Algorithm Configuration: A Hands-on Tutorial

Frank Hutter   Marius Lindauer

University of Freiburg

AAAI 2016, Phoenix, USA

What is this tutorial about?

- Algorithm configuration in a nutshell: optimization of free parameters
- What kinds of parameters?
  - $\rightsquigarrow$ any that you would otherwise tune yourself (&more)

What is this tutorial about?

- Algorithm configuration in a nutshell: optimization of free parameters
- What kinds of parameters?
  - ⇝ any that you would otherwise tune yourself (&more)

Examples of free parameters in various subfields of AI

- Tree search (in particular for SAT): pre-processing, branching heuristics, clause learning & deletion, restarts, data structures, ...

What is this tutorial about?

- Algorithm configuration in a nutshell: optimization of free parameters
- What kinds of parameters?
  - ⇝ any that you would otherwise tune yourself (&more)

Examples of free parameters in various subfields of AI

- Tree search (in particular for SAT): pre-processing, branching heuristics, clause learning & deletion, restarts, data structures, ...
- Local search: neighbourhoods, perturbations, tabu length, annealing...

What is this tutorial about?

- Algorithm configuration in a nutshell: optimization of free parameters
- What kinds of parameters?
  - $\rightsquigarrow$ any that you would otherwise tune yourself (&more)

Examples of free parameters in various subfields of AI

- Tree search (in particular for SAT): pre-processing, branching heuristics, clause learning & deletion, restarts, data structures, ...
- Local search: neighbourhoods, perturbations, tabu length, annealing...
- Genetic algorithms: population size, mating scheme, crossover operators, mutation rate, local improvement stages, hybridizations, ...

What is this tutorial about?

- Algorithm configuration in a nutshell: optimization of free parameters
- What kinds of parameters?
  - ⇝ any that you would otherwise tune yourself (&more)

Examples of free parameters in various subfields of AI

- Tree search (in particular for SAT): pre-processing, branching heuristics, clause learning & deletion, restarts, data structures, ...
- Local search: neighbourhoods, perturbations, tabu length, annealing...
- Genetic algorithms: population size, mating scheme, crossover operators, mutation rate, local improvement stages, hybridizations, ...
- Machine Learning: pre-processing, regularization (type & strength), minibatch size, learning rate schedules, optimizer & its parameters, ...

What is this tutorial about?

- Algorithm configuration in a nutshell: optimization of free parameters
- What kinds of parameters?
  - ⇝ any that you would otherwise tune yourself (&more)

Examples of free parameters in various subfields of AI

- Tree search (in particular for SAT): pre-processing, branching heuristics, clause learning & deletion, restarts, data structures, ...
- Local search: neighbourhoods, perturbations, tabu length, annealing...
- Genetic algorithms: population size, mating scheme, crossover operators, mutation rate, local improvement stages, hybridizations, ...
- Machine Learning: pre-processing, regularization (type & strength), minibatch size, learning rate schedules, optimizer & its parameters, ...
- Deep learning (in addition): #layers (& layer types), #units/layer, dropout constants, weight initialization and decay, pre-training, ...

What is this tutorial about?

- Algorithm configuration in a nutshell: optimization of free parameters
- What kinds of parameters?
  - $\rightsquigarrow$ any that you would otherwise tune yourself (&more)

Examples of free parameters in various subfields of AI

- Tree search (in particular for SAT): pre-processing, branching heuristics, clause learning & deletion, restarts, data structures, ...
- Local search: neighbourhoods, perturbations, tabu length, annealing...
- Genetic algorithms: population size, mating scheme, crossover operators, mutation rate, local improvement stages, hybridizations, ...
- Machine Learning: pre-processing, regularization (type & strength), minibatch size, learning rate schedules, optimizer & its parameters, ...
- Deep learning (in addition): #layers (& layer types), #units/layer, dropout constants, weight initialization and decay, pre-training, ...
- and many more ...

Hands-on!

Focus on basics

- Less material, more in-depth
- Target audience: focus on beginners
- No special background assumed
- Please ask questions
- All literature references are hyperlinks

Goal: you can use algorithm configuration in your research

- All demos use the virtual machine (VM) we distributed
- If you downloaded the VM you can follow along live!

Hands-on!

Focus on basics

- Less material, more in-depth
- Target audience: focus on beginners
- No special background assumed
- Please ask questions
- All literature references are hyperlinks

Goal: you can use algorithm configuration in your research

- All demos use the virtual machine (VM) we distributed
- If you downloaded the VM you can follow along live!
- How many of you downloaded the VM?

Hands-on!

Focus on basics

- Less material, more in-depth
- Target audience: focus on beginners
- No special background assumed
- Please ask questions
- All literature references are hyperlinks

Goal: you can use algorithm configuration in your research

- All demos use the virtual machine (VM) we distributed
- If you downloaded the VM you can follow along live!
- How many of you downloaded the VM?
  - Don't try now: it's 3GB and you can follow what we do on the screen

Outline

Outline

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

Outline

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 $\theta_4 = 1$ is forbidden

Algorithm Parameters

Parameter Types

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

Parameter space has structure

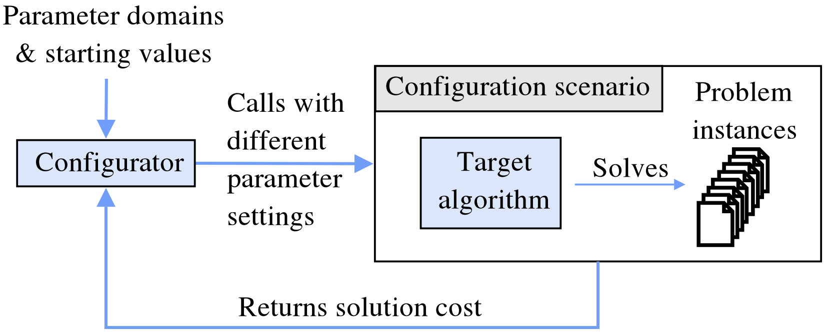
- E.g., parameter θ_2 of heuristic H is only active if H is used $(\theta_1 = H)$
- In this case, we say θ_2 is a conditional parameter with parent θ_1
- Sometimes, some combinations of parameter settings are forbidden e.g., the combination of $\theta_3 = 1$ and $\theta_4 = 1$ is forbidden

Parameters give rise to a structured space of configurations

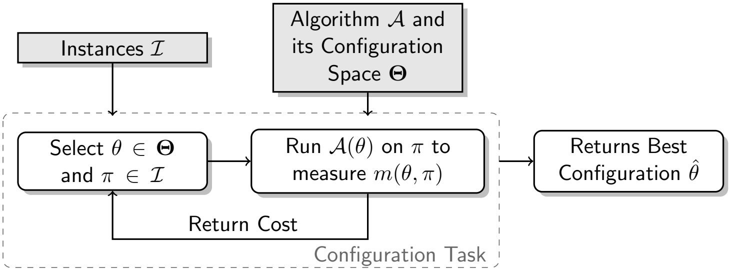
- Many configurations (e.g., SAT solver *lingeling* with 10^{947})
- Configurations often yield qualitatively different behaviour

→ Algorithm Configuration (as opposed to “parameter tuning”)

Algorithm Configuration Visualized



Algorithm Configuration – in More Detail

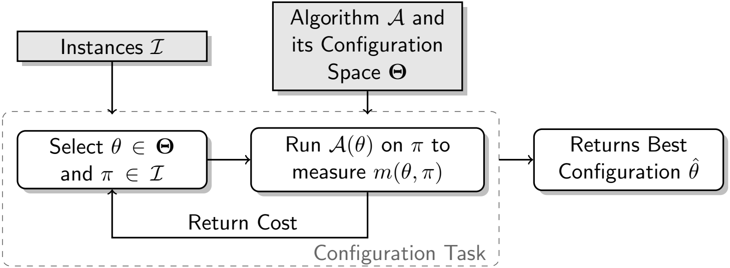


Definition: algorithm configuration

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

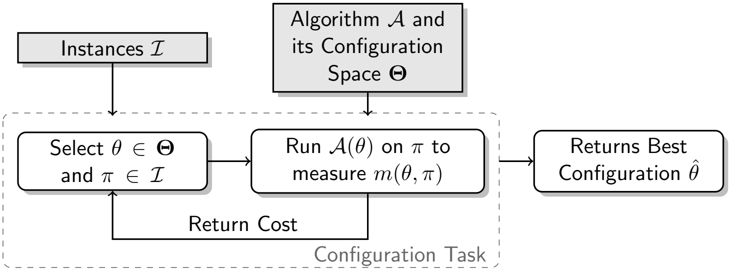


Definition: algorithm configuration

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 $m : \boldsymbol{\Theta} \times \mathcal{I} \rightarrow \mathbb{R}$,

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 $m : \mathbf{\Theta} \times \mathcal{I} \to \mathbb{R}$,

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

Algorithm Configuration – Full Formal Definition

Definition: algorithm configuration

An instance of the algorithm configuration problem is a 5-tuple $(\mathcal{A}, \mathbf{\Theta}, \mathcal{D}, \kappa, m)$ where:

- $\mathcal{A}$ is a parameterized algorithm;
- $\mathbf{\Theta}$ is the parameter configuration space of $\mathcal{A}$;
- $\mathcal{D}$ is a distribution over problem instances with domain $\mathcal{I}$;

Algorithm Configuration – Full Formal Definition

Definition: algorithm configuration

An instance of the algorithm configuration problem is a 5-tuple $(\mathcal{A}, \mathbf{\Theta}, \mathcal{D}, \kappa, m)$ where:

- $\mathcal{A}$ is a parameterized algorithm;
- $\mathbf{\Theta}$ is the parameter configuration space of $\mathcal{A}$;
- $\mathcal{D}$ is a distribution over problem instances with domain $\mathcal{I}$;
- $\kappa < \infty$ is a cutoff time, after which each run of $\mathcal{A}$ will be terminated if still running

Algorithm Configuration – Full Formal Definition

Definition: algorithm configuration

An instance of the algorithm configuration problem is a 5-tuple $(\mathcal{A}, \mathbf{\Theta}, \mathcal{D}, \kappa, m)$ where:

- $\mathcal{A}$ is a parameterized algorithm;
- $\mathbf{\Theta}$ is the parameter configuration space of $\mathcal{A}$;
- $\mathcal{D}$ is a distribution over problem instances with domain $\mathcal{I}$;
- $\kappa < \infty$ is a cutoff time, after which each run of $\mathcal{A}$ will be terminated if still running
- $m : \mathbf{\Theta} \times \mathcal{I} \rightarrow \mathbb{R}$ is a function that measures the observed cost of running $\mathcal{A}(\theta)$ on an instance $\pi \in \mathcal{I}$ with cutoff time κ

Algorithm Configuration – Full Formal Definition

Definition: algorithm configuration

An instance of the algorithm configuration problem is a 5-tuple $(\mathcal{A}, \mathbf{\Theta}, \mathcal{D}, \kappa, m)$ where:

- $\mathcal{A}$ is a parameterized algorithm;
- $\mathbf{\Theta}$ is the parameter configuration space of $\mathcal{A}$;
- $\mathcal{D}$ is a distribution over problem instances with domain $\mathcal{I}$;
- $\kappa < \infty$ is a cutoff time, after which each run of $\mathcal{A}$ will be terminated if still running
- $m : \mathbf{\Theta} \times \mathcal{I} \to \mathbb{R}$ is a function that measures the observed cost of running $\mathcal{A}(\theta)$ on an instance $\pi \in \mathcal{I}$ with cutoff time κ

The cost of a candidate solution $\theta \in \mathbf{\Theta}$ is $c(\theta) = \mathbb{E}_{\pi \sim \mathcal{D}}(m(\theta, \pi))$.
The goal is to find $\theta^* \in \arg\min_{\theta \in \mathbf{\Theta}} c(\theta)$.

Distribution vs. Set of Instances

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

Special case: distribution with finite support

- We often only have N instances from a given application
- In that case: split N instances into training and test set
- Find $\theta^* \in \arg\min_{\theta \in \boldsymbol{\Theta}} \frac{1}{N_{train}} \sum_{i=1}^{N_{train}} (m(\theta, \pi_i))$.

Distribution vs. Set of Instances

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

Special case: distribution with finite support

- We often only have N instances from a given application
- In that case: split N instances into training and test set
- Find $\theta^* \in \arg\min_{\theta \in \mathbf{\Theta}} \frac{1}{N_{train}} \sum_{i=1}^{N_{train}} (m(\theta, \pi_i))$.

Evaluation on new test instances

Same approach as in machine learning

- We configure algorithms on the training instances
- We only use test instances to assess generalization performance
  - $\rightarrow$ unbiased estimate of generalization performance for unseen instances

Our Two Running Examples

Minimize the runtime of a SAT solver for a benchmark set

- Optimize on training set:
  $\theta^* \in \arg\min_{\theta \in \mathbf{\Theta}} \frac{1}{N_{train}} \sum_{i=1}^{N_{train}} (m(\theta, \pi_i))$

Our Two Running Examples

Minimize the runtime of a SAT solver for a benchmark set

- Optimize on training set:
  $\theta^* \in \arg\min_{\theta \in \mathbf{\Theta}} \frac{1}{N_{train}} \sum_{i=1}^{N_{train}} (m(\theta, \pi_i))$

Minimize K-fold cross-validation error of a machine learning algorithm

- A cross-validation fold k plays the role of an instance
  $\theta^* \in \arg\min_{\theta \in \mathbf{\Theta}} \frac{1}{K} \sum_{k=1}^{K} (m(\theta, k))$

Some use cases for Algorithm Configuration

Automatically customize versatile algorithms

- New application domains
- Optimize speed, accuracy, memory, energy consumption, latency, ...

Some use cases for Algorithm Configuration

Automatically customize versatile algorithms

- New application domains
- Optimize speed, accuracy, memory, energy consumption, latency, ...
- All of these: possible without intimate knowledge of algorithm
- Trade costly human time for cheap CPU time

Some use cases for Algorithm Configuration

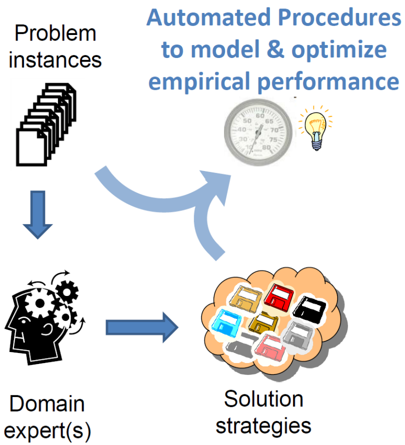
Automatically customize versatile algorithms

- New application domains
- Optimize speed, accuracy, memory, energy consumption, latency, ...
- All of these: possible without intimate knowledge of algorithm
- Trade costly human time for cheap CPU time

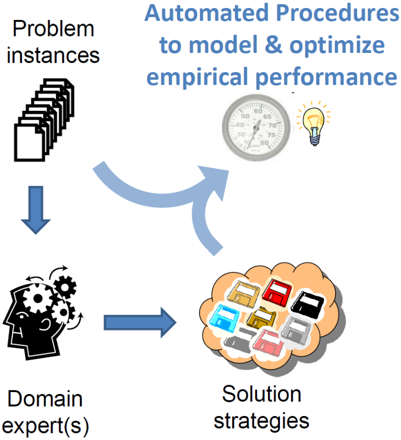
Empirical studies, evaluations & comparisons of algorithms

- Fairness: same tuning protocol for all algorithms
- Reproducibility of protocol

AC in Context: Computer-Aided Algorithm Design



AC in Context: Computer-Aided Algorithm Design



Algorithm configuration

Find best for

Performance prediction

Fit model f(,) ≈

Quantify importance of algorithm components

Algorithm portfolios

Select best for new

Outline

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

SAT (propositional satisfiability problem)

- Prototypical $\mathcal{NP}$-hard problem
- Interesting theoretically and in practical applications

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

SAT (propositional satisfiability problem)

- Prototypical $\mathcal{NP}$-hard problem
- Interesting theoretically and in practical applications

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]

SAT (propositional satisfiability problem)

- Prototypical $\mathcal{NP}$-hard problem
- Interesting theoretically and in practical applications

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]

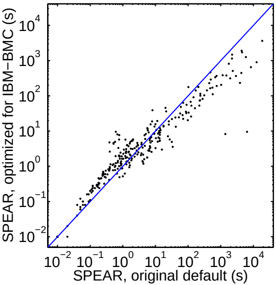
- Ran FocusedILS, 2 days $\times$ 10 machines
  - On a training set from each benchmark

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

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

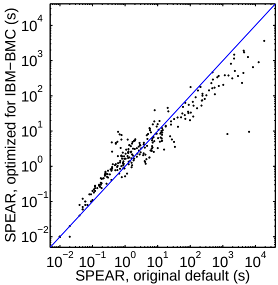


4.5-fold speedup

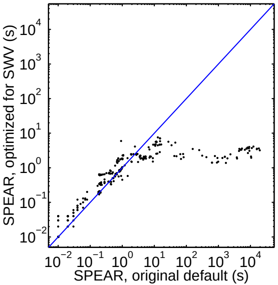
on hardware verification

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

- Ran FocusedILS, 2 days $\times$ 10 machines
  - On a training set from each benchmark
- Compared to manually-engineered default
  - 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

Mixed integer programming

Social gaming

Scheduling and Resource Allocation

Automated Machine Learning

Machine Learning is very successful in many applications.

- But it still requires human machine learning experts to
  - Preprocess the data
  - Select / engineer features
  - Select a model family
  - Optimize hyperparameters
  - Construct ensembles
  - ...

Automated Machine Learning

Machine Learning is very successful in many applications.

- But it still requires human machine learning experts to
  - Preprocess the data
  - Select / engineer features
  - Select a model family
  - Optimize hyperparameters
  - Construct ensembles
  - ...
- AutoML: taking the human expert out of the loop

Automated Machine Learning

Machine Learning is very successful in many applications.

- But it still requires human machine learning experts to
  - Preprocess the data
  - Select / engineer features
  - Select a model family
  - Optimize hyperparameters
  - Construct ensembles
  - ...
- AutoML: taking the human expert out of the loop
- Deep learning helps to automatically learn features

  But it is even more sensitive to hyperparameters

Automated Machine Learning

The AutoML approach introduced by Auto-WEKA [Thornton et al, 2013]

- Expose the choices in a machine learning framework
  - Algorithms, hyperparameters, preprocessors, ...
- Optimize C/V performance using Bayesian optimization

$\rightsquigarrow$ Obtain a true push-button solution for machine learning

Automated Machine Learning

The AutoML approach introduced by Auto-WEKA [Thornton et al, 2013]

- Expose the choices in a machine learning framework
  - Algorithms, hyperparameters, preprocessors, ...
- Optimize C/V performance using Bayesian optimization

$\rightsquigarrow$ Obtain a true push-button solution for machine learning

Extended recently in Auto-sklearn [Feurer et al, 2015]

Auto-sklearn's configuration space

name	#λ
AdaBoost (AB)	3
Bernoulli naïve Bayes	2
decision tree (DT)	3
extreml. rand. trees	5
Gaussian naïve Bayes	-
gradient boosting (GB)	6
kNN	3
LDA	2
linear SVM	5
kernel SVM	8
multinomial naïve Bayes	2
passive aggressive	3
QDA	2
random forest (RF)	5
ridge regression (RR)	2
SGD	9

name	#λ
extreml. rand. trees prepr.	5
fast ICA	4
feature agglomeration	3
kernel PCA	5
rand. kitchen sinks	2
linear SVM prepr.	5
no preprocessing	-
nystroem sampler	5
PCA	2
random trees embed.	4
select percentile	2
select rates	3
imputation	1
balancing	1
rescaling	1

Hands-on: Auto-sklearn

In your virtual machine:

Run a linear SVM and Auto-sklearn

```
$ cd AC-Tutorial/auto-sklearn/

$ vim baseline_svc.py
$ python baseline_svc.py

$ vim autosklearn-example.py
$ python autosklearn-example.py
```

Outline

Challenges of Algorithm Configuration

Structured high-dimensional parameter space

- Categorical vs. continuous parameters
- Conditionals between parameters

Challenges of Algorithm Configuration

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*

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

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

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

Component 1: Which Configuration to Evaluate?

- Need to balance diversification and intensification
- The extremes
  - Random search
  - Gradient Descent
- Stochastic local search (SLS)
- Population-based methods
- Model-based 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

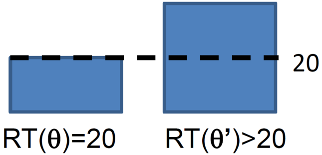
Saving Time: Aggressive Racing

- 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:

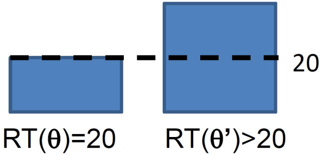
- Is θ' better than θ?
  - Example:



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")*

Algorithm 1: Manual Greedy Algorithm Configuration

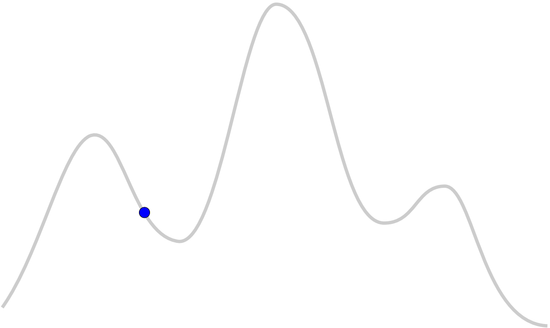
Start with some configuration θ
repeat
  Modify a single parameter
  if *results on benchmark set improve* **then**
    keep new configuration
until *no more improvement possible (or "good enough")*

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

Going Beyond Local Optima: Iterated Local Search

Animation credit: Holger Hoos

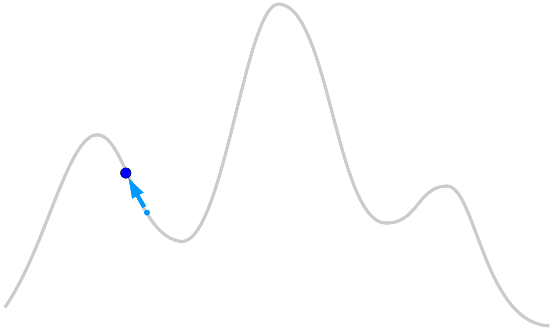
Going Beyond Local Optima: Iterated Local Search



Initialisation

Animation credit: Holger Hoos

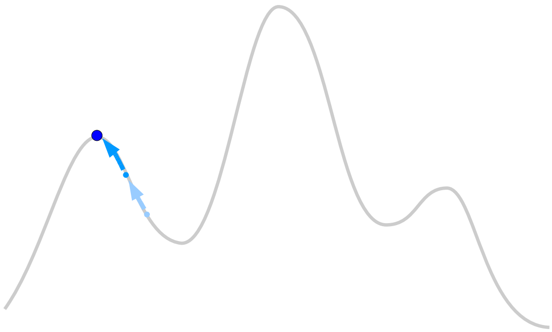
Going Beyond Local Optima: Iterated Local Search



Local Search

Animation credit: Holger Hoos

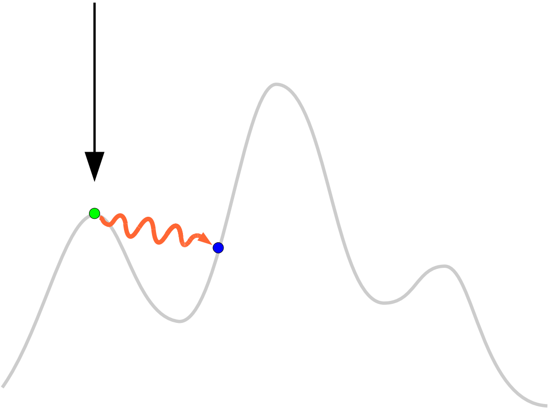
Going Beyond Local Optima: Iterated Local Search



Local Search

Animation credit: Holger Hoos

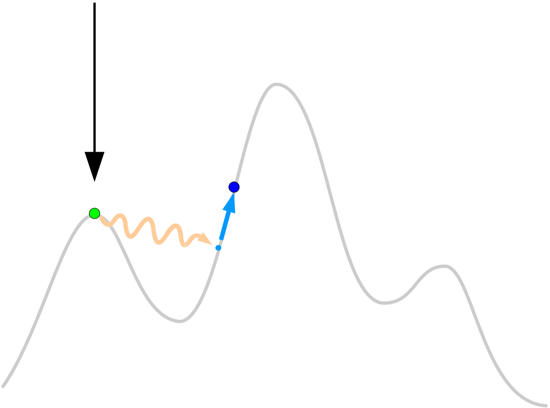
Going Beyond Local Optima: Iterated Local Search



Perturbation

Animation credit: Holger Hoos

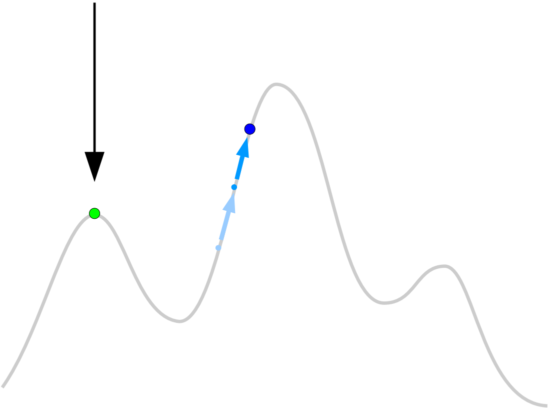
Going Beyond Local Optima: Iterated Local Search



Local Search

Animation credit: Holger Hoos

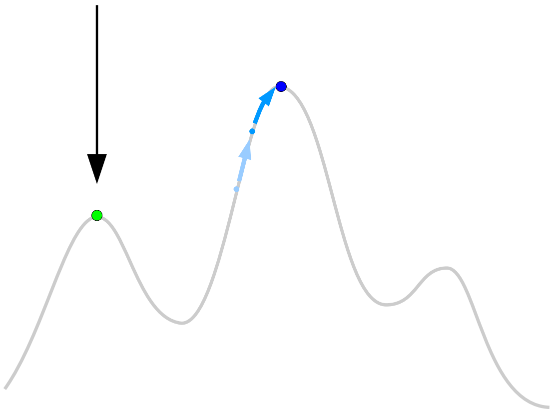
Going Beyond Local Optima: Iterated Local Search



Local Search

Animation credit: Holger Hoos

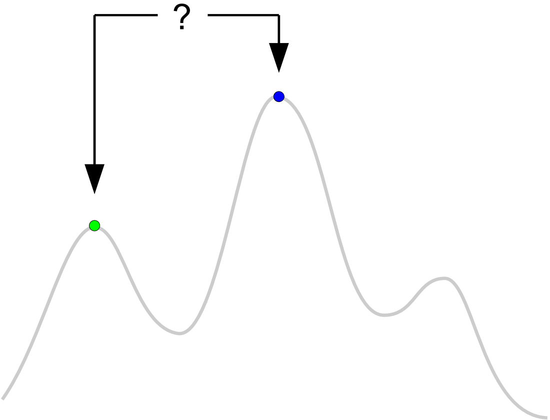
Going Beyond Local Optima: Iterated Local Search



Local Search

Animation credit: Holger Hoos

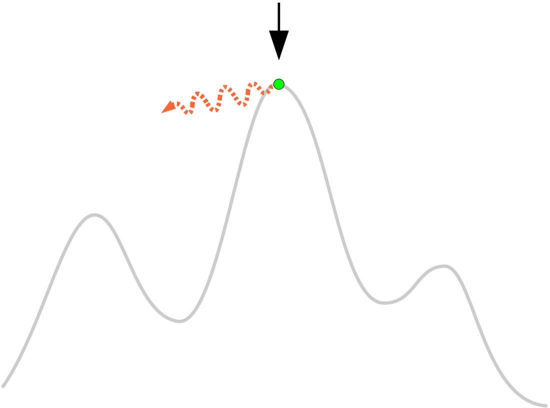
Going Beyond Local Optima: Iterated Local Search



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]

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

Theorem

Let $\boldsymbol{\Theta}$ be finite. Then, the probability that FocusedILS finds the true optimal parameter configuration $\theta^* \in \boldsymbol{\Theta}$ approaches 1 as the number of ILS iterations goes to infinity.

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

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
- 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

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

Theorem

Let $\boldsymbol{\Theta}$ be finite. Then, the probability that FocusedILS finds the true optimal parameter configuration $\theta^* \in \boldsymbol{\Theta}$ approaches 1 as the number of ILS iterations goes to infinity.

Bayesian Optimization

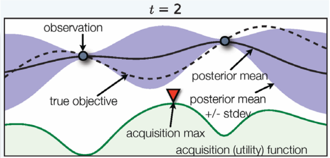
General approach

- 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

General approach

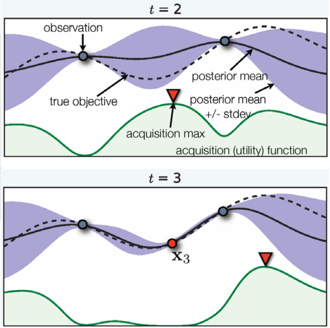
- Fit a probabilistic model to the collected function samples $\langle \theta, f(\theta) \rangle$
- Use the model to guide optimization, trading off exploration *vs* exploitation



Bayesian Optimization

General approach

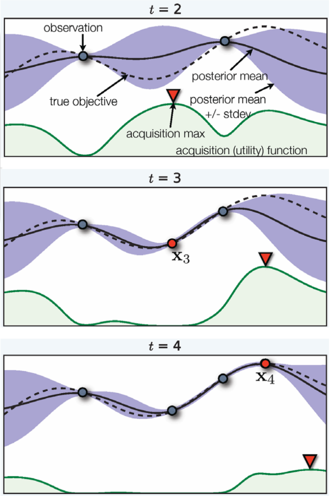
- Fit a probabilistic model to the collected function samples $\langle \theta, f(\theta) \rangle$
- Use the model to guide optimization, trading off exploration *vs* exploitation



Bayesian Optimization

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



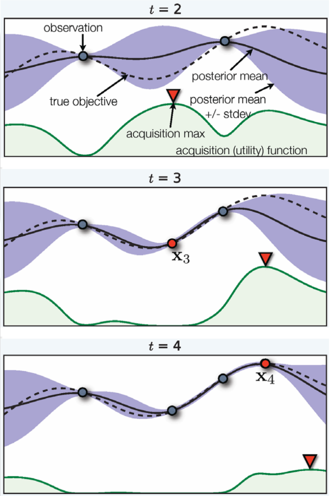
Bayesian Optimization

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: $\mathbf{x_i}$, a vector of feature values
- Observed algorithm runtime data: $\langle(\theta_i, \mathbf{x}_i, y_i)\rangle_{i=1}^N$

Find: a mapping $\hat{m} : [\theta, \mathbf{x}] \mapsto y$ 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: $\mathbf{x_i}$, a vector of feature values
- Observed algorithm runtime data: $\langle(\theta_i, \mathbf{x}_i, y_i)\rangle_{i=1}^N$

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

Fitting a Regression Tree

param 1	feature 2	param 3	runtime
false	2	red	3.7
false	2.5	blue	20
true	5.5	red	2.1
false	5.5	blue	25
false	5	red	1.2
true	4.5	green	19
true	4	blue	12
true	3.5	green	17

$\text{param}_3 \in \{\text{red}\}$

param 1	feature 2	param 3	runtime
false	2	red	3.7
true	5.5	red	2.1
false	5	red	1.2

$\text{feature}_2 \leq 3.5$

param 1	feature 2	param 3	runtime
false	2	red	3.7

$\text{feature}_2 > 3.5$

param 1	feature 2	param 3	runtime
true	5.5	red	2.1
false	5	red	1.2

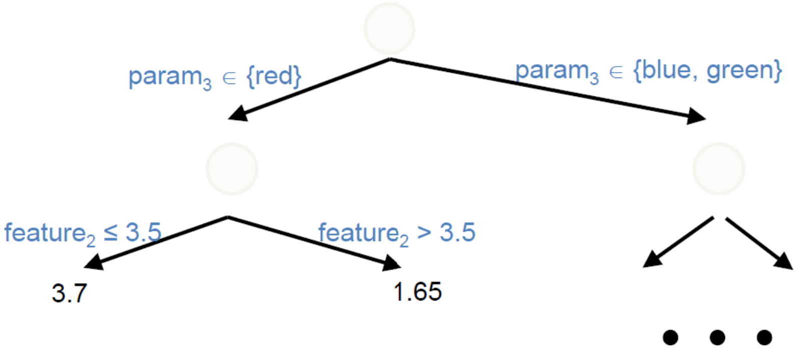
$\text{param}_3 \in \{\text{blue, green}\}$

param 1	feature 2	param 3	runtime
false	2.5	blue	20
false	5.5	blue	25
true	4.5	green	19
true	4	blue	12
true	3.5	green	17

• • •

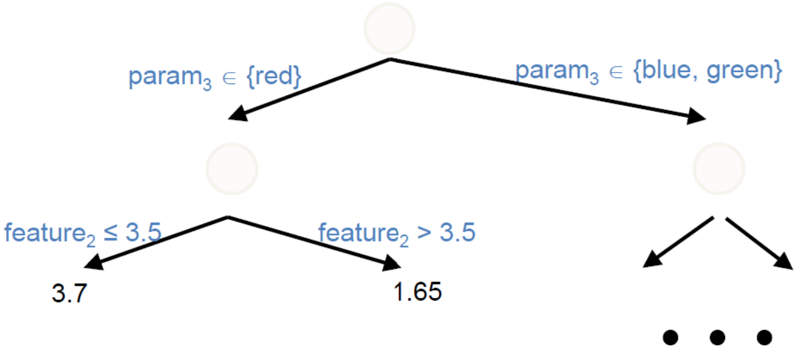
Fitting a Regression Tree

- In each internal node: only store split criterion used
- In each leaf: store mean of responses



Fitting a Regression Tree

- In each internal node: only store split criterion used
- In each leaf: store mean of responses



Prediction for a new data point: walk down the tree, return stored value

- E.g. for (param_1, feature_2, param_3) = (true, 4.7, red): 1.65

Random Forests: Sets of Regression Trees

Training

- Draw T bootstrap samples of the data
- For each bootstrap sample, fit a randomized regression tree

Random Forests: Sets of Regression Trees

Training

- Draw T bootstrap samples of the data
- For each bootstrap sample, fit a randomized regression tree

Prediction

- Predict with each of the T trees
- Return empirical mean and variance across these T predictions

Random Forests: Sets of Regression Trees

Training

- Draw T bootstrap samples of the data
- For each bootstrap sample, fit a randomized regression tree

Prediction

- Predict with each of the T trees
- Return empirical mean and variance across these T predictions

Complexity for N data points

- Training: $O(TNlog^2N)$
- Prediction: $O(TlogN)$

SMAC: Averaging Across Multiple Instances

Fit a Model to the Algorithm Performance Data

- Observed algorithm runtime data: $\langle(\theta_i, \mathbf{x}_i, y_i)\rangle_{i=1}^{N}$
- Random forest model $\hat{m} : \mathbf{\Theta} \times \mathcal{I} \rightarrow \mathbb{R}$

SMAC: Averaging Across Multiple Instances

Fit a Model to the Algorithm Performance Data

- Observed algorithm runtime data: $\langle(\theta_i, \mathbf{x}_i, y_i)\rangle_{i=1}^{N}$
- Random forest model $\hat{m} : \mathbf{\Theta} \times \mathcal{I} \to \mathbb{R}$

Aggregate over instances by marginalization

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

- Intuition: predict for each instance and then average
- More efficient implementation in random forests

SMAC: Putting it all Together

Algorithm 2: SMAC

Initialize with a single run for the default

SMAC: Putting it all Together

Algorithm 2: SMAC

Initialize with a single run for the default

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

SMAC: Putting it all 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: Putting it all Together

Algorithm 2: SMAC

Initialize with a single run for the default

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

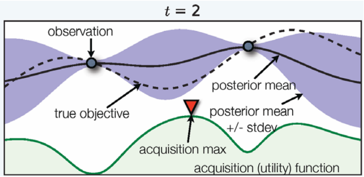
Aggregate over instances: $\hat{f}(\theta) = \mathbb{E}_{\pi \sim \mathcal{D}}(\hat{m}(\theta, \pi))$

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



SMAC: Putting it all Together

Algorithm 2: SMAC

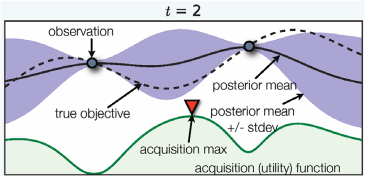
Initialize with a single run for the default

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

Race selected configurations against best known



SMAC: Putting it all Together

Algorithm 2: SMAC

Initialize with a single run for the default

repeat

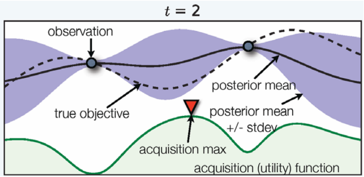
  Learn a RF model from data so far: $\hat{m} : \boldsymbol{\Theta} \times \mathcal{I} \rightarrow \mathbb{R}$

  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

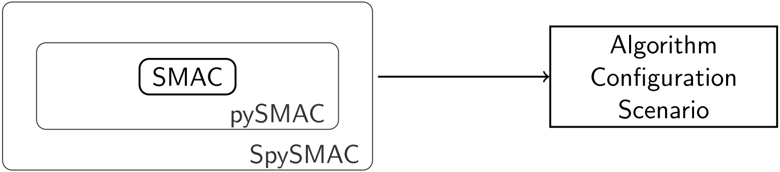
until *time budget exhausted*



Outline

1. The Algorithm Configuration Problem
2. Using AC Systems
   - SpySMAC: A SAT Python Tool for AC
   - pySMAC: A Python Interface to AC
   - SMAC: Full Flexibility for AC
3. Importance of Parameters
4. Pitfalls and Best Practices
5. Advanced Topics

SMAC, pySMAC, SpySMAC



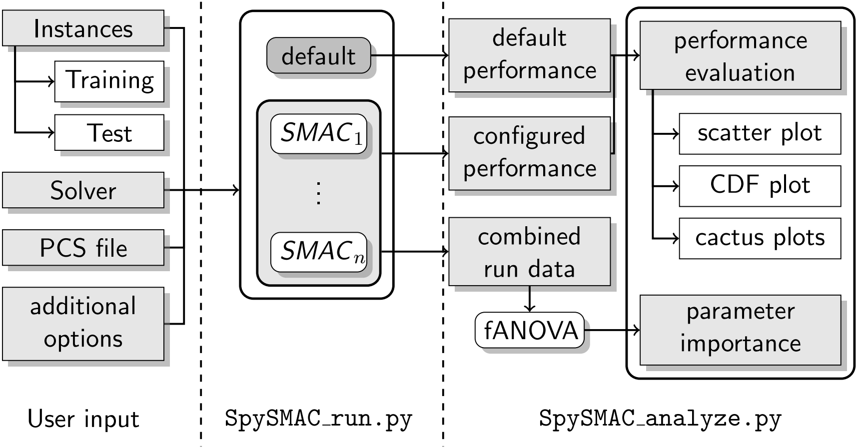
SMAC Configurator implemented in JAVA

pySMAC Python Interface to *SMAC*

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

Outline

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

```
$ minisat --help

  -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)
```

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

```
$ minisat --help

  -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)
```

Pitfall: Random seed (rnd-seed) should not be tuned!

Toy AC Scenario

- Solver: *MiniSAT*
- PCS: 8 parameters
- Instances: software verification of gzip
- Budget: 60 seconds
- Cutoff: 2 seconds
- Object: runtime
- Statistic: PAR10 (penalized average runtime, counting timeouts at t_{max} as $10 \cdot t_{max}$)

Hands-on: *SpySMAC*

In your virtual machine:

Determine optimized configuration

```
$ cd AC-Tutorial/SpySMAC/
$ python ./SpySMAC/SpySMAC_run.py
-i ./SpySMAC/examples/swv-inst/SWV-GZIP/
-b ./SpySMAC/examples/minisat/core/minisat
-p ./SpySMAC/examples/minisat/pcs.txt
-o minisat-logs
--prefix "-"
-c 2 -B 60
```

Hands-on: *SpySMAC*

In your virtual machine:

Validate default configuration

```
$ python ./SpySMAC/SpySMAC_run.py
-i ./SpySMAC/examples/swv-inst/SWV-GZIP/
-b ./SpySMAC/examples/minisat/core/minisat
-p ./SpySMAC/examples/minisat/pcs.txt
-o minisat-logs
--prefix "-"
-c 2 -B 60
--seed 0                  # <-- validate default!
```

Inputs of SpySMAC

- Parameter Configuration Space
- Algorithm
- Instances
- Options:
  - Target algorithm runtime cutoff
  - Configuration budget

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

Configuration Space – Parameter Types

Continuous

Parameter can take every value in a given range:

`param_name [lower bound, upper bound][default]`

e.g., `rnd-freq [0,1][0]`

Configuration Space – Parameter Types

Continuous

Parameter can take every value in a given range:

`param_name [lower bound, upper bound][default]`

e.g., `rnd-freq [0,1][0]`

Integer

Parameter can take every integer value in a given range:

`param_name [lower bound, upper bound][default]i`

e.g., `rfirst [1,10000000][1]i`

Configuration Space – Parameter Types

Continuous

Parameter can take every value in a given range:

`param_name [lower bound, upper bound][default]`

e.g., `rnd-freq [0,1][0]`

Integer

Parameter can take every integer value in a given range:

`param_name [lower bound, upper bound][default]i`

e.g., `rfirst [1,10000000][1]i`

Categorical

Parameter can take every value from an unordered finite set

`param_name {value1, ..., valueN}[default]`

e.g., `ccmin-mode {0,1,2}[0]`

Range Transformation

Log-Transformation

- The ranges of continuous and integer parameter can be log-transformed

$\rightarrow$ *SMAC* samples new configurations uniformly from the log-transformed range

- E.g., the difference between 0.9 and 0.8 is maybe not so important as between 0.1 and 0.2

`param_name [lower bound, upper bound][default]l`

e.g., `rinc [1.00001,1024][1.00001]l`

Structured Configuration Space

Conditionals

- 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
- θ_1 currently has to be a categorical parameter

```
param2 | param1 in {H}
```

Structured Configuration Space

Conditionals

- 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
- θ_1 currently has to be a categorical parameter

```
param2 | param1 in {H}
```

Forbidden Combinations

Sometimes, combinations of parameter settings are forbidden
e.g., the combination of $\theta_3 = 1$ and $\theta_4 = 2$ is forbidden

```
{param3=1, param4=2}
```

PCS Example: *MiniSAT*

```
$ minisat --help

  -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)
```

~/AC-Tutorial/SpySMAC/SpySMAC/examples/minisat/pcs.txt

```
rnd-freq [0,1][1]
var-decay [0.001,1][0.001]l
cla-decay [0.001,1][0.001]l
rinc [1.00001,1024][1.00001]l
gc-frac [0,1][0.00001]
rfirst [1,10000000][1]il
ccmin-mode {0,1,2}[0]
phase-saving {0,1,2}[0]
```

→ For illustration, we use a bad default configuration.

Decision: Instance set

Representative instances

- Representative of the instances you want to solve later

Decision: Instance set

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

Decision: Instance set

Homogeneity of instances

- Easier to optimize on homogeneous instances
  - There is one configuration that performs well on all instances
  - Indicator: Same characteristics, e.g., all instances encode the same problem

Decision: Instance set

Homogeneity of instances

- Easier to optimize on homogeneous instances
  - There is one configuration that performs well on all instances
  - Indicator: Same characteristics, e.g., all instances encode the same problem
- Harder on heterogeneous instances
  - A portfolio approach will perform better an these instances
  - Indicator: Different characteristics, e.g., instances encode different problems

Decision: Instance set

Homogeneity of instances

- Easier to optimize on homogeneous instances
  - There is one configuration that performs well on all instances
  - Indicator: Same characteristics, e.g., all instances encode the same problem
- Harder on heterogeneous instances
  - A portfolio approach will perform better an these instances
  - Indicator: Different characteristics, e.g., instances encode different problems

Enough instances

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

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 (i.e., 1000 times the cutoff)
  - 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 (i.e., 1000 times the cutoff)
  - 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 300 CPU seconds

Toy AC Scenario (repeated)

- Solver: *MiniSAT*
- PCS: 8 parameters
- Instances: software verification of gzip
- Budget: 60 seconds
- Cutoff: 2 seconds
- Object: runtime
- Statistic: PAR10 (penalized average runtime, counting timeouts at t_{max} as $10 \cdot t_{max}$)

Hands-on: *SpySMAC*

In your virtual machine:

Analyze and show report

```
$ python ./SpySMAC/SpySMAC_analyze.py
-i minisat-logs/
-o minisat-report
[...]
$ firefox minisat-report/index.html &
```

Running examples of algorithms to compare

- Taken from the Configurable SAT Solver Challenge [Hutter et al, 2015] (http://aclib.net/cssc2014/)

Running examples of algorithms to compare

- Taken from the Configurable SAT Solver Challenge [Hutter et al, 2015] (`http://aclib.net/cssc2014/`)
- Example 1:
  - Algorithm: Lingeling [Biere, 2013]
    - 2 versions: default vs. configured on training instances

Running examples of algorithms to compare

- Taken from the Configurable SAT Solver Challenge [Hutter et al, 2015] (`http://aclib.net/cssc2014/`)
- Example 1:
  - Algorithm: Lingeling [Biere, 2013]
    - 2 versions: default vs. configured on training instances
  - Benchmark set: Circuit-Fuzz [Brummayer et al, 2013]

Running examples of algorithms to compare

- Taken from the Configurable SAT Solver Challenge [Hutter et al, 2015] (`http://aclib.net/cssc2014/`)
- Example 1:
  - Algorithm: Lingeling [Biere, 2013]
    - 2 versions: default vs. configured on training instances
  - Benchmark set: Circuit-Fuzz [Brummayer et al, 2013]
- Example 2:
  - Algorithm: Clasp [Gebser et al, 2012]
    - 2 versions: default vs. configured on training instances
  - Benchmark set: N-Rooks [Manthey & Steinke, 2014]

Running examples of algorithms to compare

- Taken from the Configurable SAT Solver Challenge [Hutter et al, 2015] (`http://aclib.net/cssc2014/`)
- Example 1:
  - Algorithm: Lingeling [Biere, 2013]
    - 2 versions: default vs. configured on training instances
  - Benchmark set: Circuit-Fuzz [Brummayer et al, 2013]
- Example 2:
  - Algorithm: Clasp [Gebser et al, 2012]
    - 2 versions: default vs. configured on training instances
  - Benchmark set: N-Rooks [Manthey & Steinke, 2014]
- Comparing runtimes on test instances not used for configuration

Comparing algorithms based on summary statistics

Clasp on N-Rooks, $t_{max} = 300s$

	Default	Configured
Average runtime [s]	81.8	4.68
PAR10 [s]	704	4.68
Timeouts (out of 351)	81	0

PAR10: penalized average runtime, counting timeouts at t_{max} as $10 \cdot t_{max}$

Comparing algorithms based on summary statistics

Clasp on N-Rooks, $t_{max} = 300s$

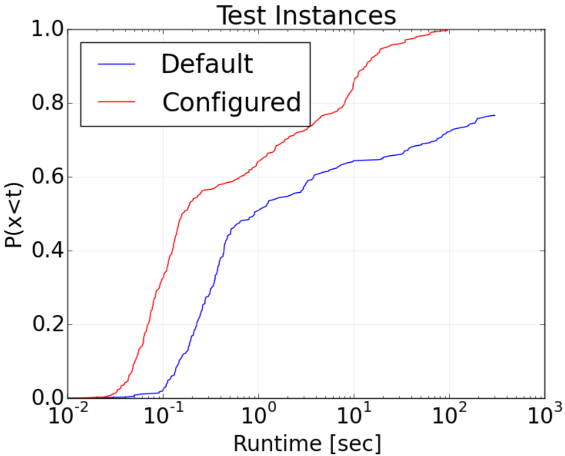
	Default	Configured
Average runtime [s]	81.8	4.68
PAR10 [s]	704	4.68
Timeouts (out of 351)	81	0

PAR10: penalized average runtime, counting timeouts at t_{max} as $10 \cdot t_{max}$

Lingeling on Circuit-Fuzz, $t_{max} = 300s$

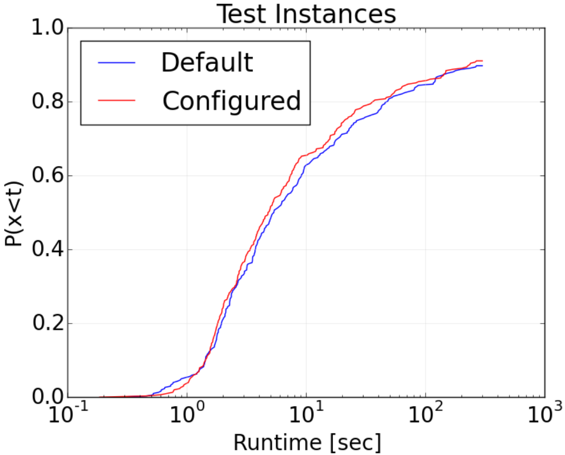
	Default	Configured
Average runtime [s]	47.8	32.0
PAR10 [s]	186	115
Timeouts (out of 585)	30	18

Comparing algorithms based on their runtime distributions



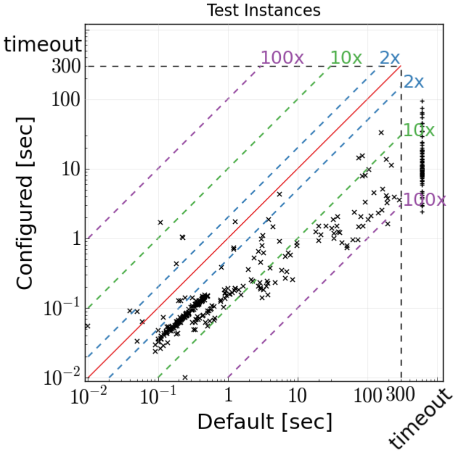
Distributions of runtime across benchmark instances
(Clasp on N-Rooks)

Comparing algorithms based on their runtime distributions



(Lingeling on Circuit-Fuzz)

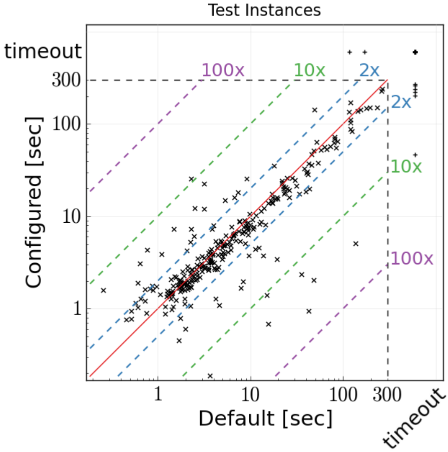
Comparing algorithms based on scatter plots



Each marker represents one instance; note the log-log axis!

(Clasp on N-Rooks; 81 vs. 0 timeouts)

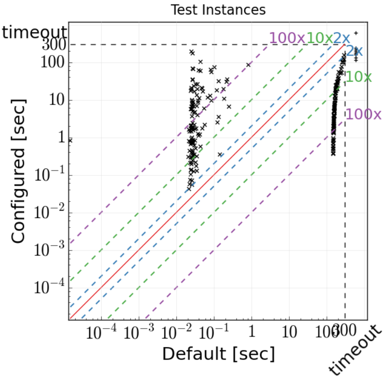
Comparing algorithms based on scatter plots



(Lingeling on Circuit-Fuzz; 30 vs. 18 timeouts)

Comparing algorithms based on scatter plots

Scatter plots can reveal clear patterns in the data

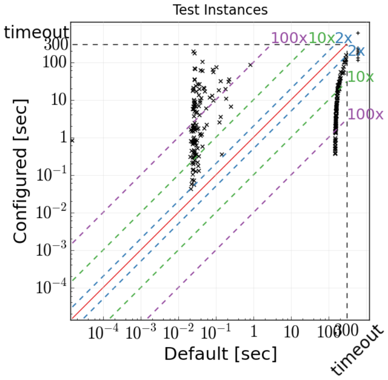


Example: an algorithm that

- first runs SLS algorithm A (good for satisfiable instances) for t seconds
- then runs complete tree search algorithm B (good for unsatisfiable instances) until time is up

Comparing algorithms based on scatter plots

Scatter plots can reveal clear patterns in the data



Example: an algorithm that

- first runs SLS algorithm A (good for satisfiable instances) for t seconds
- then runs complete tree search algorithm B (good for unsatisfiable instances) until time is up

There are 2 instance clusters:
satisfiable (left) and unsatisfiable instances (right)

Why not always using *SpySMAC*?

Advantages:

- Easy to set up – easier than plain *SMAC*
- Includes an analyzing part – not part of *SMAC*

Why not always using *SpySMAC*?

Advantages:

- Easy to set up – easier than plain *SMAC*
- Includes an analyzing part – not part of *SMAC*

However, *SpySMAC* has some limitations right now:

- Only runtime optimization
- Parses only standard output of SAT solvers

Why not always using *SpySMAC*?

Advantages:

- Easy to set up – easier than plain *SMAC*
- Includes an analyzing part – not part of *SMAC*

However, *SpySMAC* has some limitations right now:

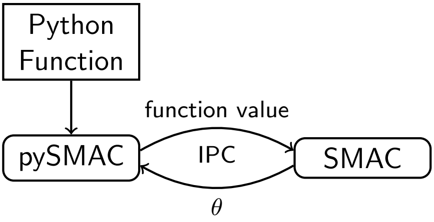
- Only runtime optimization
- Parses only standard output of SAT solvers

However, with a little Python experience, you could solve all these limitations.

BREAK - 30 Minutes!

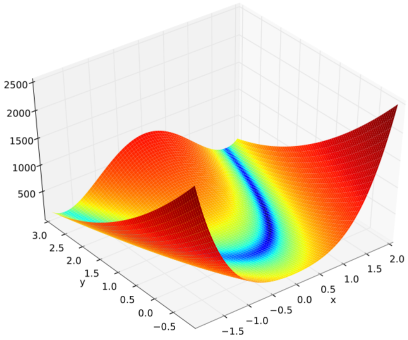
Outline

pySMAC



- "Algorithm"; black box (Python) function
- Hides most of *SMAC*'s complexity (e.g., instances)
- Still as powerful as *SMAC*

Toy Example: Rosenbrock Function



source: wikipedia.org

$$f(x, y) = (a - x)^2 + b(y - x^2)^2$$

- Well-known non-convex function
- Optimum at $(x, y) = (a, a^2)$
- Multidimensional generalization possible (here: 4D)

Optimize on Rosenbrock

Optimize Strategies:

1. *SMAC* (as shown before)
2. Random search

Optimize on Rosenbrock

Optimize Strategies:

1. *SMAC* (as shown before)
2. Random search

In your virtual machine

pySMAC Call

```
$ cd ~/AC-Tutorial/pysmac
$ python example.py
```

Optimize on Rosenbrock

Optimize Strategies:

1. *SMAC* (as shown before)
2. Random search

In your virtual machine

pySMAC Call

```
$ cd ~/AC-Tutorial/pysmac
$ python example.py
```

In the meantime, let us look into `example.py`

Advanced *pySMAC*

- Restricting your functions resources
- Optimizing runtime instead of quality
- Optimizing on a set of instances
- Validation
- Non-deterministic functions

Outline

SMAC

- Core of our algorithm configuration tools
- Implementation of the optimization algorithm

Further Inputs (wrt *SpySMAC*)

- Scenario File
- Call to algorithm through a wrapper
  - Wrapper has to honour a specified syntax

Scenario File Example

```
algo-exec python -u minisat/SATWrapper.py --mem-limit 1024\
   --script minisat/MiniSATWrapper.py
pcs-file minisat/pcs.txt
instances minisat/instances_train.txt
test-instances minisat/instances_test.txt
cutoff_time 2
wallclock-limit 120
algo-deterministic False
run-obj runtime
```

How to Wrap a Target Algorithm

Call Format

```
<algo executable> <instance name>
<instance specific information> <cutoff time>
<cutoff length> <seed> <param> <value> <param> <value>...
```

How to Wrap a Target Algorithm

Call Format

```
<algo executable> <instance name>
<instance specific information> <cutoff time>
<cutoff length> <seed> <param> <value> <param> <value>...
```

Output Format

```
Result of this algorithm run: <status>,
<runtime>, <runlength>, <quality>, <seed>
```

How To Wrap the Target Algorithm: GenericWrapper

We provide a Python script with basic functionality as a wrapper, called GenericWrapper:

- Limitation of runtime and memory with `runsolver`
- Interface to configurator
  - Parsing of input parameters
  - Output string in the required format

How To Wrap the Target Algorithm: GenericWrapper

We provide a Python script with basic functionality as a wrapper, called GenericWrapper:

- Limitation of runtime and memory with `runsolver`
- Interface to configurator
  - Parsing of input parameters
  - Output string in the required format

You only have to implement two functions:

- How to call your algorithm with the given parameter configuration?
- How to parse the output of your algorithm?

GenericWrapper: CMD Call for MiniSAT

see `minisat/MiniSATWrapper.py`

```
def get_command_line_cmd(runargs, config):
    cmd = "minisat/minisat -rnd-seed=%d" %(runargs["seed"])
    for name, value in config.items():
        cmd += " %s=%s" %(name,  value)
    cmd += " %s" %(runargs["instance"])
    return cmd
```

GenericWrapper: Output Parsing for MiniSAT

see `minisat/SATWrapper.py` (Bug in this script in the VM)

```
def process_results(self, filepointer, exit_code):
    data = filepointer.read()
    resultMap = {}
    if re.search('UNSATISFIABLE', data):
        resultMap['status'] = 'UNSAT'
    elif re.search('SATISFIABLE', data):
        resultMap['status'] = 'SAT'
    elif re.search('UNKNOWN', data):
        resultMap['status'] = 'TIMEOUT'
    else:
        resultMap['status'] = 'CRASHED'
    return resultMap
```

Hands-On: *SMAC*

```
$ cd ~/AC-Tutorial/minisat-smac/
$ ../smac-v2.10.03-master-778/smac
--scenarioFile scenario.txt
[...]
$ ../smac-v2.10.03-master-778/smac-validate
--scenarioFile scenario.txt
--random-configurations 1 --includeDefaultAsFirstRandom true
--numRun 1
[...]
```

Hands-On: *SMAC*'s Output

```
SMAC has finished. [...]
Total number of runs performed: 222,
total configurations tried: 23.
Total CPU time used: 79 s, total wallclock time used: 121 s.
SMAC's final incumbent: config 16 (internal ID: 0x34CE3),
with estimated penalized average runtime (PAR10): 0.868,
based on 24 run(s) on 24 training instance(s).
```

Hands-On: Validation Output

```
Estimated mean quality
of final incumbent config 16
on test set: 0.0168,
based on 45 run(s) on 45 test instance(s).
Sample call for the final incumbent:
cd /home/ac/AC-Tutorial/minisat-smac;
python -u minisat/SATWrapper.py --mem-limit 1024
--script minisat/MiniSATWrapper.py
swv-inst/SWV-GZIP/gzip_vc1073.cnf 0 2.0
2147483647 12660129
-ccmin-mode '2' -cla-decay '0.014589435384907675' [...]
```

Hands-On: Validation of Default

```
$ cat validationResults-cli-1-walltime.csv
"Time","Training Performance","Test Set Performance",[...]
0.0,0.0,0.4912888888888894,[...]
```

Instance Features

Instance Features

- Numerical characterizations of problem instances
- Examples:
  - SAT: #variables, #clauses,...
  - Planning: #actions, #fluents,...
  - Machine Learning: #observations, #features,...

Instance Features

Instance Features

- Numerical characterizations of problem instances
- Examples:
  - SAT: #variables, #clauses,...
  - Planning: #actions, #fluents,...
  - Machine Learning: #observations, #features,...
- Can improve the accuracy of the EPM used in *SMAC*

Types of Instance Features

Counting Features

- Compute some statistics about its characteristics

Types of Instance Features

Counting Features

- Compute some statistics about its characteristics

Probing Features

- Running an algorithm for a short amount of time
- Analyze how the algorithm behaves
- For example, number of steps to the best local minimum in a run.

Types of Instance Features

Counting Features

- Compute some statistics about its characteristics

Probing Features

- Running an algorithm for a short amount of time
- Analyze how the algorithm behaves
- For example, number of steps to the best local minimum in a run.

Please note that there are different names in literature for these types of features. For example, probing features are related to landmarking features.

Instance Features: SAT [Hutter et al, 2014]

- Problem size features (7)
- Variable-Clause graph features (10)
- Variable graph features (9)
- Clause graph features (10)
- Balance features (13)
- Proximity to horn formula (7)
- DPLL Probing Features (7)
- LP-based feature (6)
- Local search probing features (12)
- Clause learning features (18)
- Survey propagation feature (18)

Hands-On: Instance features for SAT

```
$ cd ~/AC-Tutorial/minisat-smac/
$ bash get_swv_minisat_features.sh > features.csv
$ ../smac-v2.10.03-master-778/smac
--scenarioFile scenario-features.txt
```

Another Toy Example: Pyperplan

- Pyperplan is a simple planner
- Developed for demonstration purposes on the University of Freiburg
- Only 2 parameters: search heuristic and strategy

```
heuristic categorical {hmax,hff,hadd,blind,hsa,lmcut,landmark}[blind]
search categorical {ehs,gbf,wastar,astar,ids,bfs}[bfs]
```

$\rightarrow$ 42 parameter configurations

- Instance set: 15 elevator instances (training)
- Cutoff time: 10 seconds

$\rightarrow$ Worst case of trying all configurations on all instances: $10 \cdot 15 \cdot 42 = 6300$ seconds

- configuration budget: 600 seconds

Hands-On: Configuration of a Planner

```
$ cd ~/AC-Tutorial/pyperplan/
$ ../smac-v2.10.03-master-778/smac
--scenarioFile scenario.txt
```

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

- 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

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

Basic Approach

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

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

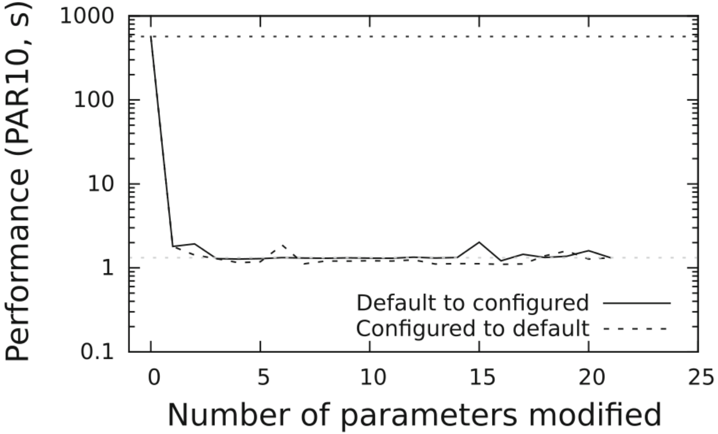
Ablation with Racing

- To determine the best flip in each iteration, use racing with a statistical test to speed up the decision

Toy example: Ablation

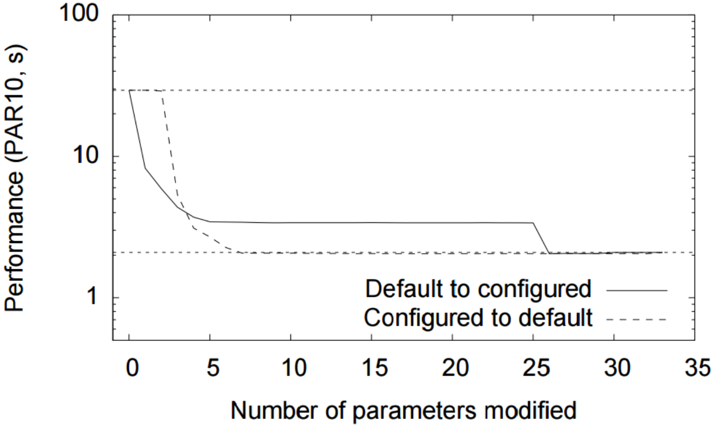
	θ_1	θ_2	θ_3	m
Default	1	1	1	100
Conf	2	2	2	10
1st Iteration				
	2	1	1	90
	1	2	1	30
	1	1	2	100
Flip θ_2				
2nd Iteration				
	2	2	1	10
	1	2	2	30
Flip θ_1				
3rd Iteration				
	2	2	2	10
Flip θ_3				

Ablation Example: *Spear* on SWV



Source: [Fawcett et al. 2013]

Ablation Example: LGP on Zenotravel



Source: [Fawcett et al. 2013]

Outline

1. The Algorithm Configuration Problem
2. Using AC Systems
3. Importance of Parameters
   - Ablation
   - Forward Selection
   - fANOVA
4. Pitfalls and Best Practices
5. Advanced Topics

Forward Selection [Hutter et al. 2013]

Idea

- Which parameters (or instance features) do we need to train a good EPM ($\hat{m} : \mathbf{\Theta} \times \mathcal{F} \rightarrow \mathbb{R}$)?
- Use forward selection to identify important parameters (/instances)

Forward Selection [Hutter et al. 2013]

Idea

- Which parameters (or instance features) do we need to train a good EPM ($\hat{m} : \mathbf{\Theta} \times \mathcal{F} \rightarrow \mathbb{R}$)?
- Use forward selection to identify important parameters (/instances)

Feature Selection via Forward Selection

- Iterative approach
- Add in each iteration the parameter (/feature) that improves the quality of the EPM the most

Forward Selection [Hutter et al. 2013]

Idea

- Which parameters (or instance features) do we need to train a good EPM ($\hat{m} : \mathbf{\Theta} \times \mathcal{F} \to \mathbb{R}$)?
- Use forward selection to identify important parameters (/instances)

Feature Selection via Forward Selection

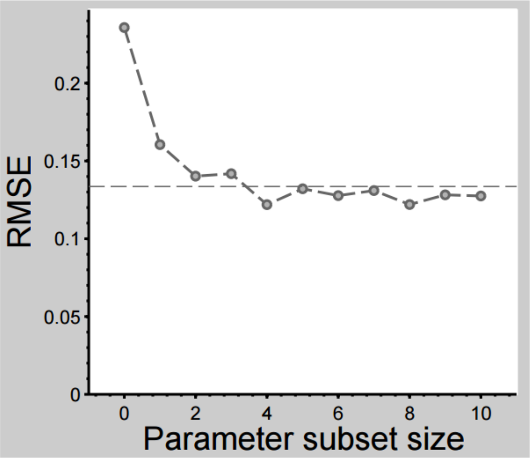
- Iterative approach
- Add in each iteration the parameter (/feature) that improves the quality of the EPM the most

Details

- Minimize RMSE (root mean squared error) on a validation set
- Limit the maximal number of selected parameters (/features)

Forward Selection [Hutter et al. 2013]

Spear on SWV



Source: [Hutter et al. 2013]

Outline

fANOVA [Hutter et al. 2014]

fANOVA [Sobol 1993]

Write performance predictions $\hat{y}$ as a sum of components:

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

$$\hat{y}(\theta_1, \ldots, \theta_n) = \quad \text{average response} + \text{main effects} +$$
$$\text{2-D interaction effects} + \text{higher order effects}$$

fANOVA [Hutter et al. 2014]

fANOVA [Sobol 1993]

Write performance predictions $\hat{y}$ as a sum of components:

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

$$\hat{y}(\theta_1, \ldots, \theta_n) = \text{average response} + \text{main effects} + \text{2-D interaction effects} + \text{higher order effects}$$

Variance Decomposition

$$\mathrm{V} = \frac{1}{||\mathbf{\Theta}||} \int_{\theta_1} \ldots \int_{\theta_n} [(\hat{y}(\theta) - \hat{f}_0)^2] d\theta_1 \ldots d\theta_n$$

fANOVA [Hutter et al. 2014]

fANOVA [Sobol 1993]

Write performance predictions $\hat{y}$ as a sum of components:

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

$$\hat{y}(\theta_1, \ldots, \theta_n) = \text{average response} + \text{main effects} + \text{2-D interaction effects} + \text{higher order effects}$$

Variance Decomposition

$$\mathrm{V} = \frac{1}{||\boldsymbol{\Theta}||} \int_{\theta_1} \ldots \int_{\theta_n} [(\hat{y}(\theta) - \hat{f}_0)^2] d\theta_1 \ldots d\theta_n$$

Application to Parameter Importance

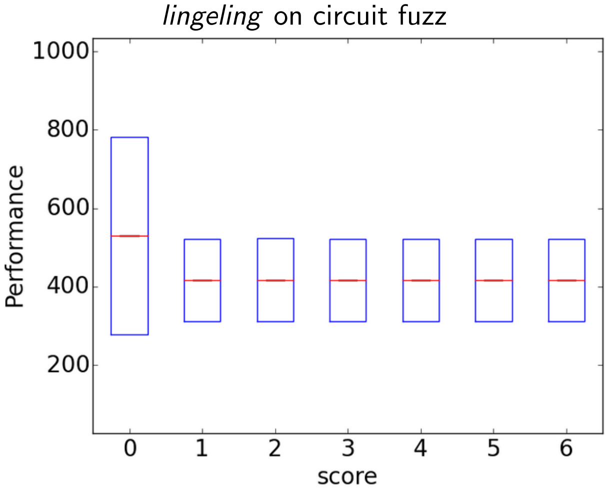
How much of the variance can be explained by a parameter (or combinations of parameters) marginalized over all other parameters?

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



0 seems to be a bad value for `score`

fANOVA Example

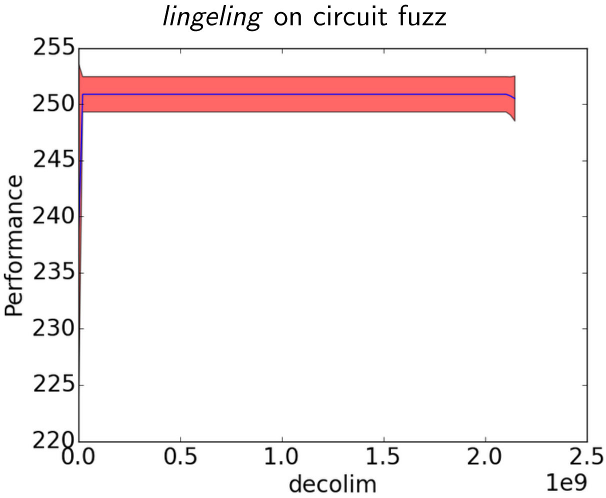
Interesting part of the configuration space

Consider only performance values better than the default

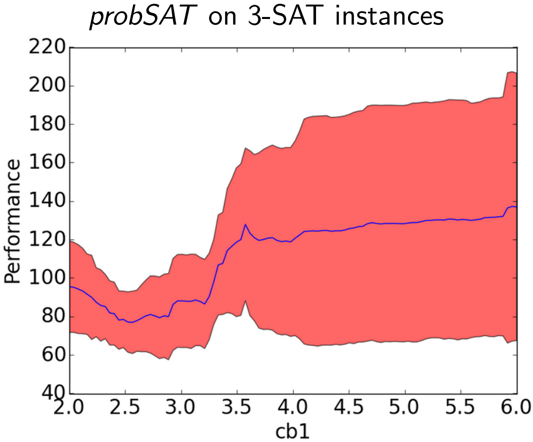
lingeling on circuit fuzz

Parameter	Importance
decolim	21.53
randecflipint	5.78
blksuccessrat	3.70
rstinoutinc	3.38
cgrmineff	3.26
actvlim	2.94
restartinit	2.83
synclslen	2.58
cgreleff	2.52
redoutvlim	2.24

fANOVA Example

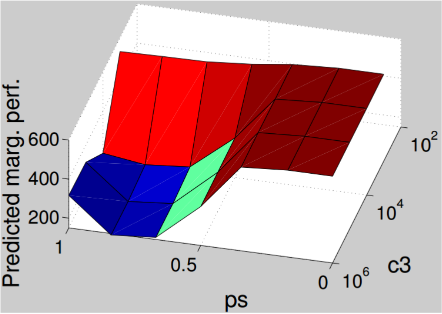


fANOVA Example



fANOVA Interaction Example

Sparrow on 5-SAT instances



fANOVA in *SpySMAC*

Look again at *SpySMAC* report from our *MiniSAT* example

```
$ firefox ~/AC-Tutorial/SpySMAC/minisat-report/index.html
```

More reports in
$\sim$`/AC-Tutorial/spysmac-reports`

Comparison Ablation, Forward Selection & *fANOVA*

Ablation

\+ Only method to compare two configurations

\- Needs a lot of algorithm runs $\rightarrow$ slow

Forward Selection

\+ EPM can be trained by the performance data collected during configuration

+/- Considers complete configuration space

\- Slow if the training of the EPM is slow (repeated process!)

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

Outline

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

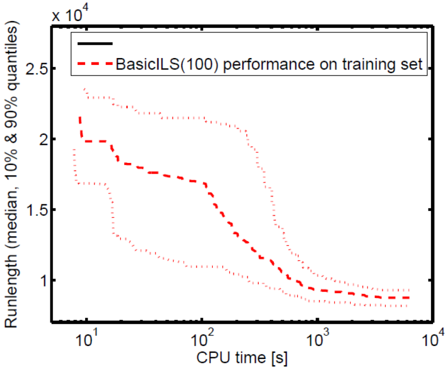
Example: minimizing SLS solver runlengths for a single SAT instance

- 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

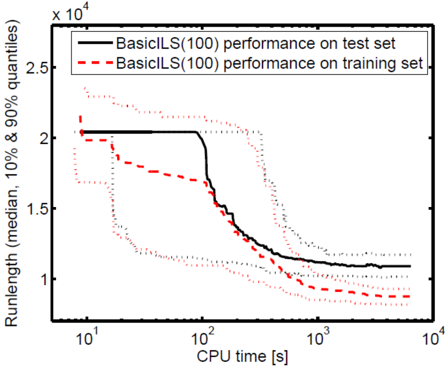
- Training cost, here based on N=100 runs with different seeds
- Test cost of $\hat{\theta}$ here based on 1000 new seeds



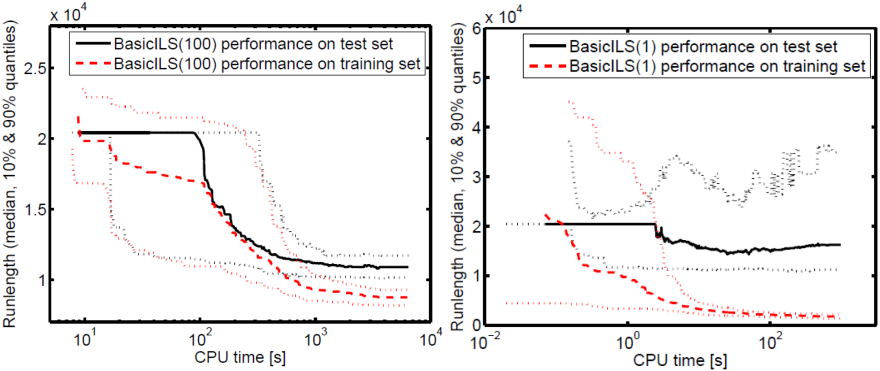
Overtuning Visualized

Example: minimizing SLS solver runlengths for a single SAT instance

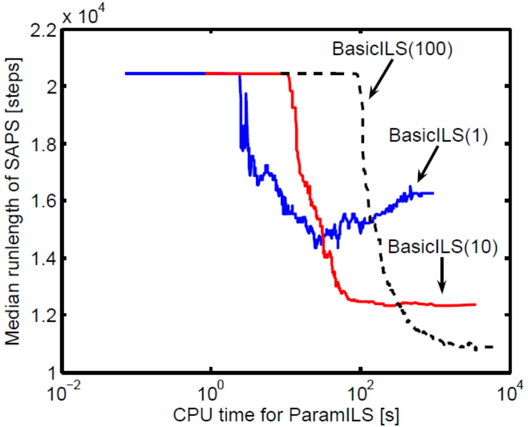
- Training cost, here based on N=100 runs with different seeds
- Test cost of $\hat{\theta}$ here based on 1000 new seeds



Overtuning is Stronger For Smaller Training Sets

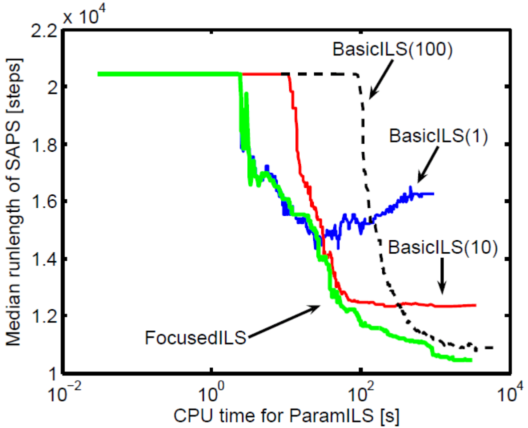


BasicILS(N) Test Results with Various N

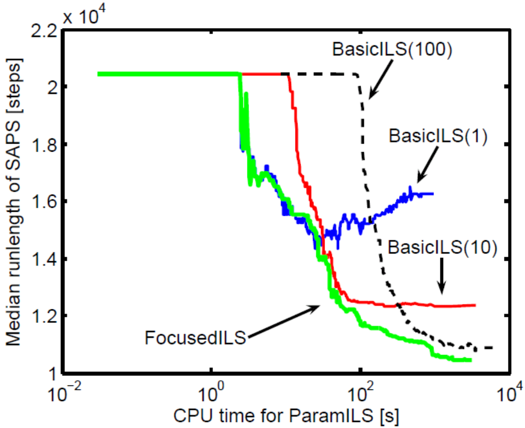


- Small N: fast evaluations $\rightarrow$ quick progress, but overtunes
- Large N: slow, but overtunes less

FocusedILS achieves the best of both worlds



FocusedILS achieves the best of both worlds



- Fast progress and no overtuning (provably, in the limit)
- General principle: focus your budget on good configurations

Hands-On: Overtuning to Fixed Seeds

Demonstration on the Simplest Case

- The real issue is overtuning to subset of instances
- But that would take too long to demo

Hands-On: Overtuning to Fixed Seeds

Demonstration on the Simplest Case

- The real issue is overtuning to subset of instances
- But that would take too long to demo

In your virtual machine:

Limit SMAC to optimize on 1 seed vs. 100 seeds

```
$ cd AC-Tutorial/saps-overtuning/

$ vim scenario1.txt
$ ../smac-v2.10.03-master-778/smac --scenarioFile scenario1.tx

$ vim scenario100.txt
$ ../smac-v2.10.03-master-778/smac --scenarioFile scenario100.
```

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

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

If you have N instances and a budget for N runs

- Do 1 run for each of the instances, with a different seed each
- This simultaneously captures variance in instance & seed space
- Provably yields lowest-variance estimator of mean performance

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

How to Wrap a Target Algorithm

Don't trust any target algorithm

- Will it terminate in the time you specify?
- Will it correctly report its time?
- Will it never use more memory than specified?
- Will it yield correct results with all parameter settings?

How to Wrap a Target Algorithm

Don't trust any target algorithm

- Will it terminate in the time you specify?
- Will it correctly report its time?
- Will it never use more memory than specified?
- Will it yield correct results with all parameter settings?

Good Practice

Wrap target runs with tool controlling time and memory

- E.g., `runsolver` [Roussel et al, 2012].
- Our genericWrapper.py already does this for you

How to Wrap a Target Algorithm

Don't trust any target algorithm

- Will it terminate in the time you specify?
- Will it correctly report its time?
- Will it never use more memory than specified?
- Will it yield correct results with all parameter settings?

Good Practice

Wrap target runs with tool controlling time and memory

- E.g., `runsolver` [Roussel et al, 2012].
- Our genericWrapper.py already does this for you

Good Practice

Verify correctness of target runs

- Detect crashes & penalize them
- Our genericWrapper.py already does the penalization for you

Pitfalls in Wrappers #1

Pitfall

Blindly minimizing target algorithm runtime

$\rightsquigarrow$ Typically, you will minimize the time to crash

Pitfalls in Wrappers #1

Pitfall

Blindly minimizing target algorithm runtime

$\rightsquigarrow$ Typically, you will minimize the time to crash

Anecdote 1

- In 2007, with Daniel Le Berre, I configured SAT4J
- I got huge improvements, emailed Daniel the configuration found
- But there was a bug
  - Some preprocessings were incompatible with some data structures
  - Leading to a very quick (wrong) result UNSAT
  - One solution: declare forbidden combinations

Pitfalls in Wrappers #1

Pitfall

Blindly minimizing target algorithm runtime

$\rightsquigarrow$ Typically, you will minimize the time to crash

Anecdote 1

- In 2007, with Daniel Le Berre, I configured SAT4J
- I got huge improvements, emailed Daniel the configuration found
- But there was a bug
  - Some preprocessings were incompatible with some data structures
  - Leading to a very quick (wrong) result UNSAT
  - One solution: declare forbidden combinations

Lesson learned: verify correctness of target runs

- E.g., for SAT: compare to known solubility status
- E.g., for SAT: check assignment found (polytime)

Pitfalls in Wrappers #2

Pitfall

Blindly minimizing target algorithm runtime

$\rightsquigarrow$ Typically, you will minimize the time to crash

Anecdote 2

- In 2010, I optimized several MIP solvers [Hutter et al, CPAIOR 2010]
- I found many bugs
- Non-default configurations are often poorly tested
- You can use algorithm configuration for testing!

Pitfalls in Wrappers #2

Pitfall

Blindly minimizing target algorithm runtime

$\rightsquigarrow$ Typically, you will minimize the time to crash

Anecdote 2

- In 2010, I optimized several MIP solvers [Hutter et al, CPAIOR 2010]
- I found many bugs
- Non-default configurations are often poorly tested
- You can use algorithm configuration for testing!

Anecdote 3

Some published work does indeed blindly minimize runtime

- Avoid this in your own work
- Ask about this in reviews etc

Pitfalls in Wrappers #3

Pitfall

Trusting the target algorithm to handle its time and memory

Anecdote 4

Configuring commercial MIP solvers was an eye-opener

- Some runs didn't end when not terminated externally
- Reported runtimes were incorrect (sometimes negative!)
- Jobs crashed on the cluster because memory limit was not honoured

Disclaimer: code versions from 2010.

Pitfalls in Wrappers #3

Pitfall

Trusting the target algorithm to handle its time and memory

Anecdote 4

Configuring commercial MIP solvers was an eye-opener

- Some runs didn't end when not terminated externally
- Reported runtimes were incorrect (sometimes negative!)
- Jobs crashed on the cluster because memory limit was not honoured

Disclaimer: code versions from 2010.

Lesson Learned

- Once bitten, twice shy. Don't trust your target algorithm!
- Use runsolver (or similar) to control time & memory

Pitfalls in Wrappers #4

Pitfall

Using different wrappers for comparing different configurators

Pitfalls in Wrappers #4

Pitfall

Using different wrappers for comparing different configurators

Anecdote 5

While developing SMAC, I compared it to ParamILS for tuning UBCSAT

- Somehow, SMAC achieved runtimes 20% better than 'possible'
- I had even used the same wrapper
- Only difference: ParamILS used an absolute path to the instance on the file system, SMAC a relative one

Pitfalls in Wrappers #4

Pitfall

Using different wrappers for comparing different configurators

Anecdote 5

While developing SMAC, I compared it to ParamILS for tuning UBCSAT

- Somehow, SMAC achieved runtimes 20% better than 'possible'
- I had even used the same wrapper
- Only difference: ParamILS used an absolute path to the instance on the file system, SMAC a relative one
- Explanation:
  - UBCSAT saved its callstring in the heap space before the instance data
  - Thus, the length of the callstring affected memory locality
  - Thus, more cache misses when using absolute paths $\rightarrow$ 20% greater runtime
  - Now fixed in UBCSAT

Pitfalls in Wrappers #5

Pitfall

Using different wrappers for comparing different configurators

Anecdote 6

A new configurator was introduced and compared to SMAC

- Wrapper for SMAC had several bugs, breaking the comparison
- Authors did not notice; paper published

Pitfalls in Wrappers #5

Pitfall

Using different wrappers for comparing different configurators

Anecdote 6

A new configurator was introduced and compared to SMAC

- Wrapper for SMAC had several bugs, breaking the comparison
- Authors did not notice; paper published

Bug 1: regular expressions

```
if ('s SATISFIABLE\n' in lines) or ('s UNSATISFIABLE' in lines)
```

- Missing \n after UNSATISFIABLE
  $\rightsquigarrow$ all UNSAT instances counted as TIMEOUT

Pitfalls in Wrappers #5

Pitfall

Using different wrappers for comparing different configurators

Anecdote 6

A new configurator was introduced and compared to SMAC

- Wrapper for SMAC had several bugs, breaking the comparison
- Authors did not notice; paper published

Bug 2: terminating the target algorithm

```
p = subprocess.Popen(cmd, shell=True)
...
os.kill(p.pid, signal.SIGKILL)
```

- Main problem: using shell=true
  - Leads to killing the shell, but not the target algorithm inside it

Lessons Learned for Comparisons of Configurators

It is far too easy to introduce a subtle difference in a new wrapper

Check: solver callstrings have to be identical for same configuration

- If possible, extend our genericWrapper.py from AClib

Use standard scenarios for comparing configurators: AClib

- We have a benchmark library of > 300 AC scenarios
- http://www.aclib.net
- Based on a git repository, maintained by several research groups
- As in other communities, reporting results on known benchmarks should be mandatory
- Please contribute new scenarios

Outline

1. The Algorithm Configuration Problem
2. Using AC Systems
3. Importance of Parameters
4. Pitfalls and Best Practices
   - Overtuning
   - Wrapping the Target Algorithm
   - General Advice
5. Advanced Topics

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

Correct

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

There often is no single great overall configuration
(see advanced topics for combinations with algorithm selection)

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`

Further tips and tricks with SMAC

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

Outline

1. The Algorithm Configuration Problem
2. Using AC Systems
3. Importance of Parameters
4. Pitfalls and Best Practices
5. Advanced Topics
   - Algorithm Configuration on Heterogeneous Data
   - More on Automated Machine Learning

Outline

Algorithm Configuration on Heterogeneous Instances

Known Problem

Algorithm configuration only performs well on homogeneous instance sets

- It only aims to find the single best configuration
- For heterogeneous instances, there might not be a single great configuration
  - That's why algorithm portfolios are so successful

Algorithm Configuration on Heterogeneous Instances

Known Problem

Algorithm configuration only performs well on homogeneous instance sets

- It only aims to find the single best configuration
- For heterogeneous instances, there might not be a single great configuration
  - That's why algorithm portfolios are so successful

Known Solution: Combine Algorithm Configuration & Portfolios

1. Use algorithm configuration to determine a set of complementary configurations
2. Build a portfolio out of these

Manual Expert Approach

Basic Assumption

Heterogeneous instance set can be divided into homogeneous subsets

Manual Expert Approach

Basic Assumption

Heterogeneous instance set can be divided into homogeneous subsets

Manual Expert

- An expert knows the homogeneous subsets (e.g., origin of instances)
- Determine a well-performing configuration on each subset
  - E.g., using algorithm configuration

  $\leadsto$ portfolio of configurations
- Use algorithm selection to select the right configuration for each instance

Instance-Specific Algorithm Configuration: *ISAC*

[Kadioglu et al. 2010]

Idea

Training:

1. Cluster instances into homogeneous subsets (using g-means in the instance feature space)
2. Apply algorithm configuration on each instance set

Instance-Specific Algorithm Configuration: *ISAC*

[Kadioglu et al. 2010]

Idea

Training:

1. Cluster instances into homogeneous subsets (using g-means in the instance feature space)
2. Apply algorithm configuration on each instance set

Test:

1. Determine the nearest cluster (k-NN with $k = 1$) in feature space
2. Apply optimized configuration of this cluster

Instance-Specific Algorithm Configuration: *ISAC*

[Kadioglu et al. 2010]

Idea

Training:

1. Cluster instances into homogeneous subsets (using g-means in the instance feature space)
2. Apply algorithm configuration on each instance set

Test:

1. Determine the nearest cluster (k-NN with $k = 1$) in feature space
2. Apply optimized configuration of this cluster

Assumes that instances with similar features can be solved well by similar configurations

ISAC+ [Malitsky et al. 2014]

Observations

- No need to restrict selection to the configuration found on a cluster
- Arbitrary algorithm selection approach possible

ISAC+ [Malitsky et al. 2014]

Observations

- No need to restrict selection to the configuration found on a cluster
- Arbitrary algorithm selection approach possible

Idea

1. Cluster instances
2. Apply algorithm configuration on each cluster
3. Build a portfolio out of all these configurations

Hydra [Xu et al. 2010]

Idea

- Iteratively add configurations to a portfolio $\mathcal{P}$, start with $\mathcal{P} = \emptyset$
- In each iteration: add a configuration that is complementary to $\mathcal{P}$

Hydra [Xu et al. 2010]

Idea

- Iteratively add configurations to a portfolio $\mathcal{P}$, start with $\mathcal{P} = \emptyset$
- In each iteration: add a configuration that is complementary to $\mathcal{P}$

Marginal contribution of a configuration θ to a portfolio $\mathcal{P}$

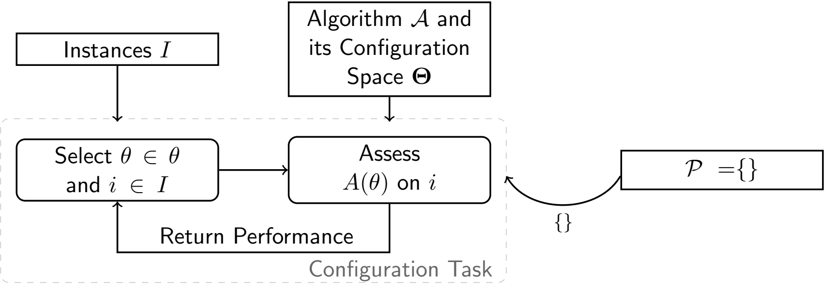
$$m(\mathcal{P}) - m(\mathcal{P} \cup \{\theta\})$$

Idea

- Iteratively add configurations to a portfolio $\mathcal{P}$, start with $\mathcal{P} = \emptyset$
- In each iteration: add a configuration that is complementary to $\mathcal{P}$

Marginal contribution of a configuration θ to a portfolio $\mathcal{P}$

$$m(\mathcal{P}) - m(\mathcal{P} \cup \{\theta\})$$

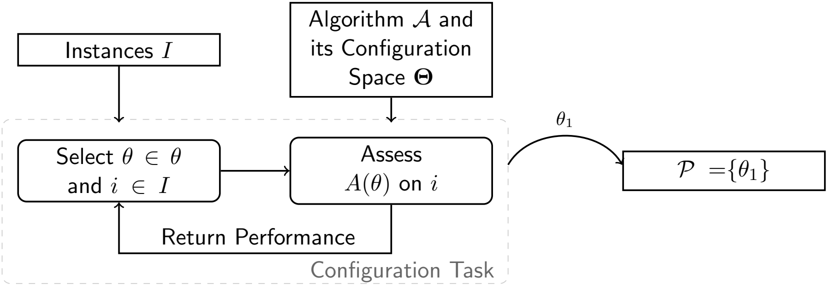


Idea

- Iteratively add configurations to a portfolio $\mathcal{P}$, start with $\mathcal{P} = \emptyset$
- In each iteration: add a configuration that is complementary to $\mathcal{P}$

Marginal contribution of a configuration θ to a portfolio $\mathcal{P}$

$$m(\mathcal{P}) - m(\mathcal{P} \cup \{\theta\})$$

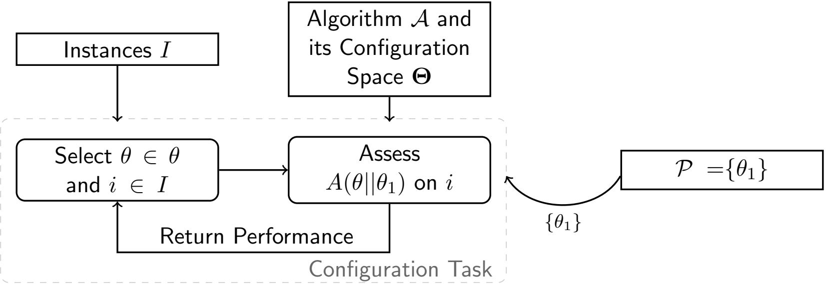


Idea

- Iteratively add configurations to a portfolio $\mathcal{P}$, start with $\mathcal{P} = \emptyset$
- In each iteration: add a configuration that is complementary to $\mathcal{P}$

Marginal contribution of a configuration θ to a portfolio $\mathcal{P}$

$$m(\mathcal{P}) - m(\mathcal{P} \cup \{\theta\})$$



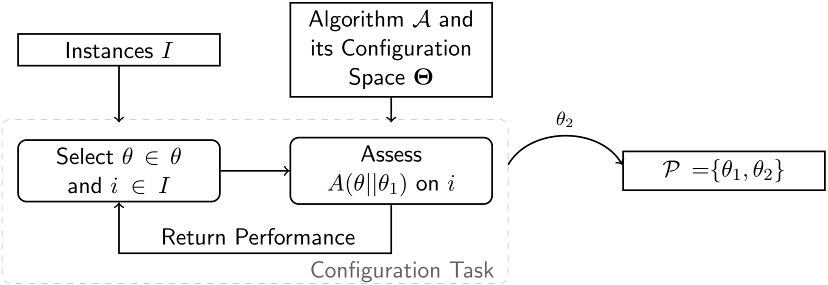
Hydra [Xu et al. 2010]

Idea

- Iteratively add configurations to a portfolio $\mathcal{P}$, start with $\mathcal{P} = \emptyset$
- In each iteration: add a configuration that is complementary to $\mathcal{P}$

Marginal contribution of a configuration θ to a portfolio $\mathcal{P}$

$$m(\mathcal{P}) - m(\mathcal{P} \cup \{\theta\})$$

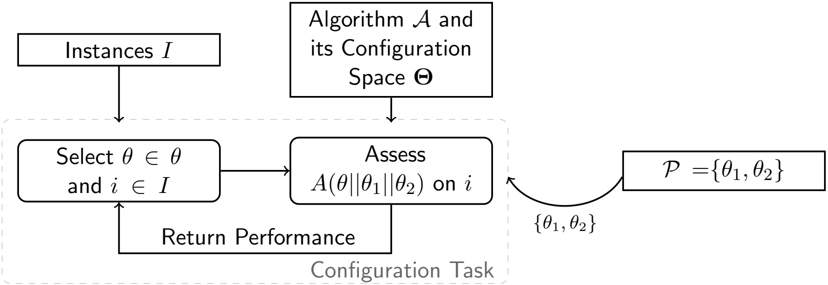


Idea

- Iteratively add configurations to a portfolio $\mathcal{P}$, start with $\mathcal{P} = \emptyset$
- In each iteration: add a configuration that is complementary to $\mathcal{P}$

Marginal contribution of a configuration θ to a portfolio $\mathcal{P}$

$$m(\mathcal{P}) - m(\mathcal{P} \cup \{\theta\})$$



Cedalion [J. Seipp et al 2014]

Idea

- Optimize a schedule of configurations with algorithm configuration

Cedalion [J. Seipp et al 2014]

Idea

- Optimize a schedule of configurations with algorithm configuration

Approach

- Iteratively add a configuration with a time slot t to a schedule $\mathcal{S} \oplus \langle \theta, t \rangle$

Cedalion [J. Seipp et al 2014]

Idea

- Optimize a schedule of configurations with algorithm configuration

Approach

- Iteratively add a configuration with a time slot t to a schedule $\mathcal{S} \oplus \langle \theta, t \rangle$
- The time slot is a further parameter in the configuration space
- Optimize marginal contribution per time spent:

$$\frac{m(\mathcal{S}) - m(\mathcal{S} \oplus \langle \theta, t \rangle)}{t}$$

Submodularity

Oberservation

- Performance metrics of *Hydra* and *Cedalion* are submodular
  - Diminishing returns: a configuration improves a smaller portfolio more than a larger one

Submodularity

Oberservation

- Performance metrics of *Hydra* and *Cedalion* are submodular
  - Diminishing returns: a configuration improves a smaller portfolio more than a larger one

Definition (Submodularity of f)

For every X, Y with $X \subseteq Y$ and every x we have that
$f(X \cup \{x\}) - f(X) \geq f(Y \cup \{x\}) - f(Y)$

Submodularity

Oberservation

- Performance metrics of *Hydra* and *Cedalion* are submodular
  - Diminishing returns: a configuration improves a smaller portfolio more than a larger one

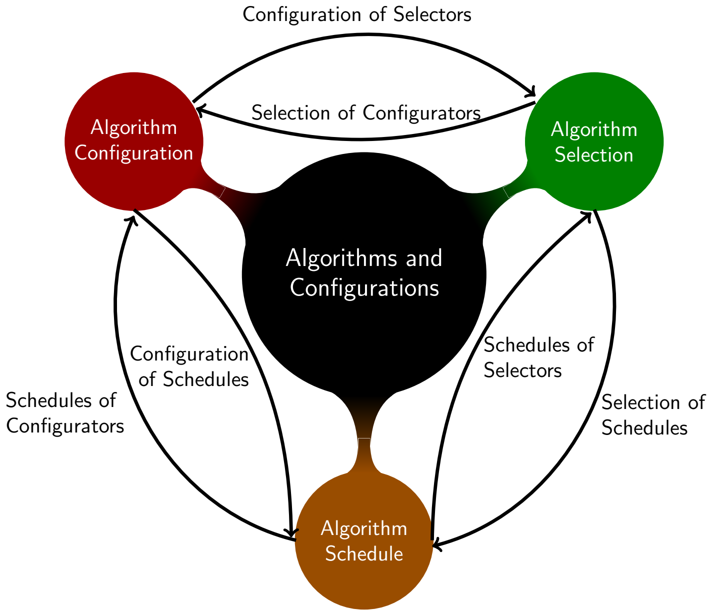
Definition (Submodularity of f)

For every X, Y with $X \subseteq Y$ and every x we have that
$f(X \cup \{x\}) - f(X) \geq f(Y \cup \{x\}) - f(Y)$

Advantage

We can bound the error of the portfolio/schedule
(see [Streeter & Golovin '07]

Further Combinations



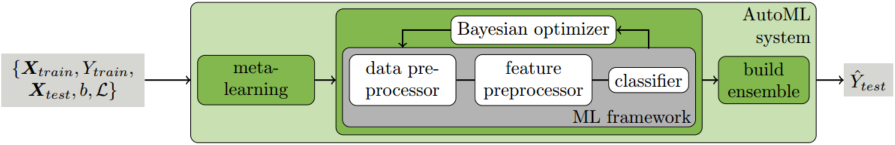
Further Combinations

- *ACPP*: Automatic construction of a parallel portfolio solver [Hoos et al. 2012]
- *AutoFolio*: configuration of an algorithm selector [Lindauer et al. 2015]
- *Sunny*: Predict an algorithm schedule for a given instance [Amadini et al. '13-'15]
- Predict the best configuration for a given instance (e.g., [Bossek et al. 2015])

Outline

Extensions to AutoWEKA's AutoML approach

- Meta-learning to warmstart Bayesian optimization
- Automated posthoc ensemble construction to combine the models Bayesian optimization evaluated



Hands-on: Auto-sklearn

In your virtual machine:

Run a linear SVM and Auto-sklearn

```
$ cd AC-Tutorial/auto-sklearn/

$ vim autosklearn-example_restricted_to_svc.py
$ python autosklearn-example_restricted_to_svc.py

$ vim autosklearn-example_with_cv.py
$ python autosklearn-example_with_cv.py
```

Auto-sklearn: Ready for Prime Time

State-of-the-art AutoML framework

- Best approach in ongoing ChaLearn AutoML challenge
- Outperformed 100s of teams of human experts

Auto-sklearn: Ready for Prime Time

State-of-the-art AutoML framework

- Best approach in ongoing ChaLearn AutoML challenge
- Outperformed 100s of teams of human experts

Trivial to use

```
import autosklearn.classification as cls
automl = cls.AutoSklearnClassifier()
automl.fit(X_train, y_train)
y_hat = automl.predict(X_test)
```

Auto-sklearn: Ready for Prime Time

State-of-the-art AutoML framework

- Best approach in ongoing ChaLearn AutoML challenge
- Outperformed 100s of teams of human experts

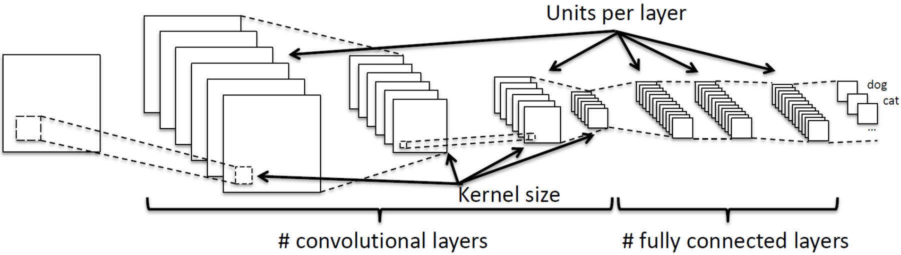
Trivial to use

```
import autosklearn.classification as cls
automl = cls.AutoSklearnClassifier()
automl.fit(X_train, y_train)
y_hat = automl.predict(X_test)
```

Availabe online

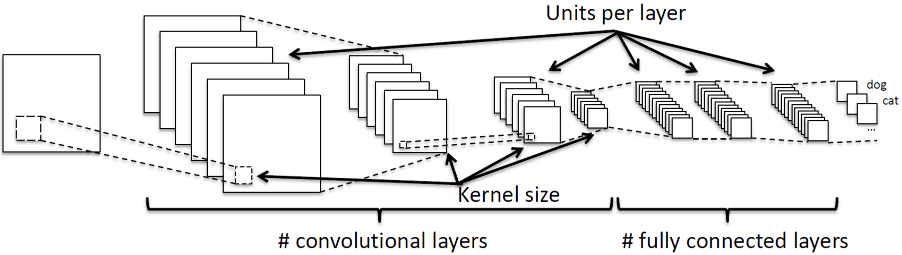
https://github.com/automl/auto-sklearn

Algorithm configuration also applies to deep learning



+ Learning rates, batch sizes, dropout rates, ...

Algorithm configuration also applies to deep learning



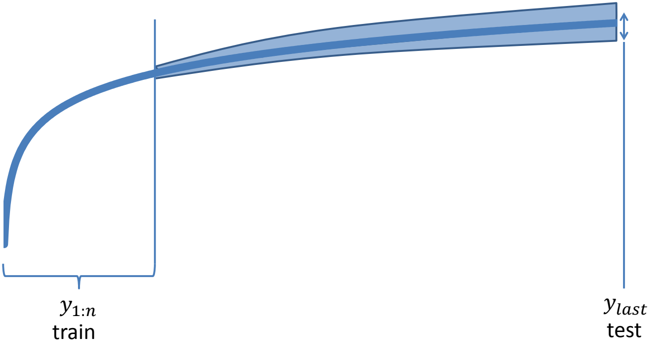
+ Learning rates, batch sizes, dropout rates, ...

State of the art for deep network hyperparameter optimization

- For few continuous hyperparameters: Bayesian optimization methods based on Gaussian processes perform best
- Many/discrete/conditional hyperparameters: SMAC performs best

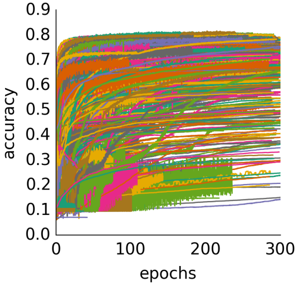
References: [Eggensperger et al, BayesOpt 2013], [Domhan et al, IJCAI 2015]

Time-permitting: extrapolating learning curves of stochastic gradient descent performance to later time steps

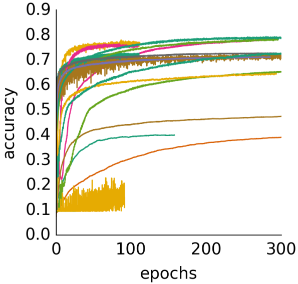


- Observe initial part of learning curve
- Probabilisticially predict remainder of learning curve
- Reference: [Domhan et al, IJCAI 2015]

Examples of SGD learning curves in deep learning



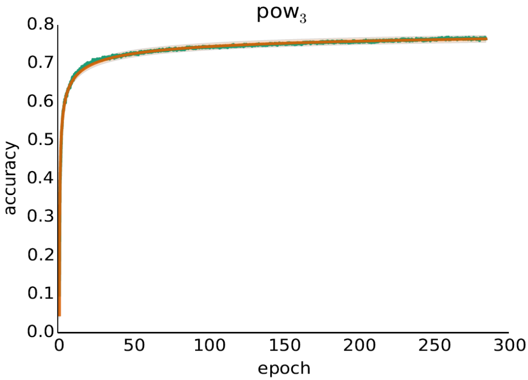
Learning curves for 1000 hyperparameter settings



Learning curves for a random sample of these

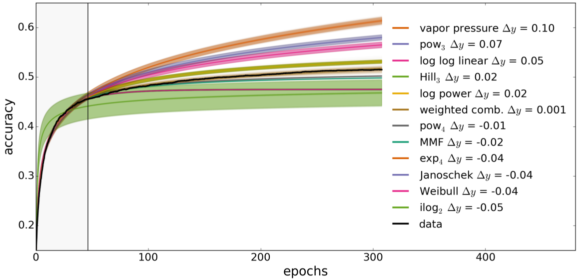
Idea for extrapolation: fitting a parametric function

- Increasing, saturating functions
- E.g., pow_3 function with 3 parameters c, a, α
  - $y = \text{pow}_3(x \mid c, a, \alpha) = c - ax^{-\alpha}$ (where x=epoch, y=accuracy)
- Fitting those parameters: optimize fit using gradient-based methods



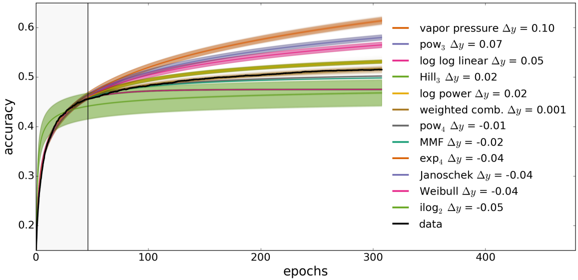
Advanced variants of extrapolation

- No single parametric family performs best
- We fit convex combinations of 11 different parametric families



Advanced variants of extrapolation

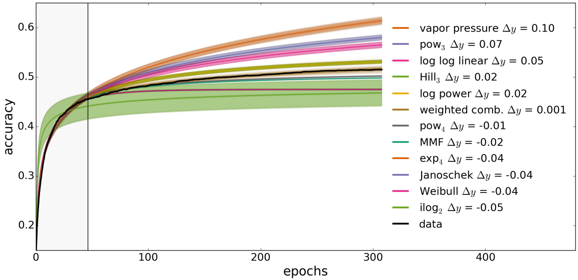
- No single parametric family performs best
- We fit convex combinations of 11 different parametric families



- In practice, we'd also like uncertainty estimates
  - Optimizing parameters yields a single fit
  - Sampling parameters yields a distribution of fits

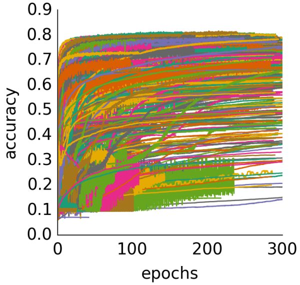
Advanced variants of extrapolation

- No single parametric family performs best
- We fit convex combinations of 11 different parametric families

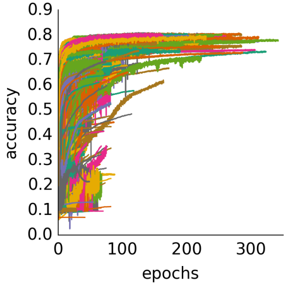


- In practice, we'd also like uncertainty estimates
  - Optimizing parameters yields a single fit
  - Sampling parameters yields a distribution of fits
    - Sample parameters according to their posterior probability (using Markov Chain Monte Carlo; beyond scope of this course)

Qualitative results



Learning curves for 1000 hyperparameter settings



Bad curves are terminated early

Quantitative results

2-fold speedup of DNN structure & hyperparameter optimization

- For several network architectures, including state-of-the-art
- For several optimizers (SMAC, TPE, random search)
- New state-of-the-art for CIFAR without data augmentation

Summary

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

Take-away messages

Algorithm configuration is very versatile

- Improves results, increases productivity
- Enables automated machine learning

What you need to use algorithm configuration

- An algorithm with exposed parameters
- An instance distribution/set
- A performance metric you care about

You know about field X. What can you contribute?

- Case study applying algorithm configuration to X
- Construct better features for X
- Build a system: Auto-X (like Auto-sklearn)