AutoML: Beyond AutoML

Per-Instance Algorithm Configuration

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Homogeneous vs. Heterogeneous Instances

Assumption of AC: Homogeneous Instance Distribution

- Algorithm configuration tools assume that the instance distribution is homogeneous (see video on "Best Practices for AC")
- Important because
  - there is a well-performing configuration for all (or most) instances
  - the racing algorithm can make educated decisions on subsets

Homogeneous vs. Heterogeneous Instances

Assumption of AC: Homogeneous Instance Distribution

- Algorithm configuration tools assume that the instance distribution is homogeneous (see video on "Best Practices for AC")
- Important because
  - there is a well-performing configuration for all (or most) instances
  - the racing algorithm can make educated decisions on subsets

Violated assumption of AC: Hetergeneous Instance Distribution

- The racing algorithm will make inconsistent (or even wrong) decisions
- There is no single well-performing configuration for all instances

Homogeneous vs. Heterogeneous Instances

Assumption of AC: Homogeneous Instance Distribution

- Algorithm configuration tools assume that the instance distribution is homogeneous (see video on "Best Practices for AC")
- Important because
  - there is a well-performing configuration for all (or most) instances
  - the racing algorithm can make educated decisions on subsets

Violated assumption of AC: Hetergeneous Instance Distribution

- The racing algorithm will make inconsistent (or even wrong) decisions
- There is no single well-performing configuration for all instances

$\rightsquigarrow$ What should we do with heterogeneous instance distributions?

Why are systems for heterogeneous instance distributions important?

1. We cannot guarantee homogeneity in practice
    1. Instances might get larger and harder
    2. The underlying task or business case might change

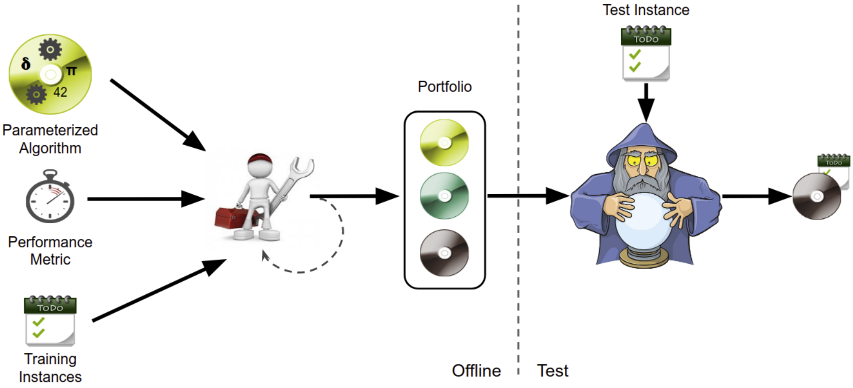
Why are systems for heterogeneous instance distributions important?

1. We cannot guarantee homogeneity in practice
   1. Instances might get larger and harder
   2. The underlying task or business case might change
2. We don't want to do algorithm configuration always from scratch

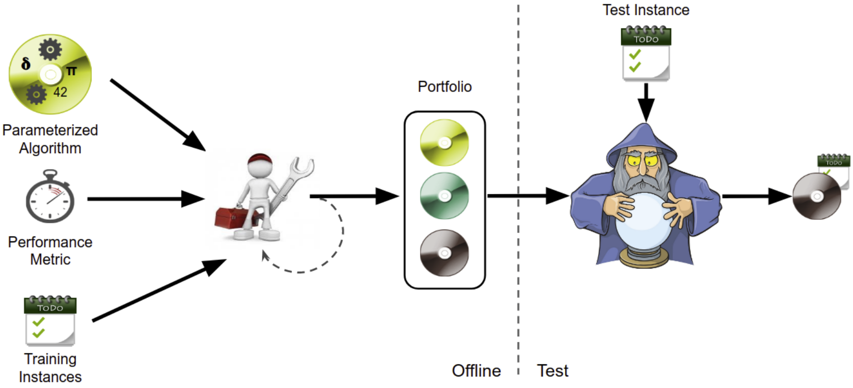
Why are systems for heterogeneous instance distributions important?

1. We cannot guarantee homogeneity in practice
    1. Instances might get larger and harder
    2. The underlying task or business case might change
2. We don't want to do algorithm configuration always from scratch
3. An adaptive configuration system would be the holy grail
    - $\rightsquigarrow$ hard to achieve

PIAC: Per-Instance Algorithm Configuration



PIAC: Per-Instance Algorithm Configuration



- You can use whichever kind of algorithm selection (wizard) you want
- Challenge: Building a portfolio
- Use case: Instances are heterogeneous

PIAC: Manual Expert Approach

Basic Assumption

Heterogeneous instance set can be divided into homogeneous subsets

PIAC: 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
  $\rightarrow$ portfolio of configurations
- Use Algorithm Selection to select a well-performing configuration on 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 (here GGA) on each instance set

Instance-Specific Algorithm Configuration: ISAC [Kadioglu et al. 2010]

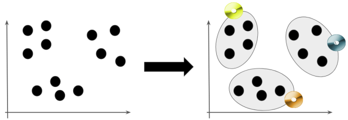
Idea

Training:

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



Hydra [Xu et al. 2010]

Idea

- Iteratively add configurations to a portfolio $\mathbf{P}$, start with $\mathbf{P} = \emptyset$
- In each iteration, determine configuration that is complementary to $\mathbf{P}$
  - $\leadsto$ Maximize marginal contribution to $\mathbf{P}$

Hydra [Xu et al. 2010]

Idea

- Iteratively add configurations to a portfolio $\mathbf{P}$, start with $\mathbf{P} = \emptyset$
- In each iteration, determine configuration that is complementary to $\mathbf{P}$
  - $\rightsquigarrow$ Maximize marginal contribution to $\mathbf{P}$

Marginal contribution of a configuration $\boldsymbol{\lambda}$ to a portfolio $\mathbf{P}$:

$$c(\mathbf{P}) - c(\mathbf{P} \cup \{\boldsymbol{\lambda}\})$$

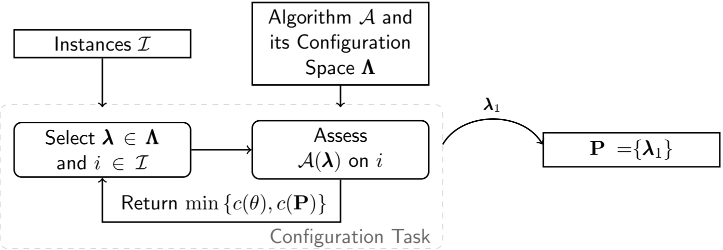
Hydra [Xu et al. 2010]

Idea

- Iteratively add configurations to a portfolio $\mathbf{P}$, start with $\mathbf{P} = \emptyset$
- In each iteration, determine configuration that is complementary to $\mathbf{P}$
  - $\leadsto$ Maximize marginal contribution to $\mathbf{P}$

Marginal contribution of a configuration $\boldsymbol{\lambda}$ to a portfolio $\mathbf{P}$:

$$c(\mathbf{P}) - c(\mathbf{P} \cup \{\boldsymbol{\lambda}\})$$



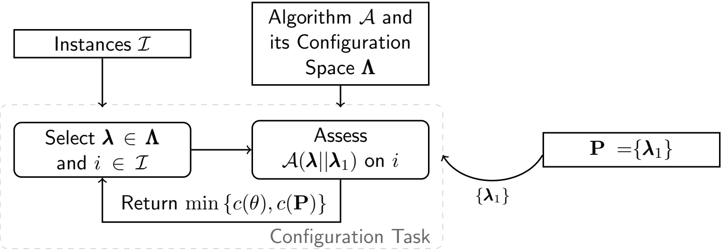
Hydra [Xu et al. 2010]

Idea

- Iteratively add configurations to a portfolio $\mathbf{P}$, start with $\mathbf{P} = \emptyset$
- In each iteration, determine configuration that is complementary to $\mathbf{P}$
  - $\rightsquigarrow$ Maximize marginal contribution to $\mathbf{P}$

Marginal contribution of a configuration $\boldsymbol{\lambda}$ to a portfolio $\mathbf{P}$:

$$c(\mathbf{P}) - c(\mathbf{P} \cup \{\boldsymbol{\lambda}\})$$



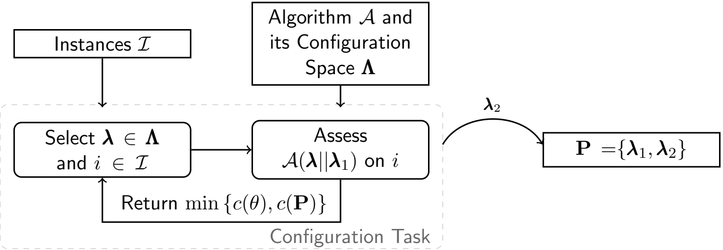
Hydra [Xu et al. 2010]

Idea

- Iteratively add configurations to a portfolio $\mathbf{P}$, start with $\mathbf{P} = \emptyset$
- In each iteration, determine configuration that is complementary to $\mathbf{P}$
  - $\leadsto$ Maximize marginal contribution to $\mathbf{P}$

Marginal contribution of a configuration $\boldsymbol{\lambda}$ to a portfolio $\mathbf{P}$:

$$c(\mathbf{P}) - c(\mathbf{P} \cup \{\boldsymbol{\lambda}\})$$



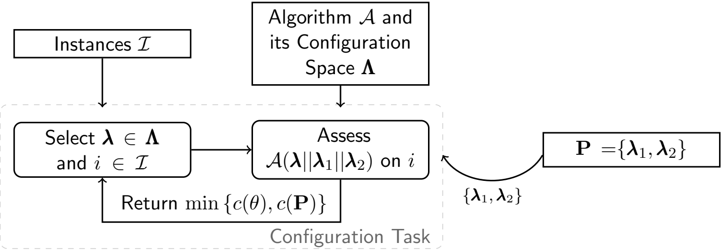
Hydra [Xu et al. 2010]

Idea

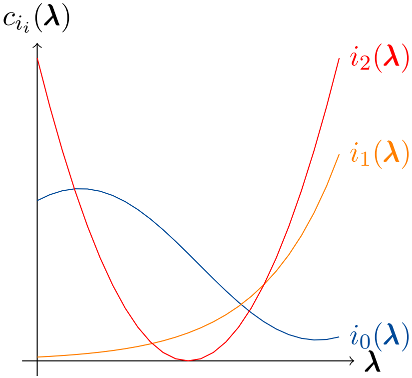
- Iteratively add configurations to a portfolio $\mathbf{P}$, start with $\mathbf{P} = \emptyset$
- In each iteration, determine configuration that is complementary to $\mathbf{P}$
  - $\leadsto$ Maximize marginal contribution to $\mathbf{P}$

Marginal contribution of a configuration $\boldsymbol{\lambda}$ to a portfolio $\mathbf{P}$:

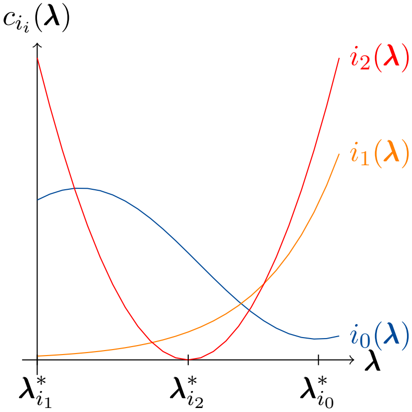
$$c(\mathbf{P}) - c(\mathbf{P} \cup \{\boldsymbol{\lambda}\})$$



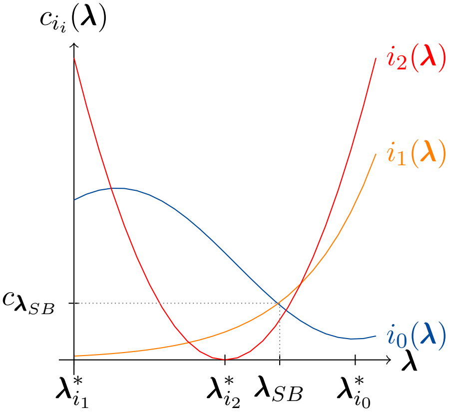
Heterogeneous example



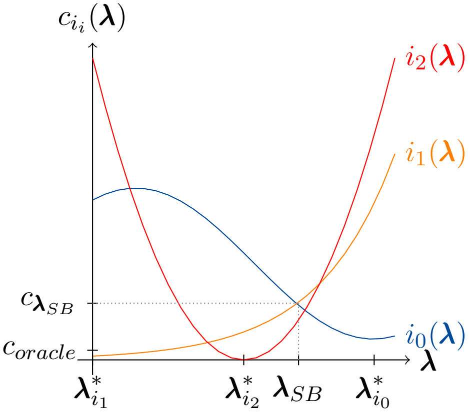
Heterogeneous example



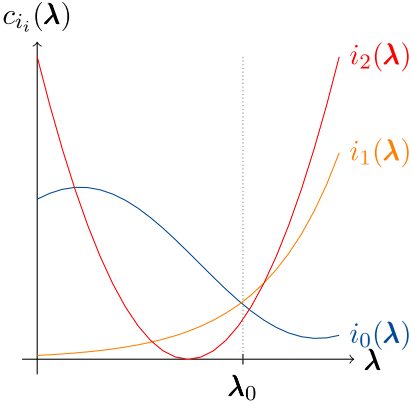
Heterogeneous example



Heterogeneous example

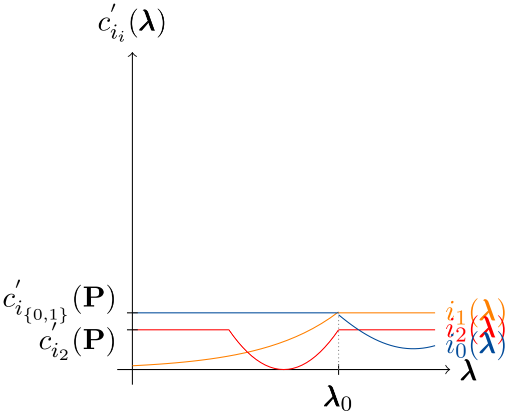


Hydra: Iteration 1



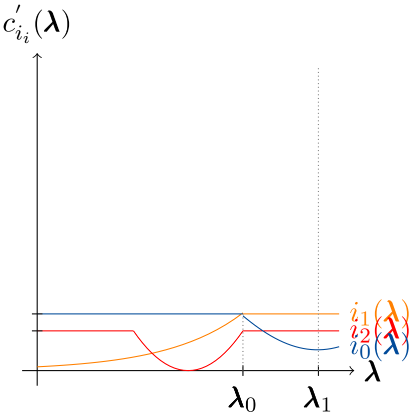
Search initial well performing configuration. Add $\boldsymbol{\lambda}_0$ to $\mathbf{P}$

Hydra: Iteration 1



Update metric

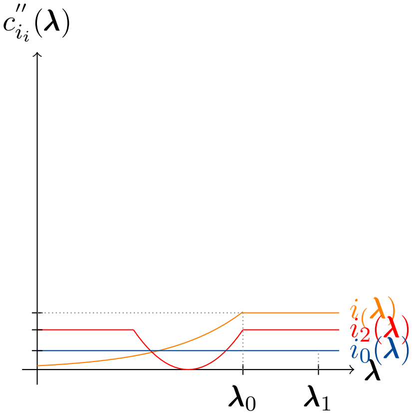
Hydra: Iteration 2



Search well performing configuration complementary to $\mathbf{P}$.

Add $\boldsymbol{\lambda}_1$ to $\mathbf{P}$.

Hydra: Iteration 2



Update metric

Cedalion [Seipp et al. 2015]

Idea

- Optimize a schedule of configurations with algorithm configuration

Cedalion [Seipp et al. 2015]

Idea

- Optimize a schedule of configurations with algorithm configuration

Approach

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

Cedalion [Seipp et al. 2015]

Idea

- Optimize a schedule of configurations with algorithm configuration

Approach

- Iteratively add a configuration with a time slot t to a schedule $\mathcal{S} \oplus \langle \boldsymbol{\lambda}, t \rangle$
- In each iteration, only optimize on instances not solved so far
- The time slot is a further parameter in the configuration space

Cedalion [Seipp et al. 2015]

Idea

- Optimize a schedule of configurations with algorithm configuration

Approach

- Iteratively add a configuration with a time slot t to a schedule $\mathcal{S} \oplus \langle \boldsymbol{\lambda}, t \rangle$
- In each iteration, only optimize on instances not solved so far
- The time slot is a further parameter in the configuration space
- Optimize marginal contribution per time spent:

$$\frac{c(\mathcal{S}) - c(\mathcal{S} \oplus \langle \boldsymbol{\lambda}, t \rangle)}{t}$$

Submodularity

Observation

- Performance metrics of Hydra and Cedalion are submodular
  - Family of functions
  - Adding an element to a set reduces the function value
  - Diminishing returns: decrease of the value reduction over time

Submodularity

Observation

- Performance metrics of Hydra and Cedalion are submodular
  - Family of functions
  - Adding an element to a set reduces the function value
  - Diminishing returns: decrease of the value reduction over time

Definition (Submodularity of f)

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

Submodularity

Observation

- Performance metrics of Hydra and Cedalion are submodular
  - Family of functions
  - Adding an element to a set reduces the function value
  - Diminishing returns: decrease of the value reduction over time

Definition (Submodularity of f)

For every $X, Y \subseteq Z$ with $X \subseteq Y$ and every $x \in Z - Y$ 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:
At most away from optimum by factor of 0.63 (see [Streeter and Golovin. 2008])

Dynamic Instance Grouping [Liu et al. 2018]

Idea

- Similar to ISAC: group instances into clusters
- Similar to Hydra: refine clusters and configurations over several iterations

Dynamic Instance Grouping [Liu et al. 2018]

Idea

- Similar to ISAC: group instances into clusters
- Similar to Hydra: refine clusters and configurations over several iterations

Main Idea

1. Group instances randomly into clusters
2. run AC on each cluster
3. Update clusters based on performance (estimates)
4. Go to 2. if budget is not empty
5. Consider all configurations ever found to create final portfolio

Dynamic Instance Grouping [Liu et al. 2018]

Idea

- Similar to ISAC: group instances into clusters
- Similar to Hydra: refine clusters and configurations over several iterations

Main Idea

1. Group instances randomly into clusters
2. run AC on each cluster
3. Update clusters based on performance (estimates)
4. Go to 2. if budget is not empty
5. Consider all configurations ever found to create final portfolio

- increase the configuration budget in each iteration
  - first clusterings will have a poor quality $\rightarrow$ small configuration time
  - later clusterings will be better $\rightarrow$ more configuration time