Efficient algorithm design via automated algorithm selection and configuration

Alexander Tornede & Marius Lindauer

Euro PhD School Data Science Meets Combinatorial Optimisation

Story Line This Session

- What do we optimize?
  - Parameters vs. Hyperparameters
  - Challenges for AutoML
- How do we optimize it?
  - Grid Search
  - Random Search
- How do we optimize it efficiently?
  - Bayesian Optimization
- How do we optimize across many problem instances?
  - Algorithm Configuration
  - Aggressive Racing
  - Other Racing Strategies
- Outlook and Software Packages

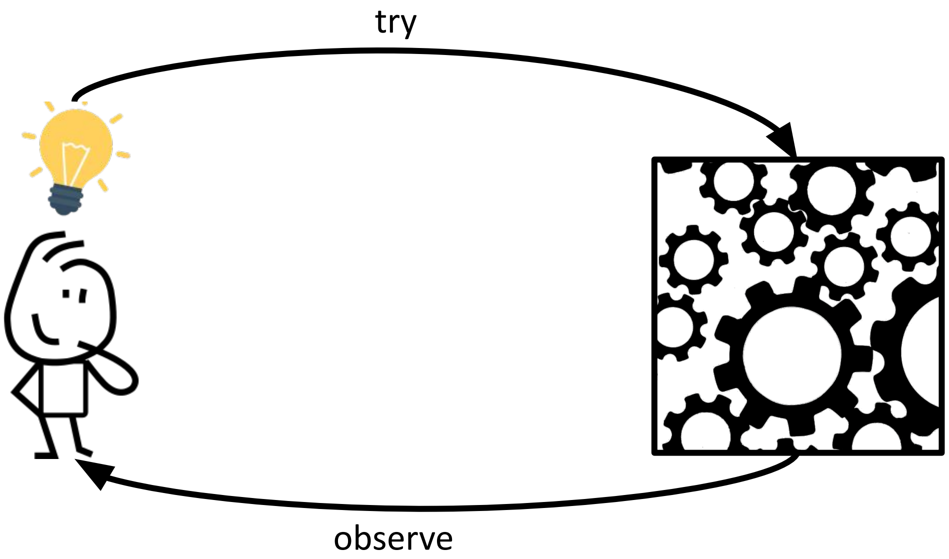
Note: This lecture is partially based on the free online lecture "Automated Machine Learning" at https://learn.ki-campus.org/courses/automl-luh2021

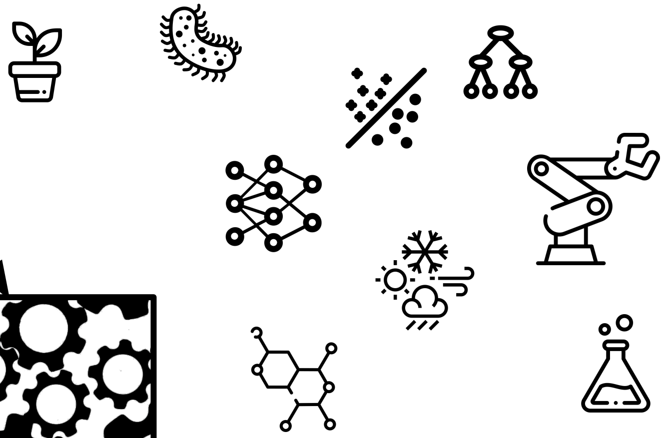
- Basics of HPO
- Bayesian Optimization for HPO

What do we optimize?

>> Here's my algorithm and data, what should I do?

Sequential Experimentation

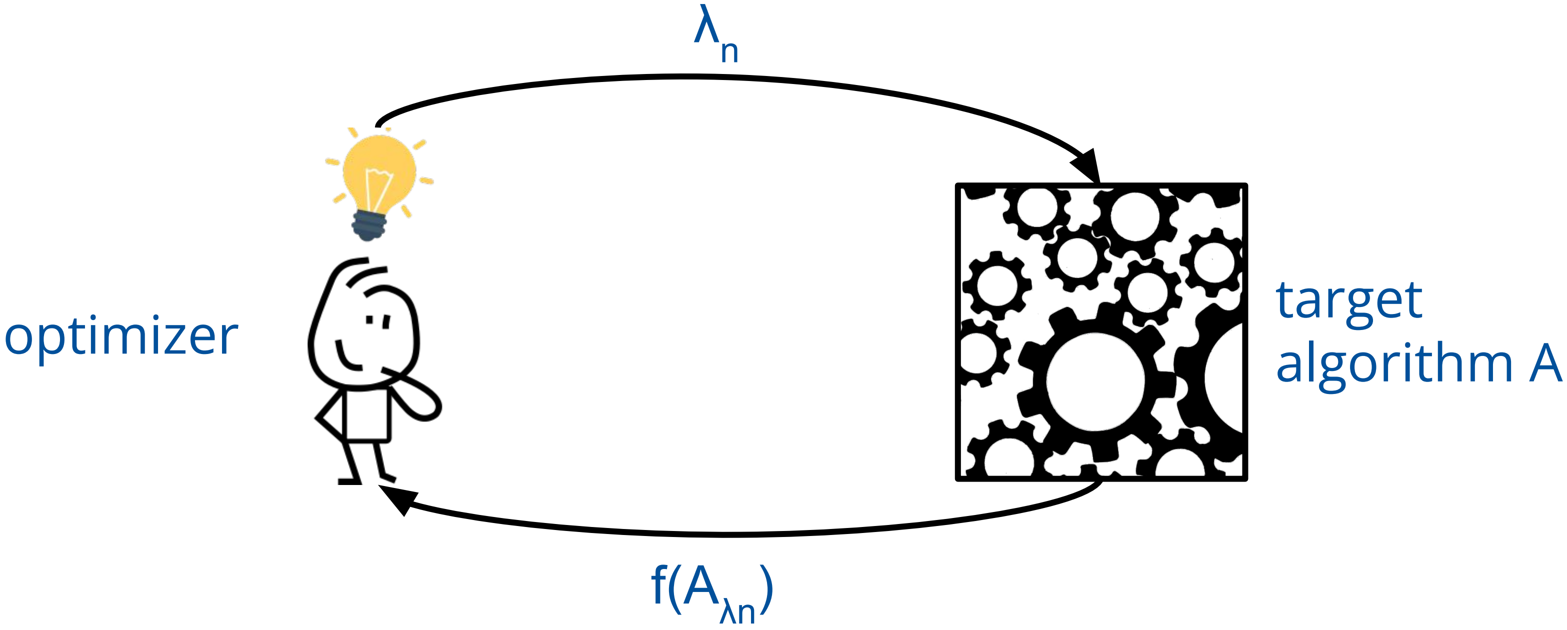




- Algorithm
- Neural Architecture
- Network Training
- Data Science Pipeline
- Simulator
- Robot

Hyperparameter Optimization

Goal: Find the best performing configuration: $\boxed{\lambda^* \in \arg\min_{\lambda \in \Lambda} f(\mathcal{A}_\lambda)}$



Example: Machine Learning

- **Given a dataset**, we want to train a neural network
- We need to choose a **learning rate and architecture**
- The “learner” takes the input data, and returns a fitted network

→ We are interested in **generalization error**!

→ We need to look at how our trained model **performs on “unseen” data**

→ We evaluate different settings and select the one that **performs best w.r.t generalization error**.

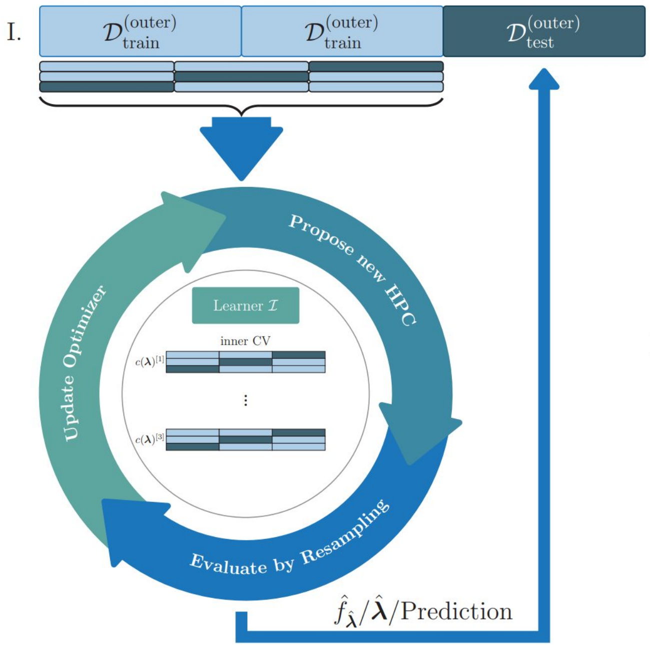


Image: Bischl et al. 2023

Hyperparameters and Parameters

Model parameters can be optimized during training and are the output of the training. Examples:

- Splits of a Decision Tree
- Weights of a Neural Network
- Coefficients of a linear model

Hyperparameters need to be set manually before training. They control the flexibility, structure and complexity of the model and training procedure. Examples:

- Max. depth of a Decision Tree
- Number of layers of a Neural Network
- K for K-Nearest Neighbours

Types of Algorithm Parameters (Hyperparameters)

Real-valued

- Learning rate for SGD to train NNs
- Bandwidth of kernel density estimates in Naive Bayes

Integer

- #Neurons in a layer of a NN
- maximum depth of a Decision Tree

Categorical

- Training Algorithm for NNs
- Split criterion for Decision Trees

Can also be on a log-scale

+ Hyperparameters can be **hierarchically dependent** on each other

Why is Hyperparameter Optimization Challenging?

Goal: Find the best performing configuration:

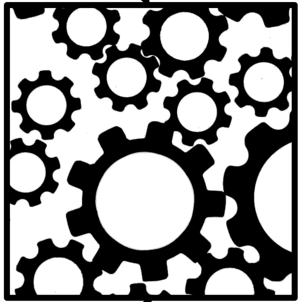
$$\boxed{\lambda^* \in \arg\min_{\lambda \in \Lambda} f(\mathcal{A}_\lambda)}$$

complex search space

No gradients
No prior knowledge

λ_n

optimizer



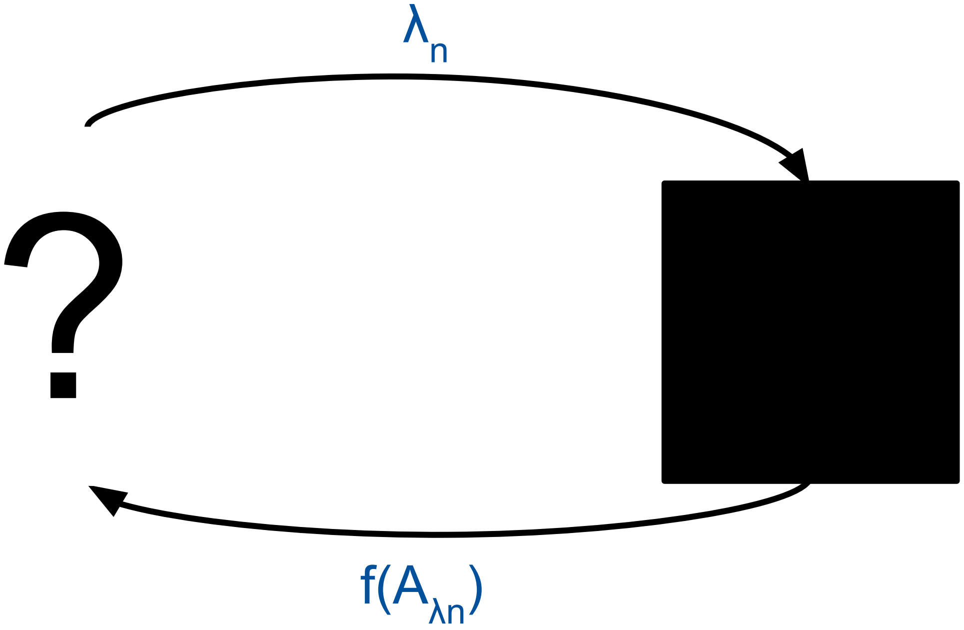
target algorithm A

$f(A_{\lambda n})$

noisy
expensive-to-evaluate

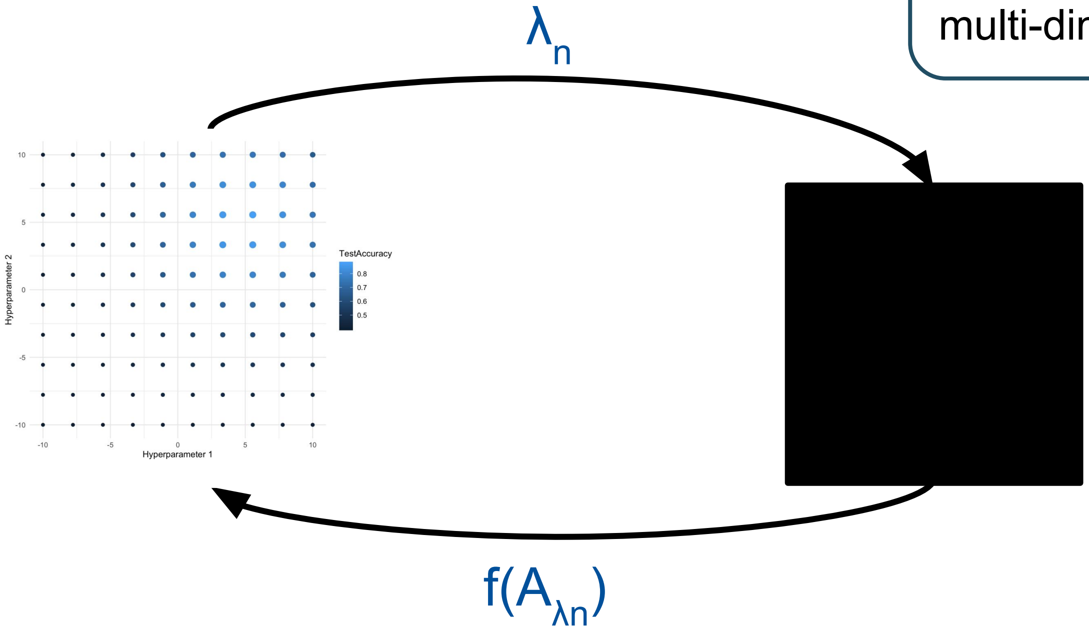
How do we optimize it?

Black-Box Optimization Problem

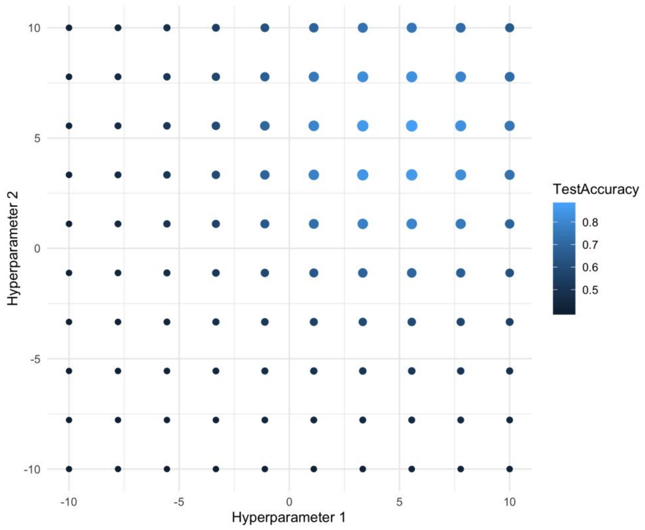


Option 1: Grid Search

Popular technique: Evaluates all combinations on a pre-defined multi-dimensional grid



Option 1: Grid Search II



Advantages

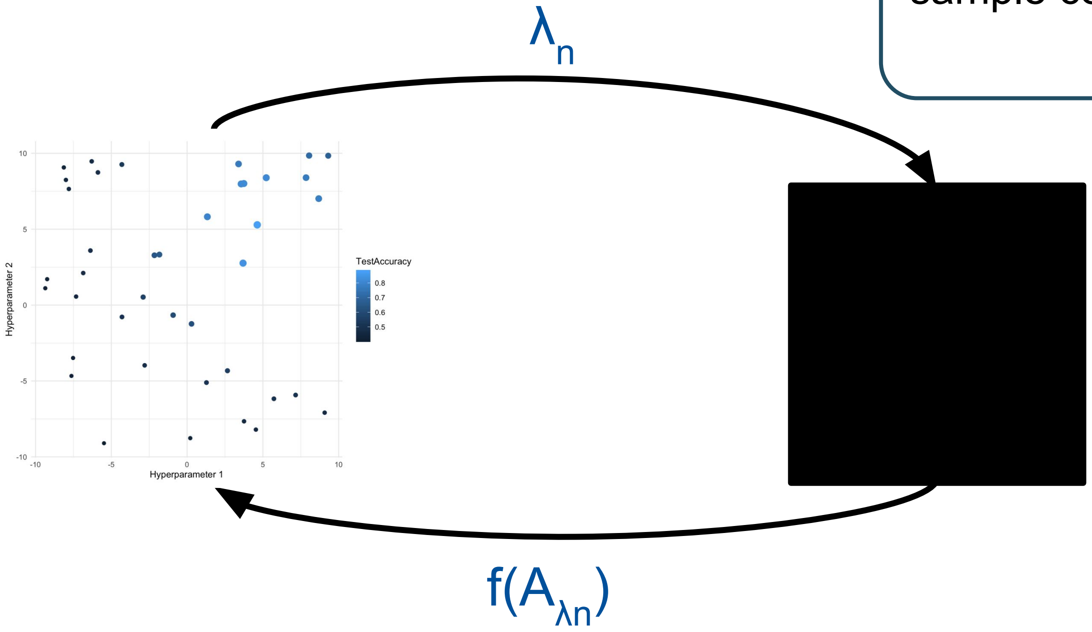
- Very easy to implement
- Very easy to parallelize
- Can handle all types of hyperparameters

Disadvantages

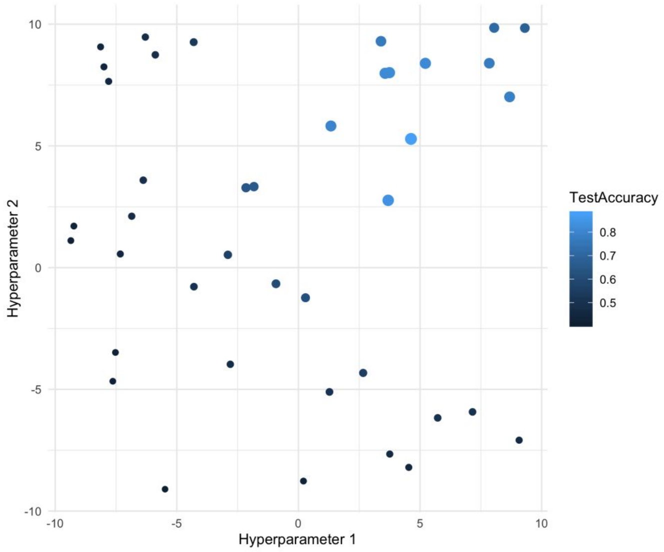
- Scales badly with #dimensions
- Inefficient: Searches irrelevant areas
- Requires to manual define discretization
- All grid points need to be evaluated

Option 2: Random Search

Variation of Grid Search: Uniformly sample configurations at random



Option 2: Random Search II



Advantages

- Very easy to implement
- Very easy to parallelize
- Can handle all types of hyperparameters
- No discretization required
- Anytime algorithm: Can be stopped and continued based on the available budget and performance goal.

Disadvantages

- Scales badly with #dimensions
- Inefficient: Searches irrelevant areas

Grid Search vs. Random Search

With a **budget** of T iterations:

Grid Search evaluates only $T^{\frac{1}{d}}$ unique values per dimension

Random Search evaluates (most likely) T different values per dimension

→ Grid search can be disadvantageous if some hyperparameters have little of no impact on the performance [Bergstra et al. 2012]

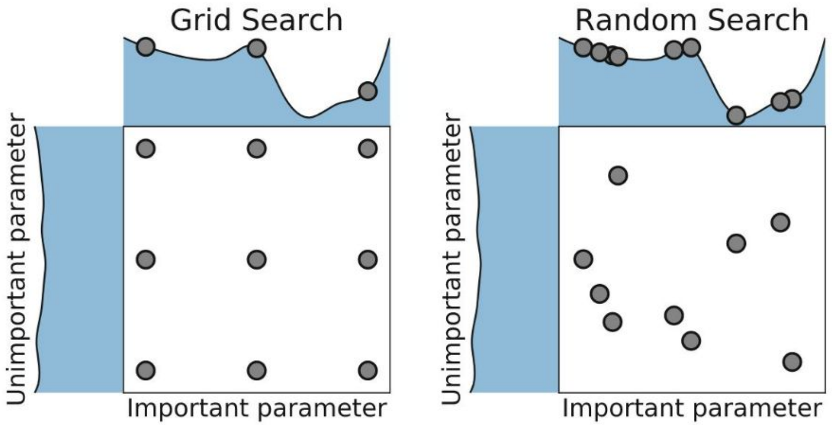


Image source: [Hutter et al. 2019]

Questions?

Kahoot Quiz I

How do we optimize it efficiently?

Model-based Optimization

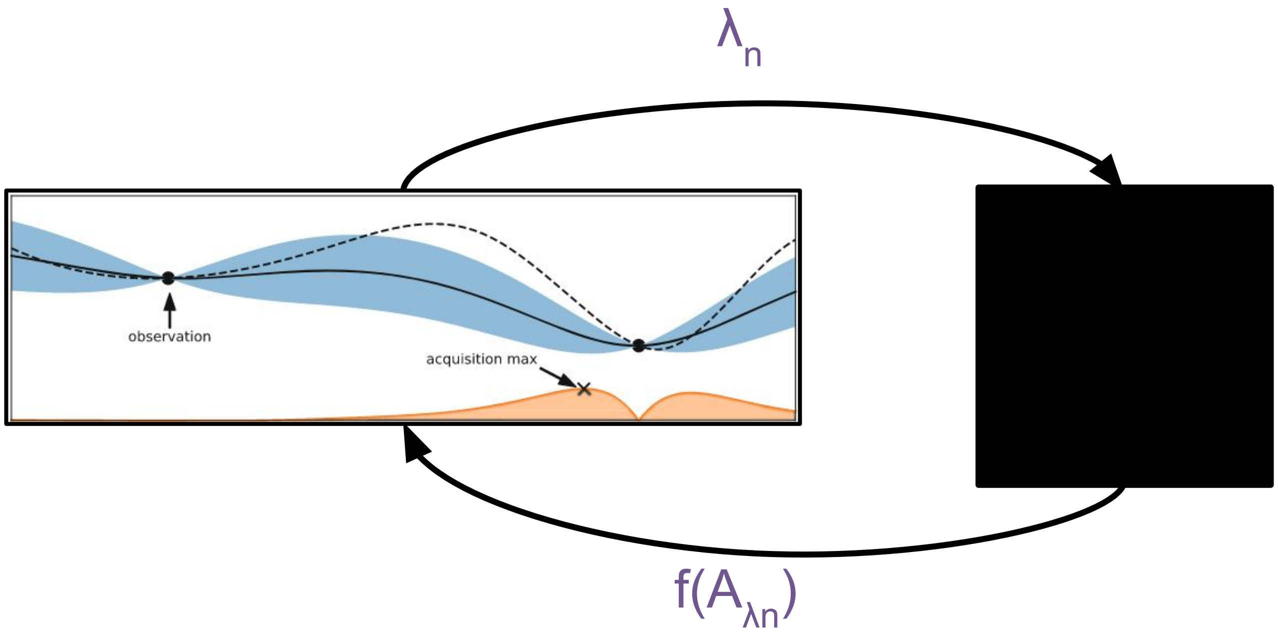
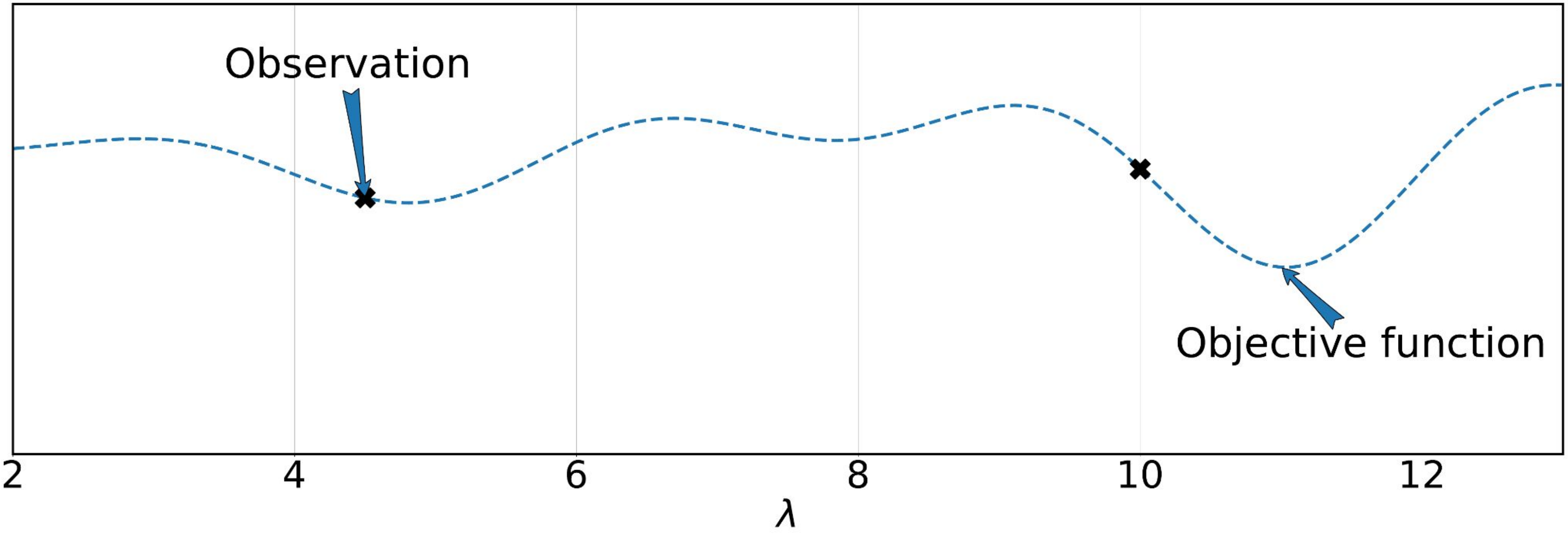
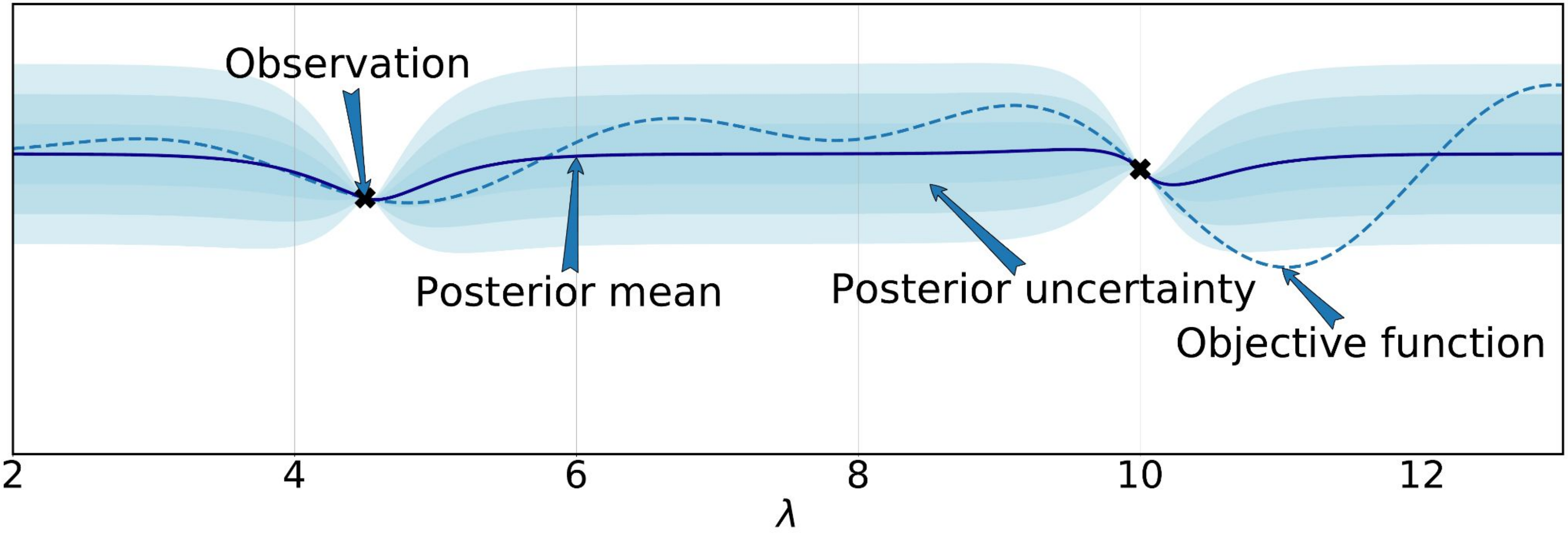


Photo by Wilhelm Gunkel on Unsplash
Image by Feurer, Hutter: Hyperparameter Optimization.
In: Automated Machine Learning

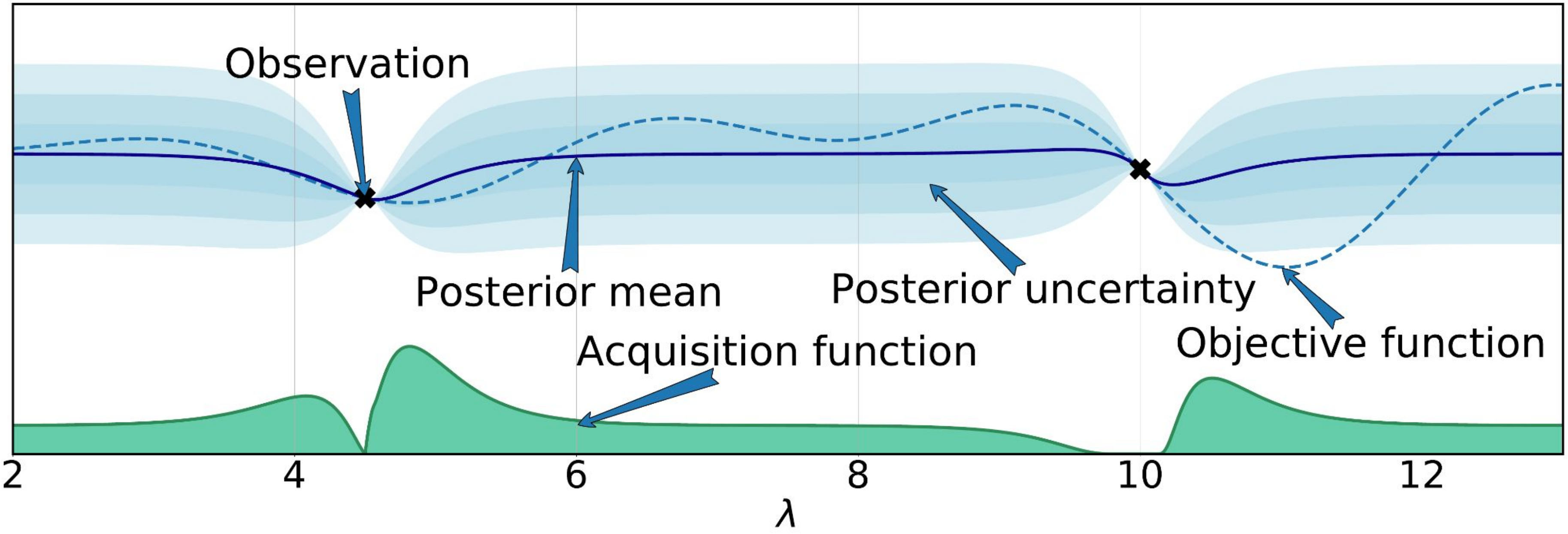
Bayesian Optimization in a Nutshell



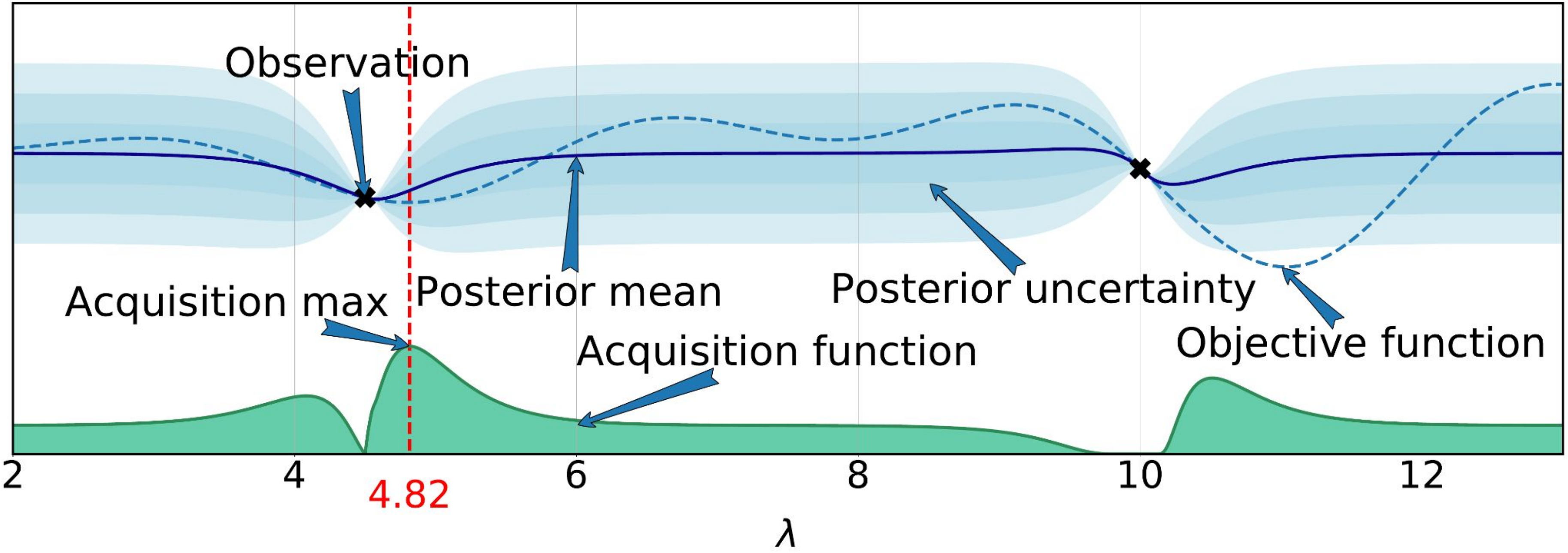
Bayesian Optimization in a Nutshell



Bayesian Optimization in a Nutshell



Bayesian Optimization in a Nutshell



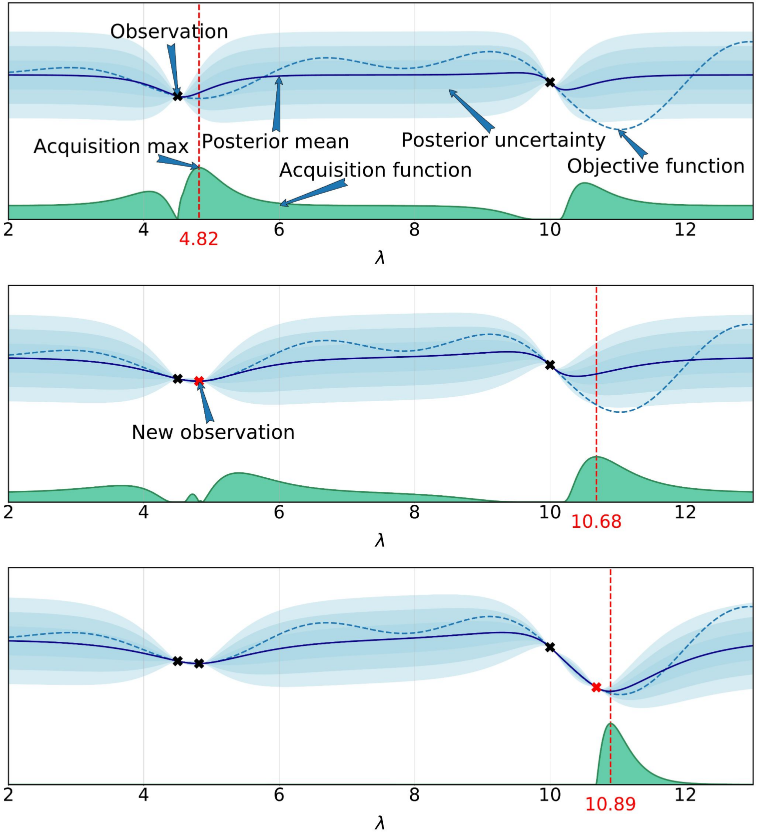
Bayesian Optimization in a Nutshell

General approach

- Fit a probabilistic model to the collected function samples $\langle \boldsymbol{\lambda}, c(\boldsymbol{\lambda}) \rangle$
- Use the model to guide optimization, trading off exploration *vs* exploitation

Popular approach in the statistics literature since Mockus et al. [1978]

- Efficient in #function evaluations
- Works when objective is nonconvex, noisy, has unknown derivatives, etc.
- Recent convergence results [Srinivas et al. 2009; Bull et al. 2011; de Freitas et al. 2012; Kawaguchi et al. 2015]



Bayesian Optimization: Pseudocode

```
BO loop
Require: Search space Λ, cost function c, acquisition function u, predictive model ĉ, maximal number of function evaluations T
Result  : Best configuration λ̂ (according to D or ĉ)
Initialize data D^(0) with initial observations
for t = 1 to T do
    Fit predictive model ĉ^(t) on D^(t−1)
    Select next query point: λ^(t) ∈ arg max_{λ∈Λ} u(λ; D^(t−1), ĉ^(t))
    Query c(λ^(t))
    Update data: D^(t) ← D^(t−1) ∪ {⟨λ^(t), c(λ^(t))⟩}
```

BO loop

Require: Search space $\boldsymbol{\Lambda}$, cost function c, acquisition function u, predictive model $\hat{c}$, maximal number of function evaluations T

Result : Best configuration $\hat{\boldsymbol{\lambda}}$ (according to $\mathcal{D}$ or $\hat{c}$)

Initialize data $\mathcal{D}^{(0)}$ with initial observations

for $t = 1$ **to** T **do**

- Fit predictive model $\hat{c}^{(t)}$ on $\mathcal{D}^{(t-1)}$
- Select next query point: $\boldsymbol{\lambda}^{(t)} \in \arg\max_{\boldsymbol{\lambda}\in\boldsymbol{\Lambda}} u(\boldsymbol{\lambda}; \mathcal{D}^{(t-1)}, \hat{c}^{(t)})$
- Query $c(\boldsymbol{\lambda}^{(t)})$
- Update data: $\mathcal{D}^{(t)} \leftarrow \mathcal{D}^{(t-1)} \cup \{\langle \boldsymbol{\lambda}^{(t)}, c(\boldsymbol{\lambda}^{(t)})\rangle\}$

Why is it called Bayesian Optimization?

- Bayesian optimization uses Bayes' theorem:

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)} \propto P(B|A) \times P(A)$$

- Bayesian optimization uses this to compute a posterior over functions:

$$P(f|\mathcal{D}_{1:t}) \propto P(\mathcal{D}_{1:t}|f) \times P(f), \qquad \text{where } \mathcal{D}_{1:t} = \{\boldsymbol{\lambda}_{1:t}, c(\boldsymbol{\lambda}_{1:t})\}$$

Meaning of the individual terms:

- $P(f)$ is the prior over functions, which represents our belief about the space of possible objective functions before we see any data
- $\mathcal{D}_{1:t}$ is the data (or observations, evidence)
- $P(\mathcal{D}_{1:t}|f)$ is the likelihood of the data given a function
- $P(f|\mathcal{D}_{1:t})$ is the posterior probability over functions given the data

Bayesian Optimization: Pros and Cons

Advantages

- Sample efficient
- Can handle noise
- Priors can be incorporated
- Does not require gradients
- Theoretical guarantees

Many extensions available:
Multi-Objective | Multi-Fidelity |
Parallelization | Warmstarting | etc.

Disadvantages

- Overhead because of model training
- Crucially relies on robust surrogate model
- Has quite a few design decisions

Main Ingredient I: The Acquisition Function

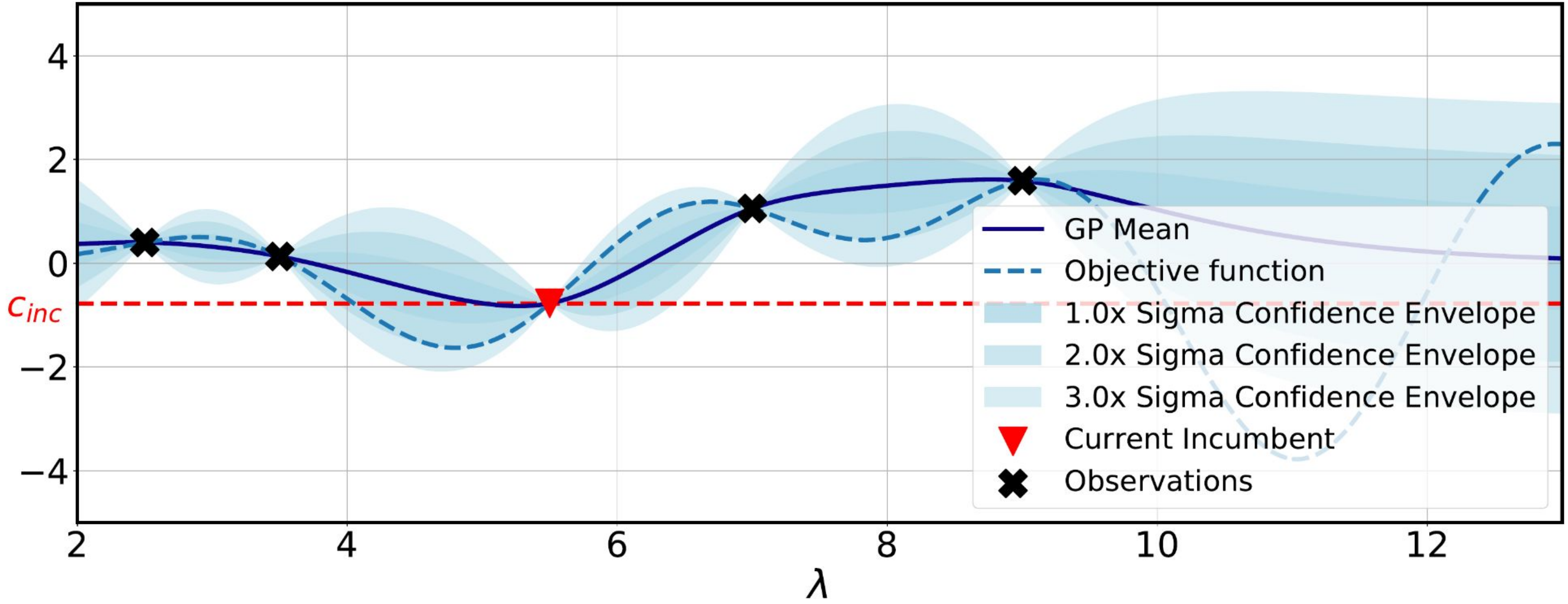
The acquisition function

- decides which configuration to evaluate next
- judges the **utility** (or **usefulness**) of evaluating a configuration (based on the surrogate model)

→ It needs to trade-off **exploration and exploitation**

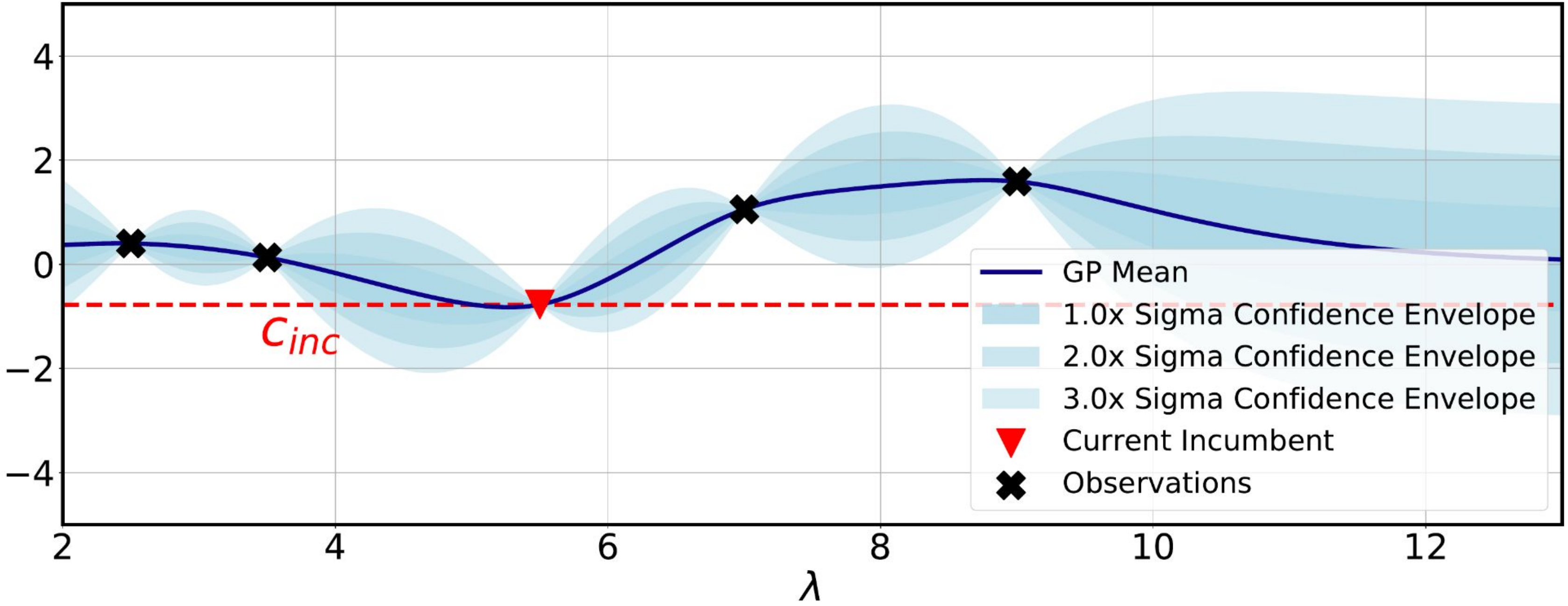
- Just picking the configuration with the lowest prediction would be too greedy
- It needs to consider the uncertainty of the surrogate model

Expected Improvement (EI)



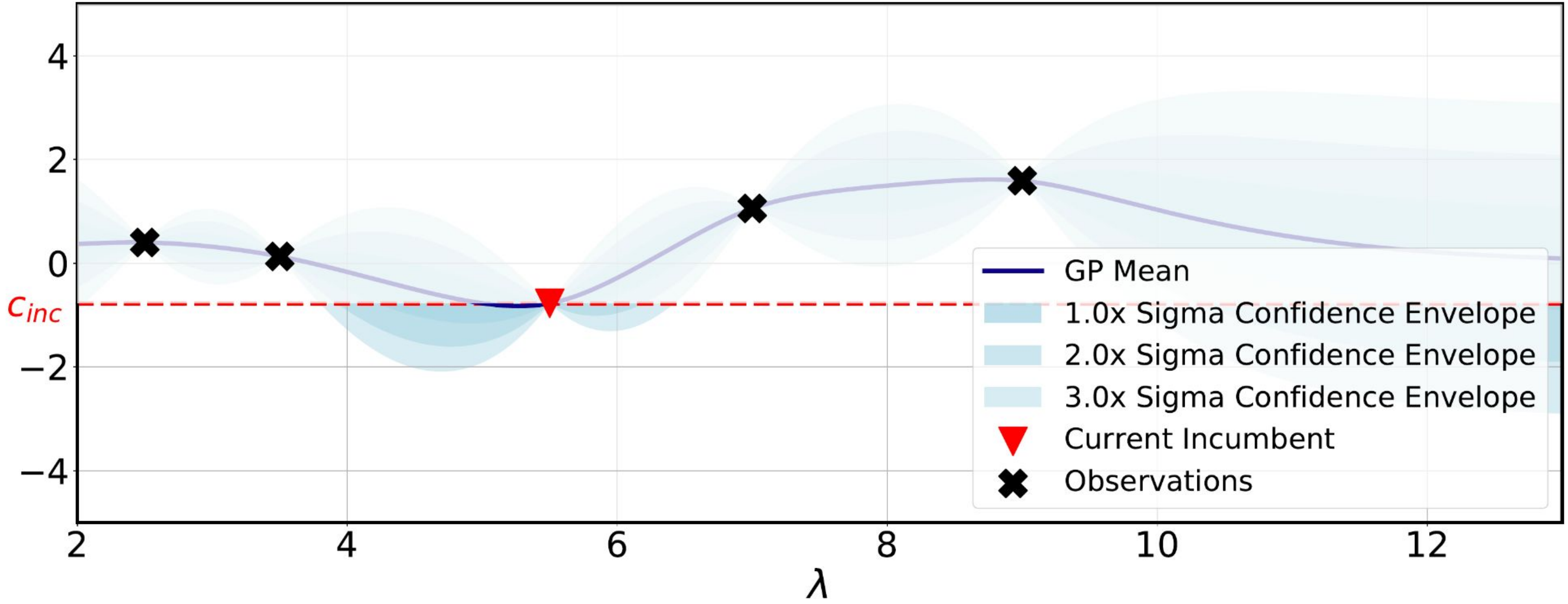
Given some observations and a fitted surrogate,

Expected Improvement (EI)



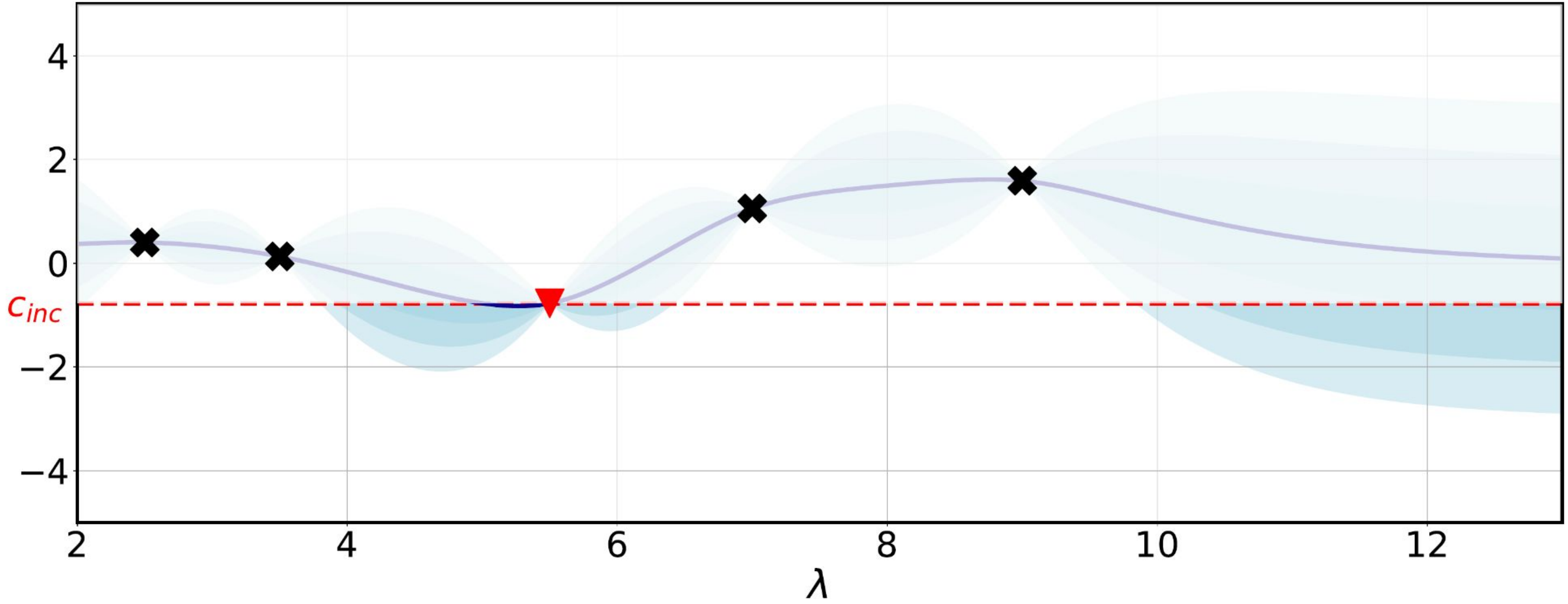
Given some observations and a fitted surrogate,

Expected Improvement (EI)



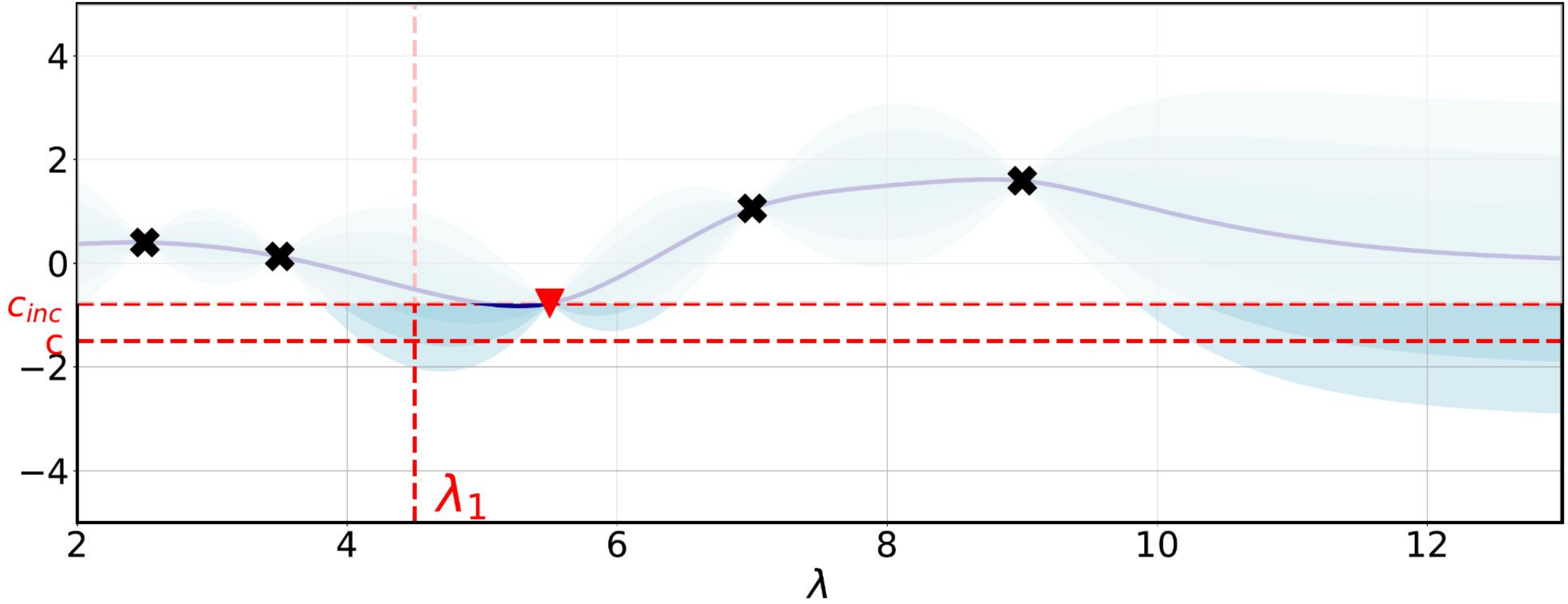
We care about *improving* over the c_{inc}.

Expected Improvement (EI)



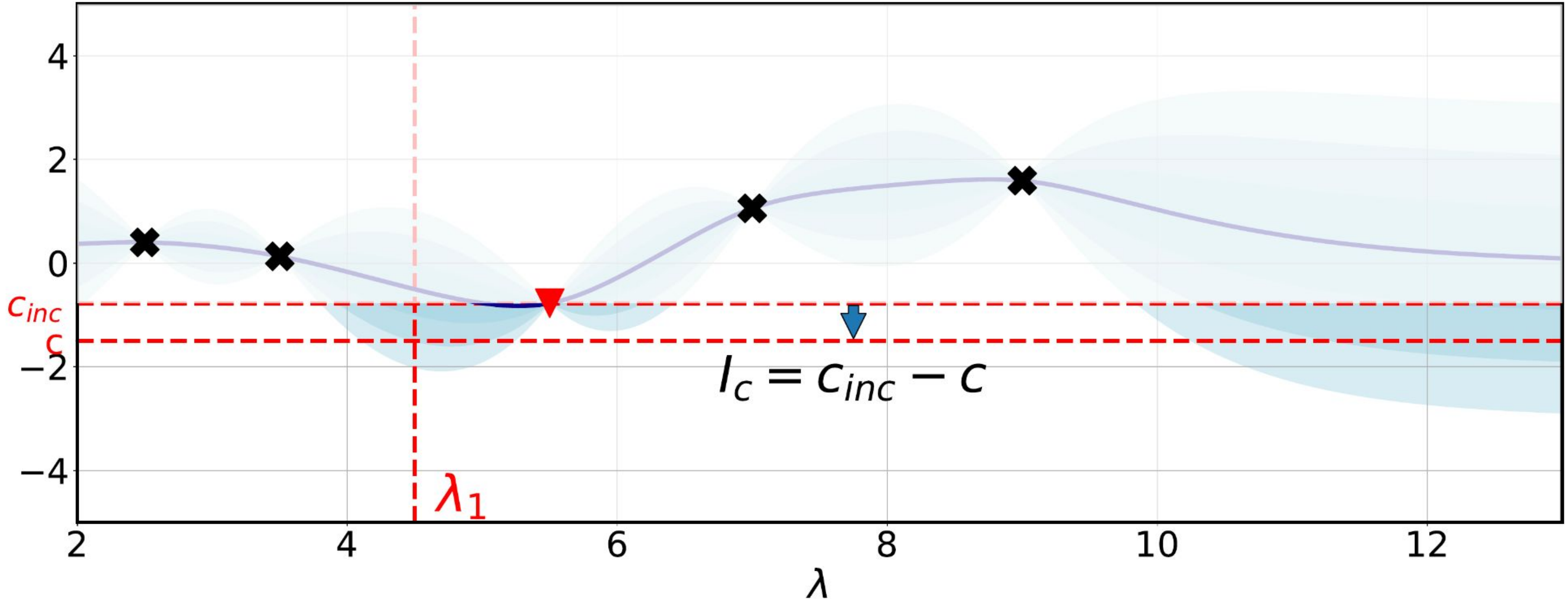
We care about *improving* over the c_{inc}.

Expected Improvement (EI)



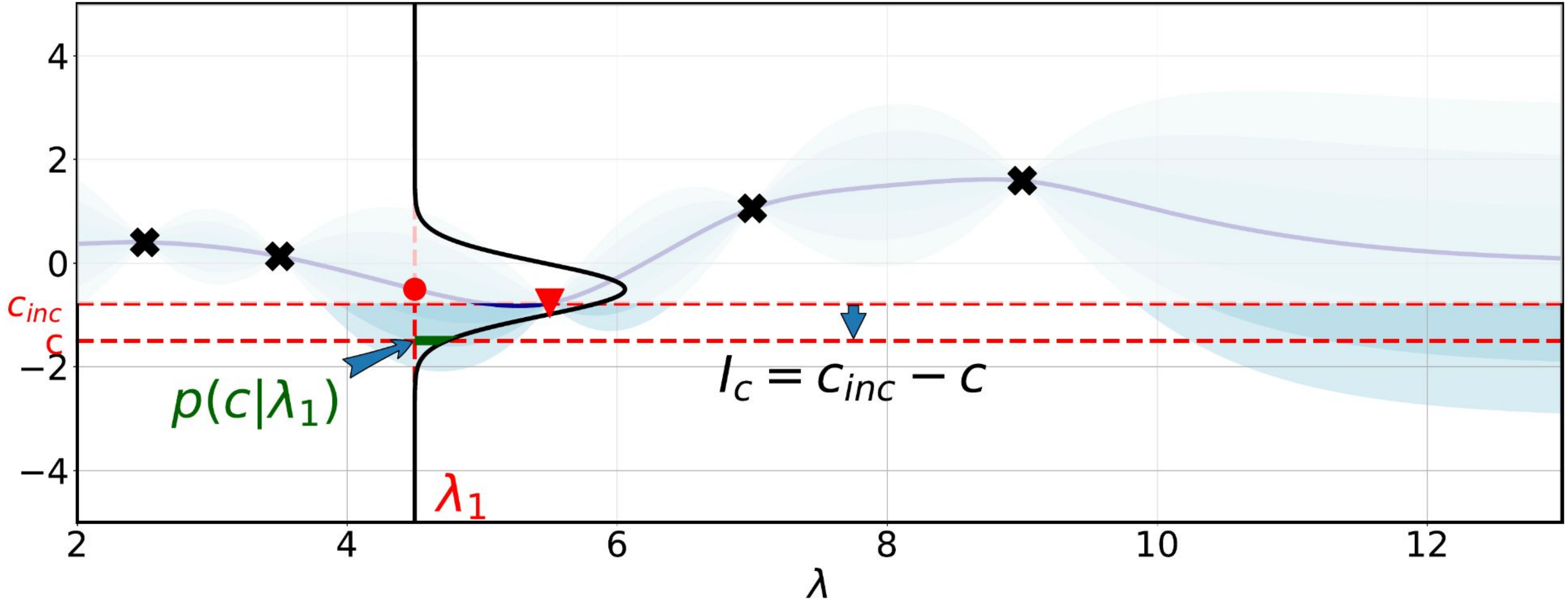
Let's look at a candidate configuration λ_1 and its hypothetical cost c.

Expected Improvement (EI)



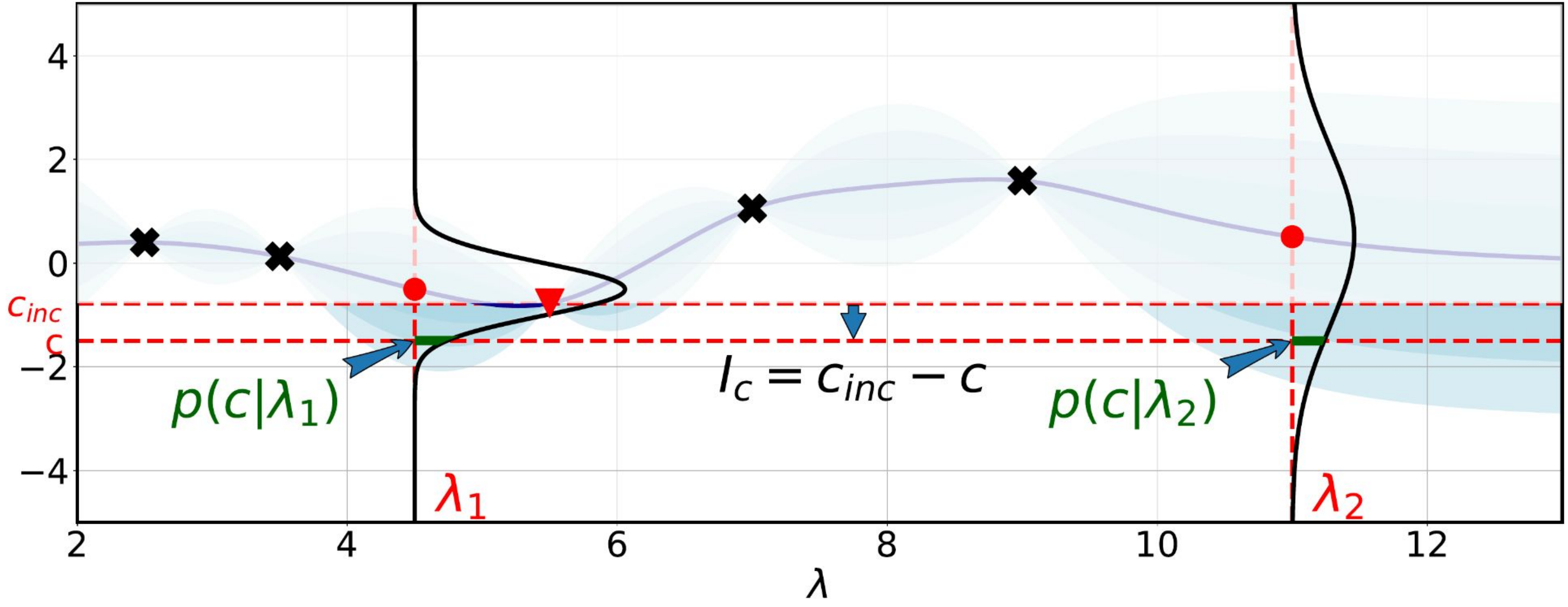
We can compute the improvement $I_c(\lambda_1)$. But how likely is it?

Expected Improvement (EI)



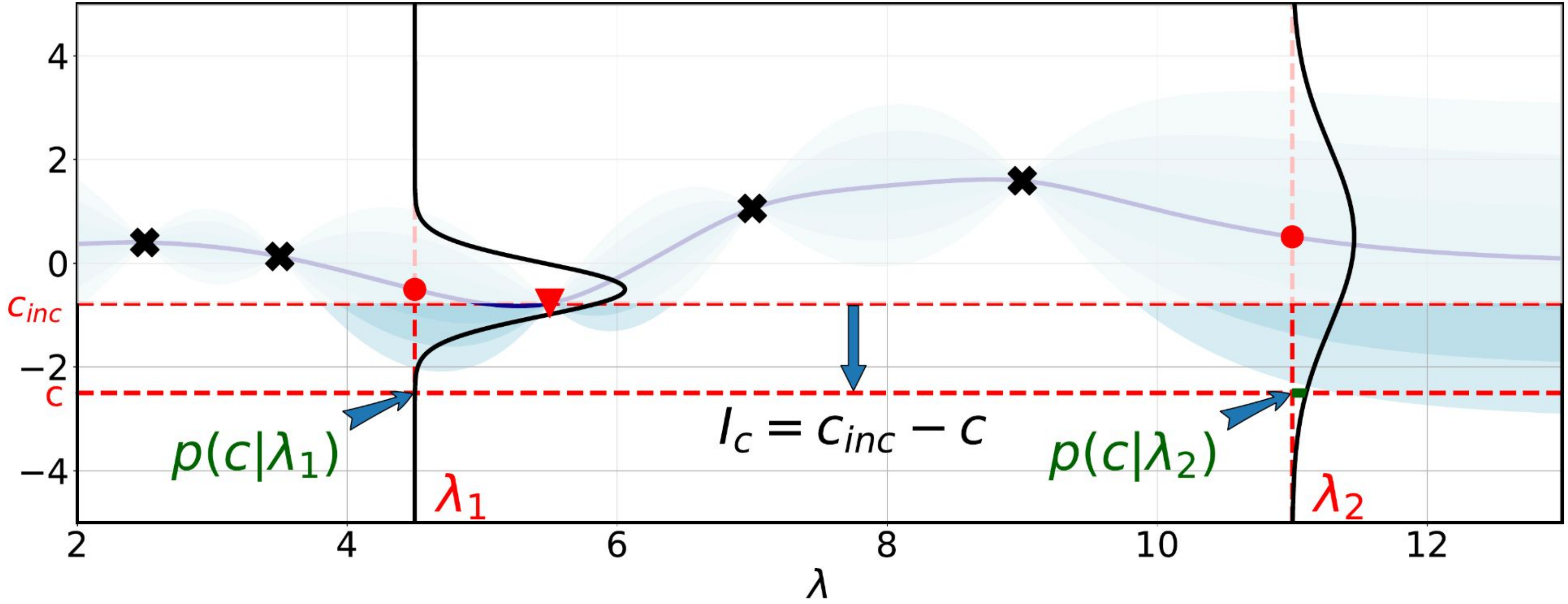
Knowing that $\hat{c}(\boldsymbol{\lambda}) = \mathcal{N}(\mu(\boldsymbol{\lambda}), \sigma^2(\boldsymbol{\lambda}))$, we can compute $p(c|\boldsymbol{\lambda})$

Expected Improvement (EI)



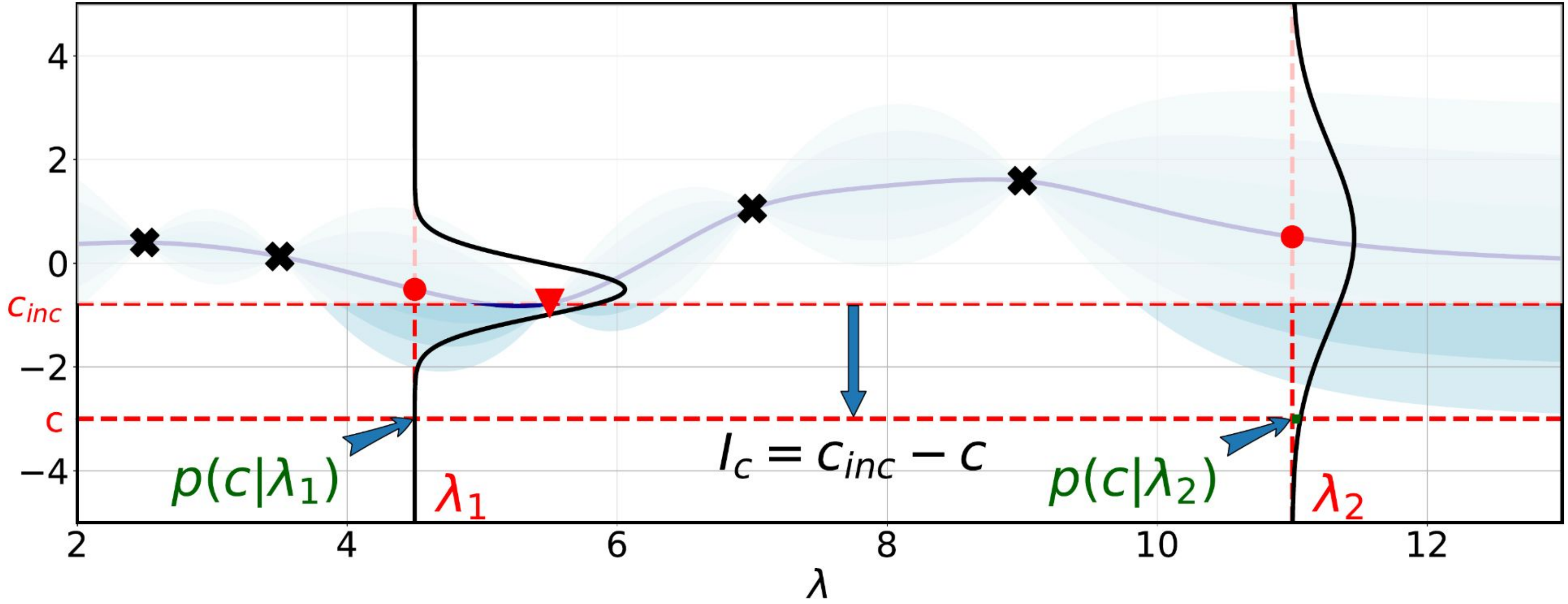
Comparing this for different configurations

Expected Improvement (EI)



and costs.

Expected Improvement (EI)



To compute EI, we sum all $p(c \mid \boldsymbol{\lambda}) \times I_c$ over all possible cost values.

Expected Improvement (EI)-Formal Definition

We define the one-step positive improvement over the current incumbent as

$$I^{(t)}(\boldsymbol{\lambda}) = \max(0, c_{inc} - c(\boldsymbol{\lambda}))$$

Expected Improvement is then defined as

$$u_{EI}^{(t)}(\boldsymbol{\lambda}) = \mathbb{E}[I^{(t)}(\boldsymbol{\lambda})] = \int_{-\infty}^{\infty} p^{(t)}(c \mid \boldsymbol{\lambda}) \times I^{(t)}(\boldsymbol{\lambda}) \; dc.$$

Since posterior is Gaussian, EI can be computed in closed form.

$$\boxed{\text{Choose } \boldsymbol{\lambda}^{(t)} \in \arg\max_{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}}(u_{EI}^{(t)}(\boldsymbol{\lambda}))}$$

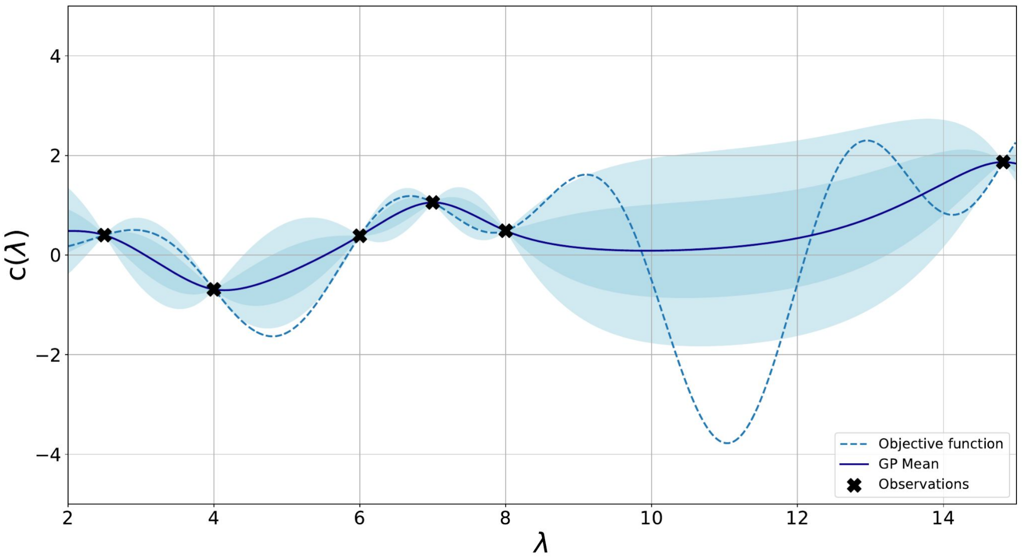
Main Ingredient II: The Surrogate Model

Required in all cases

- Regression model with uncertainty estimates
- Accurate predictions

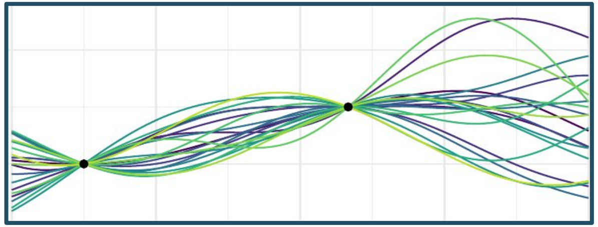
Depending on the application

- Cheap to train
- Scales well with #observations and #dimensions
- Can handle different types of hyperparameters



Types of Surrogates Models

- Gaussian Processes



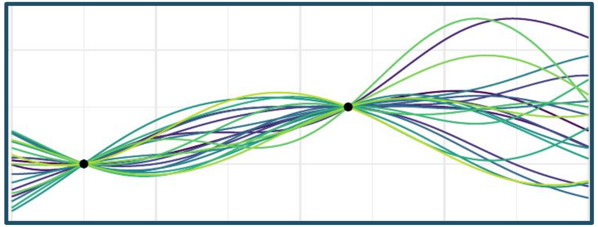
- Random Forests

- Bayesian Neural Networks

Photo by Filip Zrnzević on Unsplash
Photo by Alina Grubnyak on Unsplash

Gaussian Processes

$$m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})]$$
$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}\left[(f(\mathbf{x}) - \mathbb{E}[f(\mathbf{x})])\,(f(\mathbf{x}') - \mathbb{E}[f(\mathbf{x}')])\right]$$
$$f(\mathbf{x}) \sim \mathcal{G}\left(m(\mathbf{x}), k\left(\mathbf{x}, \mathbf{x}'\right)\right)$$



Advantages

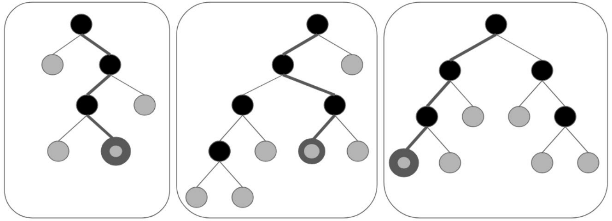
- Smooth uncertainty estimates
- Strong sample efficiency
- Expert knowledge can be encoded in the kernel
- Accurate predictions

Disadvantages

- Cost scales cubically with #observations
- Weak performance for high dimensionality
- Not easily applicable in discrete, categorical or conditional spaces
- Sensitive wrt its own hyperparameters

→ These make GPs the most commonly used model for Bayesian optimization

Tree-Based Methods



Advantages

- Scales well with #dimensions and #observations
- Training can be parallelized and is fast
- Can easily handle discrete, categorical and conditional spaces
- Robust wrt. its own hyperparameters

Disadvantages

- Poor uncertainty estimates
- Poor extrapolation (constant)
- Expert knowledge can not be easily incorporated

→ These make RFs a robust option in high dimensions, a high number of evaluations and for mixed spaces

Photo by Filip Zrnzević on Unsplash
Photo by Alina Grubnyak on Unsplash

Bayesian Neural Networks

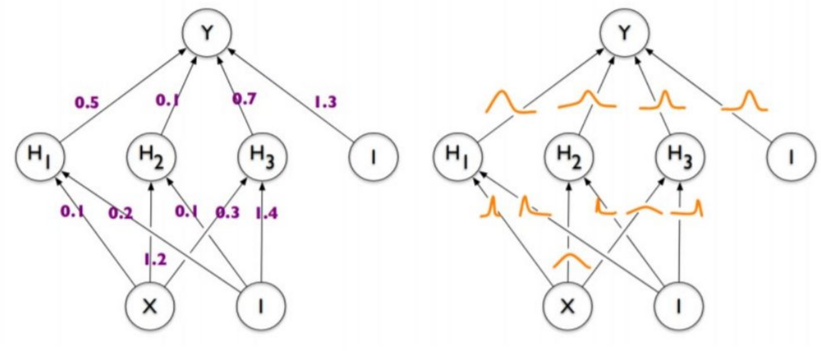


Image source: [Blundell et al. 2015]

Advantages

- Scales linear #observations
- (Can yield) smooth uncertainty estimates
- Flexibility wrt. discrete and categorical spaces

Disadvantages

- Needs many #observations
- Uncertainty estimates often worse than for GPs
- Many hyperparameters
- No robust off-the-shelf model

→ These make BNNs a promising alternative. [Li et al. 2023]

Photo by Filip Zrnzević on Unsplash
Photo by Alina Grubnyak on Unsplash

Alternatives to Bayesian Optimization

- Genetic Algorithms / Evolutionary Algorithms
  - E.g., GGA [Ansótegui et al. 2009, Ansótegui et al. 2015, Ansótegui et al. 2021]
- Estimation of Distributions
  - E.g., irace [López-Ibáñez et al. 2016]
- Reinforcement Learning
  - E.g., Neural architecture search with RL [Zoph and Le. 2017]
- Golden Parameter Search [Pushak and Hoos. 2020]
  - Assumes a benign landscape [Pushak and Hoos. 2018]

Are AC Landscapes Benign? [Pushak and Hoos, 2018]

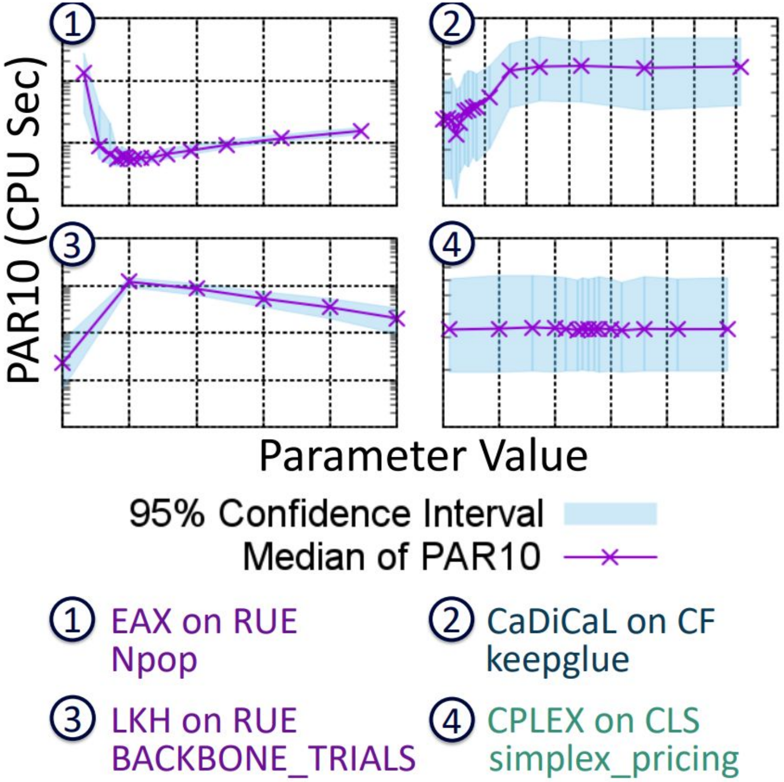
Instance set responses:

- Nearly all unimodal
- Nearly all convex
- Relatively “smooth”

Individual instance responses:

- Mostly uni-modal
- Mostly convex
- More “noisy

⇒ What kind of optimizers do we really need?



Questions?

Kahoot Quiz II

How to optimize across instances?

The Problem of Generalization & Instances

- **So far**: Obtaining a single configuration performing well on a single task (e.g., dataset, SAT instances, MIP instance, …)
  - **Problem**: Why should we search for a well-performing configuration if we have solved the task already?
    - This makes sense for optimization tasks (e.g., machine learning) where we might find better solutions.
    - Makes little to no sense for decision problems (e.g., SAT, ASP, Planning, …)
- **New Objective**:

  <u>Find a configuration that performs well on a distribution of instances</u>

  - The obtained configuration should also perform well on new instances
    → Generalization of the configuration's performance

The Algorithm Configuration Problem

Given:

- A configuration space
- A set of instances (drawn from some instance distribution)
- A cost metric (wlog. to be minimized)

The configuration problem is:

$$\operatorname{argmin}_{\lambda \in \Lambda} \frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} c(\lambda, i)$$

⇒ Assumes that there is a single good configuration for all (homogeneous) instances!

How to find good configuration we know already.

But how do we compare n configurations across instances?

Naive Algorithm Configuration

- Comparing two configurations against each other on set of instances (being sampled from the underlying instance distribution)

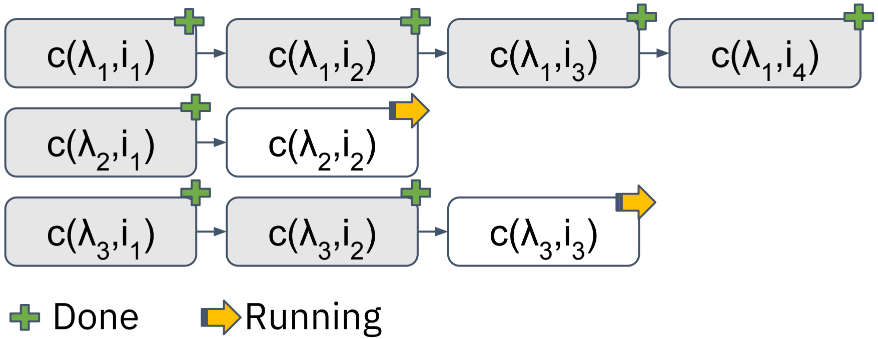
$c(\lambda_1, i_1)$	$c(\lambda_1, i_2)$	$c(\lambda_1, i_3)$	$c(\lambda_1, i_4)$	$c(\lambda_1, i_5)$	$c(\lambda_1, i_6)$	$c(\lambda_1, i_7)$	…
$c(\lambda_2, i_1)$	$c(\lambda_2, i_2)$	$c(\lambda_2, i_3)$	$c(\lambda_2, i_4)$	$c(\lambda_2, i_5)$	$c(\lambda_2, i_6)$	$c(\lambda_2, i_7)$	…

- If we consider runtime as our cost c, then we pay for evaluation two configurations:

$$\sum_{i \in \mathcal{I}} c(\lambda_1, i) + \sum_{i \in \mathcal{I}} c(\lambda_2, i)$$

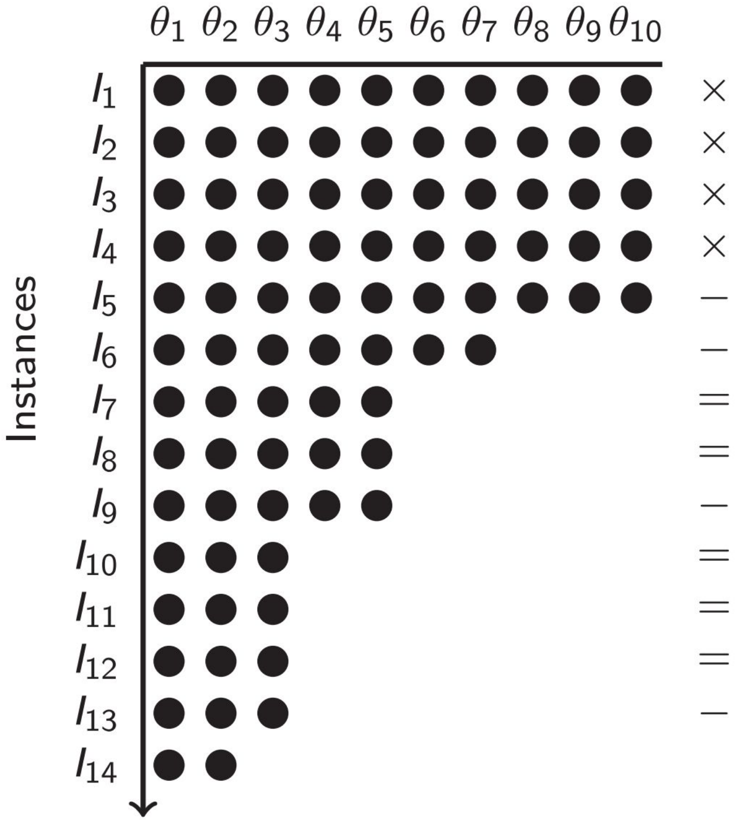
- If we have a few thousand instances and each of them takes minutes, it is crazy expensive to only compare two configurations

Simple Parallel Racing [Ansotegui et al. 2009]



- Run *n* configurations in parallel on *k* instances
- Whoever is done first, won the race
  - primarily, designed for runtime as cost metric
- Increase the number of instances for next race (e.g., geometric schedule)

Racing with Statistical Tests [López-Ibáñez et al. 2016]



- “X” no statistical test to collect sufficient evidence first
- “–” at least one underperforming configuration was discarded
- “=” no configuration was discarded

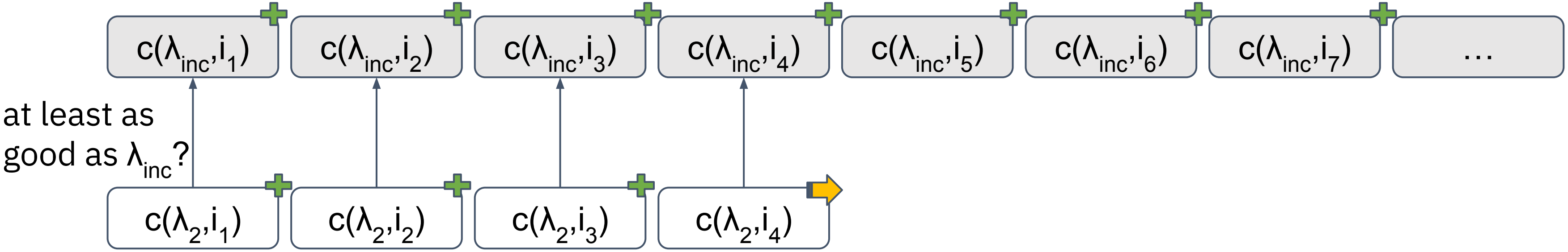
⇒ irace

— — —

- Refresh the slots of discarded configurations to have efficient parallelization [Xiao et al. 2023]

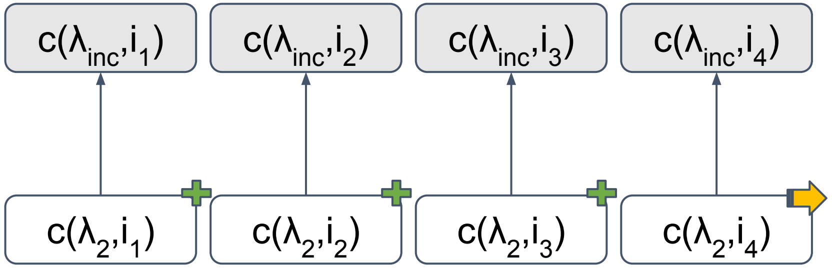
Aggressive Racing [Hutter et al. 2009]

- Idea: If a configuration underperforms compared to the current incumbent configuration, directly reject it.



- For n instances being evaluated on λ_2 (and λ_{inc})
  - **if** $\frac{1}{|\mathcal{I}_n|}\sum_{i\in\mathcal{I}_n} c(\lambda_{\text{inc}}, i) \geq \frac{1}{|\mathcal{I}_n|}\sum_{i\in\mathcal{I}_n} c(\lambda_2, i)$
    - Evaluate λ_2 on more instances (doubling the amount each time)
  - **otherwise**
    - Reject λ_2 and sample a new configuration to challenge λ_{inc}

Adaptive Capping for Aggressive Racing



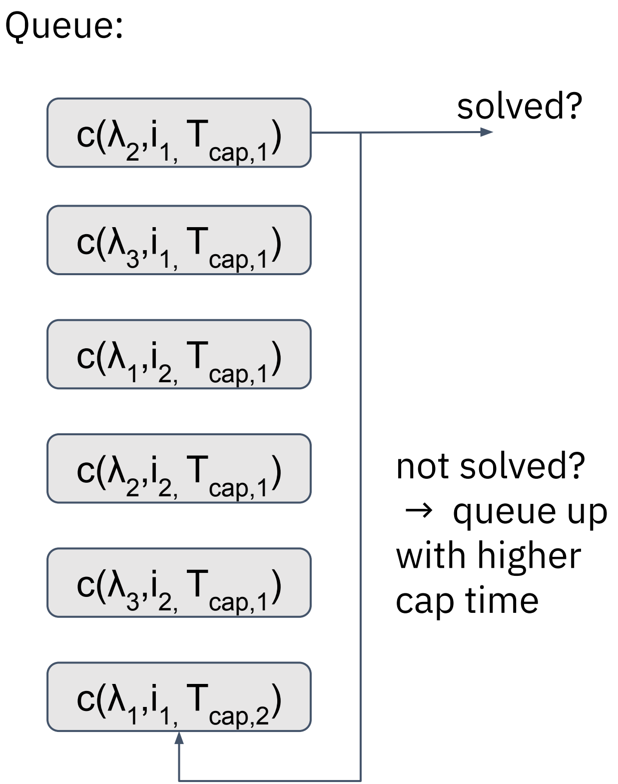
- Assuming, runtime optimization
- If we know the runtime of λ_{inc} on all *k* instances, and the runtime for λ_2 on some instances, how much time (captime) do we need at most to invest to check whether λ_2 can outperform λ_{inc}?

$$\sum_{i \in \mathcal{I}_1} c(\lambda_{\text{inc}}, i) - \sum_{i \in \mathcal{I}_2} c(\lambda_2, i) \text{ where } \mathcal{I}_1 \supseteq \mathcal{I}_2$$

Structured Procrastination

[Klein et al. 2017, 2019, Weisz et al. 2021]

- **Assumptions**:
  - Runtime optimization
  - Identify the best configurations on arbitrarily many instances
- **Problem** with previous approaches:
  - If assume a maximal captime for each configuration run, the algorithm configurator can waste arbitrarily much time on bad configurations
  - No theoretical guarantees on efficiency
- **Idea**: Instead of top-down capping runs, start with very small captimes and increase them iteratively
  - → unsuccessful runs will be procrastinated in favor of short runs of other configurations
- **⇒ Theoretical guarantees!**



AC-Band [Brandt et al. 2023]

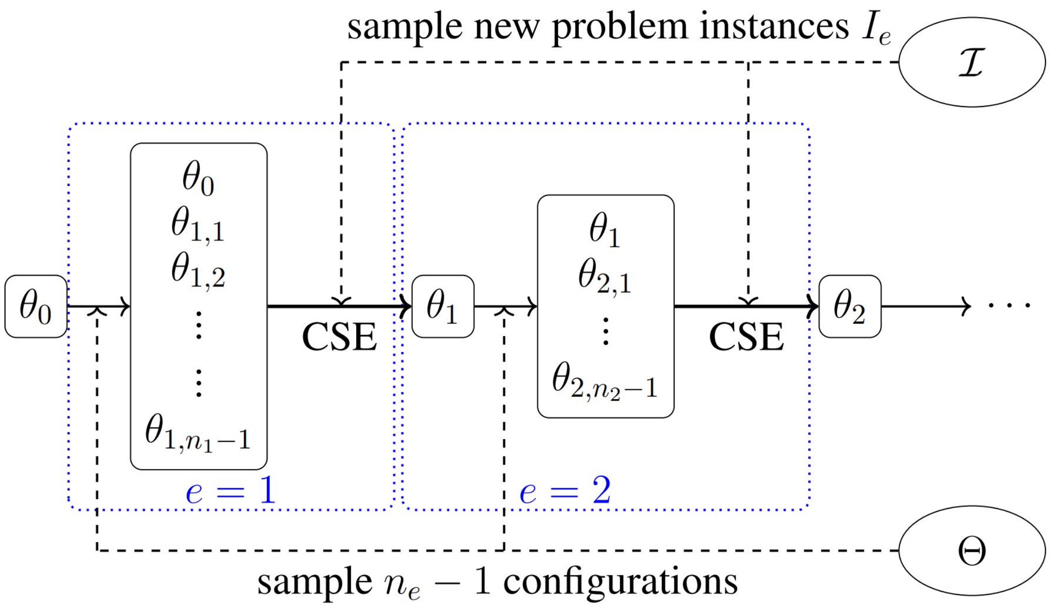
Tradeoff:

- too aggressive: rejecting good configurations
- too gracious: wasting compute time

Idea:

- motivated by Hyperband, start with many configurations on few instances → decrease number of configurations and increase number of instances

⇒ theoretical guarantees



Recommended Software

- SMAC [Hutter et al. 2011, Lindauer et al. 2021]
  - SMAC2 in Java [not maintained anymore]
  - SMAC3 in Python [active development]
- GGA [Ansótegui et al. 2009, Ansótegui et al. 2015, Ansótegui et al. 2021]
  - GGA in Python → PyDGGA
- irace [López-Ibáñez et al. 2016]
  - irace in R [active development]
  - irace in Python WIP
- AClib [Hutter et al. 2014]
  - Benchmark library for AC problems

Beyond traditional AC

Further Variations

- Real-time AC [Fitzgerald et al. 2014, Weiss & Tierney. 2022]
  - stream of instances
- Multi-objective algorithm configuration [Blot et al. 2016]
  - several objectives, such as runtime, memory, CO2-footprint, ...
  - Soon, there will be a multi-objective configurator based on Bayesian Optimization
- Dynamic algorithm configuration [Adriansen et al. 2022]
  - dynamically adapt configuration while the algorithm is running
- Per-Instance Algorithm Configuration e.g. [Xu et al. 2011]
  - Learn to map instance features to configurations
- Neural Architecture Search [Elsken et al. 2019]
  - Finding a well-performing architecture of a deep neural network
- CASH [Thornton et al. 2013]
  - Structured search spaces for AutoML (pipelines)

Journal of Artificial Intelligence Research 75 (2022) 425-487 Submitted 02/2022; published 10/2022

A Survey of Methods for Automated Algorithm Configuration

Elias Schede ELIAS.SCHEDE@UNI-BIELEFELD.DE
Decision and Operation Technologies Group,
Bielefeld University, Bielefeld, Germany

Jasmin Brandt JASMIN.BRANDT@UPB.DE
Alexander Tornede ALEXANDER.TORNEDE@UPB.DE
Department of Computer Science,
Paderborn University, Paderborn, Germany

Marcel Wever MARCEL.WEVER@IFI.LMU.DE
Institute of Informatics, LMU Munich &
Munich Center for Machine Learning, Munich, Germany

Viktor Bengs VIKTOR.BENGS@IFI.LMU.DE
Institute of Informatics,
LMU Munich, Munich, Germany

Eyke Hüllermeier EYKE@LMU.DE
Institute of Informatics, LMU Munich &
Munich Center for Machine Learning, Munich, Germany

Kevin Tierney KEVIN.TIERNEY@UNI-BIELEFELD.DE
Decision and Operation Technologies Group,
Bielefeld University, Bielefeld, Germany

Join our COSEAL network on algorithm selection, configuration and related topics, if you are interested to work more on these topics:

COSEAL.NET

Kahoot Quiz III

Find Us

AutoML.org

@AutoML_org
automl
@AutoML_org

LUH|AI

@AIHannover
LUH-AI
@luh-ai

Funded by:

European Research Council
Established by the European Commission

DFG Deutsche Forschungsgemeinschaft

Federal Ministry of Education and Research

Federal Ministry for Economic Affairs and Energy

Federal Ministry for the Environment, Nature Conservation, Nuclear Safety and Consumer Protection

Niedersachsen

Backup Slides

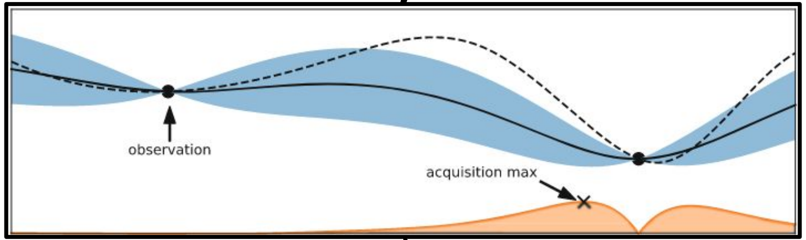
Bayesian Optimization: Extensions

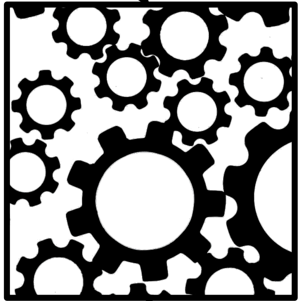
Different models to handle new design spaces

λ_n

Increase efficiency by using lower fidelities

Multi-objective





Consider constraints

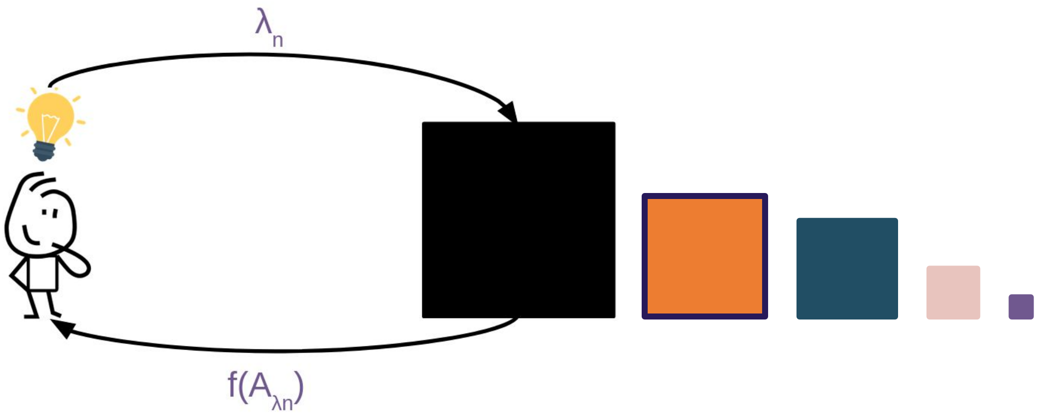
Leverage observations across experiments

$f(A_{\lambda n})$

Take user priors into account

Photo by Wilhelm Gunkel on Unsplash
Image by Feurer, Hutter: Hyperparameter Optimization
In: Automated Machine Learning

Multi-Fidelity Bayesian Optimization



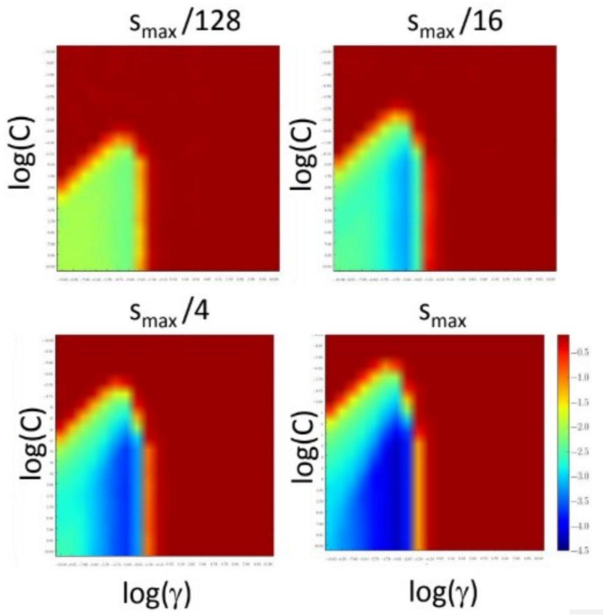
Often, the black-box

- is an **iterative** process,
- has **cheaper approximations** available,
- or can be evaluated **partially**

→ We can collect information about the actual objective value with less costly evaluations

Two Motivating Examples

Performance of a SVM on different subsets of MNIST



Learning curves of fully connected NNs on CIFAR-10

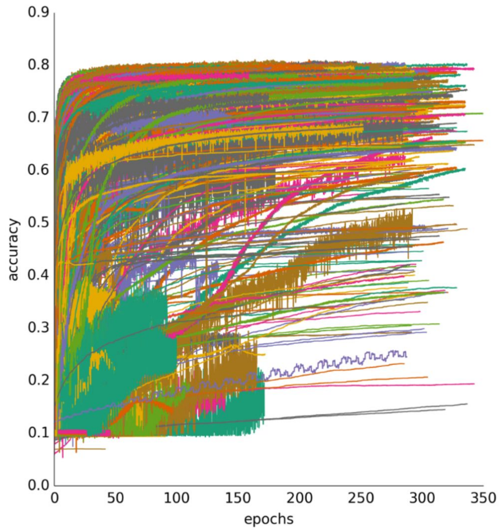
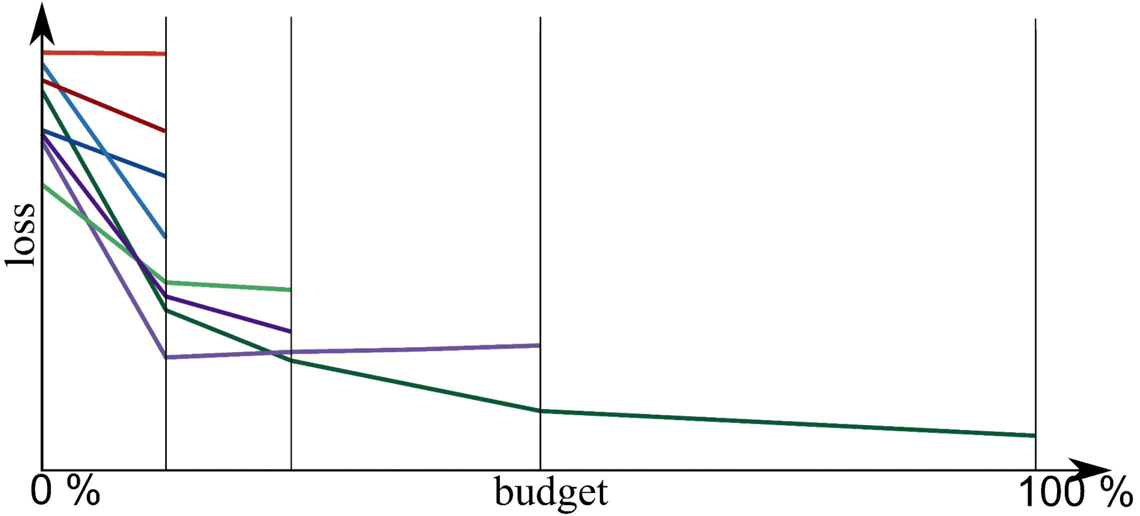


Image Source: [Domhan et al., 2015]

Successive Halving



[Jamieson and Talwalkar. 2016]

- A very simple algorithm:
  - Sample N configurations uniformly at random & evaluate them on the cheapest fidelity
  - Keep the best half (or third), move them to the next fidelity
  - Iterate until the most expensive fidelity (= original expensive black box)

Hyperband

What if the information on the lowest fidelity is not informative?

→ Run multiple iterations of SH, starting at different “lowest” fidelities.

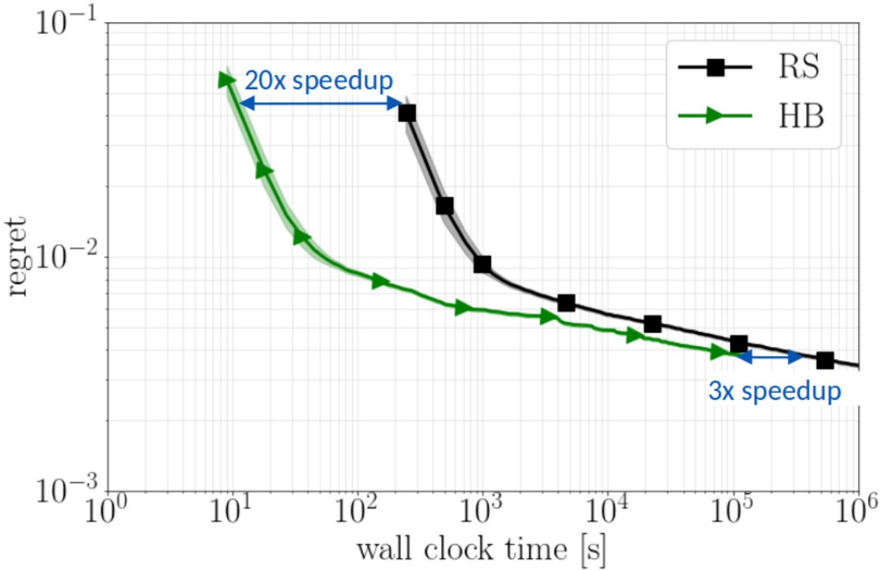


image credit: [Falkner et al. 2018]

BOHB: Hyperband X Bayesian Optimization

Idea: Use Bayesian Optimization to choose configurations [Falkner et al. 2018]

- BO to achieve strong performance
- HB to achieve good anytime performance

→ easy parallelization

→ with interleaved random sampling it keeps theoretical guarantees of HB

→ with RFs as surrogate model, it performs better than with TPE [Lindauer et al. 2022]

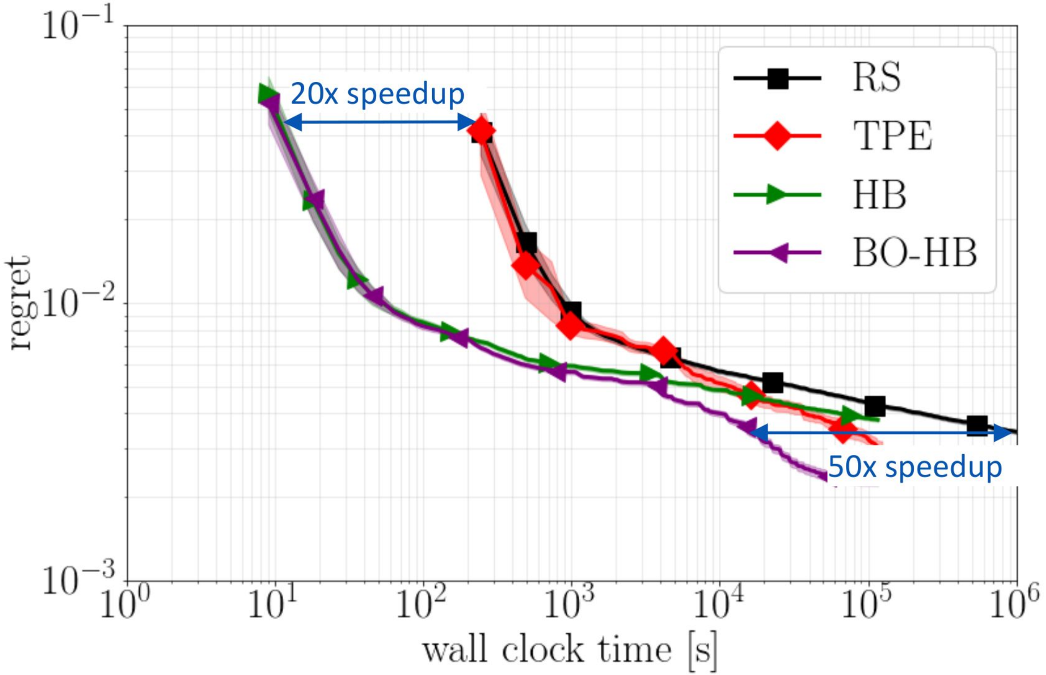


image credit: [Falkner et al. 2018]

Landscape of Multi-Fidelity HPO Methods

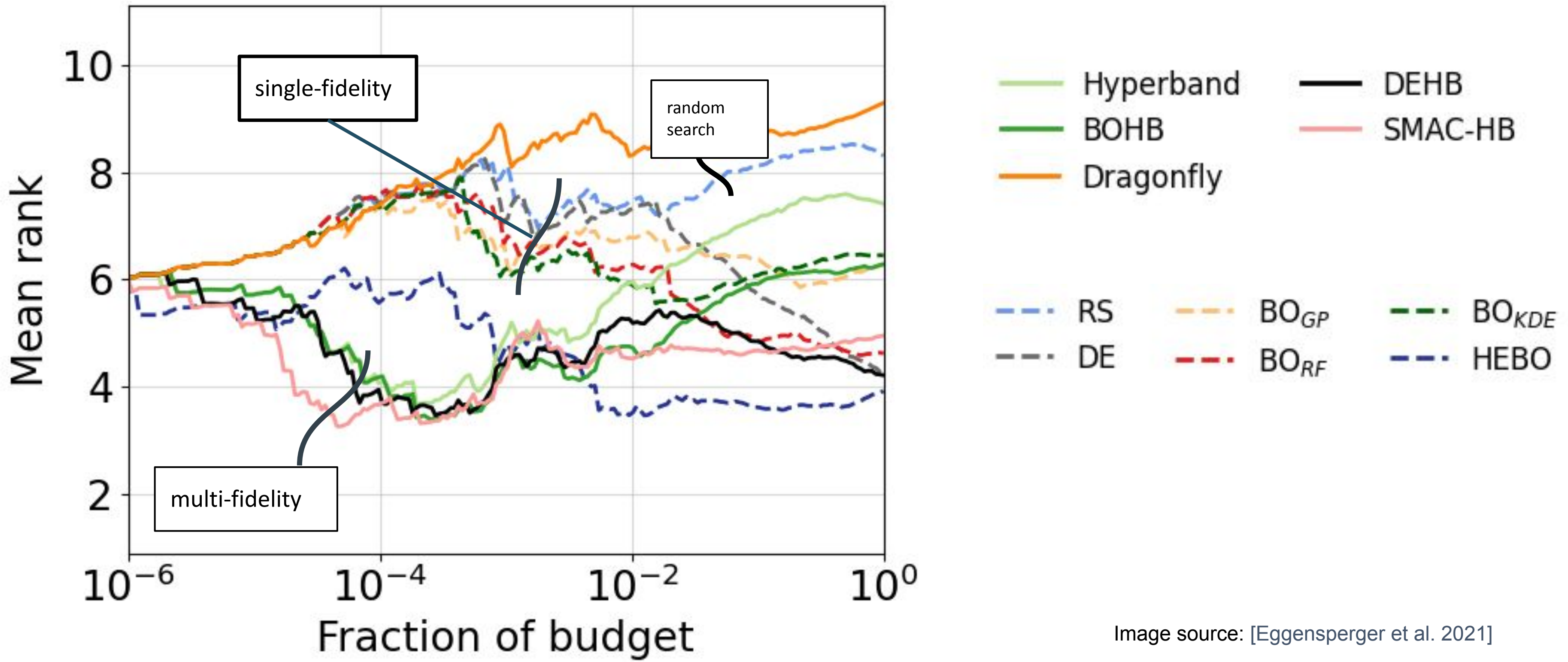


Image source: [Eggensperger et al. 2021]