Algorithm Configuration

Challenges, Methods and Perspectives

Marius Lindauer   André Biedenkapp

What can you expect?

Content Overview over well-established and new research directions

Structure Combination of research insights and practical recommendations

Hands-On Some simple coding examples for algorithm configuration s.t. you can directly start to play with it

Outlook to new research directions and open challenges

Algorithms have Parameters!

Evolutionary Algorithms population size, selection strategies, mutation operators, recombination operators, step size, probabilities, ...

Algorithms have Parameters!

Evolutionary Algorithms population size, selection strategies, mutation operators, recombination operators, step size, probabilities, ...

Satisfiability Solving Search strategies (local vs. tree-based), variable selection heuristic, restart probability, deletion heuristics, ...

Algorithms have Parameters!

Evolutionary Algorithms population size, selection strategies, mutation operators, recombination operators, step size, probabilities, . . .

Satisfiability Solving Search strategies (local vs. tree-based), variable selection heuristic, restart probability, deletion heuristics, . . .

Machine Learning model class (linear, non-linear, tree, ...), model complexity, regularization, preprocessing, data augmentation, . . .

Algorithms have Parameters!

Evolutionary Algorithms population size, selection strategies, mutation operators, recombination operators, step size, probabilities, ...

Satisfiability Solving Search strategies (local vs. tree-based), variable selection heuristic, restart probability, deletion heuristics, ...

Machine Learning model class (linear, non-linear, tree, ...), model complexity, regularization, preprocessing, data augmentation, ...

Deep Learning network depth, network width, network architecture, activation function, optimizer, learning rate, momentum, ...

Algorithms have Parameters!

Evolutionary Algorithms population size, selection strategies, mutation operators, recombination operators, step size, probabilities, . . .

Satisfiability Solving Search strategies (local vs. tree-based), variable selection heuristic, restart probability, deletion heuristics, . . .

Machine Learning model class (linear, non-linear, tree, ...), model complexity, regularization, preprocessing, data augmentation, . . .

Deep Learning network depth, network width, network architecture, activation function, optimizer, learning rate, momentum, . . .

Remark: The ML folks call an algorithm's parameters hyperparameters.

But wait! (I)

🤔 But *my* algorithm has no parameters!

But wait! (I)

🤔 But *my* algorithm has no parameters!

- Correct: There are some algorithms that do not have parameters
- However: Algorithm parameters are simply not exposed to the user in some cases.
- Programming by Optimization could be a paradigm to address these hidden parameters [Hoos 2012]

But wait! (II)

- 🤔 Can't I simply use its default settings?

But wait! (II)

🤔 Can't I simply use its default settings?

Domain	Default vs. Optimized
Answer Set Solving [Gebser et al. 2011]	up to $14\times$ speedup
AI Planning [Vallati et al. 2013]	up to $40\times$ speedup
Mixed Integer Programming [Hutter et al. 2010]	up to $52\times$ speedup
Satisfiability Solving [Hutter et al. 2017]	up to $3000\times$ speedup

But wait! (II)

🤔 Can't I simply use its default settings?

Domain	Default vs. Optimized
Answer Set Solving [Gebser et al. 2011]	up to $14\times$ speedup
AI Planning [Vallati et al. 2013]	up to $40\times$ speedup
Mixed Integer Programming [Hutter et al. 2010]	up to $52\times$ speedup
Satisfiability Solving [Hutter et al. 2017]	up to $3000\times$ speedup
Minimum Vertex Cover [Wagner et al. 2017]	up to 9% absolute impr.
Machine Learning [Feuer et al. 2015]	up to 35% absolute impr.
Deep Learning [Zimmer et al. 2020]	up to 49% absolute impr.

But wait! (II)

🤔 Can't I simply use its default settings?

Domain	Default vs. Optimized
Answer Set Solving [Gebser et al. 2011]	up to $14\times$ speedup
AI Planning [Vallati et al. 2013]	up to $40\times$ speedup
Mixed Integer Programming [Hutter et al. 2010]	up to $52\times$ speedup
Satisfiability Solving [Hutter et al. 2017]	up to $3000\times$ speedup
Minimum Vertex Cover [Wagner et al. 2017]	up to 9% absolute impr.
Machine Learning [Feuer et al. 2015]	up to 35% absolute impr.
Deep Learning [Zimmer et al. 2020]	up to 49% absolute impr.

Remark: Default configurations can nevertheless *sometimes* be very strong.

But wait! (III)

🤔 I don't want to bother with parameters,
because manual parameter tuning is . . .

1. time-consuming
2. tedious
3. error-prone

But wait! (III)

🤔 I don't want to bother with parameters,
because manual parameter tuning is . . .

1. time-consuming
2. tedious
3. error-prone

🤔 Can't I simply use these AI algorithms everyone is talking about?

But wait! (III)

🤔 I don't want to bother with parameters, because manual parameter tuning is ...

1. time-consuming
2. tedious
3. error-prone

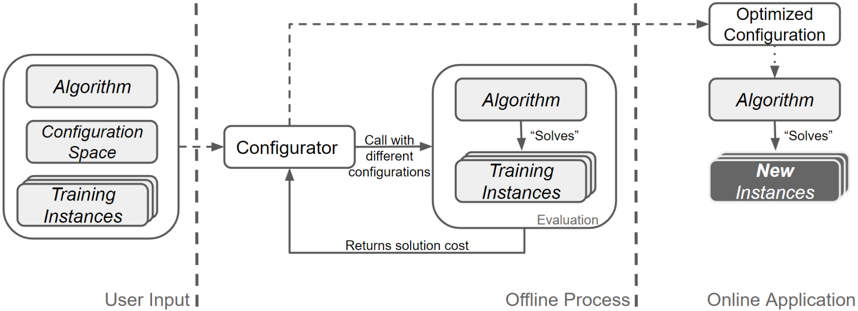
🤔 Can't I simply use these AI algorithms everyone is talking about?

$\rightsquigarrow$ Sure! Let's use automatic algorithm configuration!

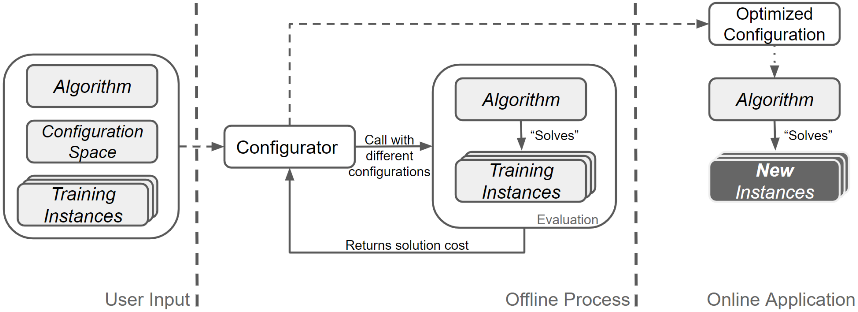
Outline

1. What is Algorithm Configuration?
2. How can we solve AC?
3. What to watch out for?
4. Is Solving AC hard?
5. Can we learn how to solve AC?
6. Top-Down or Bottom-Up Censoring?
7. Evolving from Static to Dynamic
8. What's next?

Algorithm Configuration Visualized



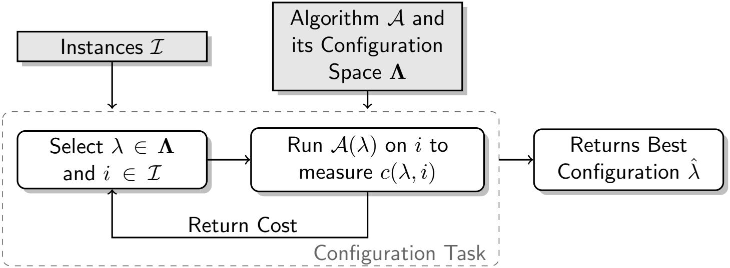
Algorithm Configuration Visualized



Offline tuning phase: Tune parameters on some training instances

Online application phase: Apply the found configuration to new instances

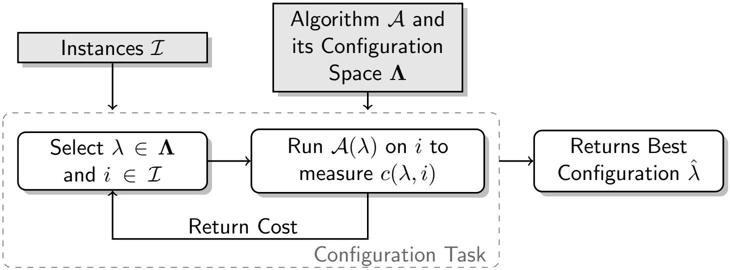
Algorithm Configuration – in More Detail



Definition

Given a parameterized algorithm $\mathcal{A}$ with possible (hyper-)parameter settings $\mathbf{\Lambda}$,

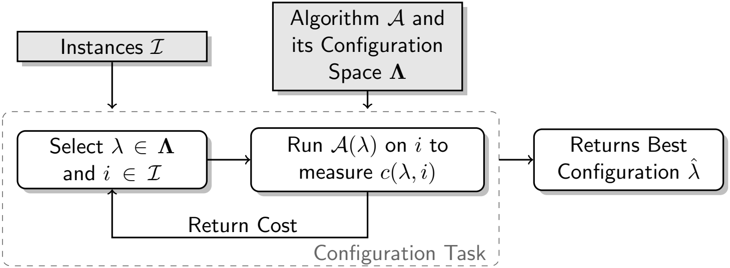
Algorithm Configuration – in More Detail



Definition

Given a parameterized algorithm $\mathcal{A}$ with possible (hyper-)parameter settings $\mathbf{\Lambda}$, a set of training problem instances $\mathcal{I}$,

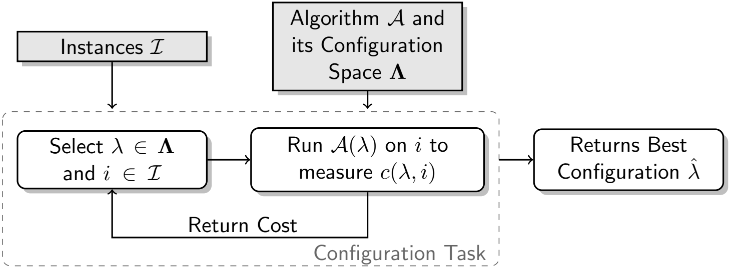
Algorithm Configuration – in More Detail



Definition

Given a parameterized algorithm $\mathcal{A}$ with possible (hyper-)parameter settings $\mathbf{\Lambda}$, a set of training problem instances $\mathcal{I}$, and a cost metric $c : \mathbf{\Lambda} \times \mathcal{I} \to \mathbb{R}$,

Algorithm Configuration – in More Detail



Definition

Given a parameterized algorithm $\mathcal{A}$ with possible (hyper-)parameter settings $\mathbf{\Lambda}$, a set of training problem instances $\mathcal{I}$, and a cost metric $c : \mathbf{\Lambda} \times \mathcal{I} \to \mathbb{R}$, the algorithm configuration problem is to find a (hyper-)parameter configuration $\lambda^* \in \mathbf{\Lambda}$ that minimizes c across the instances in $\mathcal{I}$.

Challenges of Algorithm Configuration

- Structured high-dimensional parameter space
  - categorical vs. continuous parameters
  - conditionals between parameters
  - between 5 and > 300 parameters
  - low effective dimensionality

Challenges of Algorithm Configuration

- Structured high-dimensional parameter space
  - categorical vs. continuous parameters
  - conditionals between parameters
  - between 5 and > 300 parameters
  - low effective dimensionality
- Stochastic optimization
  - Randomized algorithms: optimization across various seeds
  - Distribution of benchmark instances (often wide range of hardness)
  - Subsumes so-called *multi-armed bandit problem*

Challenges of Algorithm Configuration

- Structured high-dimensional parameter space
  - categorical vs. continuous parameters
  - conditionals between parameters
  - between 5 and > 300 parameters
  - low effective dimensionality

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

- Generalization across instances
  - apply algorithm configuration to homogeneous instance sets
  - Instance sets can also be heterogeneous,
    i.e., no single configuration performs well on all instances

  $\rightsquigarrow$ combination of algorithm configuration and selection

Challenges of Algorithm Configuration

- Structured high-dimensional parameter space
  - categorical vs. continuous parameters
  - conditionals between parameters
  - between 5 and > 300 parameters
  - low effective dimensionality

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

- Generalization across instances
  - apply algorithm configuration to homogeneous instance sets
  - Instance sets can also be heterogeneous,
    i.e., no single configuration performs well on all instances

  $\rightsquigarrow$ combination of algorithm configuration and selection

$\rightsquigarrow$ Hyperparameter optimization is a subproblem of algorithm configuration

[Eggensperger et al. 2019]

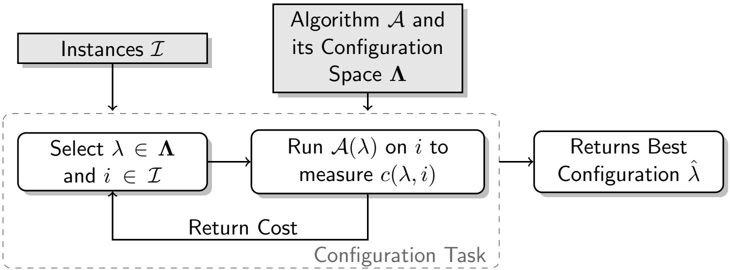
Let's configure some algorithms! (I)

Use this Google CoLab link to configure some example algorithms.

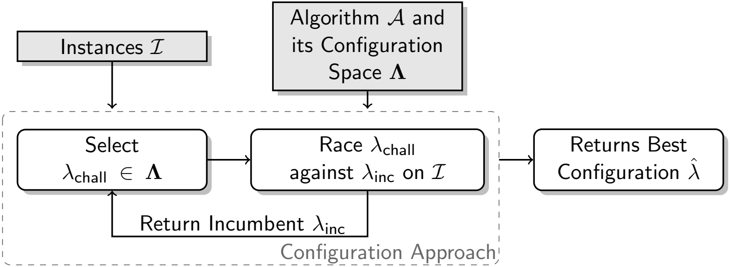
Outline

1. What is Algorithm Configuration?
2. How can we solve AC?
3. What to watch out for?
4. Is Solving AC hard?
5. Can we learn how to solve AC?
6. Top-Down or Bottom-Up Censoring?
7. Evolving from Static to Dynamic
8. What's next?

The AC Problem Again

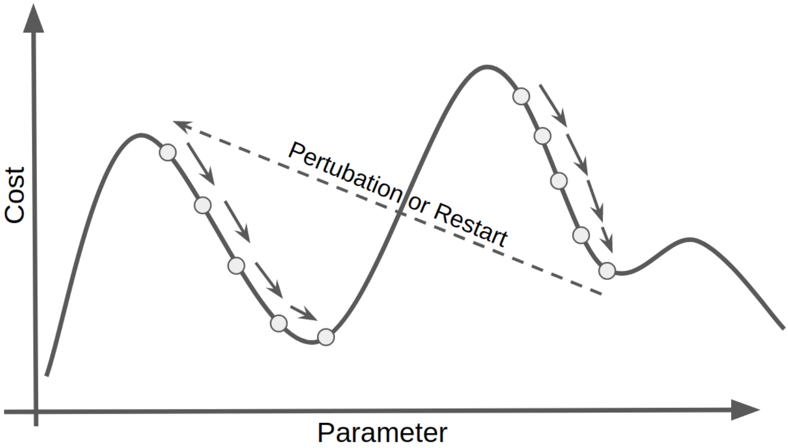


The Two Main Components



Sampling of Challengers: Iterated Local Search

[Hutter et al. 2007]



Sampling of Challengers: Bayesian Optimization

[Hutter et al. 2011]

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

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

```
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);
    Update data: 𝒟^(t) ← 𝒟^(t−1) ∪ {⟨λ, c(λ)⟩}
```

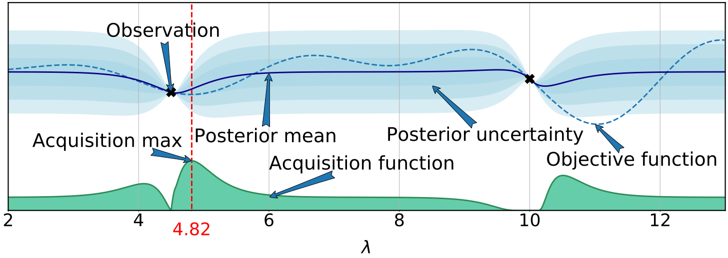
Initialize data $\mathcal{D}^{(0)}$ with initial observations

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

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

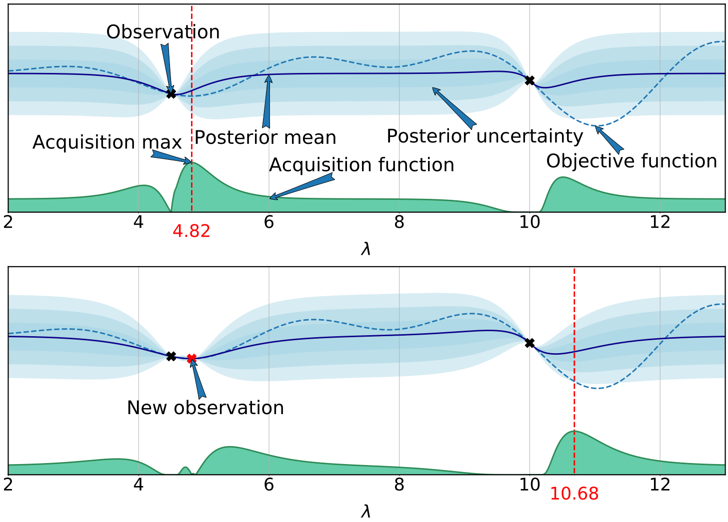
Sampling of Challengers: Bayesian Optimization

[Hutter et al. 2011]



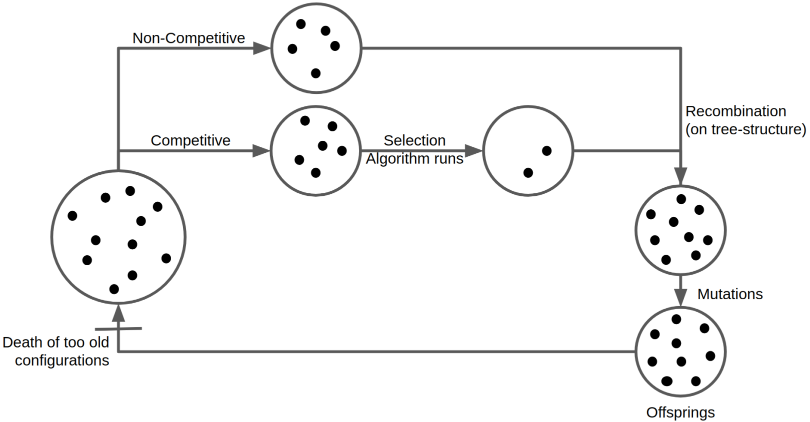
Sampling of Challengers: Bayesian Optimization

[Hutter et al. 2011]



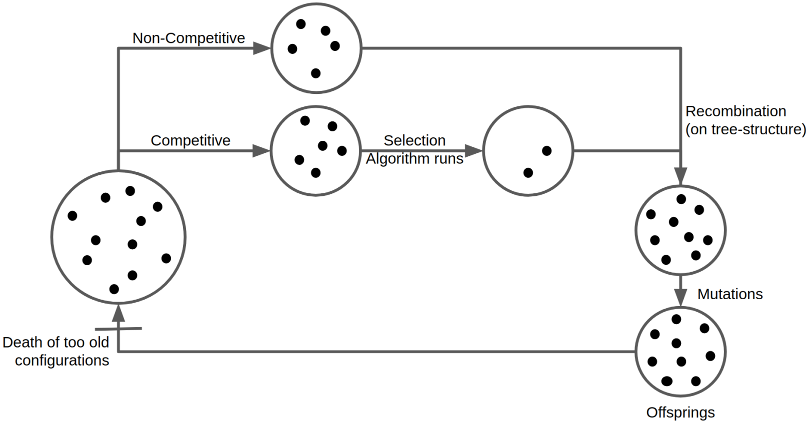
Sampling of Challengers: Genetic Algorithms

[Ansótegui et al. 2009]



Sampling of Challengers: Genetic Algorithms

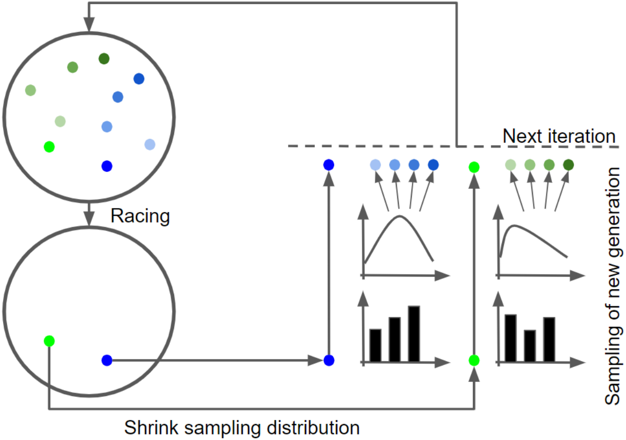
[Ansótegui et al. 2009]



- can be extended by using a surrogate model [Ansótegui et al. 2015]

Sampling of Challengers: Est. of Distributions

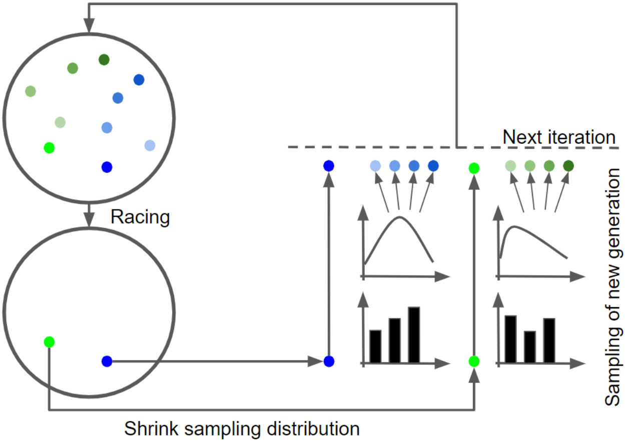
[López-Ibáñez et al. 2016]



- the sampling distribution around well-performing configurations gets more peaked after each iteration

Sampling of Challengers: Est. of Distributions

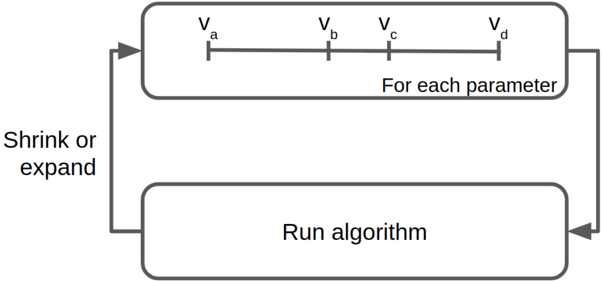
[López-Ibáñez et al. 2016]



- the sampling distribution around well-performing configurations gets more peaked after each iteration
- can be extended by using a surrogate model [Pérez Cáceres et al. 2017]

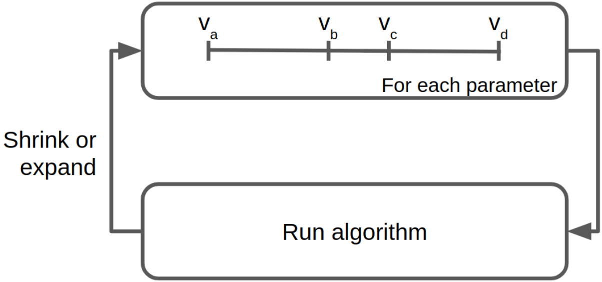
Sampling of Challengers: Golden Search

[Pushak and Hoos. 2020]



Sampling of Challengers: Golden Search

[Pushak and Hoos. 2020]



Assumes:

- uni-modal & convex search space
- independent parameters

$\rightsquigarrow$ more often applicable than one might expect [Pushak and Hoos. 2020]

Let's configure some algorithms! (II)

Use this Google CoLab link to configure some example algorithms.

Racing of Challengers: Main Idea

Observations

Obs I : We cannot afford to evaluate all configurations $\lambda \in \mathbf{\Lambda}$ on all instances $\mathcal{I}$

Racing of Challengers: Main Idea

Observations

Obs I : We cannot afford to evaluate all configurations $\lambda \in \mathbf{\Lambda}$ on all instances $\mathcal{I}$

Obs II : We do not want to waste time on poor λ

Racing of Challengers: Main Idea

Observations

- Obs I : We cannot afford to evaluate all configurations $\lambda \in \mathbf{\Lambda}$ on all instances $\mathcal{I}$
- Obs II : We do not want to waste time on poor λ
- Obs III : We need enough empirical evidence to distinguish between well performing λs

Racing of Challengers: Main Idea

Observations

Obs I : We cannot afford to evaluate all configurations $\lambda \in \mathbf{\Lambda}$ on all instances $\mathcal{I}$

Obs II : We do not want to waste time on poor λ

Obs III : We need enough empirical evidence to distinguish between well performing λs

Idea

Idea I : Discard poorly performing λs early on

Idea II : Run promising λs on many instances

Selection of Challengers: Aggressive Racing

[Hutter et al. 2009]

Race a challenger configuration λ against incumbent configuration $\hat{\lambda}$:

Selection of Challengers: Aggressive Racing

[Hutter et al. 2009]

Race a challenger configuration λ against incumbent configuration $\hat{\lambda}$:

Reject Challenger

Reject λ if

$$\sum_{i \in \mathcal{I}'} c(\lambda, i) > \sum_{i \in \mathcal{I}'} c(\hat{\lambda}, i)$$

where $\mathcal{I}'$ are the instances λ was evaluated on

Selection of Challengers: Aggressive Racing

[Hutter et al. 2009]

Race a challenger configuration λ against incumbent configuration $\hat{\lambda}$:

Reject Challenger

Reject λ if

$$\sum_{i \in \mathcal{I}'} c(\lambda, i) > \sum_{i \in \mathcal{I}'} c(\hat{\lambda}, i)$$

where $\mathcal{I}'$ are the instances λ was evaluated on

Update Incumbent

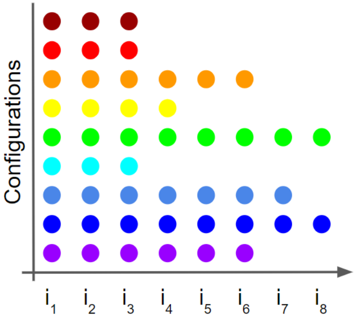
Update $\hat{\lambda} \rightarrow \lambda$ if

(i) λ was evaluated on the same instances as $\hat{\lambda}$ (i.e. same evidence level)

(ii) $\sum_{i \in \mathcal{I}'} c(\lambda, i) \leq \sum_{i \in \mathcal{I}'} c(\hat{\lambda}, i)$

Selection of Challengers: Statistical testing

[López-Ibáñez et al. 2016]

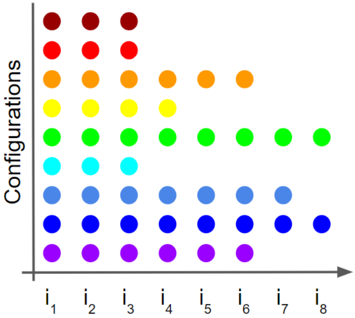


Objective: Decide from a set of configurations $\hat{\mathbf{\Lambda}}$ which one(s) is the best

1. Evaluate all $\lambda \in \hat{\mathbf{\Lambda}}$ on one or more additional instances (i.e., add more empirical evidence)

Selection of Challengers: Statistical testing

[López-Ibáñez et al. 2016]

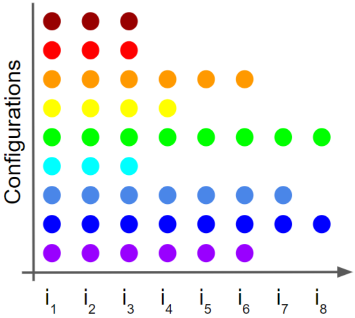


Objective: Decide from a set of configurations $\hat{\mathbf{\Lambda}}$ which one(s) is the best

1. Evaluate all $\lambda \in \hat{\mathbf{\Lambda}}$ on one or more additional instances (i.e., add more empirical evidence)
2. Drop all configurations λ from $\hat{\mathbf{\Lambda}}$ that perform statistically worse than the best configuration $\hat{\lambda} \in \hat{\mathbf{\Lambda}}$

Selection of Challengers: Statistical testing

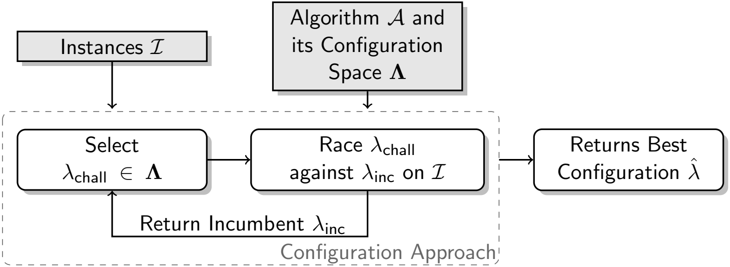
[López-Ibáñez et al. 2016]



Objective: Decide from a set of configurations $\hat{\mathbf{\Lambda}}$ which one(s) is the best

1. Evaluate all $\lambda \in \hat{\mathbf{\Lambda}}$ on one or more additional instances
   (i.e., add more empirical evidence)
2. Drop all configurations λ from $\hat{\mathbf{\Lambda}}$ that perform statistically worse than the best configuration $\hat{\lambda} \in \hat{\mathbf{\Lambda}}$
3. Continue with 1. if budget remains and $|\hat{\mathbf{\Lambda}}| > 1$

The Two Main Components



Many systems combining these two components in different ways, incl.

[Hutter et al. 2009, Ansótegui et al. 2009, Hutter et al. 2011, López-Ibáñez et al. 2016, Pushak and Hoos. 2020]

Let's configure some algorithms! (III)

Use this Google CoLab link to configure some example algorithms.

Outline

1. What is Algorithm Configuration?
2. How can we solve AC?
3. What to watch out for?
4. Is Solving AC hard?
5. Can we learn how to solve AC?
6. Top-Down or Bottom-Up Censoring?
7. Evolving from Static to Dynamic
8. What's next?

Pitfalls & Best Practices

The goals of automated algorithm configuration include:

Pitfalls & Best Practices

The goals of automated algorithm configuration include:

1. reducing the expertise required to use an algorithm

Pitfalls & Best Practices

The goals of automated algorithm configuration include:

1. reducing the expertise required to use an algorithm
2. less human-time

Pitfalls & Best Practices

The goals of automated algorithm configuration include:

1. reducing the expertise required to use an algorithm
2. less human-time
3. tuning algorithms to the task at hand

Pitfalls & Best Practices

The goals of automated algorithm configuration include:

1. reducing the expertise required to use an algorithm
2. less human-time
3. tuning algorithms to the task at hand
4. faster development of algorithms

Pitfalls & Best Practices

The goals of automated algorithm configuration include:

1. reducing the expertise required to use an algorithm
2. less human-time
3. tuning algorithms to the task at hand
4. faster development of algorithms
5. facilitating systematic and reproducible research

BUT:

1. algorithm configuration can lead to over-tuning

Pitfalls & Best Practices

The goals of automated algorithm configuration include:

1. reducing the expertise required to use an algorithm
2. less human-time
3. tuning algorithms to the task at hand
4. faster development of algorithms
5. facilitating systematic and reproducible research

BUT:

1. algorithm configuration can lead to over-tuning
2. using algorithm configuration requires (at least some) expertise in algorithm configuration

Pitfalls & Best Practices

The goals of automated algorithm configuration include:

1. reducing the expertise required to use an algorithm
2. less human-time
3. tuning algorithms to the task at hand
4. faster development of algorithms
5. facilitating systematic and reproducible research

BUT:

1. algorithm configuration can lead to over-tuning
2. using algorithm configuration requires (at least some) expertise in algorithm configuration
3. if done wrong, waste of time and compute resources

Setting up AC

9 Steps to your well-performing algorithm:

1. Define your cost metric

Setting up AC

9 Steps to your well-performing algorithm:

1. Define your cost metric
2. Define instance set

Setting up AC

9 Steps to your well-performing algorithm:

1. Define your cost metric
2. Define instance set
3. Split your instances in training and test

Setting up AC

9 Steps to your well-performing algorithm:

1. Define your cost metric
2. Define instance set
3. Split your instances in training and test
4. Define the configuration space

Setting up AC

9 Steps to your well-performing algorithm:

1. Define your cost metric
2. Define instance set
3. Split your instances in training and test
4. Define the configuration space
5. Choose your preferred configurator

Setting up AC

9 Steps to your well-performing algorithm:

1. Define your cost metric
2. Define instance set
3. Split your instances in training and test
4. Define the configuration space
5. Choose your preferred configurator
6. Implement an interface between your algorithm and the configurator

Setting up AC

9 Steps to your well-performing algorithm:

1. Define your cost metric
2. Define instance set
3. Split your instances in training and test
4. Define the configuration space
5. Choose your preferred configurator
6. Implement an interface between your algorithm and the configurator
7. Define resource limitations of your algorithm

Setting up AC

9 Steps to your well-performing algorithm:

1. Define your cost metric
2. Define instance set
3. Split your instances in training and test
4. Define the configuration space
5. Choose your preferred configurator
6. Implement an interface between your algorithm and the configurator
7. Define resource limitations of your algorithm
8. Run the configurator on your algorithm and the training instances

Setting up AC

9 Steps to your well-performing algorithm:

1. Define your cost metric
2. Define instance set
3. Split your instances in training and test
4. Define the configuration space
5. Choose your preferred configurator
6. Implement an interface between your algorithm and the configurator
7. Define resource limitations of your algorithm
8. Run the configurator on your algorithm and the training instances
9. Validate the eventually returned configuration on your test instances

Pitfall 1: Trust your algorithm

We have encountered algorithms that

- ignored resource limitations
- returned wrong solutions
- even returned negative runtimes

Pitfall 1: Trust your algorithm

We have encountered algorithms that

- ignored resource limitations
- returned wrong solutions
- even returned negative runtimes

Best Practice 1: Never trust your algorithm

Explicitly check and use external software to:

1. ensure resource limitations
2. terminate your algorithm
3. verify returned solutions
4. measure cost

Pitfall 2: File System

Algorithm configurators ...

- often produce quite a lot of log files (e.g., for each algorithm run)
- are often used on HPC clusters with a shared file system

Pitfall 2: File System

Algorithm configurators ...

- often produce quite a lot of log files (e.g., for each algorithm run)
- are often used on HPC clusters with a shared file system

$\rightsquigarrow$ It's surprisingly easy to substantially slow down a shared file system

Pitfall 2: File System

Algorithm configurators ...

- often produce quite a lot of log files (e.g., for each algorithm run)
- are often used on HPC clusters with a shared file system

$\rightsquigarrow$ It's surprisingly easy to substantially slow down a shared file system

Best Practice 2: Don't use the Shared File System

To lower the burden the file system on a HPC cluster:

- design well which files are required and which are not
- use a local (SSD) disc

Pitfall 3: Over-tuning

It's easy to overtune to different aspects, including:

- training instances
- random seeds
- machine type

Pitfall 3: Over-tuning

It's easy to overtune to different aspects, including:

- training instances
- random seeds
- machine type

It can be overcome, e.g., by

- using larger instance sets
- tuning on the target hardware

$\rightsquigarrow$ Still this is very hard in practice

Pitfall 3: Over-tuning

It's easy to overtune to different aspects, including:

- training instances
- random seeds
- machine type

It can be overcome, e.g., by

- using larger instance sets
- tuning on the target hardware

$\rightsquigarrow$ Still this is very hard in practice

Best Practice 3: Check for Over-Tuning

Check for over-tuning by validating your final configuration on

- many random seeds
- a large set of unused test instances
- different hardware

More Pitfalls and Best Practices

... can be found in [Eggensperger et al. 2019]

Outline

1. What is Algorithm Configuration?
2. How can we solve AC?
3. What to watch out for?
4. Is Solving AC hard?
5. Can we learn how to solve AC?
6. Top-Down or Bottom-Up Censoring?
7. Evolving from Static to Dynamic
8. What's next?

Worst Case

In the worst case, AC problems are very hard:

1. very hard instances with long evaluation time
2. interactions between all parameters
3. multi-modal with many local optima
4. heterogeneous instance sets

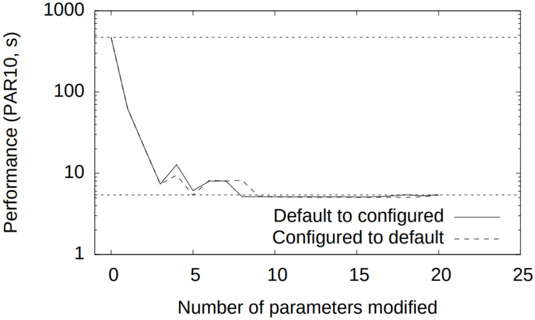
Worst Case

In the worst case, AC problems are very hard:

1. very hard instances with long evaluation time
2. interactions between all parameters
3. multi-modal with many local optima
4. heterogeneous instance sets

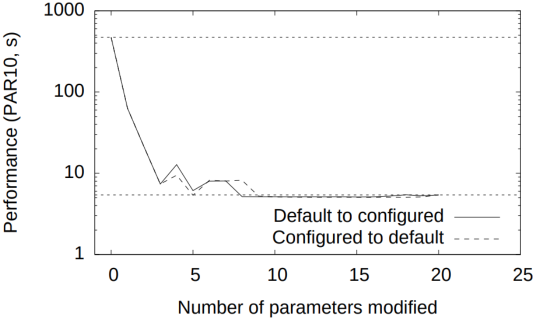
$\rightsquigarrow$ However, does this actually apply in practice?

Low Effective Dimensionality



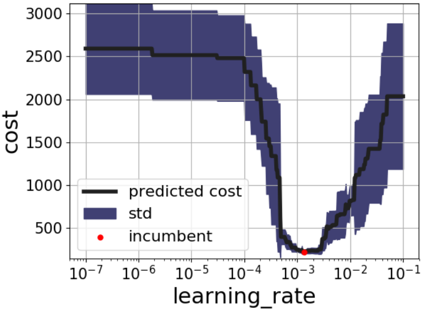
- Algorithms have often between tenths and hundreds of parameters
- Often only less than 10 matter ($\rightsquigarrow$ low effective dimenionality)
  [Bergstra and Bengio. 2012, Hutter et al. 2013, Hutter at el. 2014, Fawcet and Hoos. 2015, Biedenkapp et al. 2017]

Low Effective Dimensionality



- Algorithms have often between tenths and hundreds of parameters
- Often only less than 10 matter ($\rightsquigarrow$ low effective dimenionality)
  [Bergstra and Bengio. 2012, Hutter et al. 2013, Hutter at el. 2014, Fawcet and Hoos. 2015, Biedenkapp et al. 2017]
- Importance of parameter changes depending on instance set
  [Bergstra and Bengio. 2012, Biedenkapp et al. 2018]

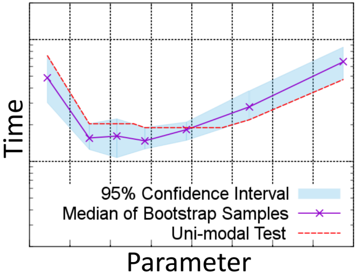
Uni-Modal? Convex? Smooth? (I)



- Intuition: parameters are either set to too high or too low
- We can analyse it for example by using fANOVA
  [Hutter at el. 2014, Biedenkapp et al. 2018]

PPO on cartpole [Lindauer et al. 2019]

Uni-Modal? Convex? Smooth? (II) [Pushak and Hoos. 2018]



Observations:

- On a single instances,
  the landscape might not be uni-modal, convex or smooth
- On average across many instances,
  the landscapes are often uni-modal, convex and smooth
  (at least on solvers for discrete combinatorial problems)

Homogeneity vs. Heterogeneity (I) [Schneider and Hoos. 2012]

Algorithm configurators...

- use racing to not evaluate each configuration on all instances

Homogeneity vs. Heterogeneity (I) [Schneider and Hoos. 2012]

Algorithm configurators...

- use racing to not evaluate each configuration on all instances
- misleading on subsets of instances if they are heterogeneous overall

Homogeneity vs. Heterogeneity (I) [Schneider and Hoos. 2012]

Algorithm configurators...

- use racing to not evaluate each configuration on all instances
- misleading on subsets of instances if they are heterogeneous overall

$\rightsquigarrow$ returned configurations often perform worse than default configurations in the validation phase

Homogeneity vs. Heterogeneity (II)

[Schneider and Hoos. 2012]

Rule of Thumb

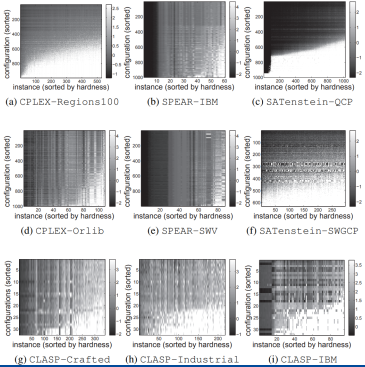
- An instance set is homogeneous if all instances encode the same task (or problem)
- An homogeneous instance can have instances of different hardness

Homogeneity vs. Heterogeneity (II)

[Schneider and Hoos. 2012]

Rule of Thumb

- An instance set is homogeneous if all instances encode the same task (or problem)
- An homogeneous instance can have instances of different hardness



(a) CPLEX-Regions100 (b) SPEAR-IBM (c) SATenstein-QCP (d) CPLEX-Orlib (e) SPEAR-SWV (f) SATenstein-SWGCP (g) CLASP-Crafted (h) CLASP-Industrial (i) CLASP-IBM

$\leftarrow$ smooth transition $\rightsquigarrow$ homogeneous

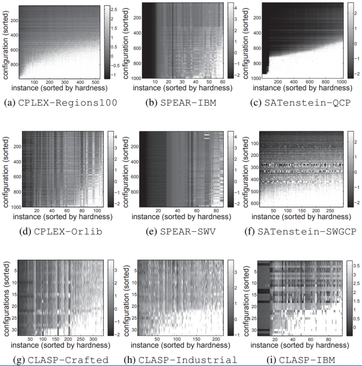
$\leftarrow$ check board pattern $\rightsquigarrow$ heterogeneous

Homogeneity vs. Heterogeneity (II)

[Schneider and Hoos. 2012]

Rule of Thumb

- An instance set is homogeneous if all instances encode the same task (or problem)
- An homogeneous instance can have instances of different hardness



(a) CPLEX-Regions100 (b) SPEAR-IBM (c) SATenstein-QCP
(d) CPLEX-Orlib (e) SPEAR-SWV (f) SATenstein-SWGCP
(g) CLASP-Crafted (h) CLASP-Industrial (i) CLASP-IBM

$\leftarrow$ smooth transition $\rightsquigarrow$ homogeneous

$\leftarrow$ check board pattern $\rightsquigarrow$ heterogeneous

$\rightsquigarrow$ Rule of thumb does not always apply!

Per-Instance Algorithm Configuration: Hydra

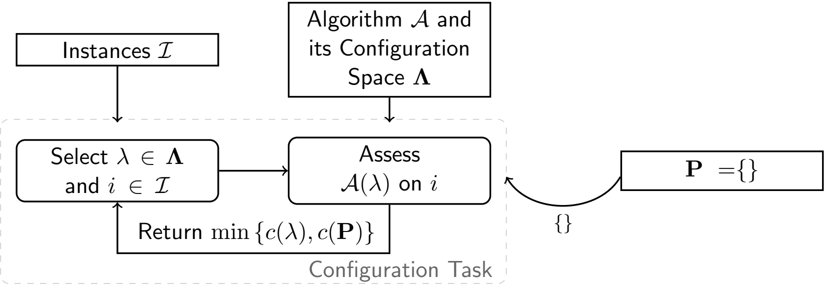
[Xu et al. 2010; Xu et al. 2011]

Idea: Find a configuration for each homogeneous subset of instances

Per-Instance Algorithm Configuration: Hydra

[Xu et al. 2010; Xu et al. 2011]

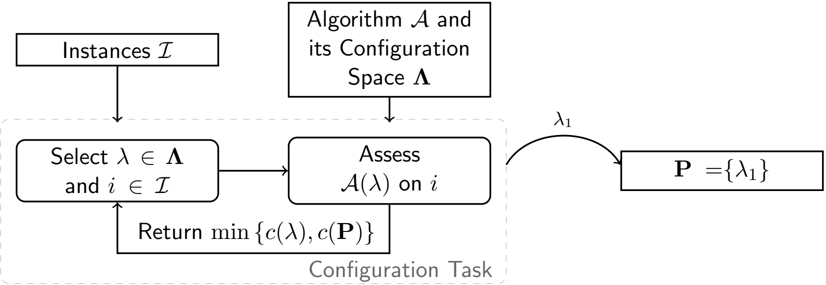
Idea: Find a configuration for each homogeneous subset of instances



Per-Instance Algorithm Configuration: Hydra

[Xu et al. 2010; Xu et al. 2011]

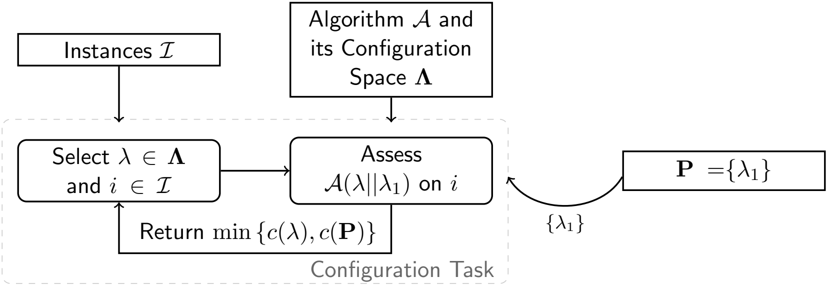
Idea: Find a configuration for each homogeneous subset of instances



Per-Instance Algorithm Configuration: Hydra

[Xu et al. 2010; Xu et al. 2011]

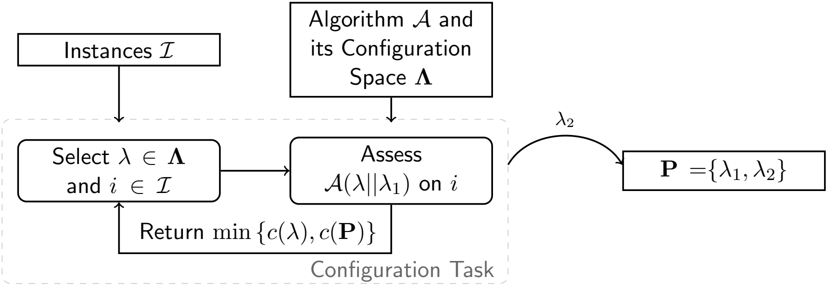
Idea: Find a configuration for each homogeneous subset of instances



Per-Instance Algorithm Configuration: Hydra

[Xu et al. 2010; Xu et al. 2011]

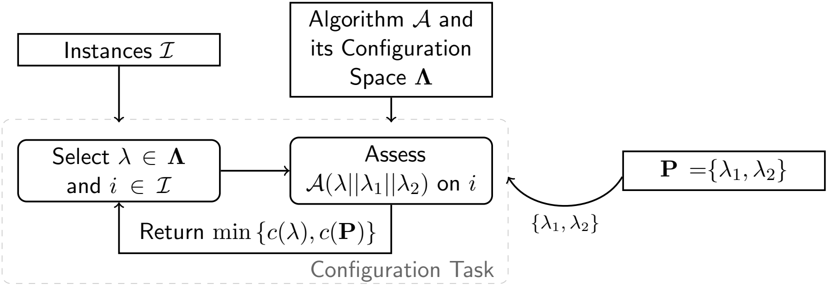
Idea: Find a configuration for each homogeneous subset of instances



Per-Instance Algorithm Configuration: Hydra

[Xu et al. 2010; Xu et al. 2011]

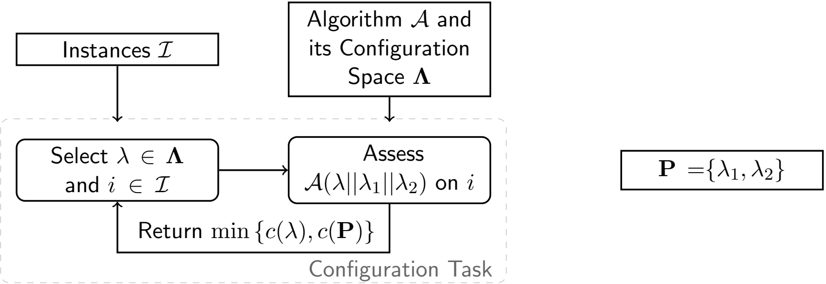
Idea: Find a configuration for each homogeneous subset of instances



Per-Instance Algorithm Configuration: Hydra

[Xu et al. 2010; Xu et al. 2011]

Idea: Find a configuration for each homogeneous subset of instances



Similar ideas:

- ISAC: 1. Cluster instances; 2. configure on each instance cluster [Kadioglu et al. 2010, Ansótegui et al. 2016]
- Combination of iterative configuration and clustering [Liu et al. 2019]

Outline

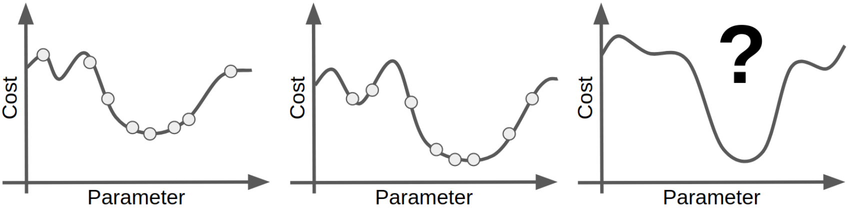
1. What is Algorithm Configuration?
2. How can we solve AC?
3. What to watch out for?
4. Is Solving AC hard?
5. Can we learn how to solve AC?
6. Top-Down or Bottom-Up Censoring?
7. Evolving from Static to Dynamic
8. What's next?

Solving AC more than once?

Assumption: We tune the parameters of the same algorithms again and again on different instance sets.

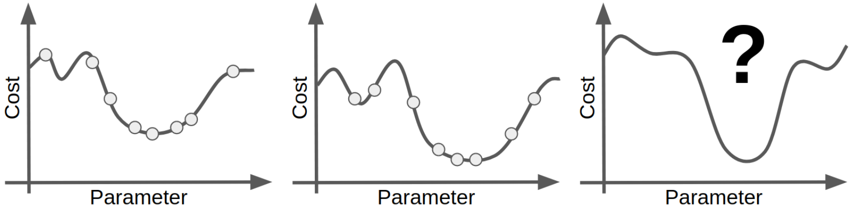
Solving AC more than once?

Assumption: We tune the parameters of the same algorithms again and again on different instance sets.



Solving AC more than once?

Assumption: We tune the parameters of the same algorithms again and again on different instance sets.



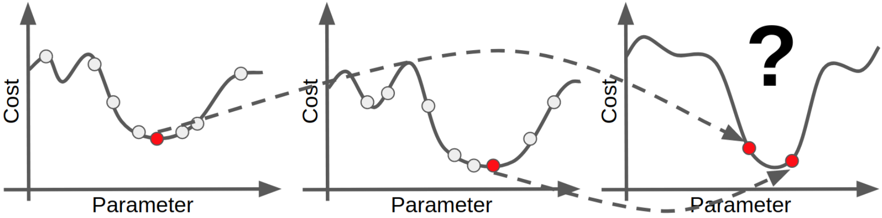
$\rightsquigarrow$ Learn on the first k instance sets,
how to optimize on the $k+1$ instance set.
(Common idea in the optimization community.)

Idea 1: Try the best ones [Lindauer and Hutter. 2018]

Idea: Take the best configurations of previous runs and try them as initial design on new instances.

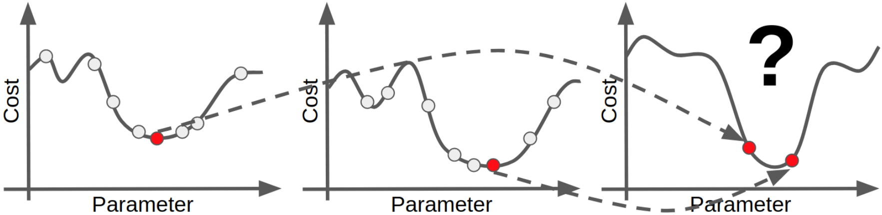
Idea 1: Try the best ones [Lindauer and Hutter. 2018]

Idea: Take the best configurations of previous runs and try them as initial design on new instances.



Idea 1: Try the best ones [Lindauer and Hutter. 2018]

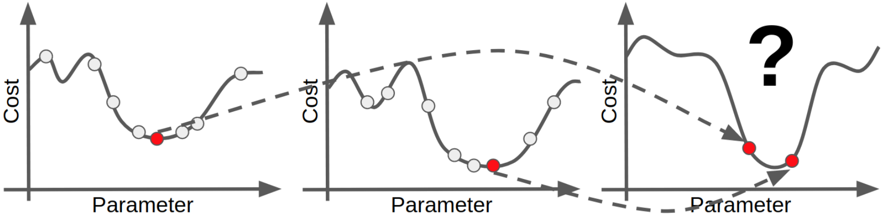
Idea: Take the best configurations of previous runs and try them as initial design on new instances.



Problem: If we have too many previous AC runs, the initial design will be too expensive.

Idea 1: Try the best ones [Lindauer and Hutter. 2018]

Idea: Take the best configurations of previous runs and try them as initial design on new instances.



Problem: If we have too many previous AC runs, the initial design will be too expensive.

Idea: Select complementary configurations ($\mathbf{\Lambda}'$) across all previous instance sets ($\bigcup_j \mathcal{I}_j$):

$$\sum_{i \in \bigcup_j \mathcal{I}_j} \min_{\lambda \in \mathbf{\Lambda}'} c(\lambda, i)$$

Idea 2: Combine Surrogate Models

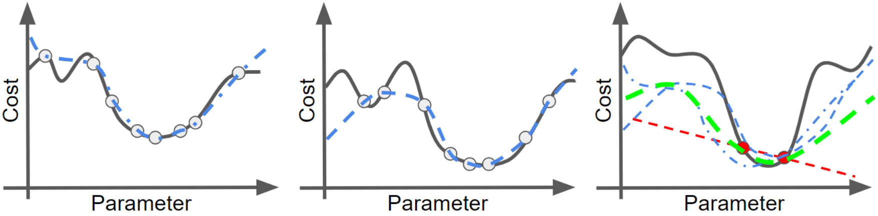
[Lindauer and Hutter. 2018]

Idea: Weighted combination of already trained surrogate models.

Idea 2: Combine Surrogate Models

[Lindauer and Hutter. 2018]

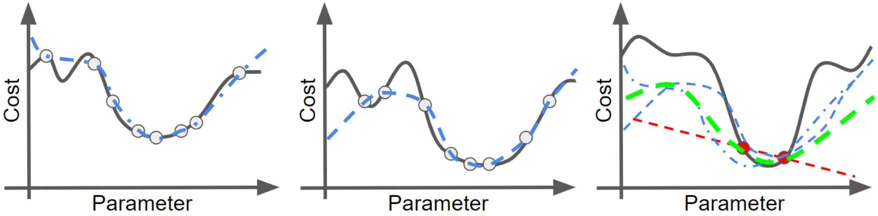
Idea: Weighted combination of already trained surrogate models.



Idea 2: Combine Surrogate Models

[Lindauer and Hutter. 2018]

Idea: Weighted combination of already trained surrogate models.



$$\hat{c}(\lambda, i) = w_0 + w_c \cdot \hat{c}_c(\lambda, i) + \sum_j w_j \cdot \hat{c}_j(\lambda, i)$$

$\rightsquigarrow$ Fit linear combination wrt $\mathbf{w}$ on hold-out set of the current obs.

Insights from Warmstarting [Lindauer and Hutter. 2018]

1. If landscape changed too much, AC should be able to recover.
   - both methods can do that

Insights from Warmstarting [Lindauer and Hutter. 2018]

1. If landscape changed too much, AC should be able to recover.
   - both methods can do that
2. Weights of combined surrogate models, prefer previous surrogates first, and later on the surrogate fitted on the current data.

Insights from Warmstarting [Lindauer and Hutter. 2018]

1. If landscape changed too much, AC should be able to recover.
   - both methods can do that
2. Weights of combined surrogate models, prefer previous surrogates first, and later on the surrogate fitted on the current data.
3. Combining both ideas lead to the best speedups.

Insights from Warmstarting [Lindauer and Hutter. 2018]

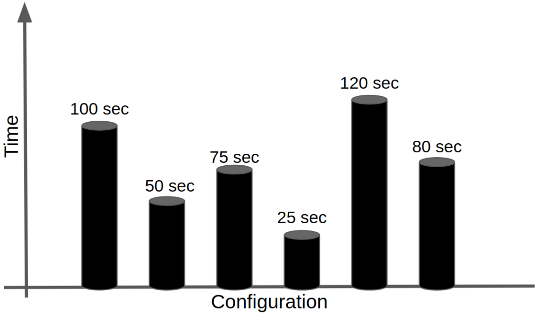
1. If landscape changed too much, AC should be able to recover.
   - both methods can do that
2. Weights of combined surrogate models, prefer previous surrogates first, and later on the surrogate fitted on the current data.
3. Combining both ideas lead to the best speedups.
4. We sped up SMAC by a factor of 4.3 on average and up to a factor of 165.

Outline

1. What is Algorithm Configuration?
2. How can we solve AC?
3. What to watch out for?
4. Is Solving AC hard?
5. Can we learn how to solve AC?
6. Top-Down or Bottom-Up Censoring?
7. Evolving from Static to Dynamic
8. What's next?

Running until the End?

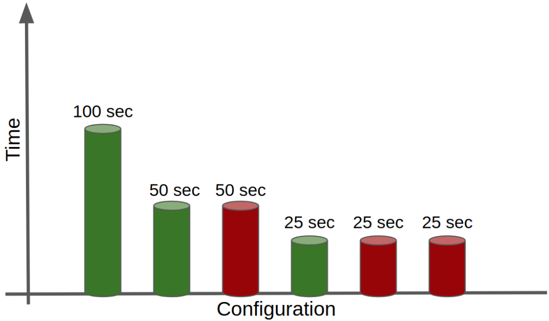
- Assumption: we would like to minimize runtime of algorithms



$\rightsquigarrow$ overall invested time: 450 sec

Top-Down Capping [Hutter et al. 2009]

- Assumption: we would like to minimize runtime of algorithms

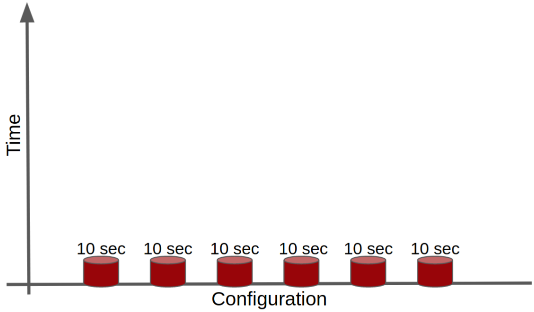


$\rightsquigarrow$ overall invested time: 275 sec (instead of 450 sec)

Bottom-Up Procrastination

[Kleinberg et al. 2017, Kleinberg et al. 2019]

- Assumption: we would like to minimize runtime of algorithms



⇝ overall invested time: 225 sec (instead of 450 and 275 sec)

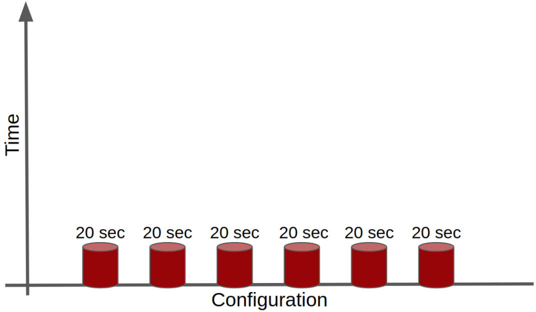
- Generalize this idea to evaluate configurations on pairs of random seeds and instances

⇝ performance guarantees of AC methods

Bottom-Up Procrastination

[Kleinberg et al. 2017, Kleinberg et al. 2019]

- Assumption: we would like to minimize runtime of algorithms



⇝ overall invested time: 225 sec (instead of 450 and 275 sec)

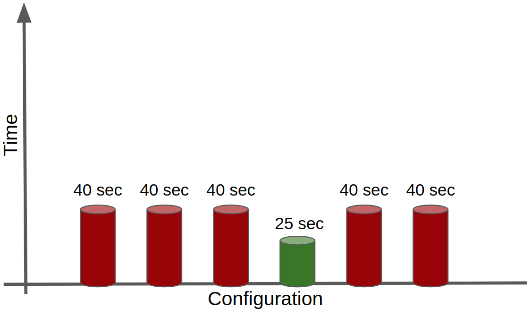
- Generalize this idea to evaluate configurations on pairs of random seeds and instances

⇝ performance guarantees of AC methods

Bottom-Up Procrastination

[Kleinberg et al. 2017, Kleinberg et al. 2019]

- Assumption: we would like to minimize runtime of algorithms



⇝ overall invested time: 225 sec (instead of 450 and 275 sec)

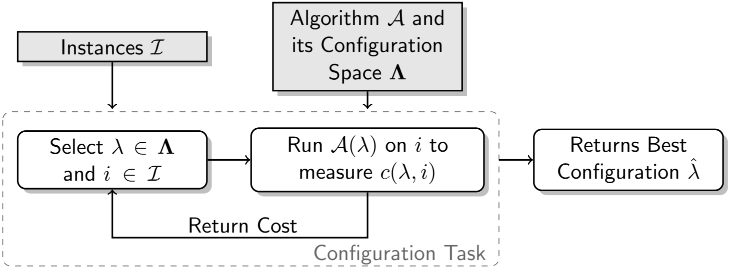
- Generalize this idea to evaluate configurations on pairs of random seeds and instances

⇝ performance guarantees of AC methods

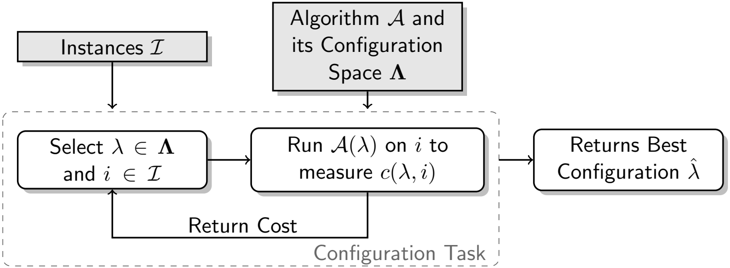
Outline

1. What is Algorithm Configuration?
2. How can we solve AC?
3. What to watch out for?
4. Is Solving AC hard?
5. Can we learn how to solve AC?
6. Top-Down or Bottom-Up Censoring?
7. Evolving from Static to Dynamic
8. What's next?

Algorithm Configuration Recap



Algorithm Configuration Recap



- Important observation: Many algorithms are iterative by design

$\rightsquigarrow$ fixed configurations are not optimal for each iteration

Algorithms have Dynamic Parameters!

Evolutionary Algorithms population size, selection strategies, mutation operators, recombination operators, step size, probabilities, ...

Algorithms have Dynamic Parameters!

Evolutionary Algorithms population size, selection strategies, mutation operators, recombination operators, step size, probabilities, ...

Algorithms have Dynamic Parameters!

Evolutionary Algorithms: population size, selection strategies, mutation operators, recombination operators, step size, probabilities, ...

Satisfiability Solving: Search strategies (local vs. tree-based), variable selection heuristic, restart probability, deletion heuristics, ...

Algorithms have Dynamic Parameters!

Evolutionary Algorithms: population size, selection strategies, mutation operators, recombination operators, step size, probabilities, ...

Satisfiability Solving: Search strategies (local vs. tree-based), variable selection heuristic, restart probability, deletion heuristic, ...

Algorithms have Dynamic Parameters!

Evolutionary Algorithms population size, selection strategies, mutation operators, recombination operators, step size, probabilities, . . .

Satisfiability Solving Search strategies (local vs. tree-based), variable selection heuristic, restart probability, deletion heuristic, . . .

Machine Learning model class (linear, non-linear, tree, ...), model complexity, regularization, preprocessing, data augmentation, . . .

Algorithms have Dynamic Parameters!

Evolutionary Algorithms population size, selection strategies, mutation operators, recombination operators, step size, probabilities, ...

Satisfiability Solving Search strategies (local vs. tree-based), variable selection heuristic, restart probability, deletion heuristic, ...

Machine Learning model class (linear, non-linear, tree, ...), model complexity, regularization, preprocessing, data augmentation, ...

Algorithms have Dynamic Parameters!

Evolutionary Algorithms population size, selection strategies, mutation operators, recombination operators, step size, probabilities, ...

Satisfiability Solving Search strategies (local vs. tree-based), variable selection heuristic, restart probability, deletion heuristic, ...

Machine Learning model class (linear, non-linear, tree, ...), model complexity, regularization, preprocessing, data augmentation, ...

Deep Learning network depth, network width, network architecture, activation function, optimizer, learning rate, momemtum, ...

Algorithms have Dynamic Parameters!

Evolutionary Algorithms population size, selection strategies, mutation operators, recombination operators, step size, probabilities, ...

Satisfiability Solving Search strategies (local vs. tree-based), variable selection heuristic, restart probability, deletion heuristic, ...

Machine Learning model class (linear, non-linear, tree, ...), model complexity, regularization, preprocessing, data augmentation, ...

Deep Learning network depth, network width, network architecture, activation function, optimizer, learning rate, momemtum, ...

Algorithms have Dynamic Parameters!

Evolutionary Algorithms[1] population size, selection strategies, mutation operators, recombination operators, step size, probabilities, ...

Satisfiability Solving Search strategies (local vs. tree-based), variable selection heuristic, restart probability, deletion heuristic, ...

Machine Learning model class (linear, non-linear, tree, ...), model complexity, regularization, preprocessing, data augmentation, ...

Deep Learning network depth, network width, network architecture, activation function, optimizer, learning rate, momemtum, ...

[1] Gregor Papa: "Dynamic Parameter Changing During the Run to Speed Up Evolutionary Algorithms"

What Can We Do? (I)

🤔 Manually design heuristics to adapt the parameter online.

- Requires substantial expert knowledge

$\rightsquigarrow$ specialized heuristics for specific domains

- Very time-consuming

What Can We Do? (I)

🤔 Manually design heuristics to adapt the parameter online.

- Requires substantial expert knowledge

$\rightsquigarrow$ specialized heuristics for specific domains

- Very time-consuming

🤔 Use Algorithm Configuration/Selection to choose a heuristic.

- Limited approach

$\rightsquigarrow$ does not make use of information during the algorithms execution

What Can We Do? (II)

🤔 Learn to configure from scratch

- Requires access to the algorithms internal statistics
- Data-driven approach

What Can We Do? (II)

🤔 Learn to configure from scratch

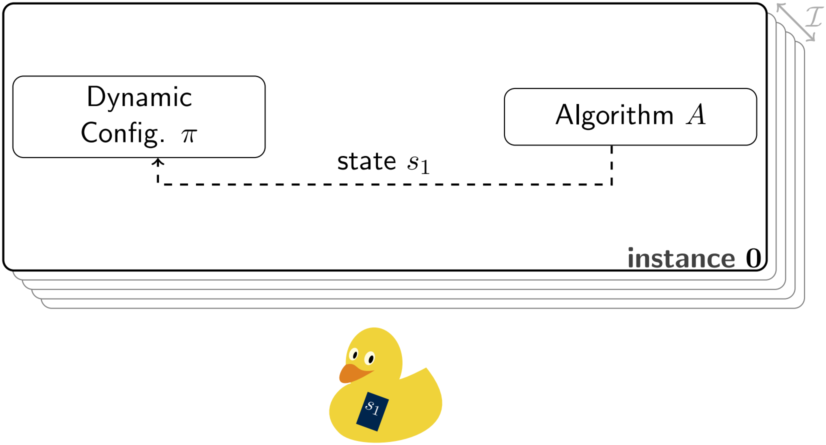
- Requires access to the algorithms internal statistics
- Data-driven approach

🤔 Warmstart from expert knowledge

- Potentially more sample efficient
- Which expert should we learn form?

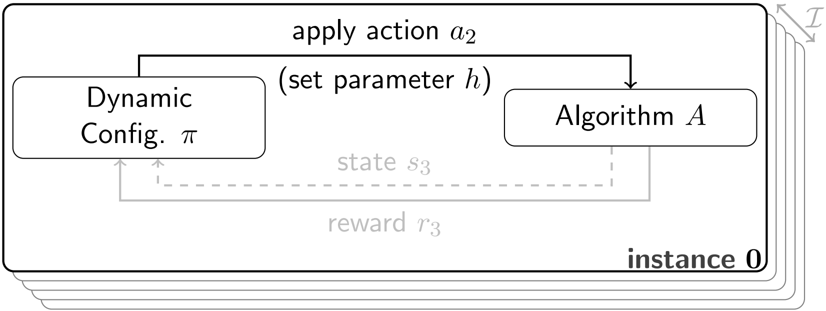
Dynamic Algorithm Configuration

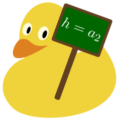
[Biedenkapp et al. 2020]



Dynamic Algorithm Configuration

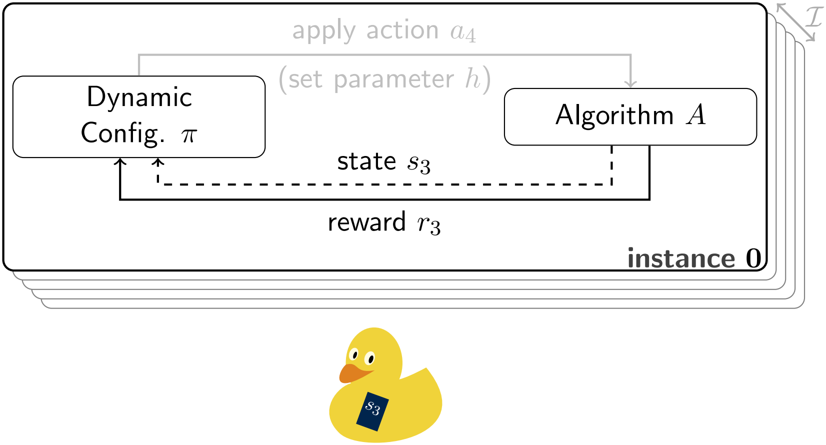
[Biedenkapp et al. 2020]





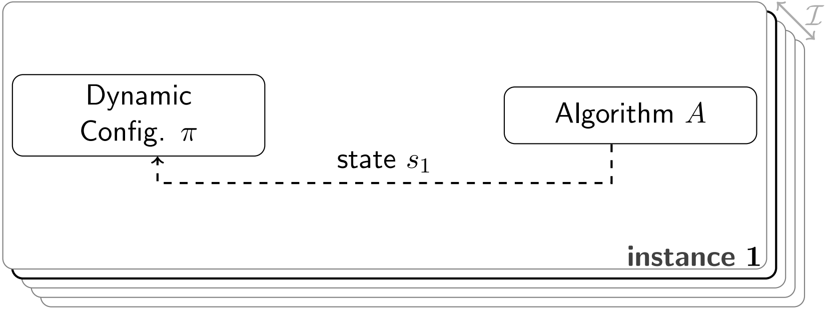
Dynamic Algorithm Configuration

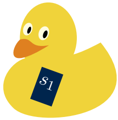
[Biedenkapp et al. 2020]



Dynamic Algorithm Configuration

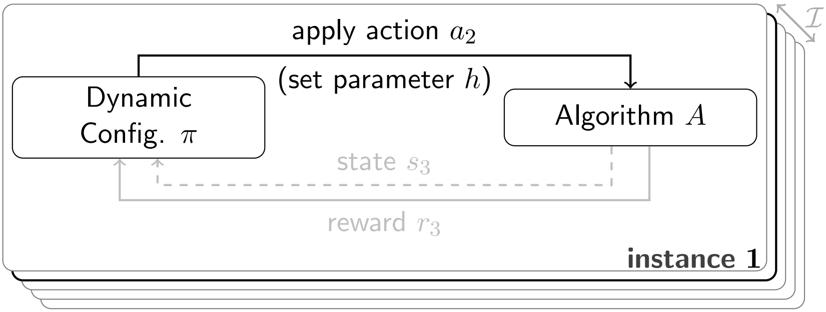
[Biedenkapp et al. 2020]

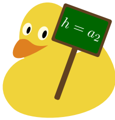




Dynamic Algorithm Configuration

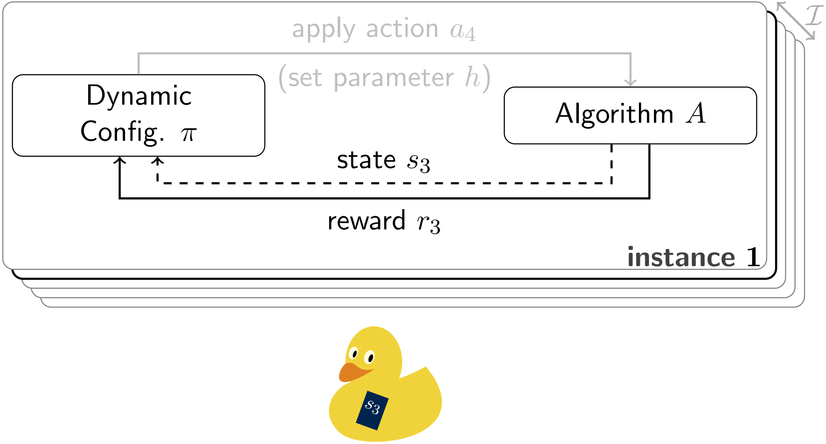
[Biedenkapp et al. 2020]





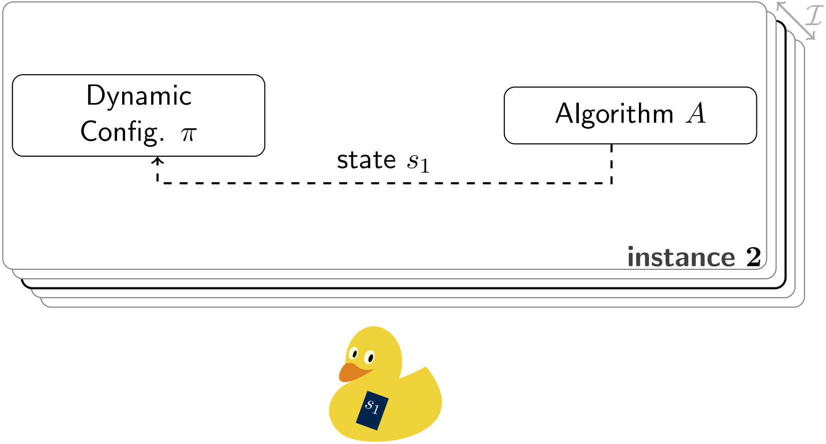
Dynamic Algorithm Configuration

[Biedenkapp et al. 2020]



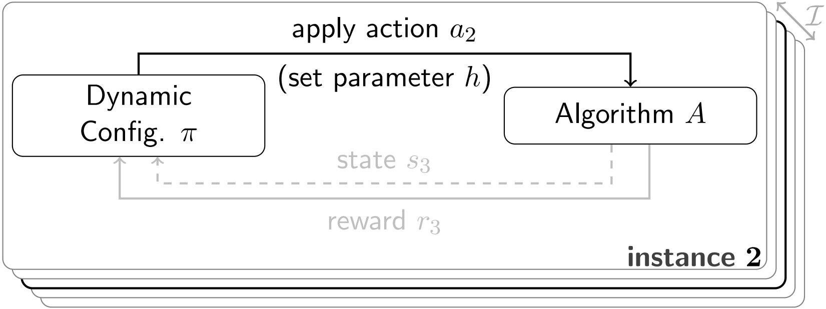
Dynamic Algorithm Configuration

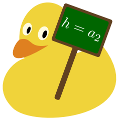
[Biedenkapp et al. 2020]



Dynamic Algorithm Configuration

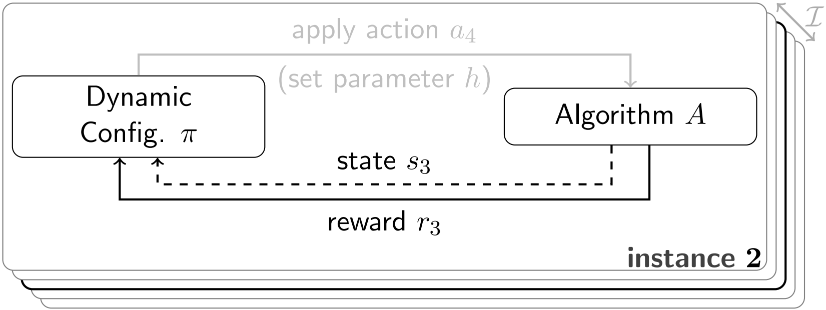
[Biedenkapp et al. 2020]

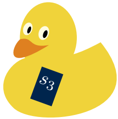




Dynamic Algorithm Configuration

[Biedenkapp et al. 2020]





$\rightsquigarrow$ How can DAC look inside the algorithm? I.e. what is a state?

Looking Inside Algorithms: Desiderata

1. Cheap to compute
2. Quantify the progress of the algorithm
3. Available at each decision point

Looking Inside Algorithms: Desiderata

1. Cheap to compute
2. Quantify the progress of the algorithm
3. Available at each decision point

$\rightsquigarrow$ Make use of internal statistics

Looking Inside Algorithms

- EAs
  - population fitness [Sharma et al. 2019, Shala et al. 2020]
  - stdev population fitness [Sharma et al. 2019]
  - cumulative evolution path length [Shala et al. 2020]
  - ...
- AI Planning [Speck et al. 2020]
  - average heuristic value
  - minimal heuristic value
  - #possible next planning states
  - ...
- NN Optimization [Daniel et al. 2016]
  - predictive change in function value
  - disagreement of function values
  - uncertainty
  - ...

Looking Inside Algorithms

- EAs $\rightsquigarrow$ better generalization
  - population fitness [Sharma et al. 2019, Shala et al. 2020]
  - stdev population fitness [Sharma et al. 2019]
  - cumulative evolution path length [Shala et al. 2020]
  - ...
- AI Planning [Speck et al. 2020] $\rightsquigarrow$ more solved instances
  - average heuristic value
  - minimal heuristic value
  - #possible next planning states
  - ...
- NN Optimization [Daniel et al. 2016] $\rightsquigarrow$ robust learning rates
  - predictive change in function value
  - disagreement of function values
  - uncertainty
  - ...

Looking Inside Algorithms

- EAs $\rightsquigarrow$ better generalization
  - population fitness [Sharma et al. 2019, Shala et al. 2020[2]]
  - stdev population fitness [Sharma et al. 2019]
  - cumulative evolution path length [Shala et al. 2020[2]]
  - ...
- AI Planning [Speck et al. 2020] $\rightsquigarrow$ more solved instances
  - average heuristic value
  - minimal heuristic value
  - #possible next planning states
  - ...
- NN Optimization [Daniel et al. 2016] $\rightsquigarrow$ robust learning rates
  - predictive change in function value
  - disagreement of function values
  - uncertainty
  - ...

[2] presented at PPSN in poster session 2

Outline

1. What is Algorithm Configuration?
2. How can we solve AC?
3. What to watch out for?
4. Is Solving AC hard?
5. Can we learn how to solve AC?
6. Top-Down or Bottom-Up Censoring?
7. Evolving from Static to Dynamic
8. What's next?

Next I: Neural Networks?

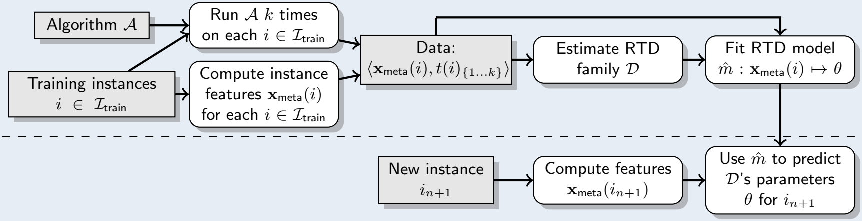
- 🤔 "everyone" uses neural networks these days.
  Models in algorithm configuration are still random forests?

Next I: Neural Networks?

"everyone" uses neural networks these days.
Models in algorithm configuration are still random forests?

Towards DNNs for AC (I): Prediction of Runtime Distributions

[Eggensperger et al. 2018]



Next II: Neural Networks?

🤔 How to handle censored data (from capping of runs) in DNNs?

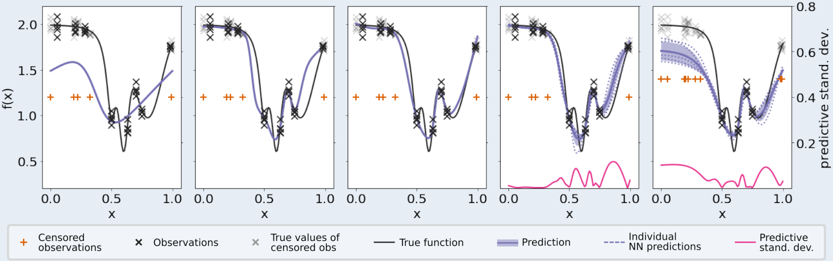
Next II: Neural Networks?

🤔 How to handle censored data (from capping of runs) in DNNs?

Towards DNNs for AC (II): Handling of Censored Data

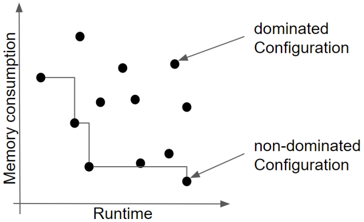
[Eggensperger et al. 2020]

$$-\log \mathcal{L}\left(\left(\hat{\mu}_i, \hat{\sigma}_i^2\right)_{i=1}^n \mid \mathcal{D}\right) = -\sum_{i=1}^{n} \log\left(\phi(Z_i)^{1-I_i}\left(1-\Phi(Z_i)\right)^{I_i}\right)$$



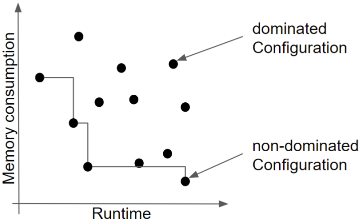
Next II: Multi-Objective Optimization?

🤔 Can we consider more than one objective?



Next II: Multi-Objective Optimization?

🤔 Can we consider more than one objective?



Challenges:

1. How to sample promising (non-dominated) configurations?
2. How to efficiently determine that a configuration is really on the Pareto front across all instances? [Blot et al. 2016]

Next III: What are interesting AC benchmarks?

🤔 I developed a new configurator,
but on which benchmarks should I benchmark it?

https://bitbucket.org/mlindauer/aclib2/

Next III: What are interesting AC benchmarks?

🤔 I developed a new configurator,
but on which benchmarks should I benchmark it?

AClib [Hutter et al. 2014]

- 84 AC scenarios
- 18 target algorithms and their configuration spaces
- 45 instance sets
- 6 AI domains
- 6 configurators

https://bitbucket.org/mlindauer/aclib2/

Next III: What are interesting AC benchmarks?

🤔 I developed a new configurator,
but on which benchmarks should I benchmark it?

AClib [Hutter et al. 2014]

- 84 AC scenarios
- 18 target algorithms and their configuration spaces
- 45 instance sets
- 6 AI domains
- 6 configurators

$\rightsquigarrow$ To make development cheaper, you can benchmark on surrogate benchmarks [Eggensperger et al. 2018]

$\rightsquigarrow$ Open which of these are interesting?

- not too easy, not too hard, diverse set, different characteristics, representative, ...

https://bitbucket.org/mlindauer/aclib2/

Take Home Message

1. Algorithm Configuration improves the performance of your algorithms

Take Home Message

1. Algorithm Configuration improves the performance of your algorithms
2. There are different state-of-the-art AC approaches

Take Home Message

1. Algorithm Configuration improves the performance of your algorithms
2. There are different state-of-the-art AC approaches
3. Using algorithm configuration is nevertheless fairly easy these days

Take Home Message

1. Algorithm Configuration improves the performance of your algorithms
2. There are different state-of-the-art AC approaches
3. Using algorithm configuration is nevertheless fairly easy these days
4. Many open questions for the research community to be answered

Thank you!

AutoML.org