hoos et al, LION 2013]

selection of a benchmark set to get a robust parameter configuration

Advanced Topics

Automatic Construction of Parallel Portfolios [Lindauer et al, AIJ 2016]
: parallel portfolio of complementary parameter configurations

Robust Benchmark Sets [Hoos et al, LION 2013]
: selection of a benchmark set to get a robust parameter configuration

SpyBug: Automated Bug Detection [Manthey et al, SAT 2016]
: search for bugs in the configuration space

Advanced Topics

Automatic Construction of Parallel Portfolios [Lindauer et al, AIJ 2016]
: parallel portfolio of complementary parameter configurations

Robust Benchmark Sets [Hoos et al, LION 2013]
: selection of a benchmark set to get a robust parameter configuration

SpyBug: Automated Bug Detection [Manthey et al, SAT 2016]
: search for bugs in the configuration space

Per-Instance Algorithm Selection [Xu et al, AAAI 2010]
: selection of a well-performing configuration for an instance at hand

Further tips and tricks

Further Tools

- see `www.ml4aad.org`

Further tips and tricks

Further Tools

- see `www.ml4aad.org`

There is extensive documentation

`http://aclib.net/smac`

- Quickstart guide, FAQ, extensive manual
- E.g., resuming SMAC runs, warmstarting with previous runs, etc.

Further tips and tricks

Further Tools

- see `www.ml4aad.org`

There is extensive documentation

`http://aclib.net/smac`

- Quickstart guide, FAQ, extensive manual
- E.g., resuming SMAC runs, warmstarting with previous runs, etc.

Ask questions in the SMAC Forum

`https://groups.google.com/forum/#!forum/smac-forum`

- It can also help to read through others' issues and solutions

Thank you!

ML4AAD