Meta-algorithmics: Automated algorithm selection, configuration, and parallel portfolios

Koen van der Blom

kvdb@cwi.nl

2026-05-13

About me

- PhD from Leiden University
- Previously postdoc at Leiden University, then Sorbonne Université
- Now postdoc at Centrum Wiskunde & Informatica (CWI)

- Research
  - Meta-algorithmics, focus on algorithm selection for black-box optimisation
  - Multi-objective optimisation
  - Real-world adoption

What is needed for an optimisation problem?

- Some compute resources
- A problem representation
- An algorithm
- The algorithm and representation to work together

Can we do better than…

- **Some** compute resources
- **A** problem representation
- **An** algorithm
- The algorithm and representation **to work** together

What if we can

- Take advantage of the **specific** compute resources we have
- Choose a **high quality** problem representation
- Choose a **good algorithm** for this problem (representation)
- **Optimise** how the algorithm and representation interact

Focus of this lecture

- Take advantage of the **specific** compute resources we have

- ~~Choose a **high quality** problem representation~~
  - Not part of this lecture, but **very** important for performance!

- Choose a **good algorithm** for this problem (representation)

- **Optimise** how **the algorithm** and representation interact
  - Focus on algorithm alone means we can only get so far!

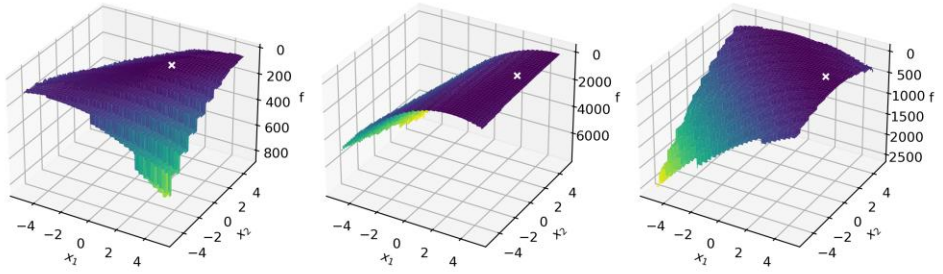
Focus of this lecture

- Use what we already have to the best possible effect
- No algorithm design, but configure already available settings
- Don’t create a new algorithm, but select the best one we have
- Don’t redesign an algorithm to parallelise it, but run copies in parallel

Main concepts

- Parallel algorithm portfolios
  - Run algorithm(s) in parallel to benefit from variance in performance
- Algorithm selection – Not to be confused with the selection operator!
  - Choose the best algorithm for the problem
- Algorithm configuration
  - Find the best parameters, e.g., mutation rate
  - But **also** find the best algorithm components, e.g., variation operator

Problem instances



Figures from https://coco-platform.org/

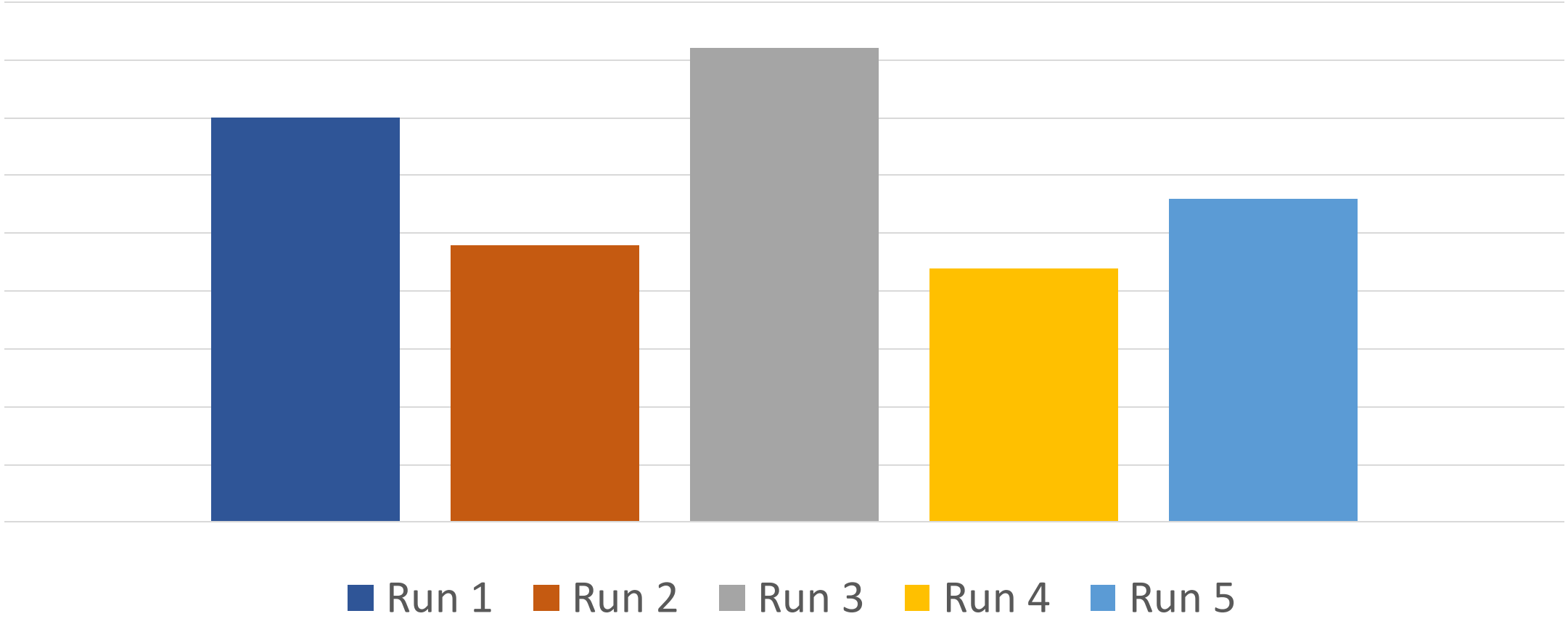
- Same high-level problem description
- Different details, e.g., variations of the same base function
- Differences affect both final solution + search space → Can be large!
- E.g., treatment plan
  - Each solution has the same components
  - But details are different for each patient

Parallel algorithm portfolios

Parallel algorithm portfolios – Why?

- If we have parallel compute resources, why not use them?
- Redesigning an algorithm to create a parallelised version
  - Can be substantial work
  - Sometimes no reasonable way to parallelise algorithmic components
- Instead: Run the algorithm(s) we have in parallel, e.g., one core each

Parallel algorithm portfolios – Why?



Parallel algorithm portfolios – Why?

- Benefits from variance in performance between different
  - Algorithms
  - Runs of the same algorithm with different parameter settings
  - Runs of the same non-deterministic algorithm (with different random seeds)
  - Runs of the same deterministic algorithm with different starting points
  - …combinations of the above
- Key: Make sure the above influences behaviour/results
  - (Otherwise it is a waste of compute/energy)

Parallel algorithm portfolios – Benefits

- Quality optimisation
  - Take the best result from all parallel runs once time is up
  - E.g., travelling salesperson problem

- Running time minimisation
  - When the first parallel run finishes we have the solution, so stop all others
  - E.g., for decision problems, like Boolean satisfiability (SAT) solving

Parallel algorithm portfolios – How?

- What is the best strategy?
  - If we know the best algorithm + configuration: Run copies of that
  - If we do not know: Run a selection of algorithms / configurations

- How many cores/runs?
  - What is available?
  - What is the cost/benefit of running more?
    Overhead vs. (small) additional improvement

Parallel algorithm portfolios – Beware!

- Overhead may reduce performance compared to single run
- Cost of monitoring each run, e.g., to see if it finished
- Demands on hardware
  - We may have multiple cores, but disk I/O, memory, etc. may not be prepared to handle (many) parallel runs

Other ideas

- Focused on: Parallel algorithm portfolios
  - Makes most sense for evolutionary algorithms

- Other closely related ideas
  - Algorithm schedules / Sequential portfolios: Run algorithms sequentially
  - Algorithm restarts: Run same algorithm again with different starting point

Parallel algorithm portfolios – Summary

- Run things in parallel to take advantage of
  - Available parallel compute resources (multi-core, distributed compute, etc.)
  - Performance variance between runs

- Benefit: Faster or better solutions (on average)

- Beware: Overhead, (other) hardware demands

Algorithm configuration – Why?

- Improve performance for problem you care about

- Default configuration/parameters sometimes:
  - Good for some problems, not for others
  - Good on average, but not **very** good for any specific problem instance
  - Not very good at all

Parameter tuning or algorithm configuration?

- (Hyper)parameter tuning/optimisation:
  - Typically focused on high-level **parameters**, e.g., mutation rate
- Algorithm configuration:
  - Consider algorithm parameters, but
  - **Also** consider larger algorithm **components**, e.g., different mutation operators
- Neural architecture search
  - Specialised methods for neural network configuration, same basic concept

Algorithm configuration – To the extreme

- Programming by optimisation [Hoos2012]
  - Intentionally turn **every** algorithmic design choice into a parameter
  - Use algorithm configuration to optimise over those parameters

- Often Many design decisions are hidden, but are worth optimising!
  - E.g., only parameter for mutation rate exposed to user, not mutation operator

- Result: Automated algorithm design
  - (Restricted to existing parts/ideas)

Algorithm configuration definition

- Given
  - A set of instances $\mathcal{I}$
  - An algorithm $A(\pi)$, whose performance depends on the parameter configuration π
  - And a parameter space Π

- Find the parameter configuration $\pi \in \Pi$ that optimises the metric m which aggregates the performance of $A(\pi)$ over all instances $\mathrm{i} \in \mathcal{I}$

Algorithm configuration – When?

- Will run multiple times, e.g., on many problem instances
  - Configuration can be costly, not worth it for a single run
- Homogeneous instance set!
  - Configure a specialist for set of similar instances
- Heterogeneous instance set? There are better options
  - Configure a generalist: Algorithm selection may perform better
  - Configure set of specialist: Even more expensive than one configuration

Algorithm configuration – How?

- Basic solutions (better than nothing)
  - Manual experimentation
  - Grid search

- Better: Automated algorithm configuration
  - SMAC [Hutter et al 2011]
  - irace [López-Ibáñez et al 2016]
  - GGA++ (GA-based!) [Ansótegui et al 2015]
  - …

Algorithm configuration – How?

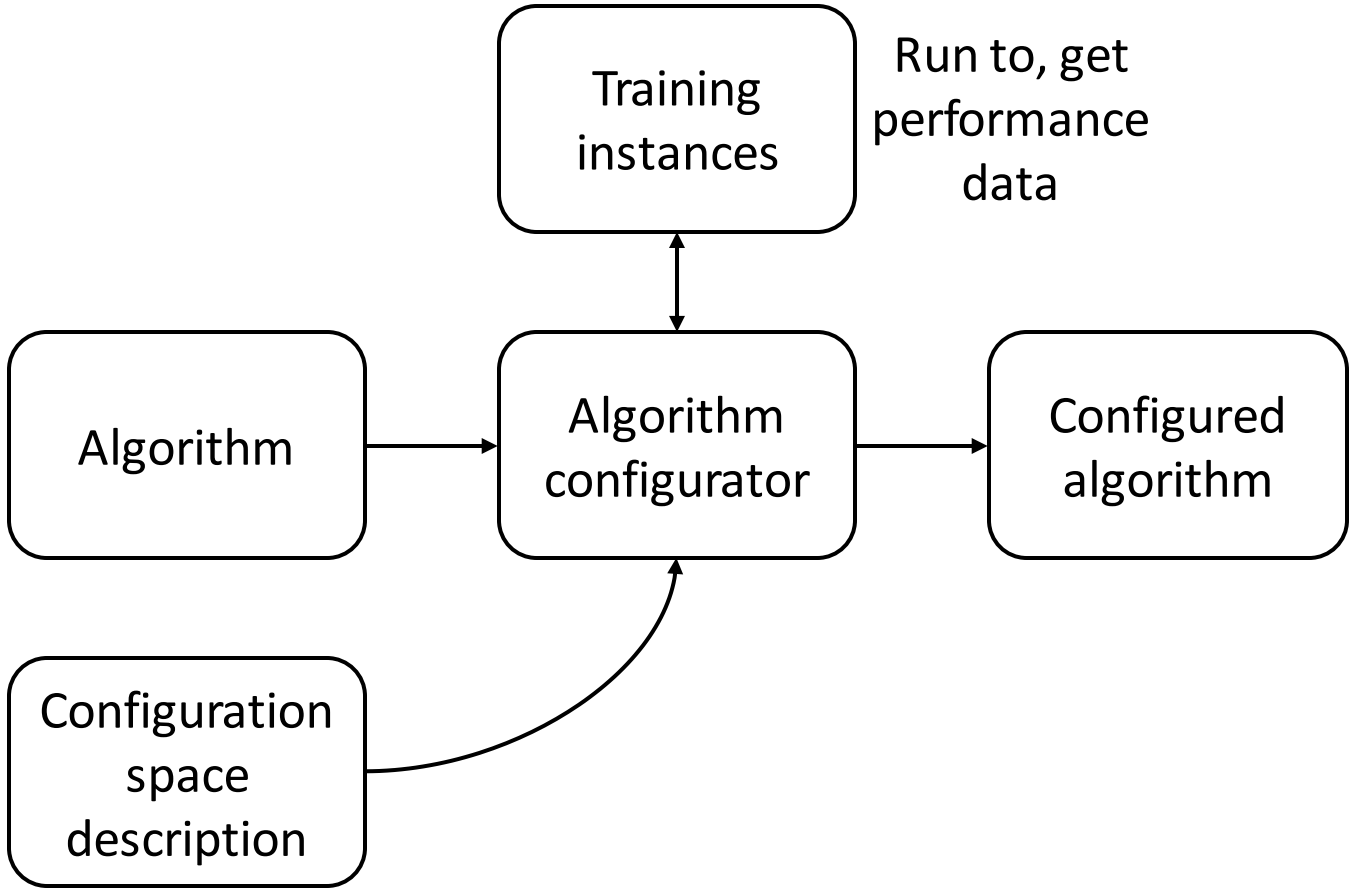
- Run different algorithm configurations on target problem instance

- Need
  - Total budget: How much time you can/want to spend on configuration
  - (Max) budget per run: How much time to spend testing one configuration
  - Description of configuration space: Which parameters + possible values
  - Training instances

Algorithm configuration pipeline

- Input
  - Algorithm
  - Description of configuration space
  - Training instances
- Output: An algorithm configuration
- Deploy: Algorithm with optimised configuration

Algorithm configuration pipeline

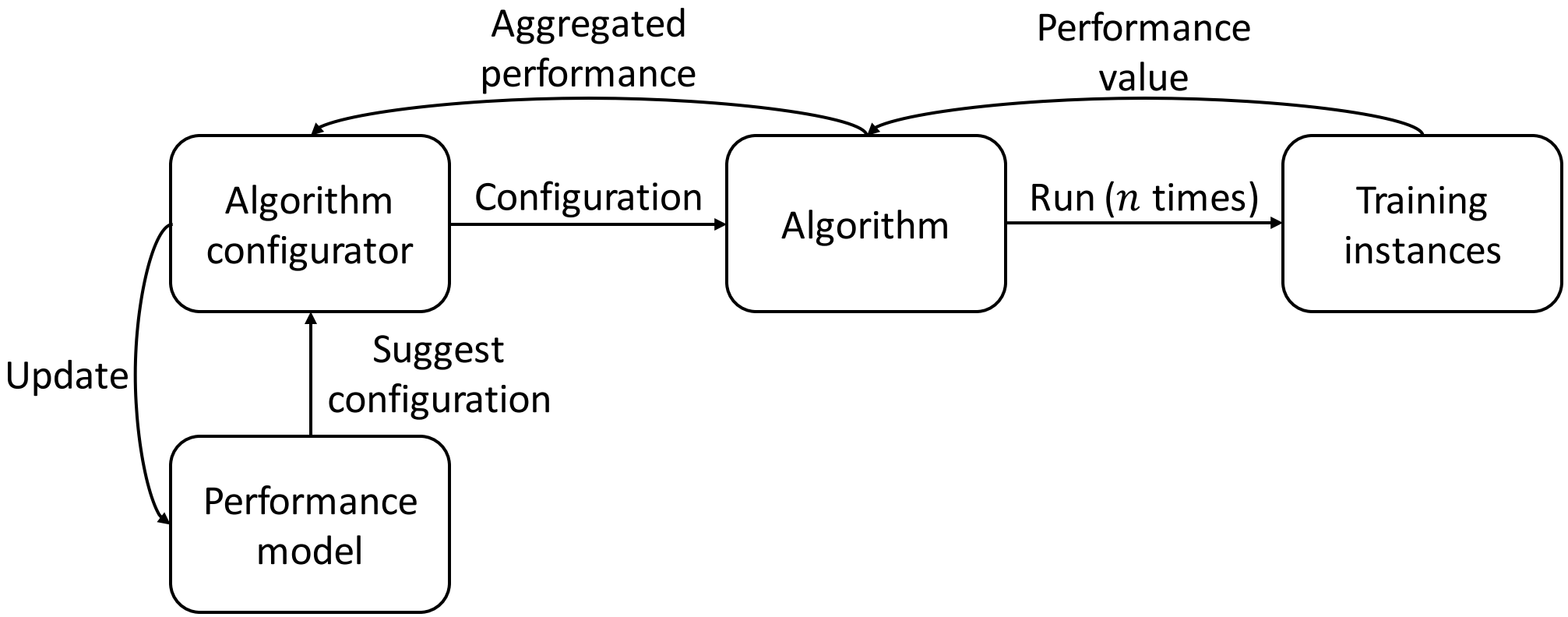


Algorithm configuration space

- Describe
  - Which parameters exist
  - Their type, e.g., integer, real, categorical
  - What (range of) values they can take
  - Any conditions, e.g., parameter a is only active if parameter $b = True$
  - Forbidden combinations, e.g., $a = 0$ is not allowed if $c > 100$

- Provide script or programming interface describing how the configurator can run the algorithm given a set of parameters

Algorithm configuration basic process



Algorithm configurator – Under the hood

- Model-based optimiser: Counteract expensive evaluations
  - Randomised search heuristic, e.g., a model-based EA
  - Bayesian optimisation
  - Sequentially updated machine learning model, e.g., random forest

- Important: Need to handle complex configuration spaces
  - Mixed variable types: Integer, real, categorical
  - High-dimensional: Possibly large number of parameters

Bayesian optimisation (model-based example)

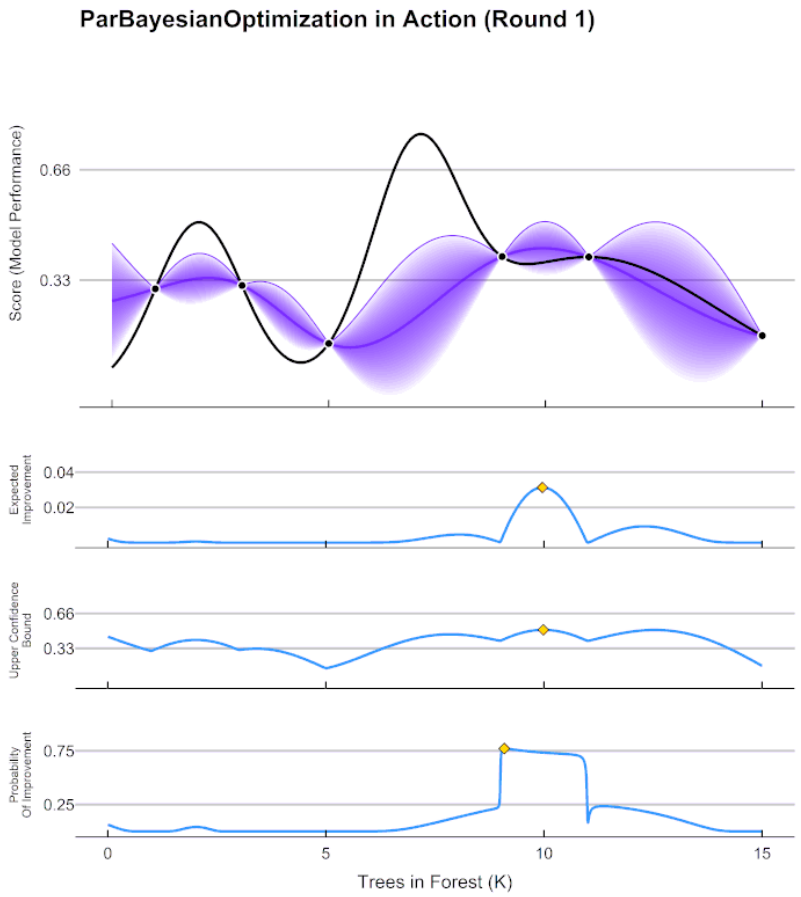
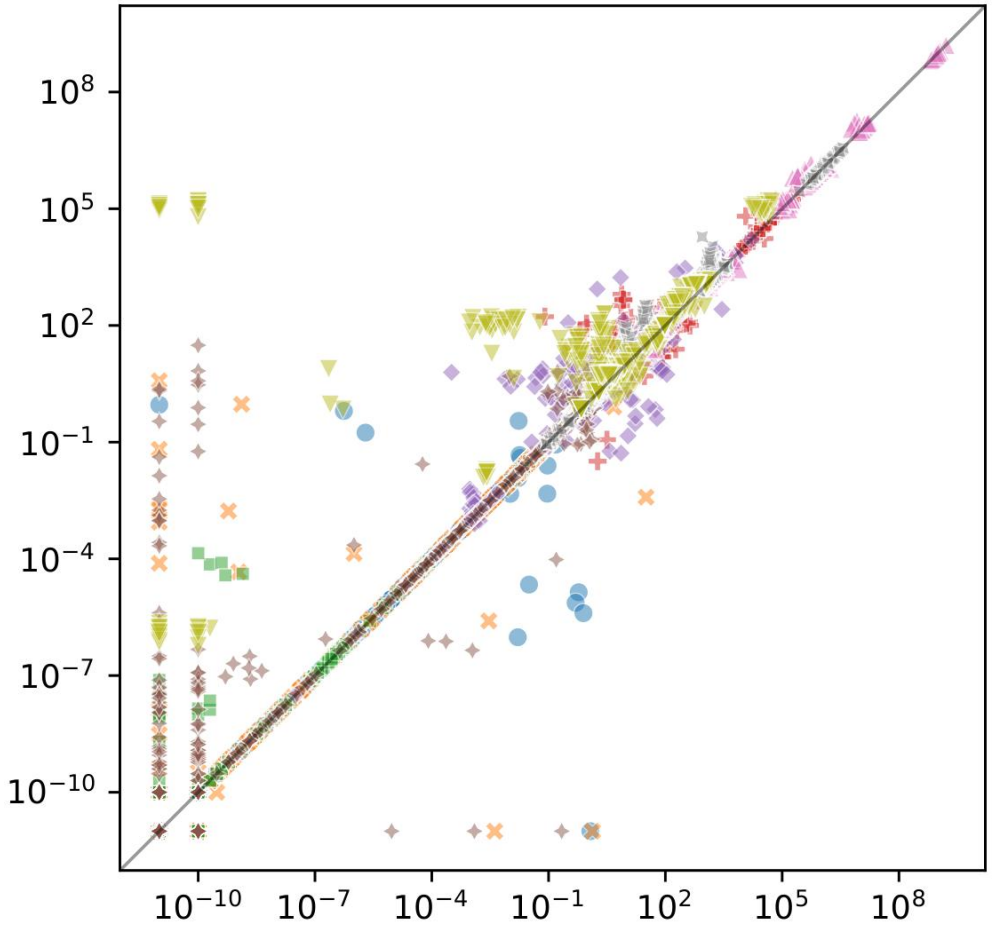


Figure by Samuel Wilson CC BY-SA 4.0

Comparing with other methods or baselines

- Plot performance of both methods
  - One point per problem instance
- Diagonal: Equal performance
- Above/below diagonal:
  One method is better
- Distance to diagonal:
  How much better/worse



Algorithm configuration – Beware!

- Carefully look at which parameters you configure, things go wrong if you configure things like – They do not generalise!
  - Random seed
  - Initial solution (set)
- If training and testing sets are different, the configuration may perform badly
- Algorithm configuration can be expensive, not always worth it

- Much more: [Eggensperger et al 2019]

Other ideas

- Focused on: Configuring algorithms
  - Most relevant to evolutionary algorithms

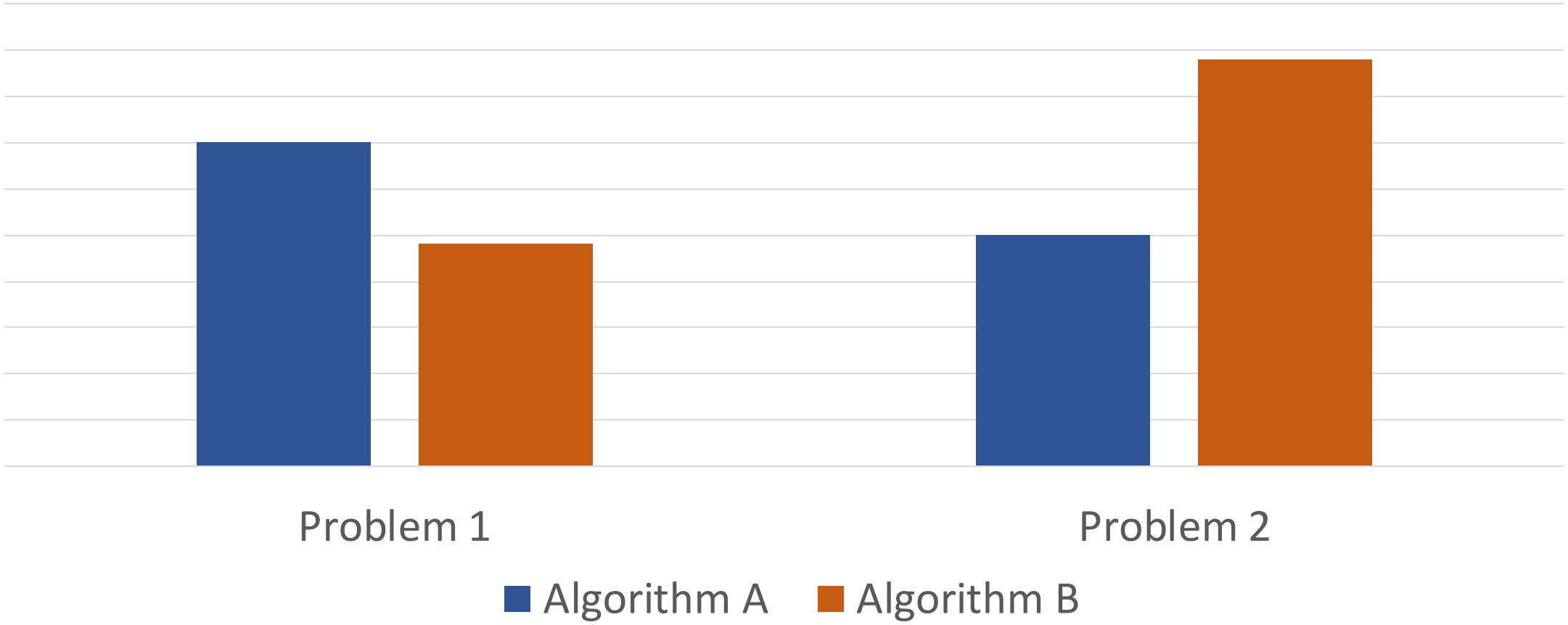
- Other related ideas
  - Dynamic algorithm configuration: Switch between configurations during run
  - Automated machine learning: Use configurators to optimise ML pipelines

Algorithm configuration – Summary

- Find good parameters and algorithm components to solve the specific problem instances you care about
- Benefit: Improved performance over default settings
- Beware: Only configure algorithm behaviour, no external components

Automated algorithm selection

Which algorithm is the best?

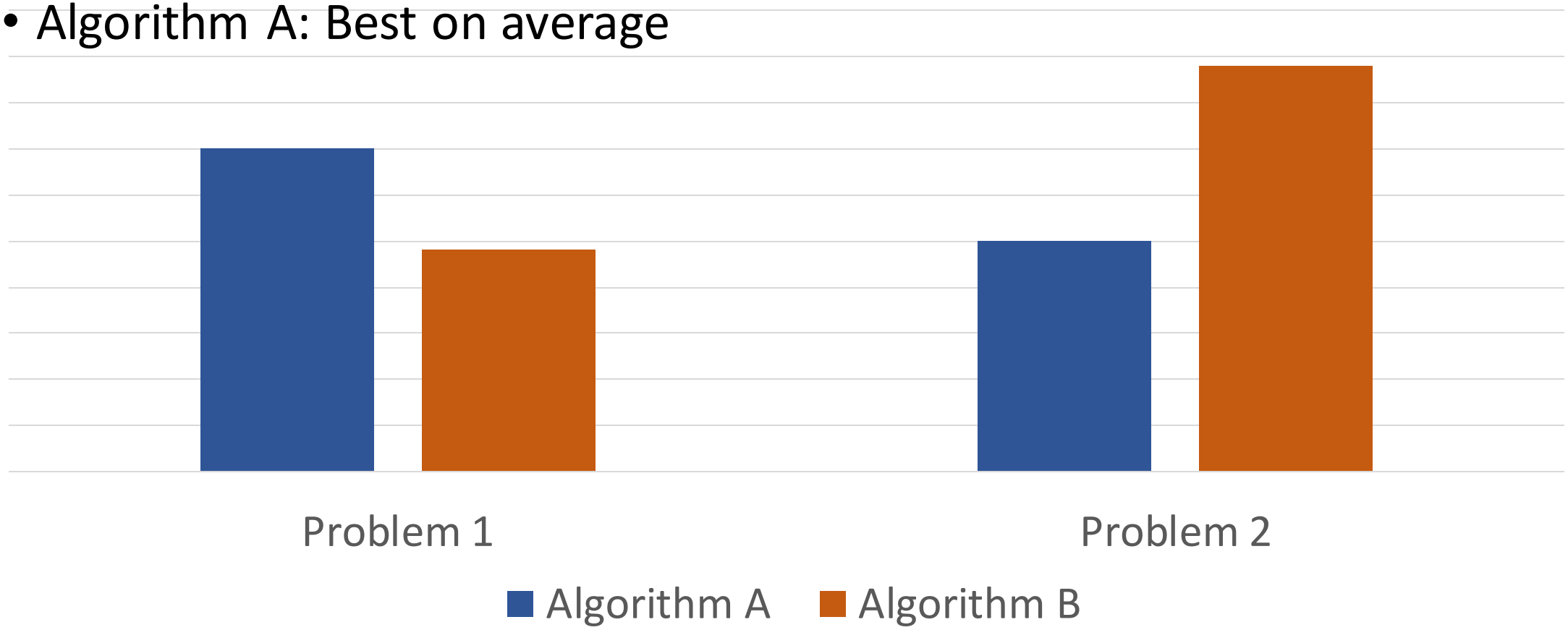


Single best solver

- The best algorithm on aggregate over all relevant problem instances
- E.g., best on average

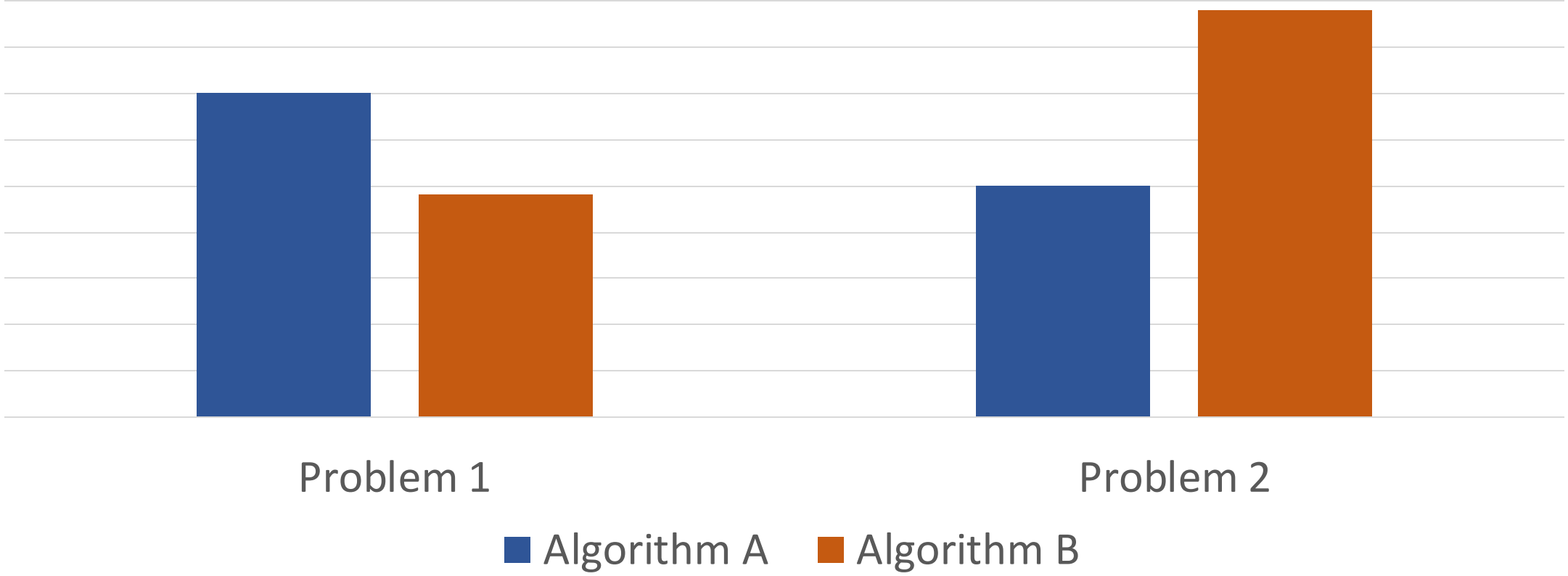
Single best solver – Can we do better?

- Algorithm A: Best on average



Single best solver – Can we do better?

- What if we can choose separately for each problem?



Algorithm selection – Why?

- Assumption
  - There is no algorithm that is the best for all problem instances

- So
  - If we are able to correctly choose the best algorithm for each instance
  - We get better performance than using **any** single algorithm for all instances

Algorithm selection – Why?

- What if this assumption is wrong?
  - *There is no algorithm that is the best for all problem instances*
- If we **don't** know the single best solver
  - We will learn it by applying algorithm selection
- If we **do** know the single best solver
  - Not beneficial (beyond verifying whether we were right)

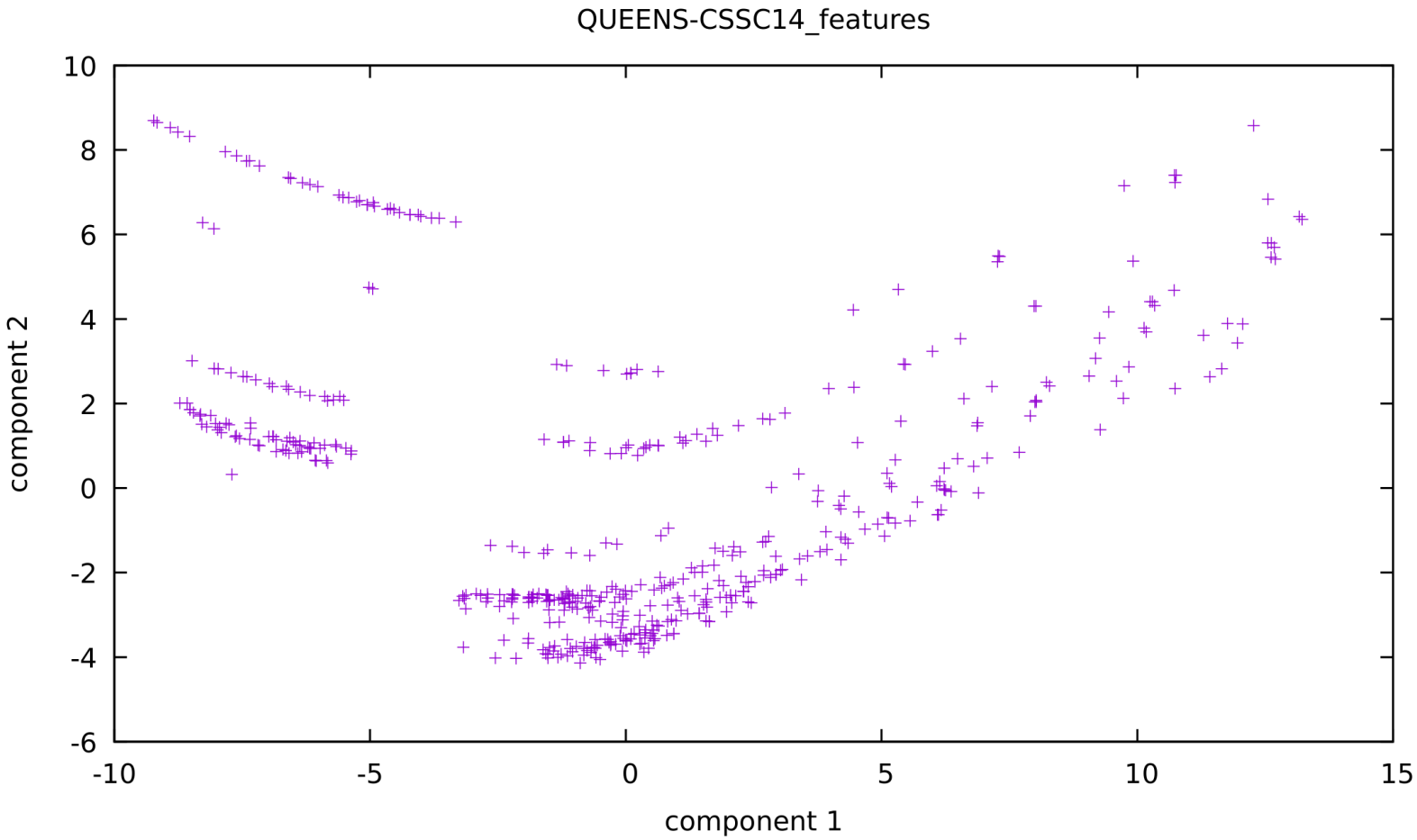
Algorithm selection – When?

- Best case: Heterogeneous instance set + complementary algorithms
  - Problem instances are somehow different from each other,
  - **and** these differences relate to algorithm performance
  - Algorithms are good for different subsets of problem instances

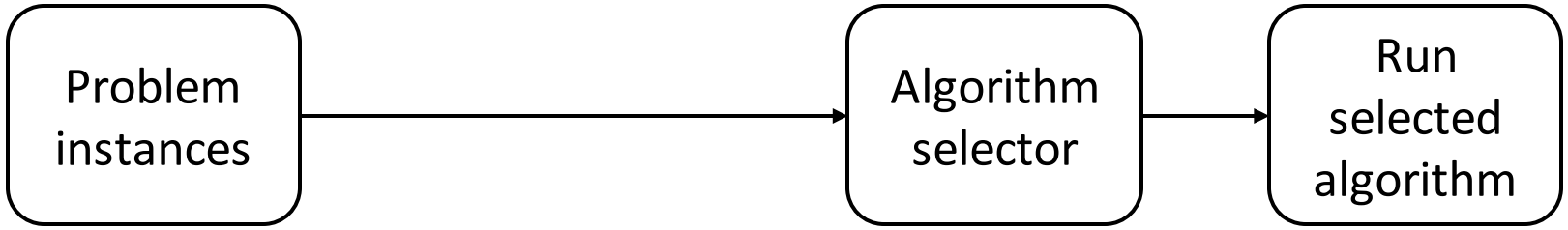
- But
  - Can still be beneficial if there is at least some complementarity

Heterogenous instance set: Clusters expected

- Variance in
  - Basic features (number of cities in TSP)
  - More complex features (even if we might not know what they mean!)

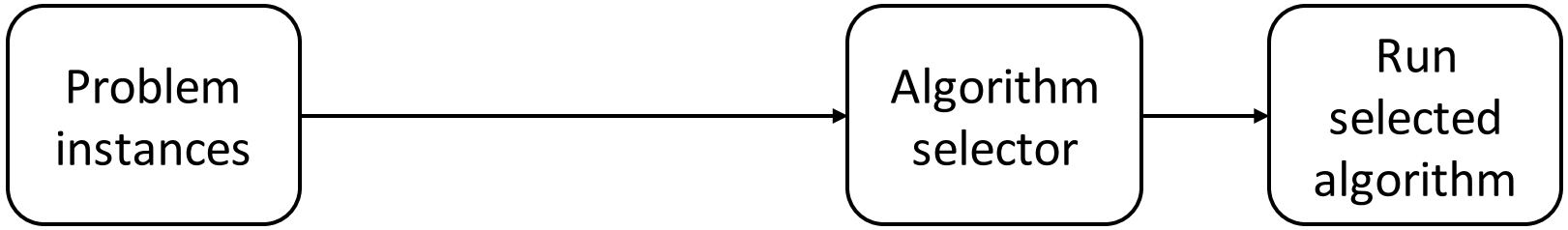


Algorithm selection pipeline



Algorithm selection pipeline

- How to distinguish between different problem instances?

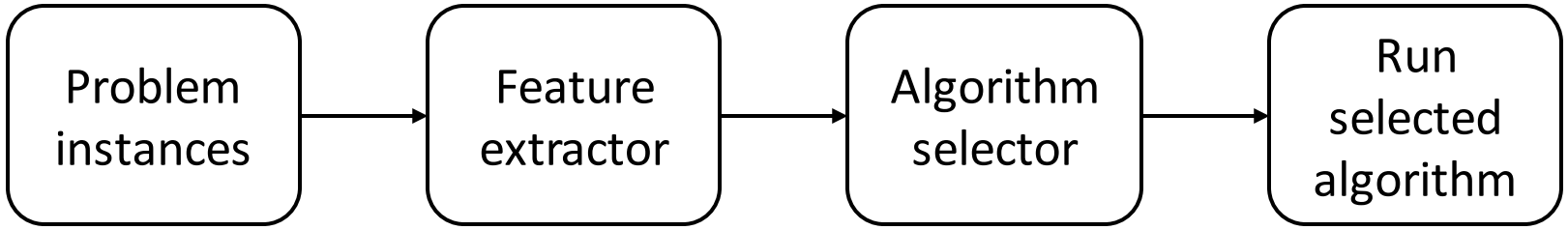


Per-instance algorithm selection (PIAS)

- Compute instance specific features
- Predict algorithm for specific instance

Algorithm selection pipeline

- Distinguish between instances by computing **instance features**



Instance features

- Simple: Read/count/…
  - Statistics, e.g., number of clauses in a SAT instance
- More complex: Compute
  - Problem property, e.g., degree of multimodality of optimisation problem

Instance ID	Feature 1	Feature 2	Feature 3
1	35	0.2	-3.4
2	12	0.8	2.1
…	…	…	…
n	22	0.4	-0.9

Deployed algorithm selection pipeline

- Input: Instance features
- Output: An algorithm

Algorithm selection – How?

- Manually?
  - Tedious
  - Error prone
  - Difficult to reproduce

- Automated: Construct a selection model to do it for us
  - AutoFolio [Lindauer et al 2015]
  - Algorithm Selection Framework (ASF) [ShavitHoos 2025]

Algorithm selection definition

- Given
  - A set of instances $\boldsymbol{\mathcal{I}}$
  - A set of algorithms $\boldsymbol{\mathcal{A}}$
  - And a metric m: $\boldsymbol{\mathcal{I}} \times \boldsymbol{\mathcal{A}} \rightarrow \mathbb{R}$
    mapping all pairs of instances and algorithms to performance values

- Find a selector S that maps all instances $i \in \boldsymbol{\mathcal{I}}$
  to an algorithm $S(i) \in \boldsymbol{\mathcal{A}}$ such that the
  aggregate performance according to m is optimised

Algorithm selection construction

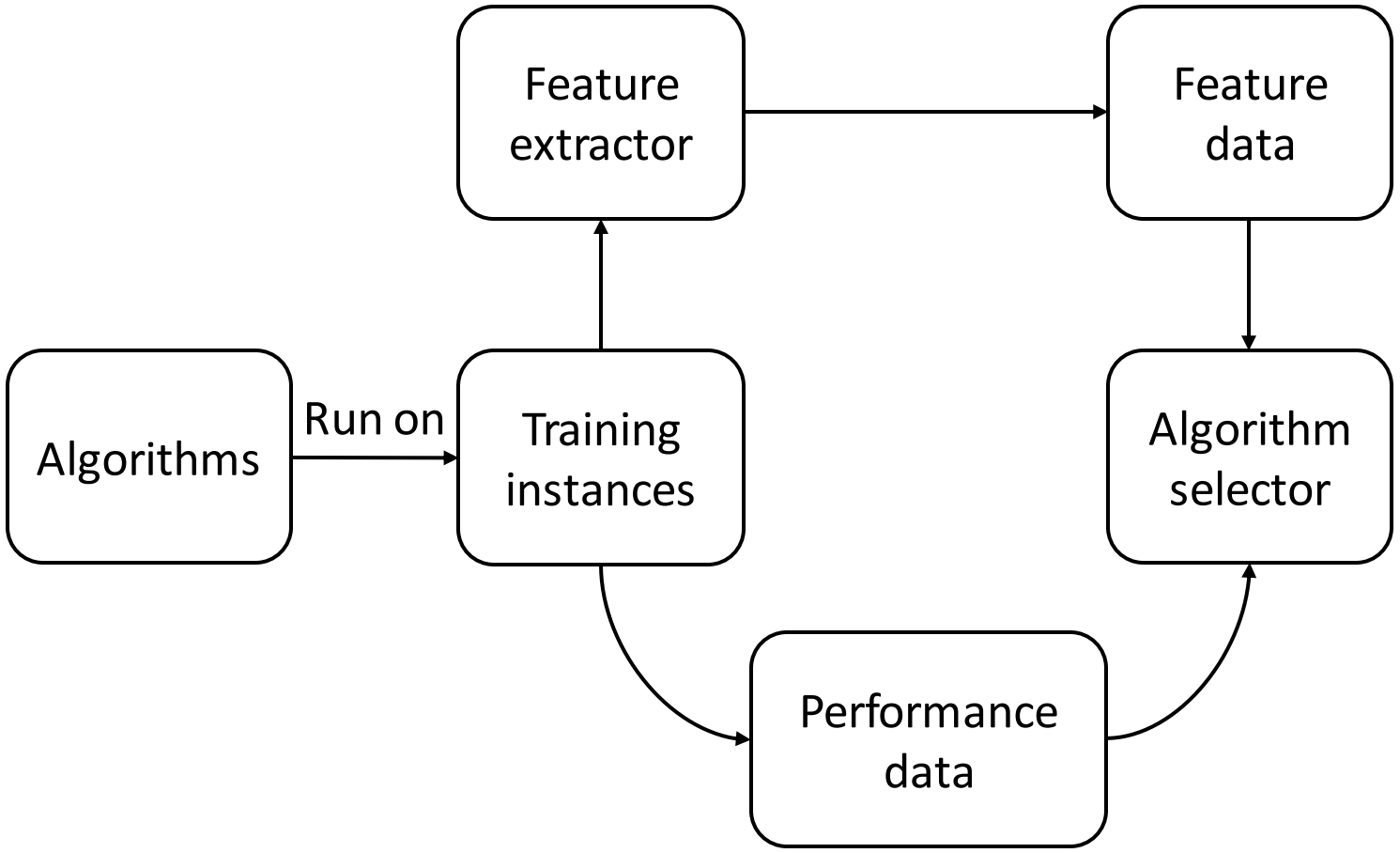
- Input
  - Training instances
  - Set of algorithms: Producing performance data on training instances
  - Feature extractor: Producing instance feature data on training instances
- Output: An algorithm selection model
  - **NOT** a single algorithm!
- Deploy: The algorithm selection model, feature extractor, algorithms

Algorithm performance data

- Measured performance of each algorithm on training instances
- Possibly multiple runs for each instance to account for nondeterminism

Instance ID	Algorithm 1	Algorithm 2	Algorithm 3
1	35	24	16
2	12	20	14
…	…	…	…
n	22	18	25

Construct an algorithm selector



Algorithm selector – Under the hood

- Machine learning model
  - Decision tree
  - Random forest

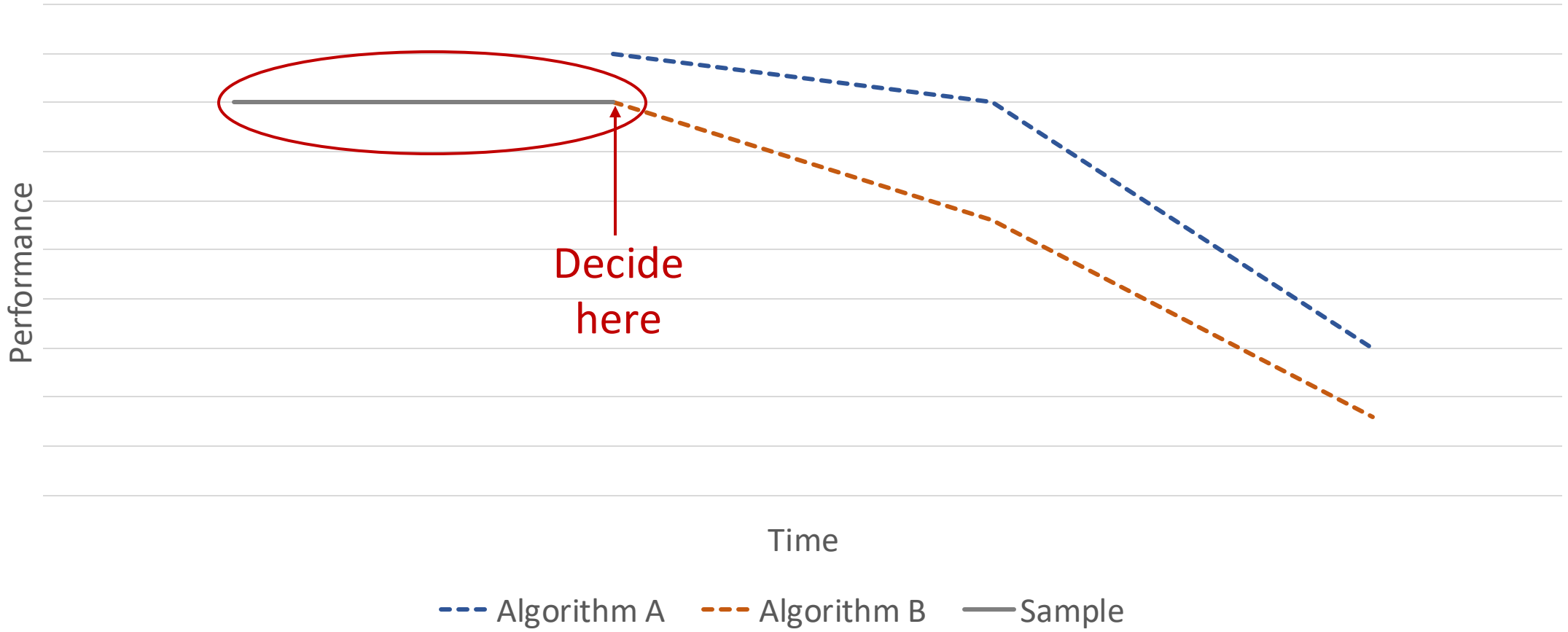
- Learn relation between
  - Instance feature values
  - Algorithm performance values

The cost of feature extraction

- Many problems like SAT solving
  - Instance description available a priori
  - Cheaply compute features from instance file

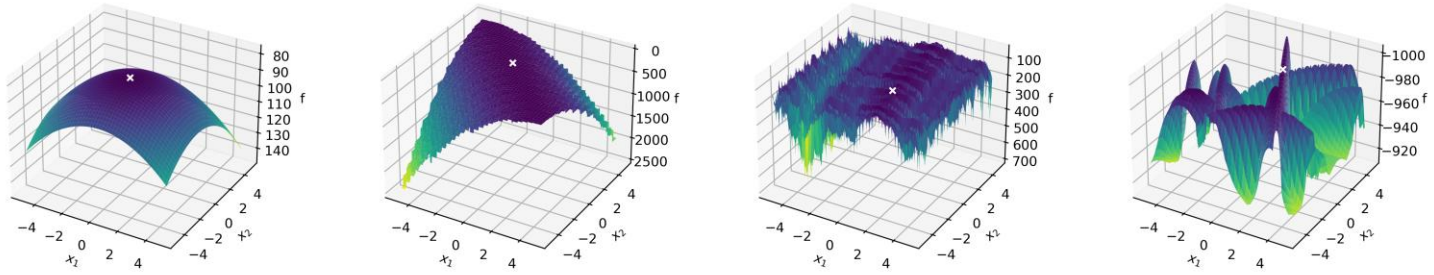
- Black-box optimisation (BBO)
  - No instance description
  - Sample solutions to compute features
  - Costly, depending on evaluation cost/budget

The cost of PIAS in black-box optimisation



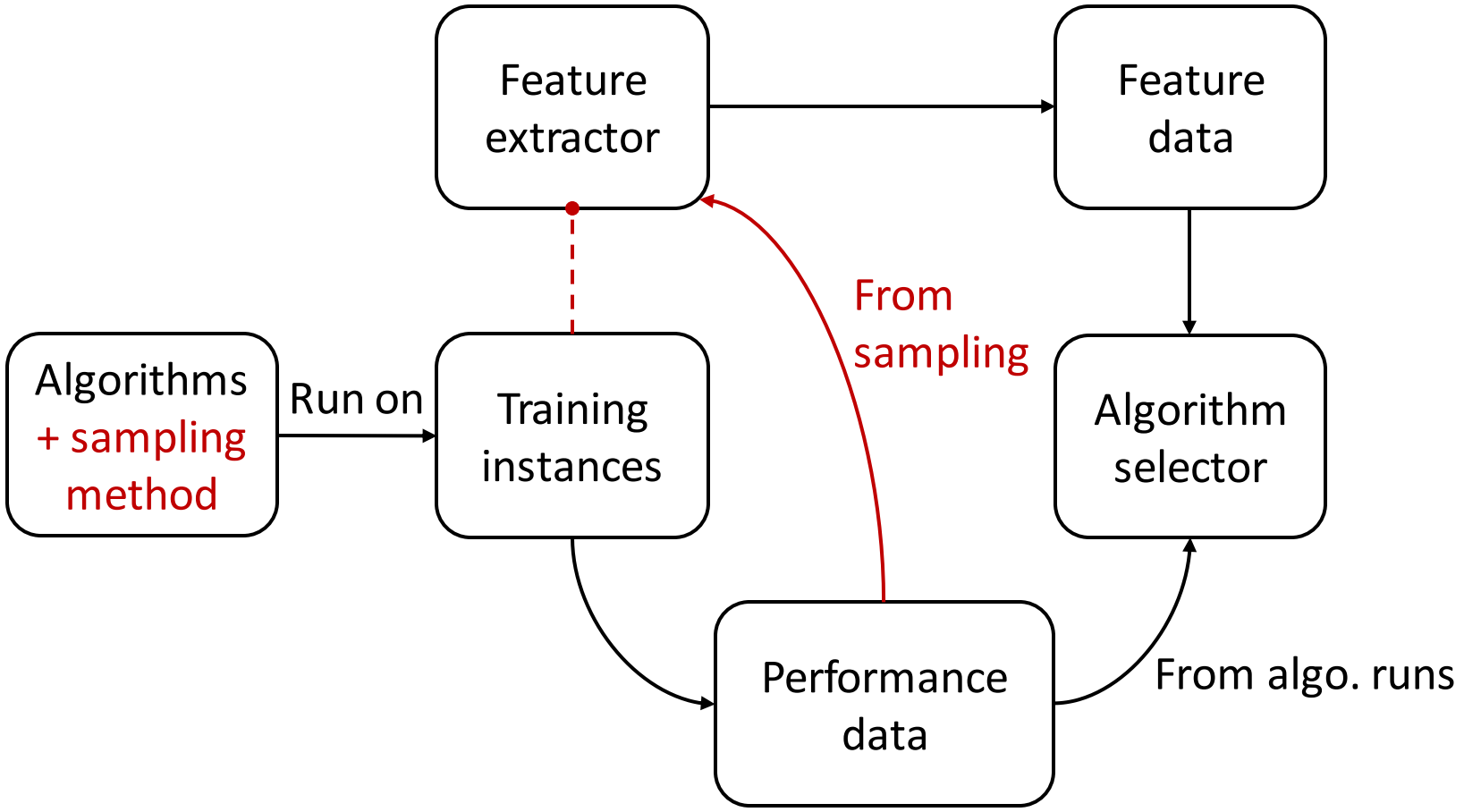
Exploratory landscape analysis

- Sample points in the search space
- Compute features based on those sampled points
- Ideally: Identify properties of the problem instance, e.g., is it:
  - Unimodal
  - Multimodal
  - Highly multimodal



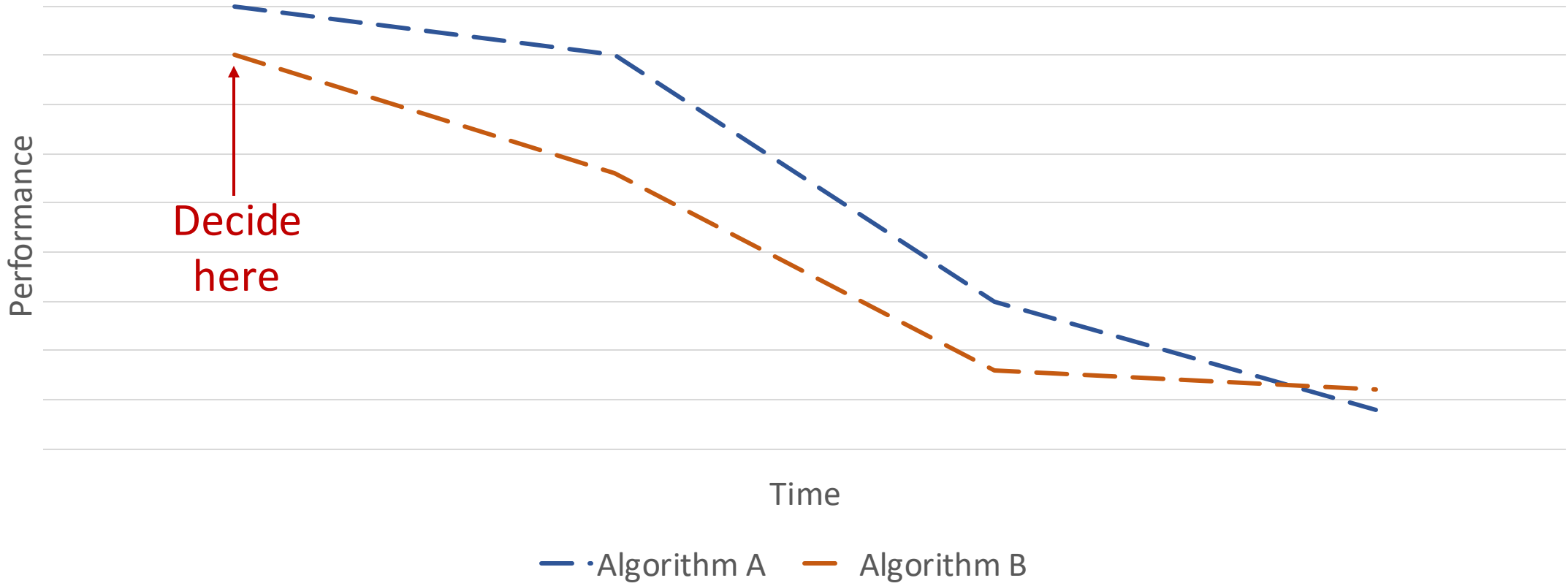
Figures from https://coco-platform.org/

Construct a selector for black-box optimisers

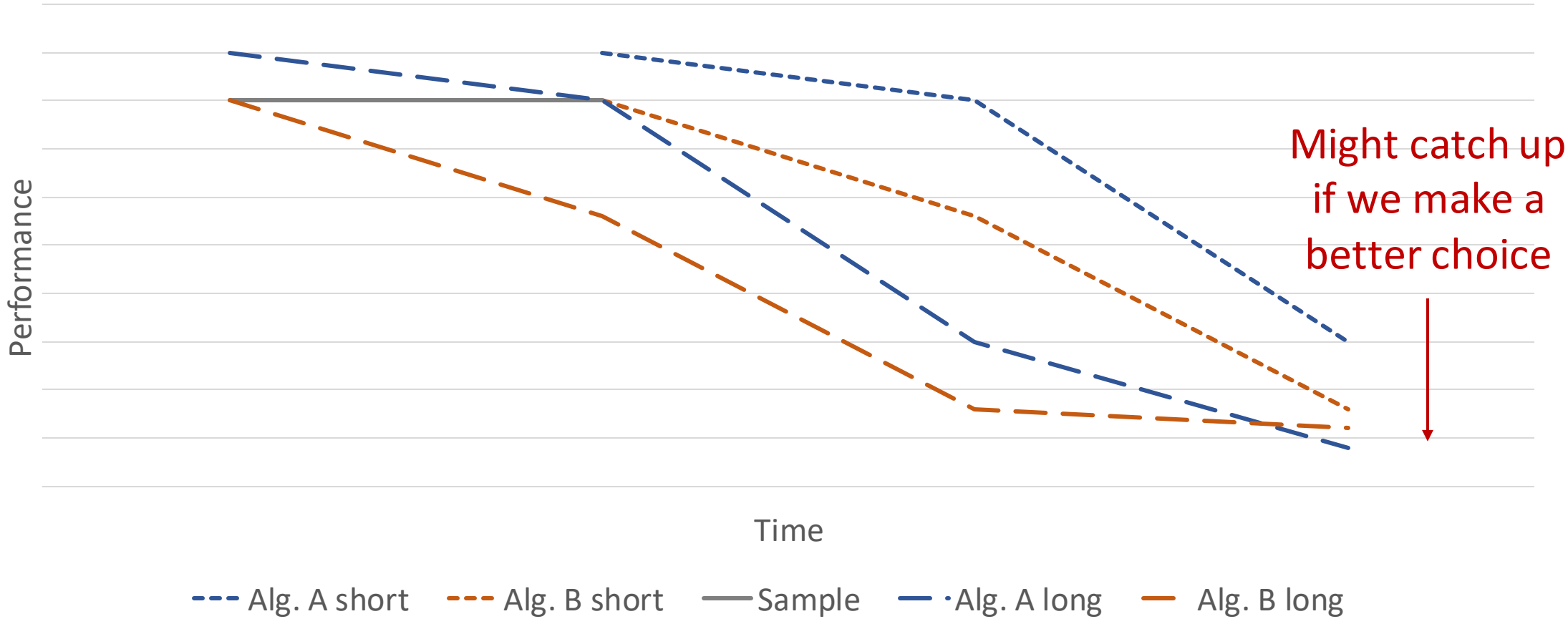


Choose a single algorithm without features

- Same idea as single best solver!



AS for BBO only worth it if we can do better!



Algorithm selector – Performance evaluation

- Once constructed, test quality of algorithm selection system

- Is it better than the baseline?
  - Maybe: The single algorithm we always used
  - Ideally: The single best solver (SBS)

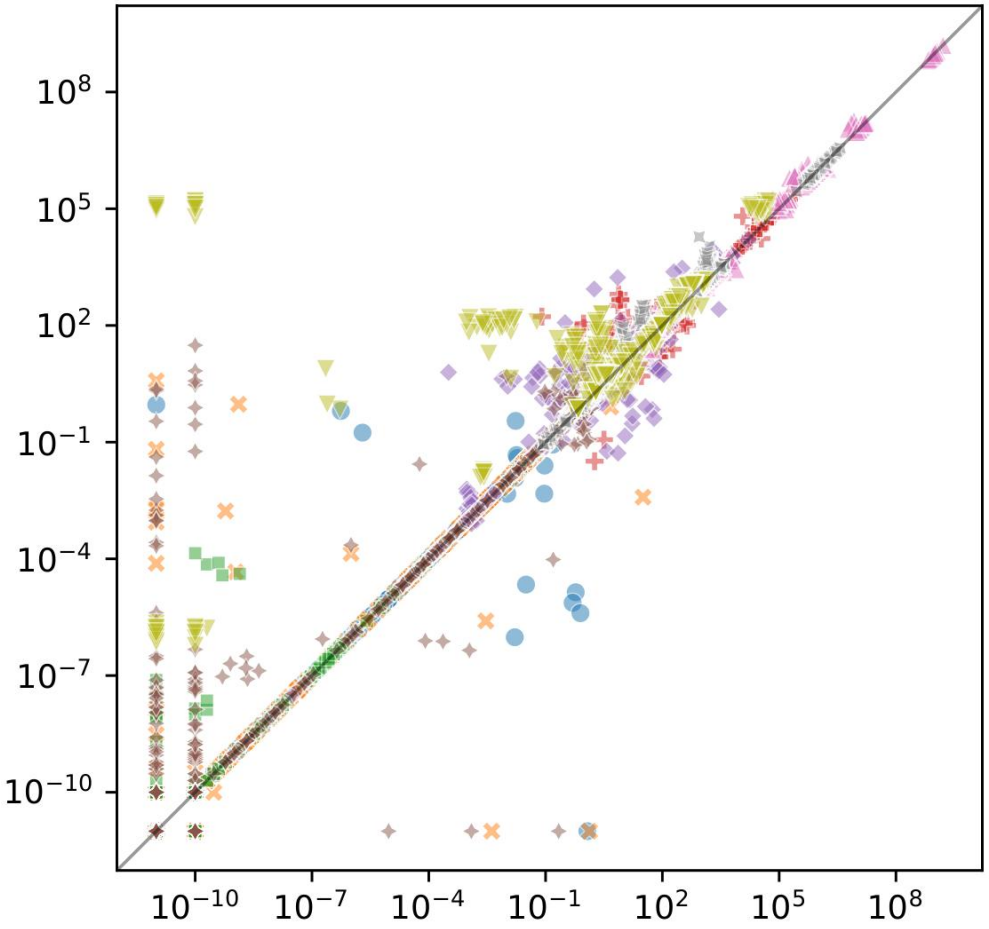
- How close do we get to the best possible performance?
  - **Virtual best solver** / oracle

Virtual best solver

- The hypothetical algorithm selector that always chooses right
- For **every** problem instance, select best performing algorithm for **that specific** instance
- **Not** a realistic target
- Assesses current quality + whether worth spending time to improve

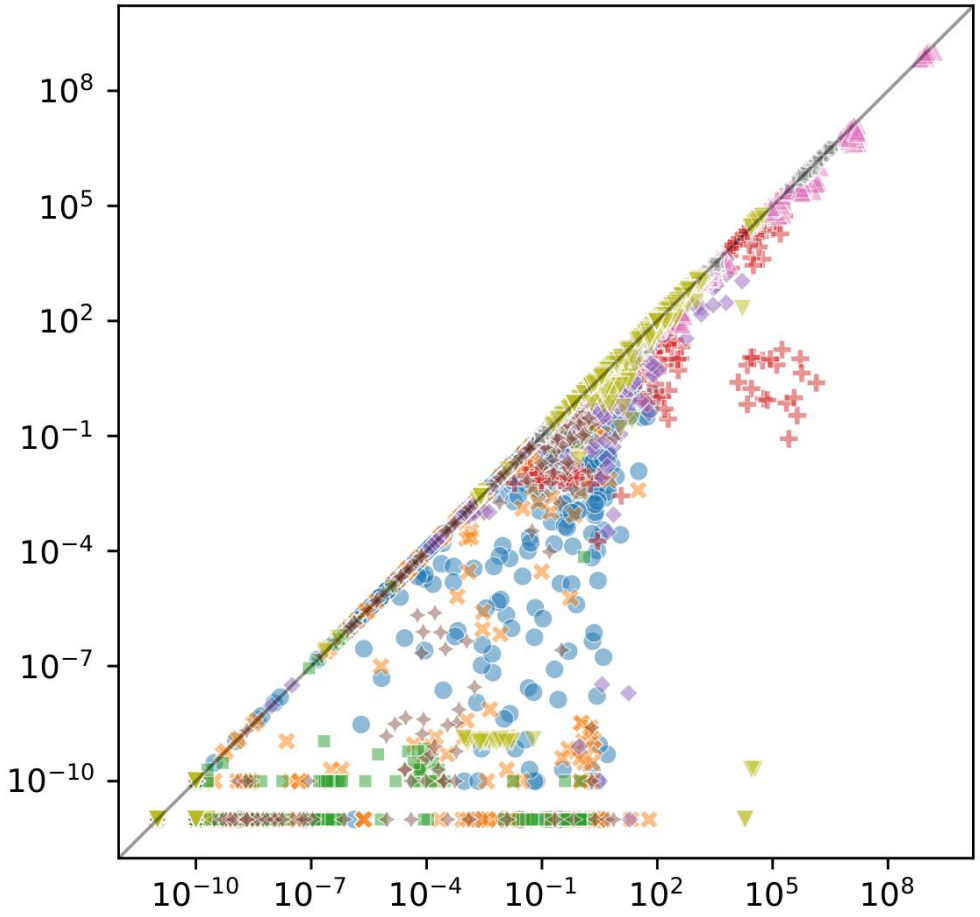
Comparing with other methods or baselines

- Plot performance of both methods
  - One point per problem instance
- Diagonal: Equal performance
- Above/below diagonal:
  One method is better
- Distance to diagonal:
  How much better/worse



Comparing an algorithm selector with VBS

- Cannot outperform it!
- Best case: Reach the diagonal
- Learn: How close the method gets to the optimum



Algorithm selection – Beware!

- If training and testing sets are different,
  we cannot trust the selector to make good choices

- Cost of feature computation/extraction, including sampling,
  should be small enough that performance gains are not negated
  - Depending on the features, the cost can be substantial!

Other ideas

- Focused on: Static algorithm selection
  - Simplest case

- Other more advanced ideas
  - Dynamic algorithm selection: Switch to another algorithm after part of budget
  - Online algorithm selection: Adapt selector to changing instance distribution

Algorithm selection – Summary

- Construct a model that chooses the best algorithm for every problem instance, to take advantage of performance complementarity between algorithms
- Benefit: Improved performance over always using the same algorithm
- Beware: Confirm the selector outperforms the baseline

Summary (Main concepts)

- Parallel algorithm portfolios
  - Run algorithm(s) in parallel to benefit from variance in performance
- Algorithm selection – Not to be confused with the selection operator!
  - Choose the best algorithm for the problem
- Algorithm configuration
  - Find the best parameters, e.g., mutation rate
  - But **also** find the best algorithm components, e.g., variation operator

References

- [Ansótegui et al 2015] – Ansótegui, C., Malitsky, Y., Samulowitz, H., Sellmann, M., & Tierney, K. (2015, July). Model-Based Genetic Algorithms for Algorithm Configuration. In IJCAI (pp. 733-739).
- [Eggensperger et al 2019] – Eggensperger, K., Lindauer, M., & Hutter, F. (2019). Pitfalls and best practices in algorithm configuration. Journal of Artificial Intelligence Research, 64, 861-893.
- [Hoos2012] – Hoos, H. H. (2012). Programming by optimization. Communications of the ACM, 55(2), 70-80.
- [Hutter et al 2011] – Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2011, January). Sequential model-based optimization for general algorithm configuration. In International conference on learning and intelligent optimization (pp. 507-523). Berlin, Heidelberg: Springer Berlin Heidelberg.
- [Lindauer et al 2015] – Lindauer, M., Hoos, H. H., Hutter, F., & Schaub, T. (2015). Autofolio: An automatically configured algorithm selector. *Journal of Artificial Intelligence Research*, *53*, 745-778.
- [López-Ibáñez et al 2016] – López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L. P., Birattari, M., & Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives, 3, 43-58.
- [ShavitHoos 2025] – Shavit, H., Hoos, H. (2025). Algorithm Selection Framework (ASF). Zenodo: https://doi.org/10.5281/zenodo.15288151

Links

- Algorithm selection tools
  - AutoFolio: https://github.com/automl/AutoFolio
  - ASF: https://hadarshavit.github.io/asf/

- Algorithm configuration tools
  - irace: https://mlopez-ibanez.github.io/irace/
  - SMAC: https://automl.github.io/SMAC3/