Sequential Model-based Optimization for General Algorithm Configuration

Frank Hutter, Holger Hoos, Kevin Leyton-Brown

University of British Columbia

LION 5, Rome
January 18, 2011

Motivation

Most optimization algorithms have parameters

- E.g. IBM ILOG CPLEX:
  - Preprocessing, balance of branching vs. cutting, type of cuts, etc.
  - 76 parameters, mostly categorical

Use machine learning to predict algorithm runtime, given

- parameter configuration used
- characteristics of the instance being solved

Use these predictions for general algorithm configuration

- E.g. optimize CPLEX parameters for given benchmark set
- Two new methods for general algorithm configuration

Related work

General algorithm configuration

- Racing algorithms, F-Race [Birattari et al., GECCO‘02-present]
- Iterated Local Search, ParamILS [Hutter et al., AAAI’07 & JAIR ‘09]
- Genetic algorithms, GGA [Ansotegui et al, CP’09]

Model-based optimization of algorithm parameters

- Sequential Parameter Optimization [Bartz-Beielstein et al., '05-present]
  - SPO toolbox: interactive tools for parameter optimization
- Our own previous work
  - SPO^+: fully automated & more robust [Hutter et al., GECCO’09]
  - TB-SPO: reduced computational overheads [Hutter et al., LION 2010]
- Here: extend to general algorithm configuration
  - Sets of problem instances
  - Many, categorical parameters

Outline

1. ROAR

2. SMAC

3. Experimental Evaluation

A key component of ROAR and SMAC

Compare a configuration θ vs. the current incumbent, θ^*:

- Racing approach:
  - Few runs for poor θ
  - Many runs for good θ
    - once confident enough: update $\theta^* \leftarrow \theta$
- Agressively rejects poor configurations θ
  - Very often after a single run

ROAR: a simple method for algorithm configuration

Main ROAR loop:

- Select a configuration θ uniformly at *random*
- Compare θ to current θ^* (*online*, one θ at a time)
  - Using *aggressive racing* from previous slide

Random

Online

Aggressive

Racing

Outline

1. ROAR

2. SMAC
   Sequential Model-based
   Algorithm Configuration

3. Experimental Evaluation

SMAC in a Nutshell

Construct a model to predict algorithm performance

- Supervised machine learning
- Gaussian processes (aka kriging)
- Random forest model $f : \Theta \to \mathbb{R}$

Use that model to select promising configurations

Compare each selected configuration to incumbent

- Using same aggressive racing as ROAR

Fitting a Regression Tree to Data: Example

param 1	param 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

$param_3 \in \{red\}$

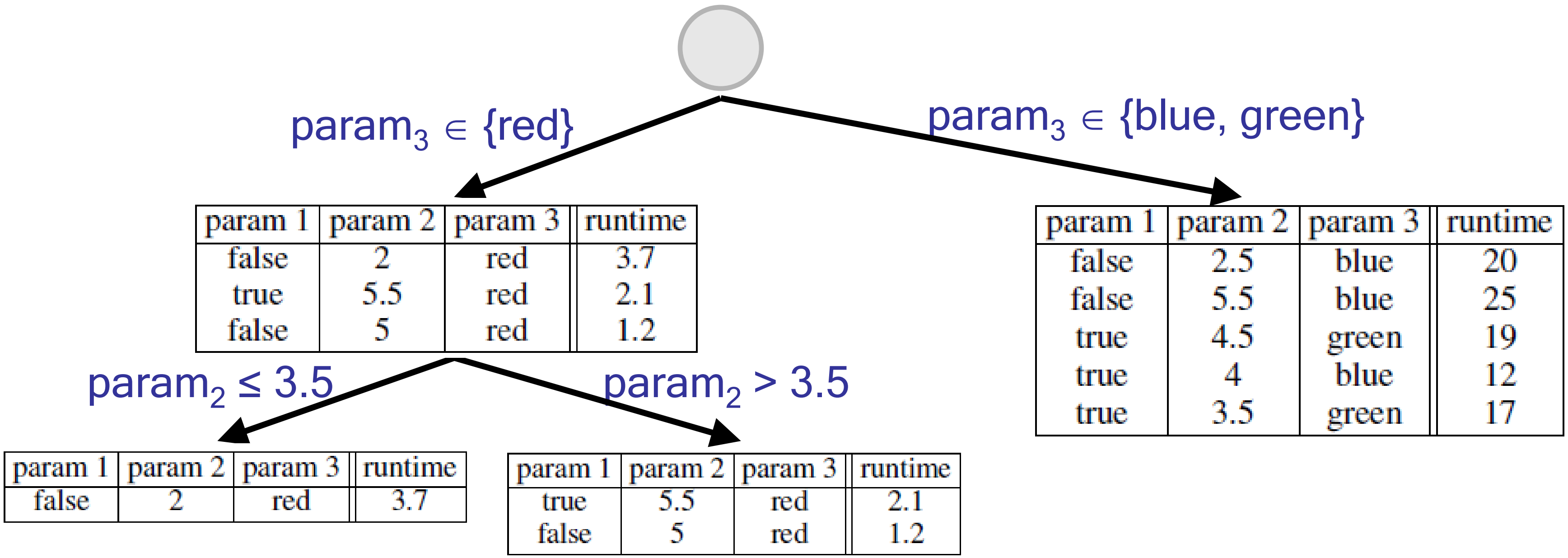
param 1	param 2	param 3	runtime
false	2	red	3.7
true	5.5	red	2.1
false	5	red	1.2

$param_3 \in \{blue, green\}$

param 1	param 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 to Data: Example

- In each internal node: only store split criterion used



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

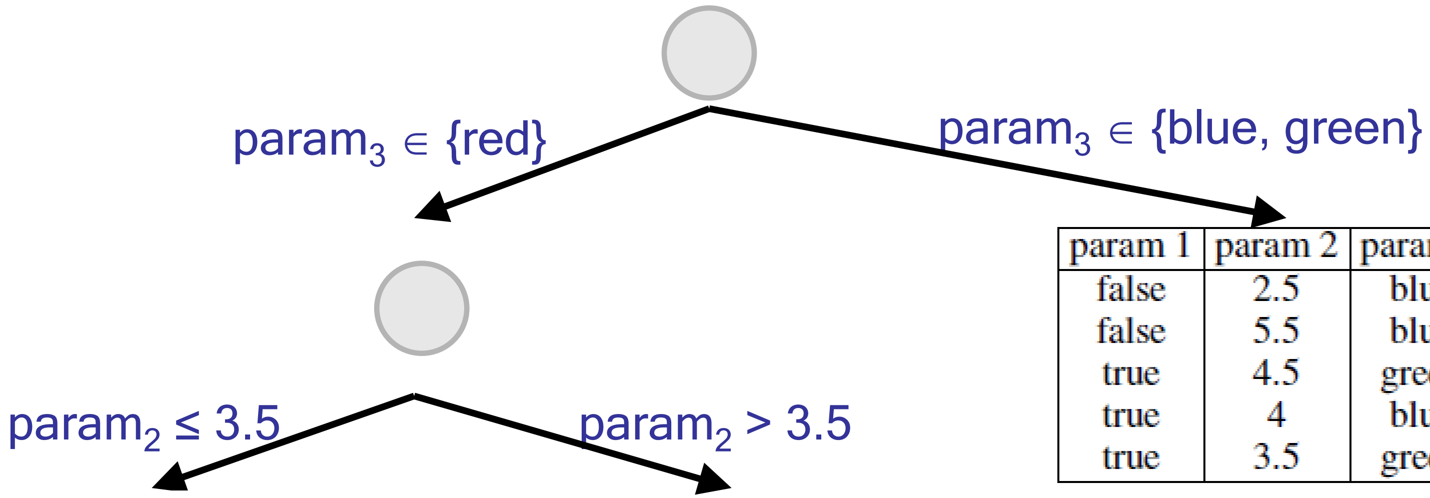
param 1	param 2	param 3	runtime
false	2.5	blue	20
false	5.5	blue	25
true	4.5	green	19
true	4	blue	12
true	3.5	green	17

param 1	param 2	param 3	runtime
false	2	red	3.7

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

Fitting a Regression Tree to Data: Example

- In each internal node: only store split criterion used



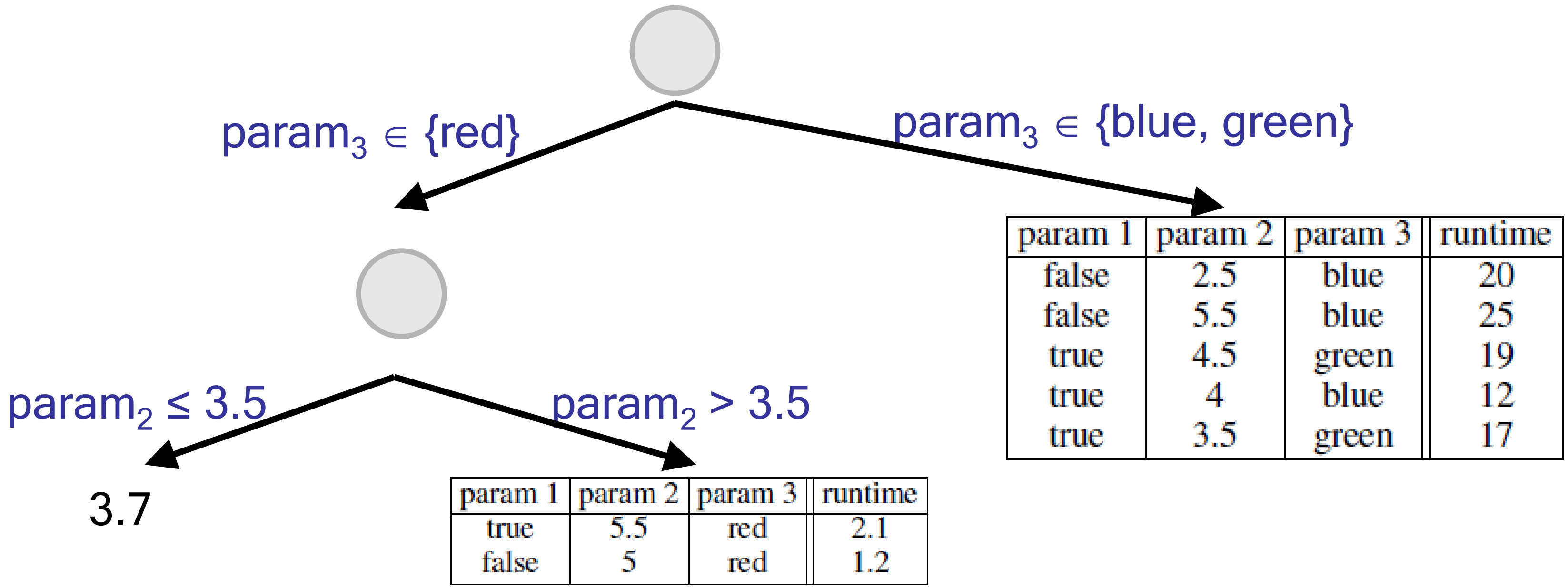
param 1	param 2	param 3	runtime
false	2	red	3.7

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

param 1	param 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 to Data: Example

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

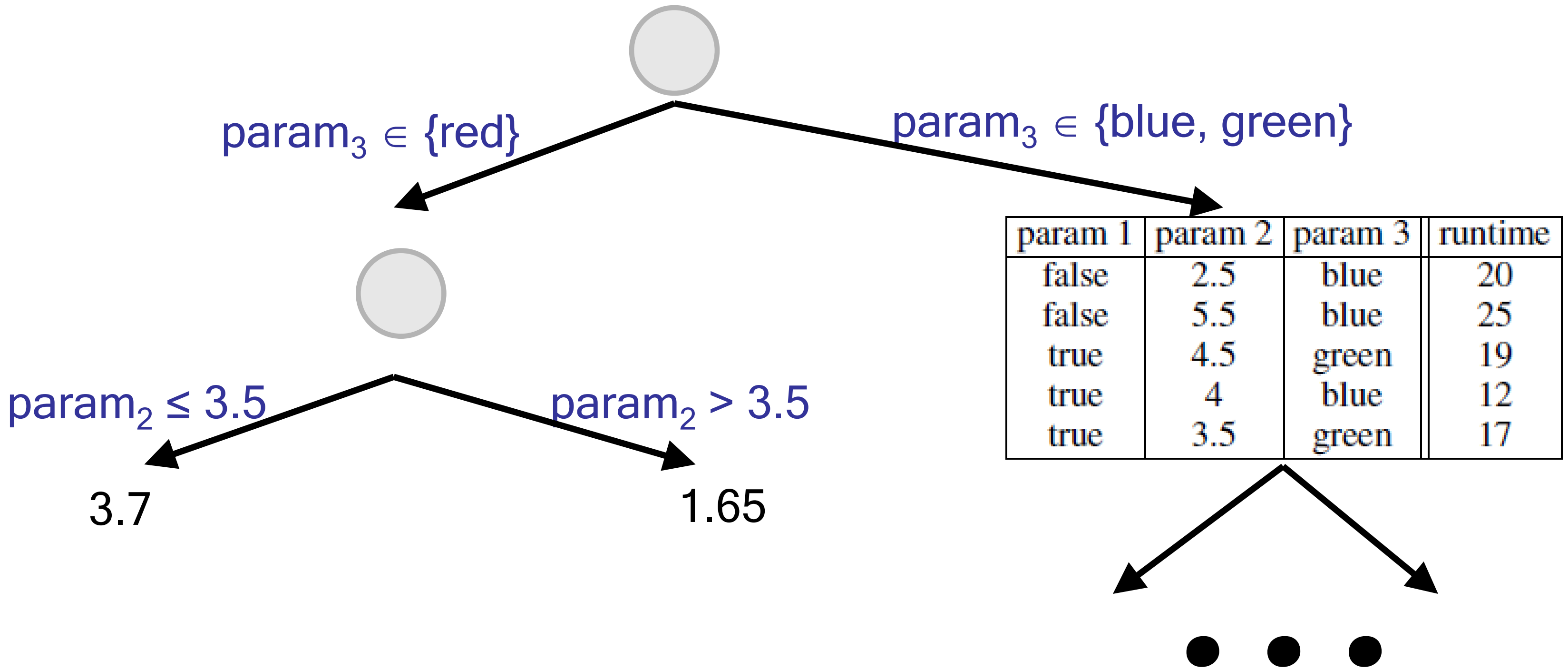


param 1	param 2	param 3	runtime
true	5.5	red	2.1
false	5	red	1.2

param 1	param 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 to Data: Example

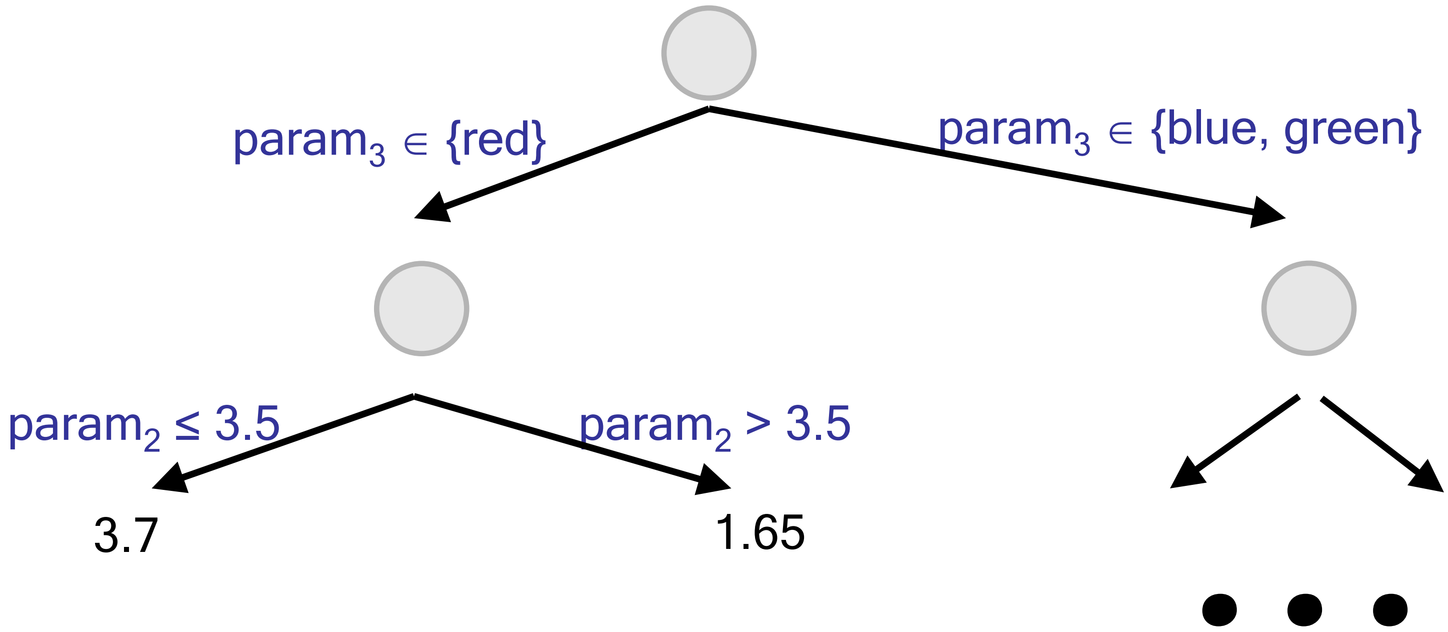
- In each internal node: only store split criterion used
- In each leaf: store mean of runtimes



param 1	param 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 to Data: Example

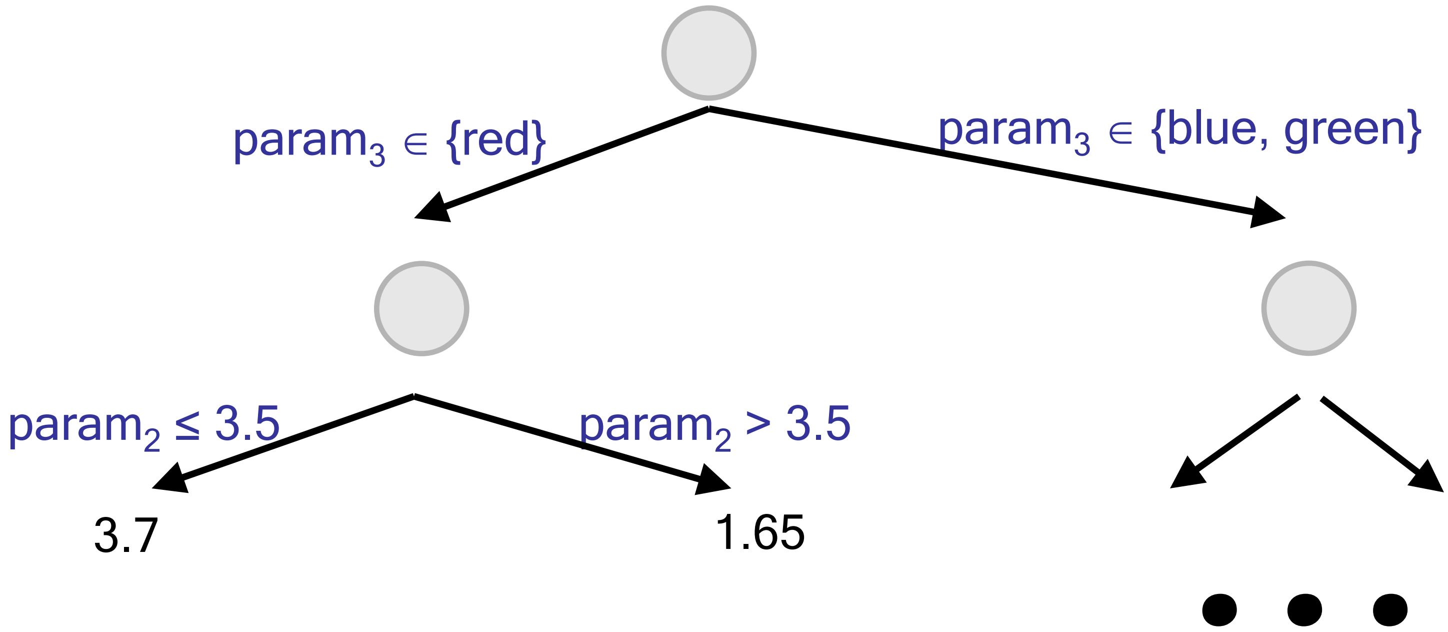
- In each internal node: only store split criterion used
- In each leaf: store mean of runtimes



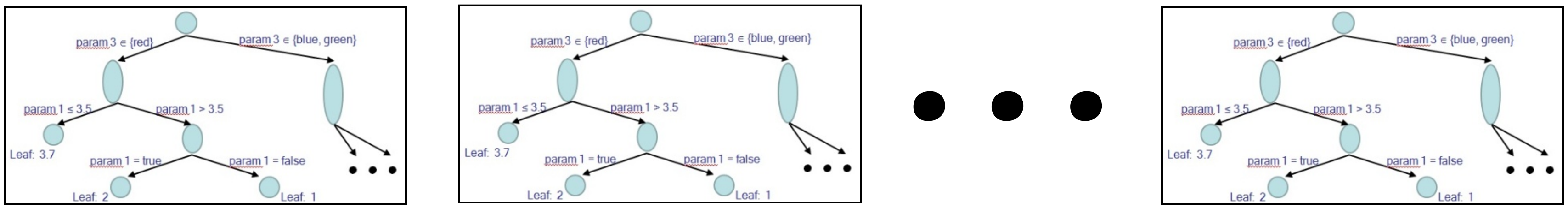
Predictions for a new parameter configuration

E.g. θ_{n+1} = (true, 4.7, red)

- Walk down tree, return mean runtime stored in leaf $\Rightarrow$ 1.65



Random Forests: sets of regression trees



Training

- Subsample the data T times (with repetitions)
- For each subsample, fit a regression tree

Prediction

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

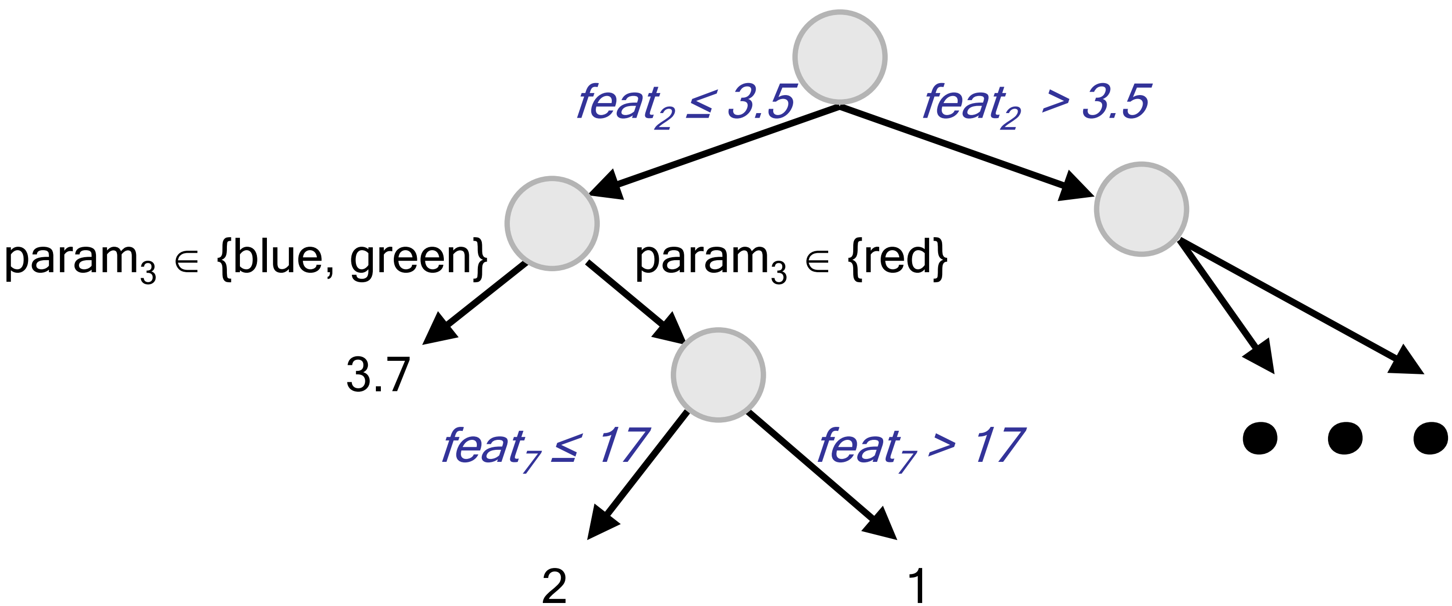
Predictions For Different Instances

Runtime data now also includes instance features:

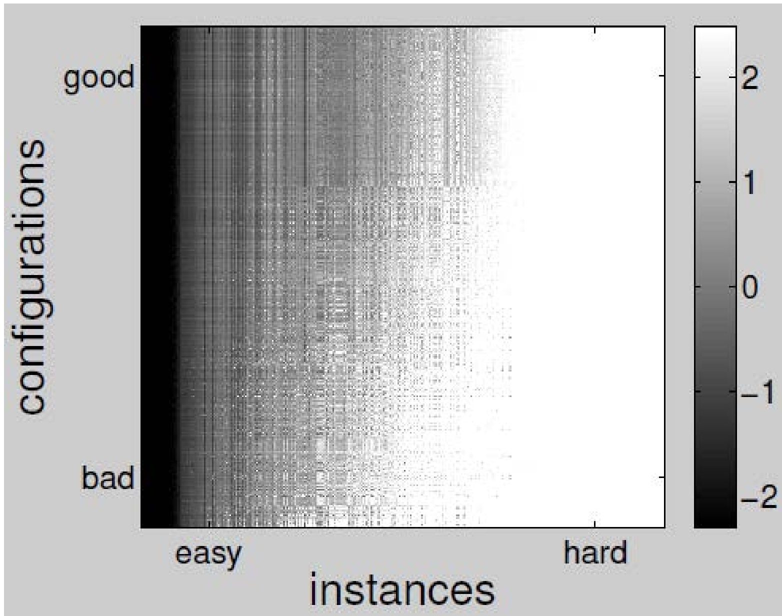
- Configuration θ_i , runtime r_i, and *instance features* $x_i = (x_{i,1}, \ldots, x_{i,m})$

Fit a model g: $\Theta \times \mathbb{R}^m \rightarrow \mathbb{R}$

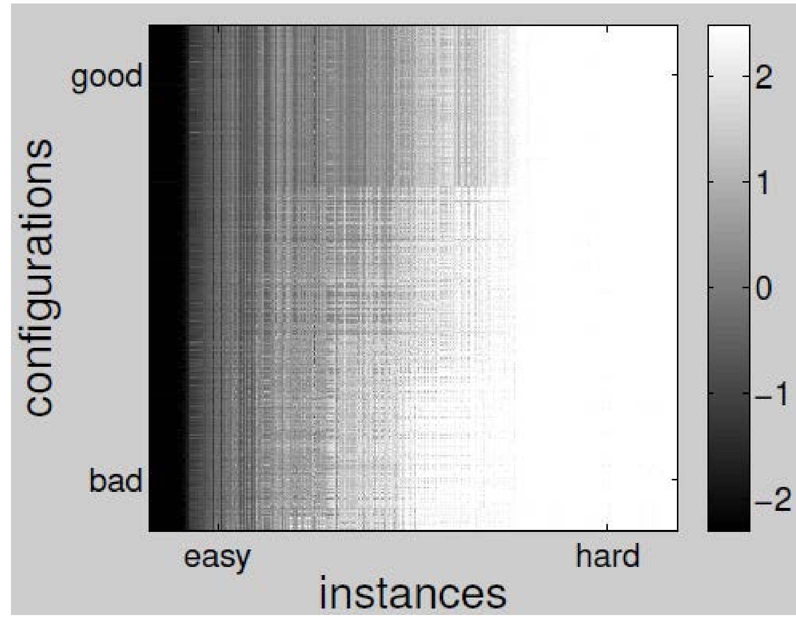
- Predict runtime for previously unseen combinations (θ_{n+1} ,x_{n+1})



Visualization of Runtime Across Instances and Parameter Configurations



True $\log_{10}$ runtime



Predicted $\log_{10}$ runtime

Darker is faster

Performance of configuration θ across instances:

- Average of θ's predicted row

Summary of SMAC Approach

Construct model to predict algorithm performance

- Random forest model $g : \mathbf{\Theta} \times \mathbb{R}^m \rightarrow \mathbb{R}$
- Marginal predictions $f : \mathbf{\Theta} \rightarrow \mathbb{R}$

Use that model to select promising configurations

- Standard "expected improvement (EI)" criterion
  - combines predicted mean and uncertainty
- Find configuration with highest EI: optimization by local search

Compare each selected configuration to incumbent θ^*

- Using same aggressive racing as ROAR
- Save all run data $\rightarrow$ use to construct models in next iteration

Outline

1. ROAR

2. SMAC

3. Experimental Evaluation

Experimental Evaluation: Setup

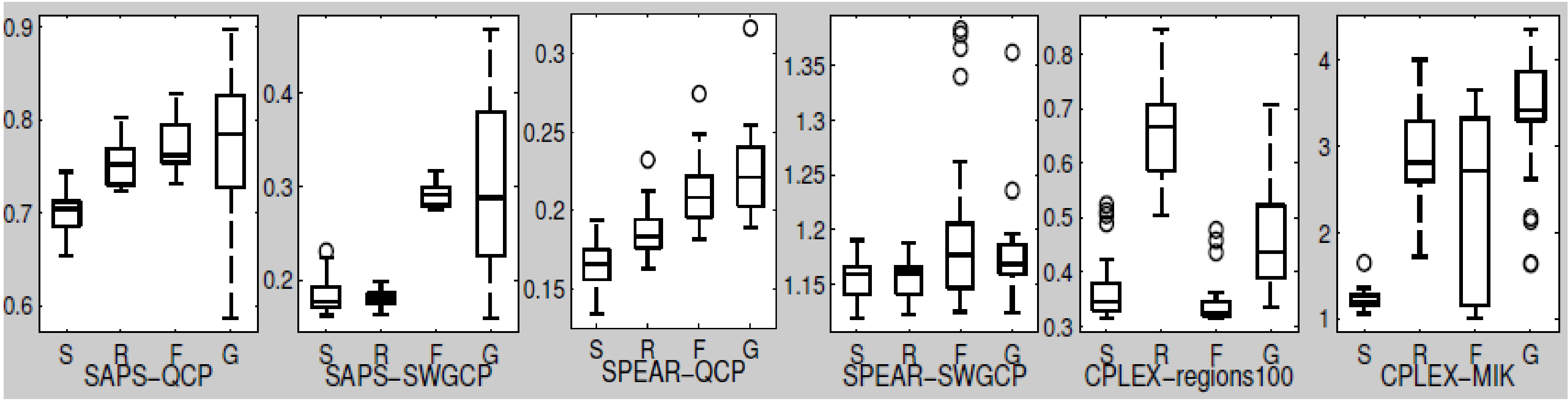
Compared SMAC, ROAR, FocusedILS, and GGA

- On 17 small configuration scenarios:
  - Local search and tree search SAT solvers SAPS and SPEAR
  - Leading commercial MIP solver CPLEX
- For each configurator and each scenario
  - 25 configuration runs with 5-hour time budget each
  - Evaluate final configuration of each run on independent test set

Over a year of CPU time

- Will be available as a reproducable experiment package in HAL
- HAL: see Chris Nell's talk tomorrow @ 17:20

Experimental Evaluation: Results



y-axis: test performance (runtime, smaller is better)

S=SMAC, R=ROAR, F=FocusedILS, G=GGA

- Improvement (means over 25 runs)
  - 0.93× – 2.25× (vs FocusedILS), 1.01× – 2.76× (vs GGA)
- Significant (never significantly worse)
  - 11/17 (vs FocusedILS), 13/17 (vs GGA)
- But: SMAC's performance depends on instance features

Conclusion

Generalized model-based parameter optimization:

- Sets of benchmark instances
- Many, categorical parameters

Two new procedures for general algorithm configuration

- Random Online Aggressive Racing (ROAR)
  - Simple yet surprisingly effective
- Sequential Model-based Algorithm Configuration (SMAC)
  - State-of-the-art configuration procedure
  - Improvements over FocusedILS and GGA

Future Work

Improve algorithm configuration further

- Cut off poor runs early (like adaptive capping in ParamILS)
  - Handle “censored” data in the models
- Combine model-free and model-based methods

Use SMAC’s models to gain scientific insights

- Importance of each parameter
- Interaction of parameters and instance features

Use SMAC’s models for per-instance algorithm configuration

- Compute instance features
- Pick configuration predicted to be best