Automatic solver configuration using SMAC

How to boost performance of your SAT solver?

Marius Lindauer[1]

University of Freiburg

SAT Industrial Day 2016, Bordeaux

[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.
```

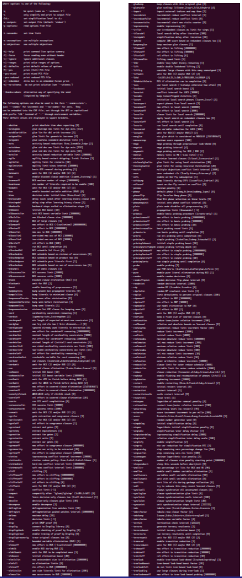
Ever looked into `--help`?

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

$\rightsquigarrow$ Robust performance across instances

$\rightsquigarrow$ Similar to downloading a solver and using its defaults

```
$ lingeling inst.cnf
```

Importance of Algorithm Configuration?

SAT Competition

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

$\rightsquigarrow$ Robust performance across instances

$\rightsquigarrow$ Similar to downloading a solver and using its defaults

```
$ lingeling inst.cnf
```

Configurable SAT Solver Challenge (CSSC)

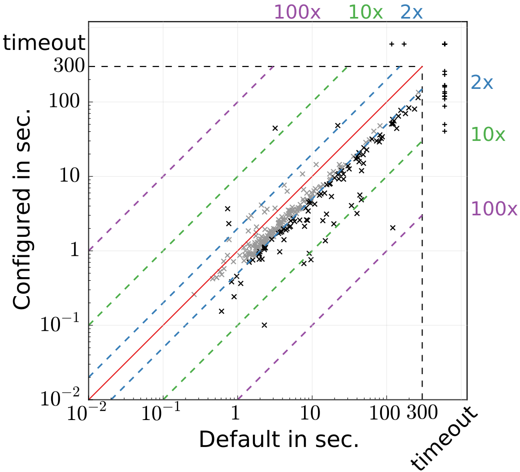
- Submission of a solver
- We tuned the parameter configuration for each instance set

$\rightsquigarrow$ Peak performance on each set

```
$ lingeling inst.cnf cce3wait=2 deco=1 saturating=73 lkhdmisifelmrtc=0
```

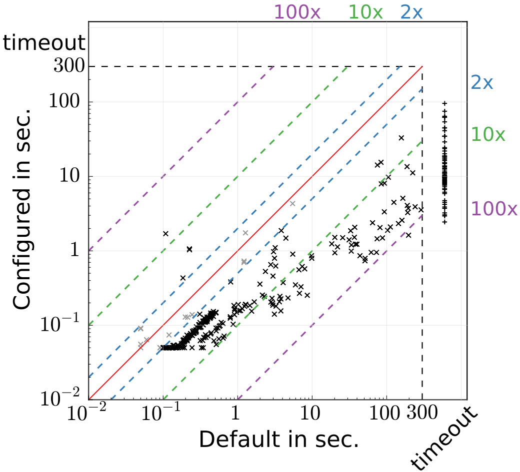
Importance of Algorithm Configuration? (from CSSC)

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



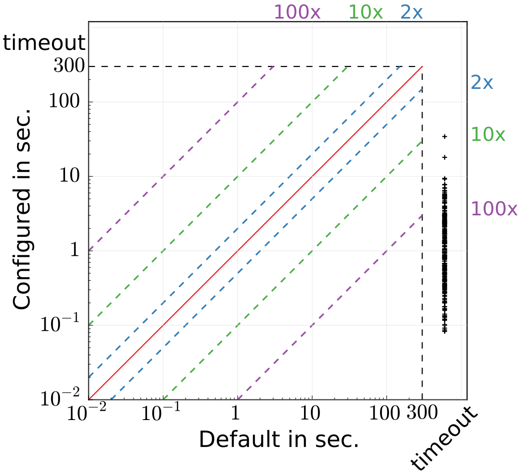
Importance of Algorithm Configuration? (from CSSC)

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



Importance of Algorithm Configuration? (from CSSC)

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



What is this talk about?

In a Nutshell: Algorithm Configuration

How to automatically determine a well-performing parameter configuration with SMAC?

What is this talk about?

In a Nutshell: Algorithm Configuration

How to automatically determine a well-performing parameter configuration with SMAC?

Focus on basics

1. State-of-the-art in algorithm configuration (Focus: *SMAC*)
2. Parameter importance
3. Advanced Topics

What is this talk about?

In a Nutshell: Algorithm Configuration

How to automatically determine a well-performing parameter configuration with SMAC?

Focus on basics

1. State-of-the-art in algorithm configuration (Focus: *SMAC*)
2. Parameter importance
3. Advanced Topics

- Please ask questions
- All literature references are hyperlinks

Slides at: `www.ml4aad.org`

Outline

Why Automated Algorithm Configuration?

Manual Tuning

- Requires a lot of expert knowledge about the solver and the instances
- Tedious and time-intensive task for a human
- Error-prone to human bias

Why Automated Algorithm Configuration?

Manual Tuning

- Requires a lot of expert knowledge about the solver and the instances
- Tedious and time-intensive task for a human
- Error-prone to human bias

Automatic Tuning

- More systematic search (less error-prone)
- Costs only computational time (after a short setup phase)
- Expensive human-time can be used for more creative tasks

Why Automated Algorithm Configuration?

Manual Tuning

- Requires a lot of expert knowledge about the solver and the instances
- Tedious and time-intensive task for a human
- Error-prone to human bias

1 week of a software developer costs: $\approx 1000 - 2000$ EURO

Automatic Tuning

- More systematic search (less error-prone)
- Costs only computational time (after a short setup phase)
- Expensive human-time can be used for more creative tasks

1 CPU week costs in the cloud: $\approx 2 - 5$ EURO

Why *SMAC*?

SMAC? [Hutter et al, LION 2011]

- Automatic algorithm configuration tool
- Combining cutting-edge optimization and machine learning methods

Why *SMAC*?

SMAC? [Hutter et al, LION 2011]

- Automatic algorithm configuration tool
- Combining cutting-edge optimization and machine learning methods

Advantages by using *SMAC*

1. Real algorithm configuration (opposed to "parameter tuning")
    - Handles different types of parameters
    - Considers instances

Why *SMAC*?

SMAC? [Hutter et al, LION 2011]

- Automatic algorithm configuration tool
- Combining cutting-edge optimization and machine learning methods

Advantages by using *SMAC*

1. Real algorithm configuration (opposed to “parameter tuning”)
    - Handles different types of parameters
    - Considers instances
2. Efficient in the number of function evaluations

Why *SMAC*?

SMAC? [Hutter et al, LION 2011]

- Automatic algorithm configuration tool
- Combining cutting-edge optimization and machine learning methods

Advantages by using *SMAC*

1. Real algorithm configuration (opposed to "parameter tuning")
   - Handles different types of parameters
   - Considers instances
2. Efficient in the number of function evaluations
3. Very efficient for runtime optimization

Outline

1. The Algorithm Configuration Problem
   - Problem Statement
   - Overview of Methods
2. Using AC Systems
3. Importance of Parameters
4. Advanced Topics

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 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

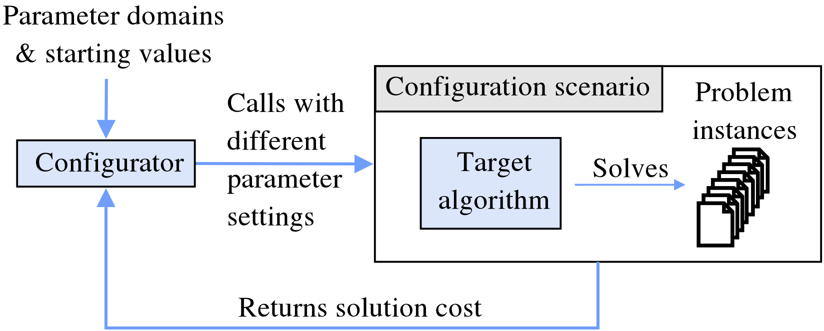
- E.g., parameter θ_2 of heuristic H is only active if H is used ($\theta_1 = H$)
- In this case, we say θ_2 is a conditional parameter with parent θ_1
- Sometimes, some combinations of parameter settings are forbidden
  e.g., the combination of $\theta_3 = 1$ and $\theta_4 = 2$ is forbidden

Parameters give rise to a structured space of configurations

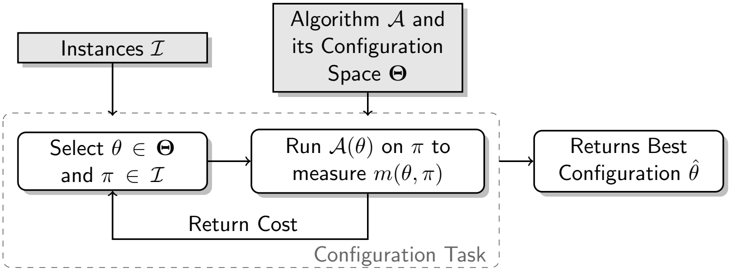
- Many configurations (e.g., SAT solver *lingeling* with 10^{947})
- Configurations often yield qualitatively different behaviour

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

Algorithm Configuration Visualized



Algorithm Configuration – in More Detail

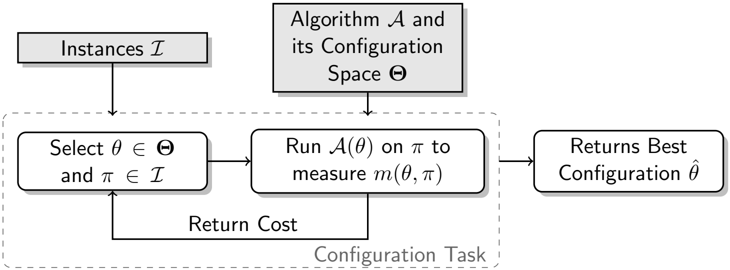


Definition: algorithm configuration

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

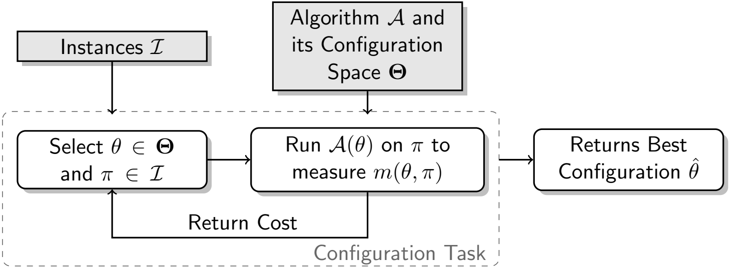


Definition: algorithm configuration

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 $m : \boldsymbol{\Theta} \times \mathcal{I} \rightarrow \mathbb{R}$ (wrt a runtime cutoff).

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 $m : \mathbf{\Theta} \times \mathcal{I} \rightarrow \mathbb{R}$ (wrt a runtime cutoff).

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

Challenges of Algorithm Configuration

Expensive Algorithm Runs

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

Structured high-dimensional parameter space

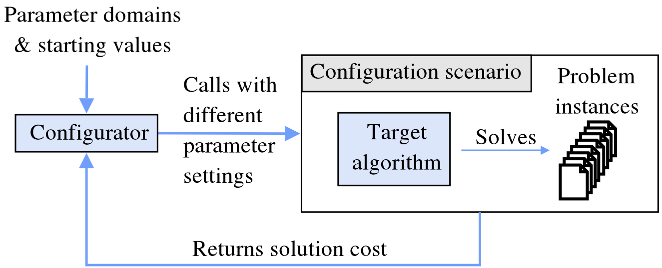
- Categorical vs. continuous parameters
- Conditionals between parameters

Stochastic optimization

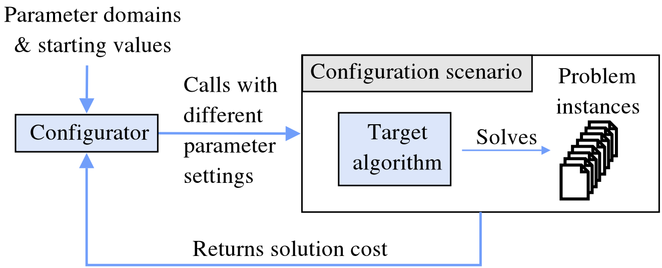
- Randomized algorithms: optimization across various seeds
- Distribution of benchmark instances (often wide range of hardness)
- Subsumes so-called *multi-armed bandit problem*

Outline

Algorithm Configuration: Components



Algorithm Configuration: Components



Required Components of an Algorithm Configurator

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: $\min_{\theta \in \mathbf{\Theta}} f(\theta)$

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

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

- Stochastic local search (SLS)
- Population-based methods
- *SMAC* $\rightarrow$ Model-based Optimization (e.g. Bayesian Optimization)
- ...

Bayesian Optimization – Detour into Machine Learning

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]

Bayesian Optimization – Detour into Machine Learning

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

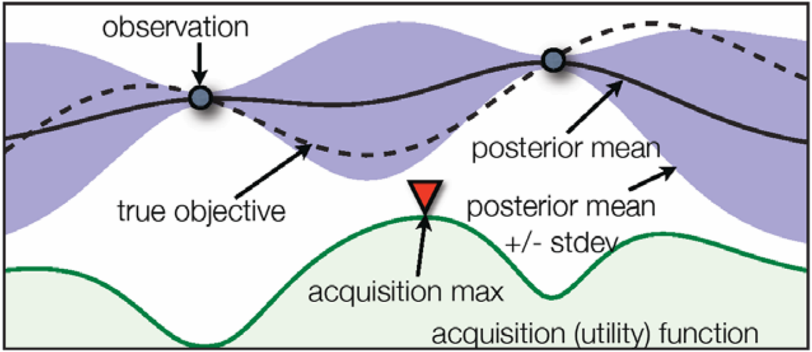
- Efficient in # function evaluations
- Works when objective is nonconvex, noisy, has unknown derivatives, etc
- Recent convergence results
  [Srinivas et al, 2010; Bull 2011; de Freitas et al, 2012; Kawaguchi et al, 2015]

General approach

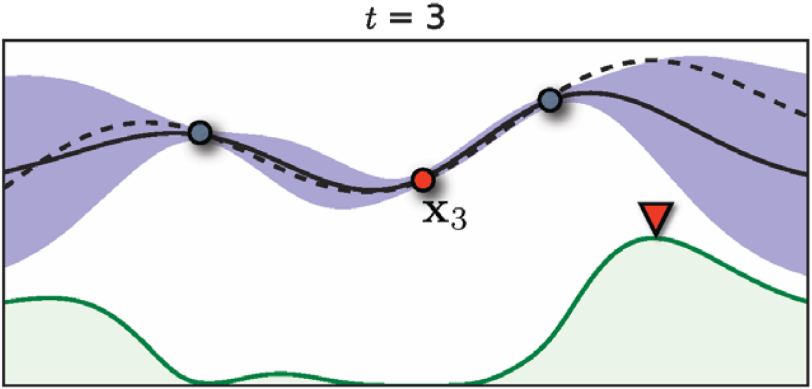
- Fit a probabilistic model to the collected function samples $\langle \theta, f(\theta) \rangle$
- Use the model to guide optimization, trading off exploration *vs* exploitation

Bayesian Optimization – Detour into Machine Learning

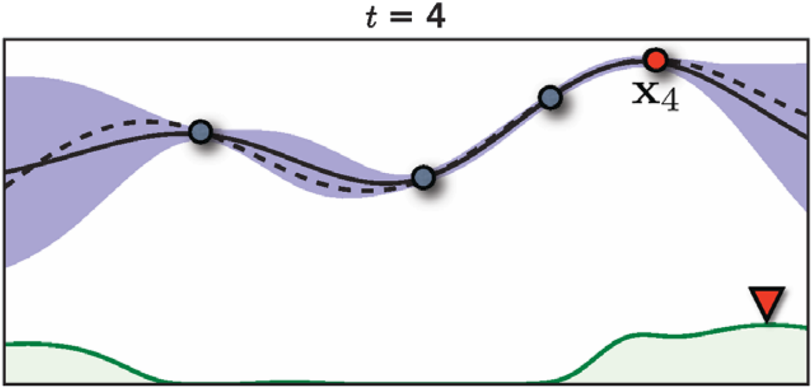
$t = 2$



Bayesian Optimization – Detour into Machine Learning



Bayesian Optimization – Detour into Machine Learning



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 (regression model)

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 (regression model)

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.

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

Also used in algorithm selection.

Component 2: How to Evaluate a Configuration?

Back to the general algorithm configuration problem

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

Component 2: How to Evaluate a Configuration?

Back to the general algorithm configuration problem

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

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 $m : \mathbf{\Theta} \times \mathcal{I} \rightarrow \mathbb{R}$
- $c(\theta) = \mathbb{E}_{\pi \sim \mathcal{D}}(m(\theta, \pi))$
- Algorithm runs are expensive!

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

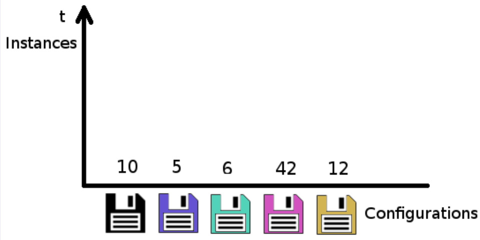
Toy Example

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

Toy Example

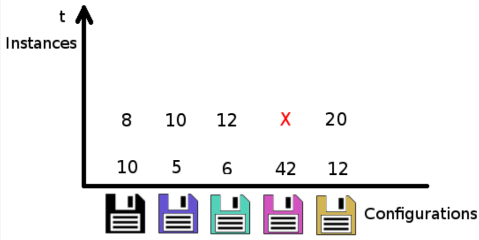


Racing algorithms: the general approach

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

Toy Example

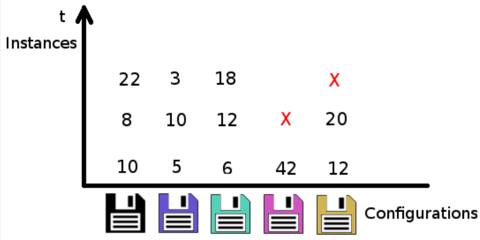


Racing algorithms: the general approach

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

Toy Example

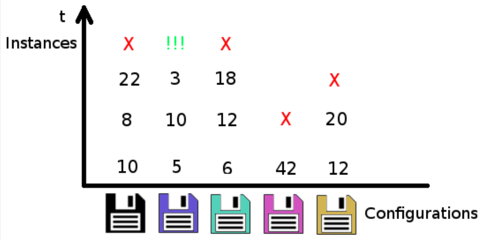


Racing algorithms: the general approach

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

Toy Example



Saving Time: Aggressive Racing

Idea

Only importance: Is a challenger better than the best known

Saving Time: Aggressive Racing

Idea

Only importance: Is a challenger better than the best known

Race new configurations against the best known

- Discard poor new configurations quickly
  - If challenger is worse on the first instance, discard it.

Saving Time: Aggressive Racing

Idea

Only importance: Is a challenger better than the best known

Race new configurations against the best known

- Discard poor new configurations quickly
  - If challenger is worse on the first instance, discard it.
- Evaluate best configurations with many runs

Saving Time: Aggressive Racing

Idea

Only importance: Is a challenger better than the best known

Race new configurations against the best known

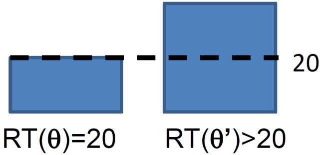
- Discard poor new configurations quickly
  - If challenger is worse on the first instance, discard it.
- Evaluate best configurations with many runs
- No requirement for statistical domination

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:

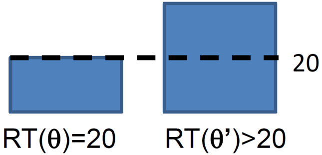
- Is θ' better than θ?
  - Example:



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

SMAC: Plug in everything together

Algorithm 2: SMAC

Initialize with a single run for the default

SMAC: Plug in everything together

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: Plug in everything together

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

SMAC: Plug in everything together

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

SMAC: Plug in everything together

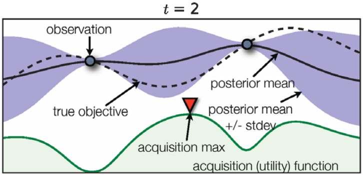
Algorithm 2: SMAC

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

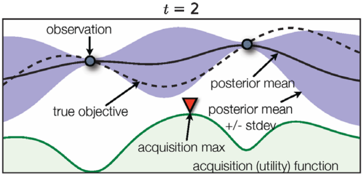
Initialize with a single run for the default

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

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



SMAC: Plug in everything together

Algorithm 2: SMAC

Initialize with a single run for the default

repeat

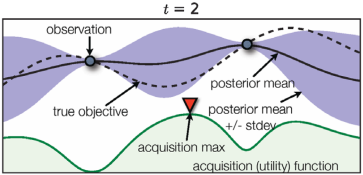
  Learn a RF model from data so far: $\hat{m} : \mathbf{\Theta} \times \mathcal{I} \rightarrow \mathbb{R}$

  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*



Overview on algorithm configuration systems

- Sequential Model-based Algorithm Configuration (SMAC)
  [Hutter et al, since 2011]
  - + State-of-the-art for runtime optimization

Overview on algorithm configuration systems

- Sequential Model-based Algorithm Configuration (SMAC) [Hutter et al, since 2011]
  + State-of-the-art for runtime optimization
- ParamILS [Hutter et al, 2007 & 2009]
  - Biased local search in configuration space
  + Easy parallelizable

Overview on algorithm configuration systems

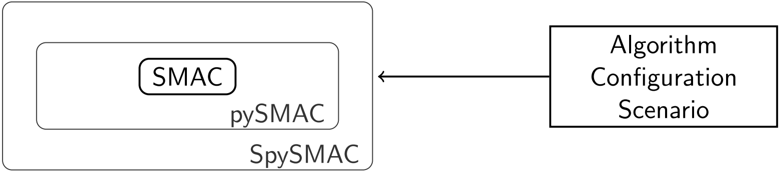
- Sequential Model-based Algorithm Configuration (SMAC)
  [Hutter et al, since 2011]
  + State-of-the-art for runtime optimization
- ParamILS [Hutter et al, 2007 & 2009]
  - Biased local search in configuration space
  + Easy parallelizable
- Gender-based Genetic Algorithm (GGA) [Ansotegui et al, 2009 & 2015]
  - Parameters are genes and populations consists of parameter settings
  + already parallelized
  - Requires adaption of GGA settings

Overview on algorithm configuration systems

- Sequential Model-based Algorithm Configuration (SMAC) [Hutter et al, since 2011]
  + State-of-the-art for runtime optimization
- ParamILS [Hutter et al, 2007 & 2009]
  - Biased local search in configuration space
  + Easy parallelizable
- Gender-based Genetic Algorithm (GGA) [Ansotegui et al, 2009 & 2015]
  - Parameters are genes and populations consists of parameter settings
  + already parallelized
  - Requires adaption of GGA settings
- Iterated F-Race [López-Ibáñez et al, 2011]
  - Sampling of configurations and racing them
  + Well maintained package in R
  - Not suited for runtime optimization

Outline

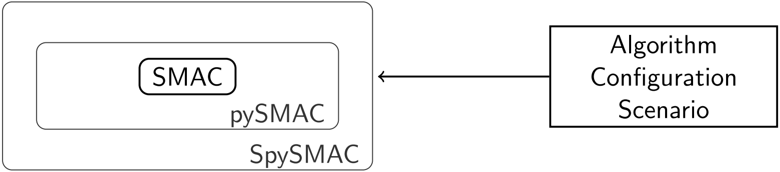
SMAC, pySMAC, SpySMAC



SMAC Configurator implemented in JAVA

- Flexible but harder to setup

SMAC, pySMAC, SpySMAC



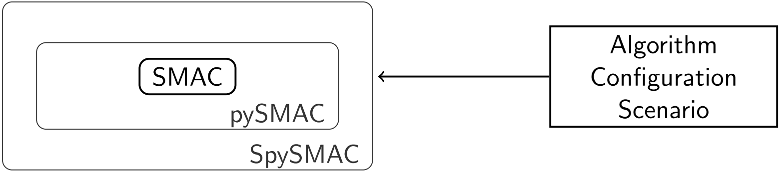
SMAC Configurator implemented in JAVA

- Flexible but harder to setup

pySMAC Python Interface to *SMAC*

- Very easy if you want to optimize a Python function

SMAC, pySMAC, SpySMAC



SMAC Configurator implemented in JAVA

- Flexible but harder to setup

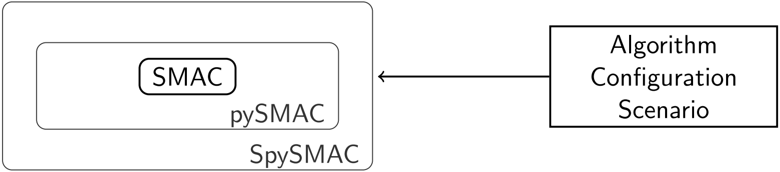
pySMAC Python Interface to *SMAC*

- Very easy if you want to optimize a Python function

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

- Less flexibility, but adapted to SAT solving

SMAC, pySMAC, SpySMAC



SMAC Configurator implemented in JAVA

- Flexible but harder to setup

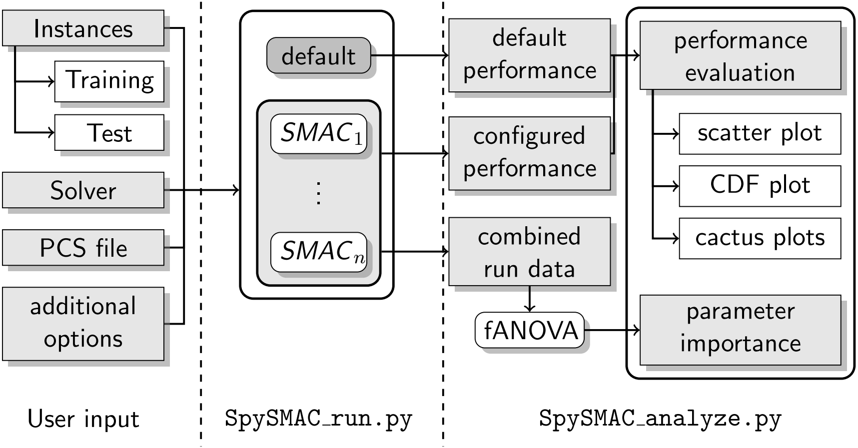
pySMAC Python Interface to *SMAC*

- Very easy if you want to optimize a Python function

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

- Less flexibility, but adapted to SAT solving

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

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
- *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)
```

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)

Different Types of Parameters

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

Specifying Parameter Configuration Spaces (PCS)

Different Types of Parameters

- 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: 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: 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

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

Ablation [Fawcett et al. 2013]

Idea

- Comparison of two parameter configurations
- Starting from the default, we change the value of the parameters
- Which of these changes were important?

→ Ablation compares parameter flips between default and optimized configuration

Ablation [Fawcett et al. 2013]

Idea

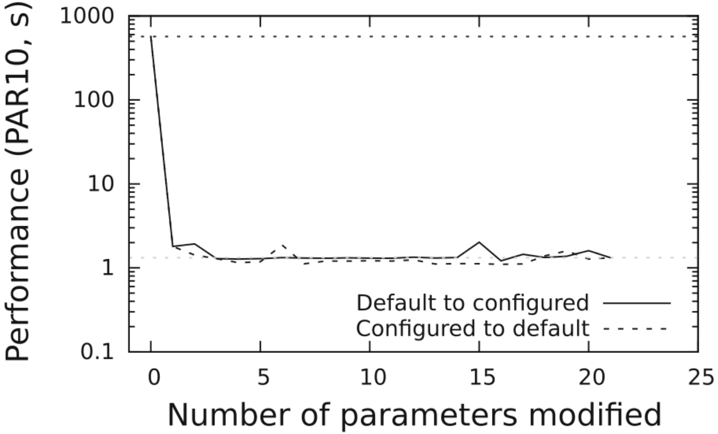
- Comparison of two parameter configurations
- Starting from the default, we change the value of the parameters
- Which of these changes were important?

→ Ablation compares parameter flips between default and optimized 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

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

fANOVA [Sobol 1993]

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

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

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:

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

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

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:

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

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

Application to Parameter Importance

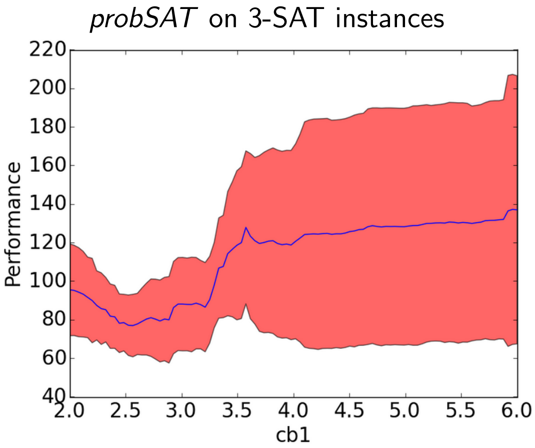
How much of the variance can be explained by a parameter (or combinations of parameters) marginalized over all other parameters?

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

Limitations of Algorithm Configurations

- (Current) Algorithm configurators can only be effectively applied to homogeneous instance sets
  - there exists a configuration that performs well on all instances

Limitations of Algorithm Configurations

- (Current) Algorithm configurators can only be effectively applied to homogeneous instance sets
  - there exists a configuration that performs well on all instances
- Robustness on heterogeneous instance sets is also important
- How to deal with heterogeneous instance sets?
  - Parallel portfolios, i.e., running a set of complementary algorithms/configurations in parallel
  - Algorithm Selection

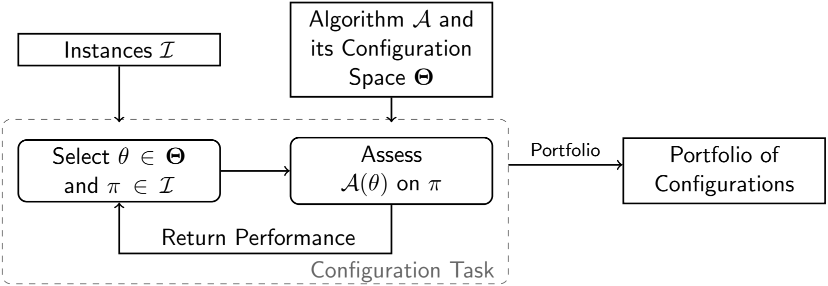
ACPP

Given sequential algorithm $\mathcal{A}$ and configuration space $\mathbf{\Theta}$, automatically find a complementary portfolio of configurations running in parallel.

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

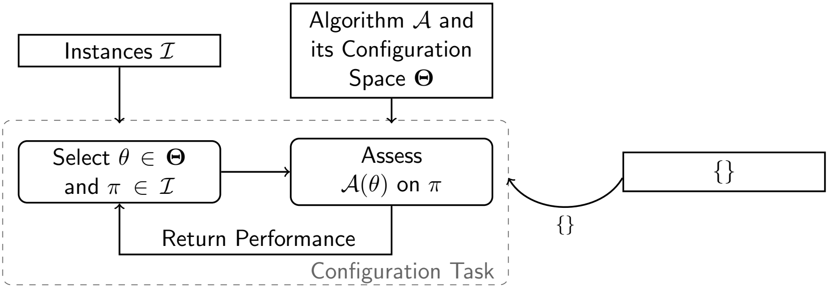
ACPP

Given sequential algorithm $\mathcal{A}$ and configuration space $\mathbf{\Theta}$, automatically find a complementary portfolio of configurations running in parallel.



ACPP

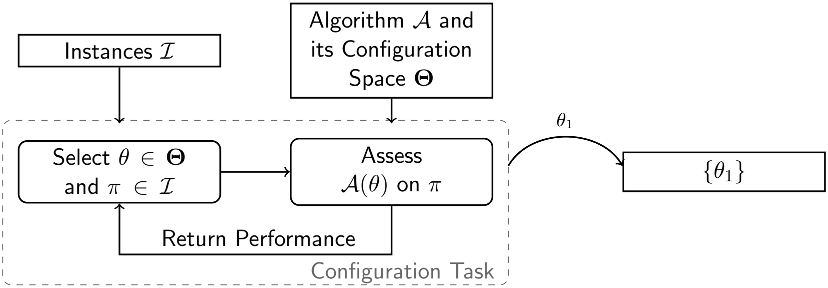
Given sequential algorithm $\mathcal{A}$ and configuration space $\boldsymbol{\Theta}$, automatically find a complementary portfolio of configurations running in parallel.



parHydra method (inspired by *Hydra* [Xu et al, AAAI 2010])

ACPP

Given sequential algorithm $\mathcal{A}$ and configuration space $\boldsymbol{\Theta}$, automatically find a complementary portfolio of configurations running in parallel.

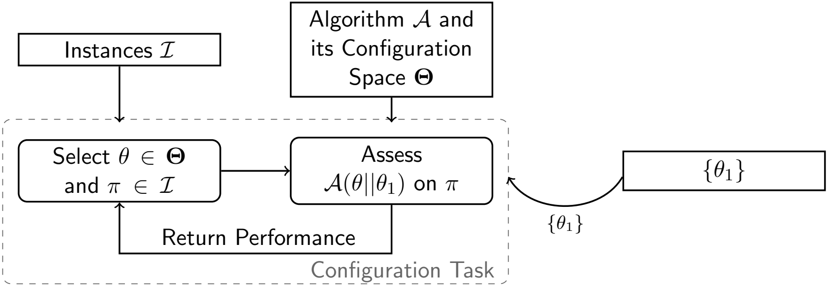


`parHydra` method (inspired by *Hydra* [Xu et al, AAAI 2010])

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

ACPP

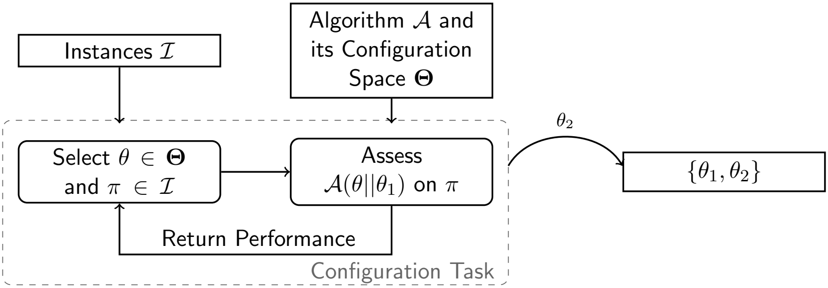
Given sequential algorithm $\mathcal{A}$ and configuration space $\boldsymbol{\Theta}$, automatically find a complementary portfolio of configurations running in parallel.



parHydra method (inspired by *Hydra* [Xu et al, AAAI 2010])

ACPP

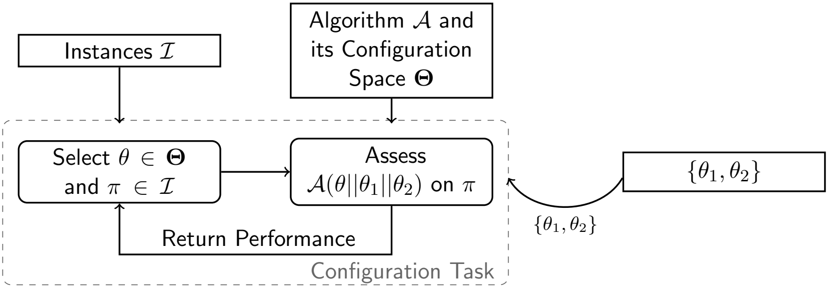
Given sequential algorithm $\mathcal{A}$ and configuration space $\boldsymbol{\Theta}$, automatically find a complementary portfolio of configurations running in parallel.



`parHydra` method (inspired by *Hydra* [Xu et al, AAAI 2010])

ACPP

Given sequential algorithm $\mathcal{A}$ and configuration space $\mathbf{\Theta}$, automatically find a complementary portfolio of configurations running in parallel.



`parHydra` method (inspired by *Hydra* [Xu et al, AAAI 2010])

Results on SAT Challenge 2012 [Lindauer et al, AIJ 2016]

	lingeling on industrial		*clasp* on hard combinatorial	
	#TOs	PAR1	#TOs	PAR1
Sequential Solvers				
Default	72	373	137	481
Configured	68	368	140	473
Parallel Solvers with 8 cores				
Default	64	345	96	358

Results on SAT Challenge 2012 [Lindauer et al, AIJ 2016]

	lingeling on industrial		*clasp* on hard combinatorial	
	#TOs	PAR1	#TOs	PAR1
Sequential Solvers				
Default	72	373	137	481
Configured	68	368	140	473
Parallel Solvers with 8 cores				
Default	64	345	96	358
parHydra	55	303	96	353

Adding Clause Sharing (CS) [Lindauer et al, AIJ 2016]

Solver set	*clasp* (hard combinatorial) #TOs	PAR1
Parallel Solvers with 8 cores		
Default	96	358
Default+CS	90	333
parHydra	96	353

Adding Clause Sharing (CS) [Lindauer et al, AIJ 2016]

Solver set	*clasp* (hard combinatorial) #TOs	PAR1
Parallel Solvers with 8 cores		
Default	96	358
Default+CS	90	333
parHydra	96	353
parHydra+default CS	90	347

Adding Clause Sharing (CS) [Lindauer et al, AIJ 2016]

Solver set	*clasp* (hard combinatorial) #TOs	PAR1
Parallel Solvers with 8 cores		
Default	96	358
Default+CS	90	333
parHydra	96	353
parHydra+default CS	90	347
parHydra+tuned CS	88	346

Per-Instance Algorithm Configuration

Complementary Configurations via Algorithm Configuration

Hydra [Xu et al, AAAI 2010] iteratively build a complementary portfolio

ISAC [Kadioglu et al, ECAI 2010] use unsupervised clustering to get instance clusters and determine a configuration for each of them

Per-Instance Algorithm Configuration

Complementary Configurations via Algorithm Configuration

Hydra [Xu et al, AAAI 2010] iteratively build a complementary portfolio

ISAC [Kadioglu et al, ECAI 2010] use unsupervised clustering to get instance clusters and determine a configuration for each of them

Algorithm Selection [Rice, 1976]

Using machine learning, predict the best performing algorithm (or parameter configuration) for a given instance.

Per-Instance Algorithm Configuration

Complementary Configurations via Algorithm Configuration

Hydra [Xu et al, AAAI 2010] iteratively build a complementary portfolio

ISAC [Kadioglu et al, ECAI 2010] use unsupervised clustering to get instance clusters and determine a configuration for each of them

Algorithm Selection [Rice, 1976]

Using machine learning, predict the best performing algorithm (or parameter configuration) for a given instance.

$\rightsquigarrow$ *Riss BlackBox* [Alfonso & Manthey, 2014] won 3 medals with such an approach in the SAT Competition 2014

Robust Benchmark Sets [Hoos et al., LION 2013]

Heterogenous vs. Homogeneous Instances

- Homogeneous: there is one configuration performing well on all
  - Similar properties, e.g., from the same encoding
- Heterogeneous: no configuration performing well on all
  - perfect for algorithm selection
  - hard to configure on such instances

Robust Benchmark Sets [Hoos et al., LION 2013]

Heterogenous vs. Homogeneous Instances

- Homogeneous: there is one configuration performing well on all
  - Similar properties, e.g., from the same encoding
- Heterogeneous: no configuration performing well on all
  - perfect for algorithm selection
  - hard to configure on such instances

Heterogeneous Benchmark Sets for Robust Configuration

1. Large variety of instances
   - Different properties should be uniformly represented

Robust Benchmark Sets [Hoos et al., LION 2013]

Heterogenous vs. Homogeneous Instances

- Homogeneous: there is one configuration performing well on all
  - Similar properties, e.g., from the same encoding
- Heterogeneous: no configuration performing well on all
  - perfect for algorithm selection
  - hard to configure on such instances

Heterogeneous Benchmark Sets for Robust Configuration

1. Large variety of instances
   - Different properties should be uniformly represented
2. Adapted instance hardness
   - Too hard: will not solve many instances, no traction
   - Too easy: will results generalize to harder instances?
   - Rule of thumb: mix of hardness ranges

Robust Benchmark Sets [Hoos et al., LION 2013]

Heterogenous vs. Homogeneous Instances

- Homogeneous: there is one configuration performing well on all
  - Similar properties, e.g., from the same encoding
- Heterogeneous: no configuration performing well on all
  - perfect for algorithm selection
  - hard to configure on such instances

Heterogeneous Benchmark Sets for Robust Configuration

1. Large variety of instances
   - Different properties should be uniformly represented
2. Adapted instance hardness
   - Too hard: will not solve many instances, no traction
   - Too easy: will results generalize to harder instances?
   - Rule of thumb: mix of hardness ranges
3. Free of duplicates

Further Tips and Tricks

Further Tools & Material

- see `www.ml4aad.org`
- "Teaching" $\rightsquigarrow$ Even more extensive slides from AAAI'16 tutorial

Further Tips and Tricks

Further Tools & Material

- see `www.ml4aad.org`
- "Teaching" $\rightsquigarrow$ Even more extensive slides from AAAI'16 tutorial

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 & Material

- see www.ml4aad.org
- "Teaching" $\rightsquigarrow$ Even more extensive slides from AAAI'16 tutorial

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

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
[Auction]omics

IBM
ILOG
Mixed integer programming

Social gaming
zynga

Scheduling and Resource Allocation
actenum

Thank you!

ML4AAD