My perspective on

Programming by Configuration:

An “emerging” paradigm in automated algorithm design

Steven Adriaensen

AI-lab, Vrije Universiteit Brussel, Belgium

Outline

1. Algorithm Design
   – What is the algorithm design problem?
   – How is it currently being solved?

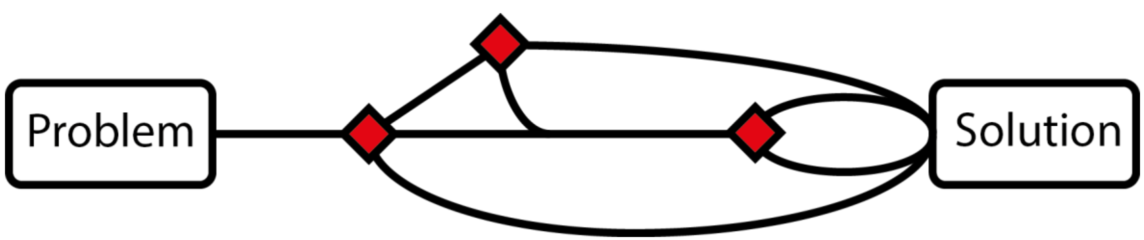
2. Programming by Configuration (PbC)
   – In a nutshell…
   – Limitations, how they could be addressed, and my own research in this direction…

1. Algorithm Design Problem (ADP)

There are **many ways** to solve a given problem.

- Multiple ways to formulate a problem
- Multiple (parametrized?) solvers exist.
- Multiple implementations of a single solution approach.

➔ **When solving a problem we face *design choices***



What is the ***best* way**?

- Minimizing execution time
- Maximizing solution quality

"The problem of how to best solve problems"

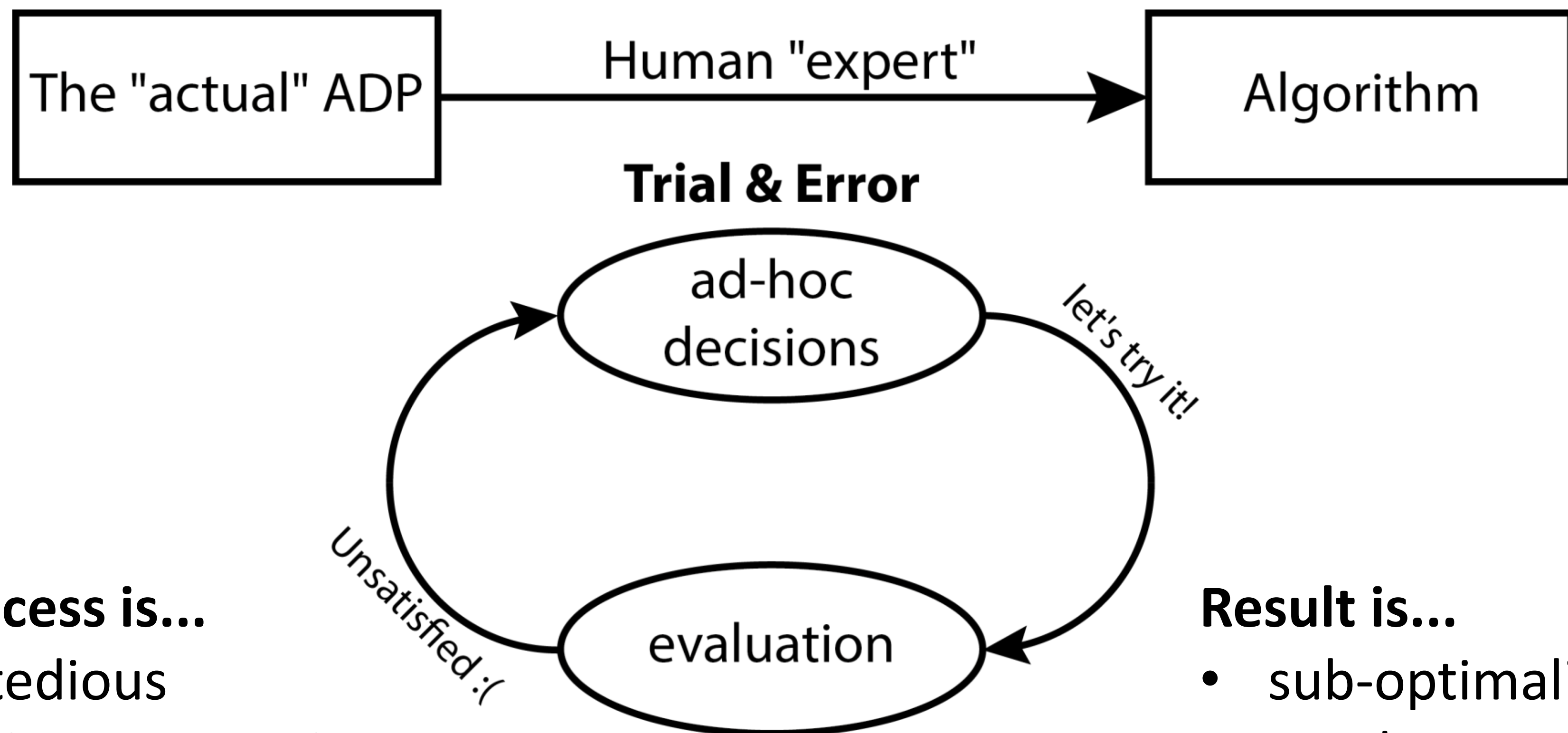
1. Contemporary Solution Approaches

ADPs and attempts to solve them are ubiquitous and fragmented…
Algorithm Configuration, Instance-based Selection, (Dynamic) Portfolios, Parameter Control, Reactive Search, Hyper-heuristics, Search-based Software Engineering, Intelligent Compilers, Machine Learning, Reinforcement Learning, Learning Classifier Systems, Program Synthesis, Genetic Programming, Ant Programming, Logical Programming, Probabilistic Programming, Neural Turing Machines,…

➔ **How to best solve the ADP is an ADP itself!**
(idea: apply recursively: configuring/selecting configurators, meta-learning,…)

Research objective: *Towards enabling a more* ***unified approach*** *to automated algorithm design, maximally* ***exploiting the nature of the problem at hand.***

1. Manually



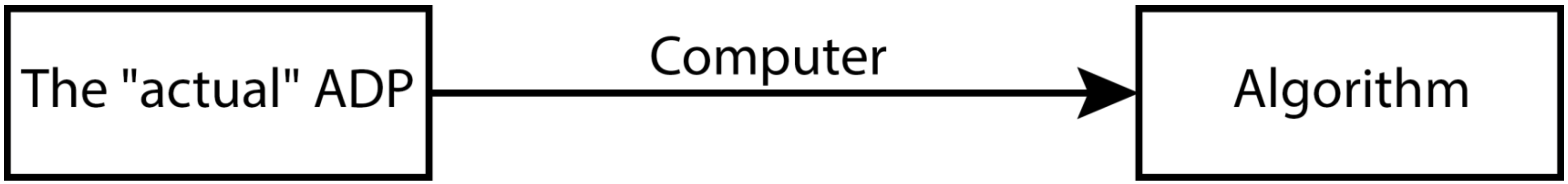
Process is...

- tedious
- time-consuming
- costly
- untraceable

Result is...

- sub-optimal?
- overly complex?
- unreliable?

1. Automated



Idea:

What? Let a computer design its own programs

Why? Computers are faster, cheaper and unbiased

How? Provide an algorithm for the ADP…
i.e. formalizing a design process.

Fully automated: Program Synthesis, Genetic Programming, Declarative Programming, and (more recently) Neural Turing Machines.

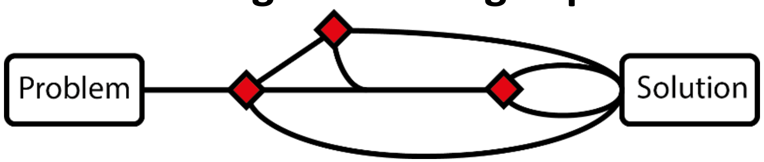
➔ Scalability issues…

1. Semi-automated

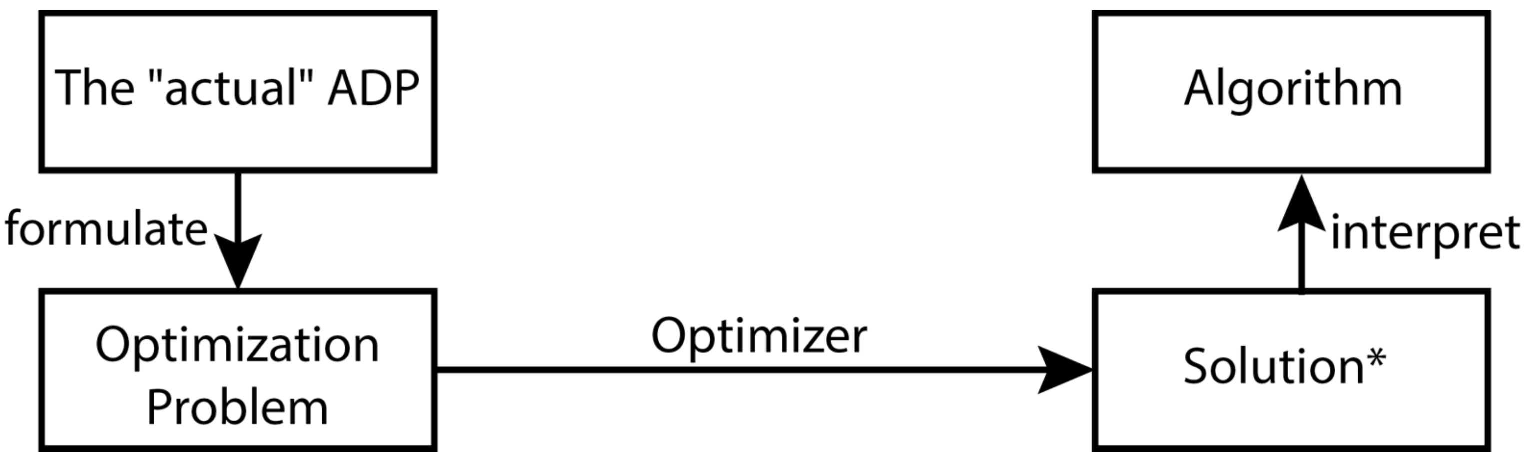
Programming by Optimization (PbO) (Holger Hoos, 2012)

1. Leave difficult decisions open at design time

Program a Design Space ⇔ **Single Algorithm**



2. Generate the best algorithm instance for a specific use-case ***automatically***.



1. Who makes which design choices?

Expert Knowledge Available to make Design Decision?	**Manual**	**Fully Automatic**	**Semi-automatic (e.g. PbO)**
Low			
High			

1. Who makes which design choices?

Expert Knowledge Available to make Design Decision?

Manual

Fully Automatic

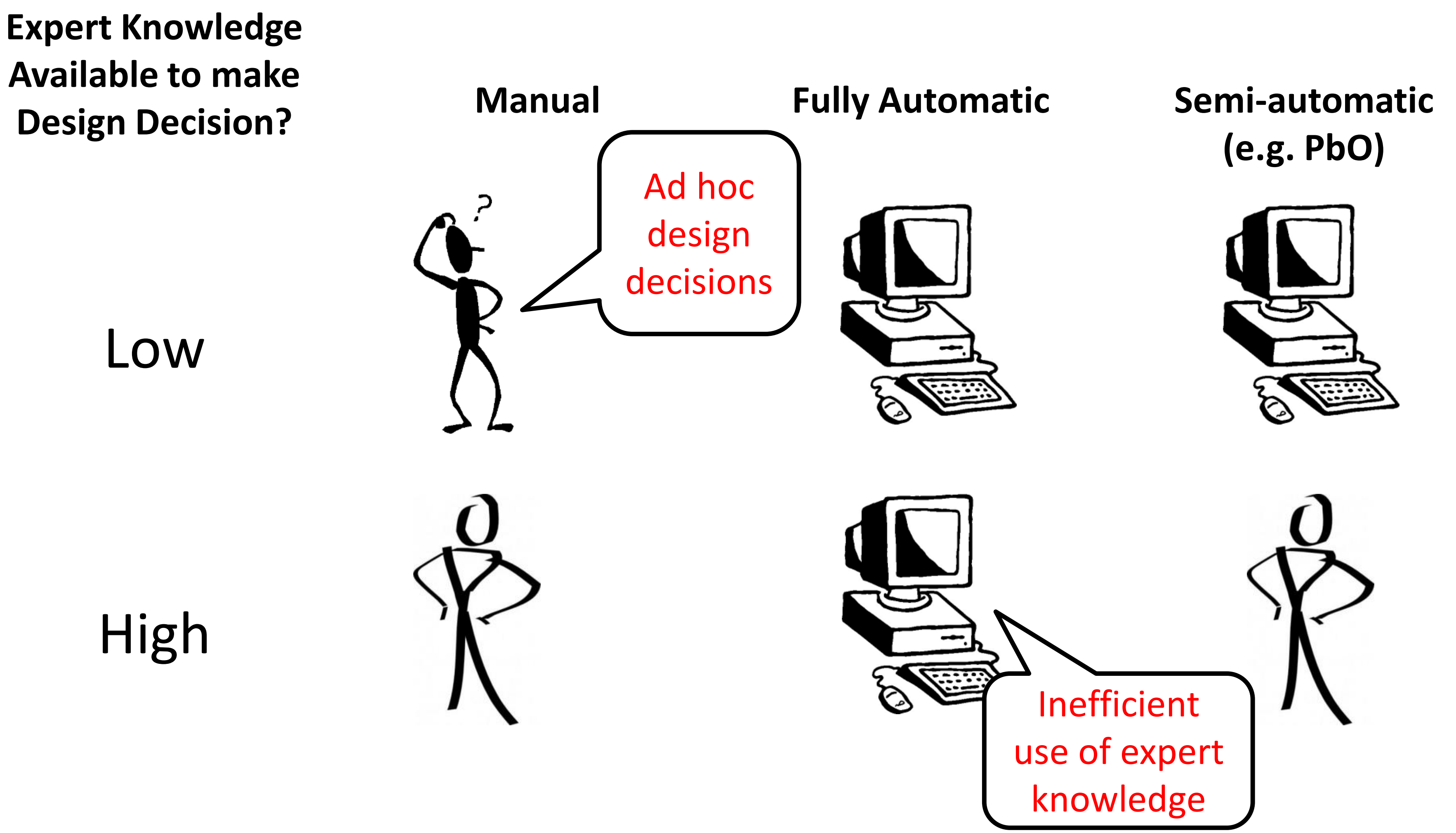
Semi-automatic (e.g. PbO)

Low

Ad hoc design decisions

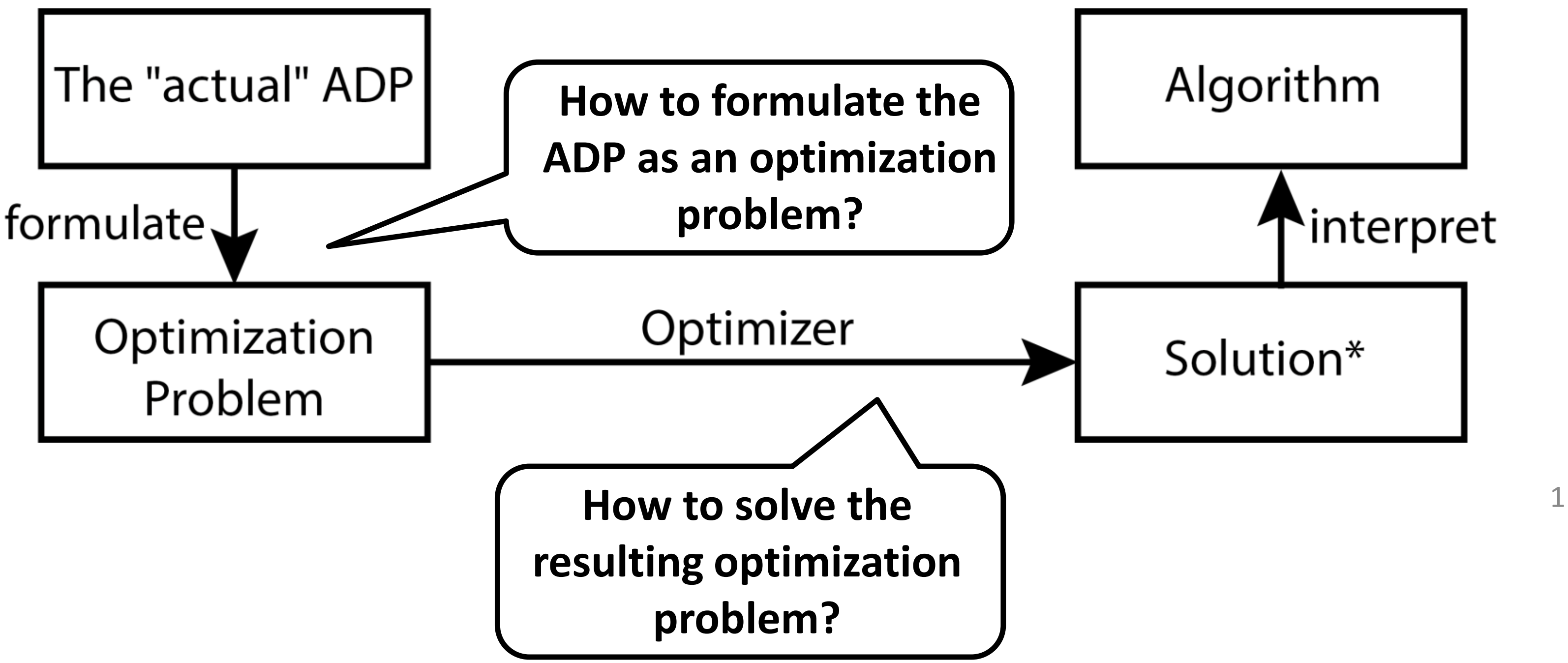
High

1. Who makes which design choices?



1. Programming by Optimization (PbO)

Open design choices:



1. Per-set Algorithm Selection Problem (set-ASP)

Given

A: algorithm space

X: input space

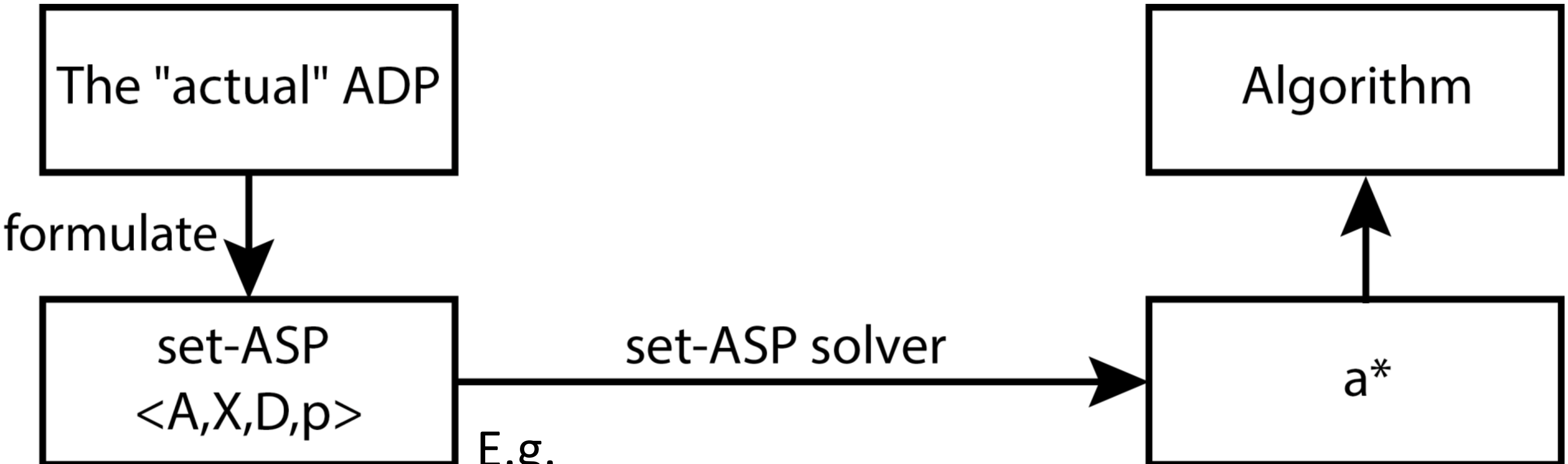
D: input distribution ("*use case*")

$p: X \times A \rightarrow \mathbb{R}$: performance evaluation function

Find

$$a^* = \arg\max_{a \epsilon A} \sum_{x \epsilon X} D(x) * \mathbf{E}[p(x,a)]$$

1. Set-ASP reduction



E.g.

- configurators (ParamILS, iRace, GGA, SMAC etc.)
- Genetic Programming (GP),
- generative hyper-heuristics
- SBSE (program optimization)

…

1. Per-input Algorithm Selection Problem (input-ASP, Rice, 1976)

Given

A: algorithm space

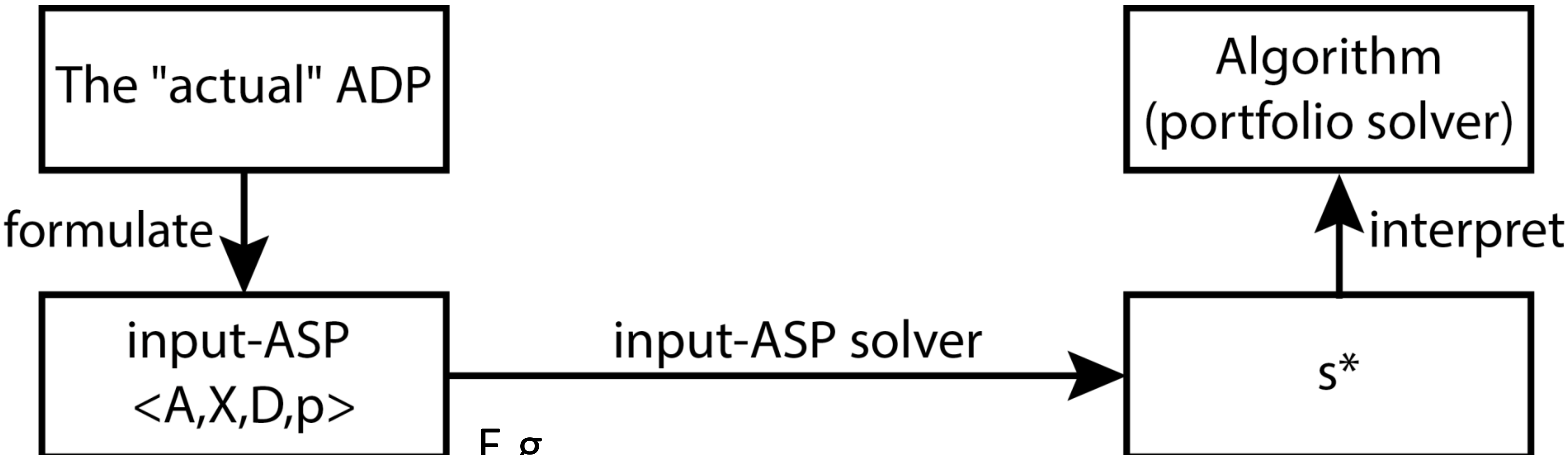
X: input space

$p: X \times A \rightarrow \mathbb{R}$: performance evaluation function

Find s^* satisfying

$$s^*(x) = \underset{a \epsilon A}{\arg\max}\, \mathbf{E}[p(x,a)]$$

1. Input-ASP reduction

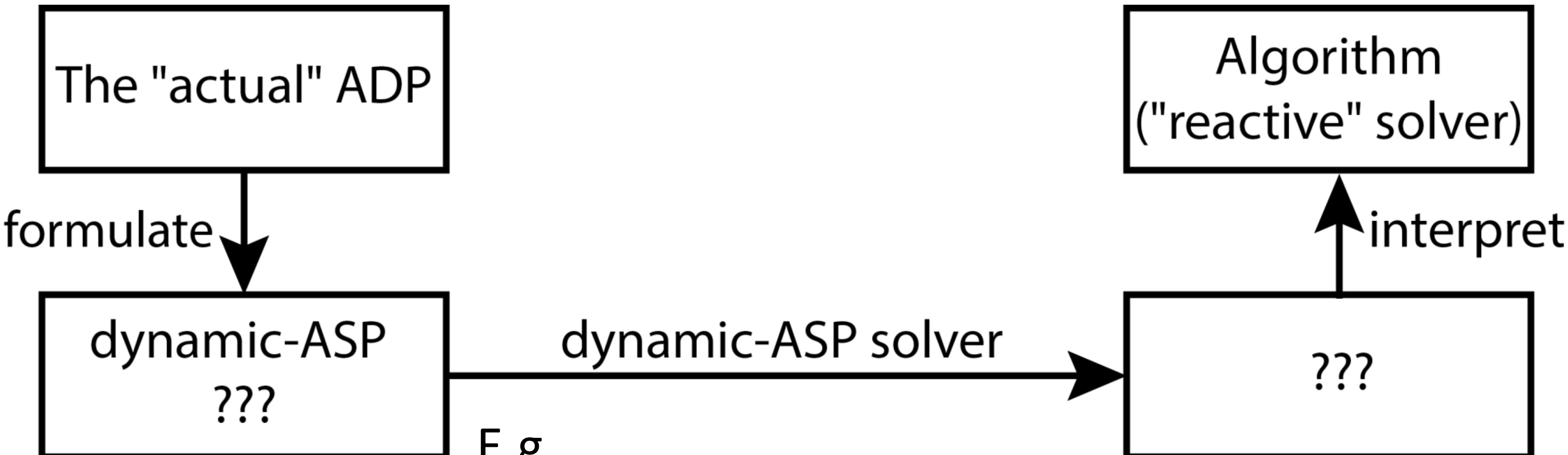


E.g.

- portfolio builders
- input specific configurators (Hydra, ISAC etc.)
- context-aware compilers

…

1. Dynamic-ASP reduction



E.g.
- (dynamic) portfolios,
- parameter control,
- (selection) hyper-heuristics

…

1. Dynamic Algorithm Selection Problem (Adriaensen et. al, IJCAI, 2016)

Given:

- **Design Space:** Non-Deterministic TM
- **Desirability Execution:** Rewards associated with moves performed by TM

Find:

A **policy** π maximizing the expected future reward.

A function mapping
- input
- transitions (leading up to choice point)

to one of the possible next transitions.

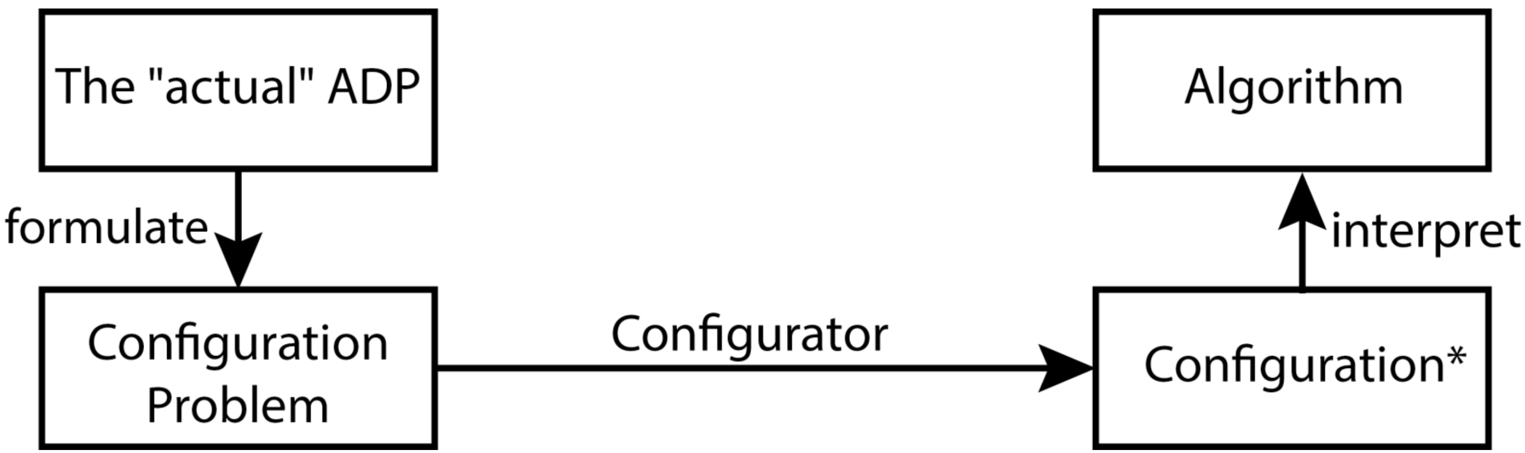
1. Reinforcement Learning Perspective

Algorithm Selection Problems	**Reinforcement Learning (RL) Problems**
Set-ASP (offline)	Best-arm Identification Problem
Set-ASP (online)	Multi-armed Bandit Problem
Input-ASP	Contextual Bandit Problem
Dynamic ASP	Markov Decision Problem

Cross-transfer:
- RL literature may help you understand and solve these problems better!
- RL community also needs to consider ASP methods in practical applications...

2. Programming by Configuration

Formulate the ADP as a Configuration Problem



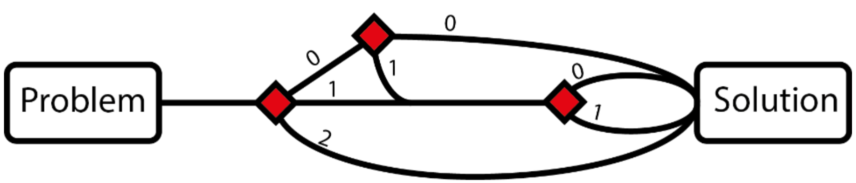
E.g. ParamILS, iRace, GGA, SMAC

19

2. Programming by Configuration

ADP ↔ ACP

ADP	ACP
Open design choices	**Parameters**
Alternative decisions	**Range of values**
Design space	**Configuration space**



C = {0,1,2} x {0,1} x {0,1}

Design

Configuration

c = (0,1,1)

Success Story

Hard Combinatorial Optimization:
- Spear SAT-solver: 500x speedup
- SATenstein: 1.6x to 218x speedup

…

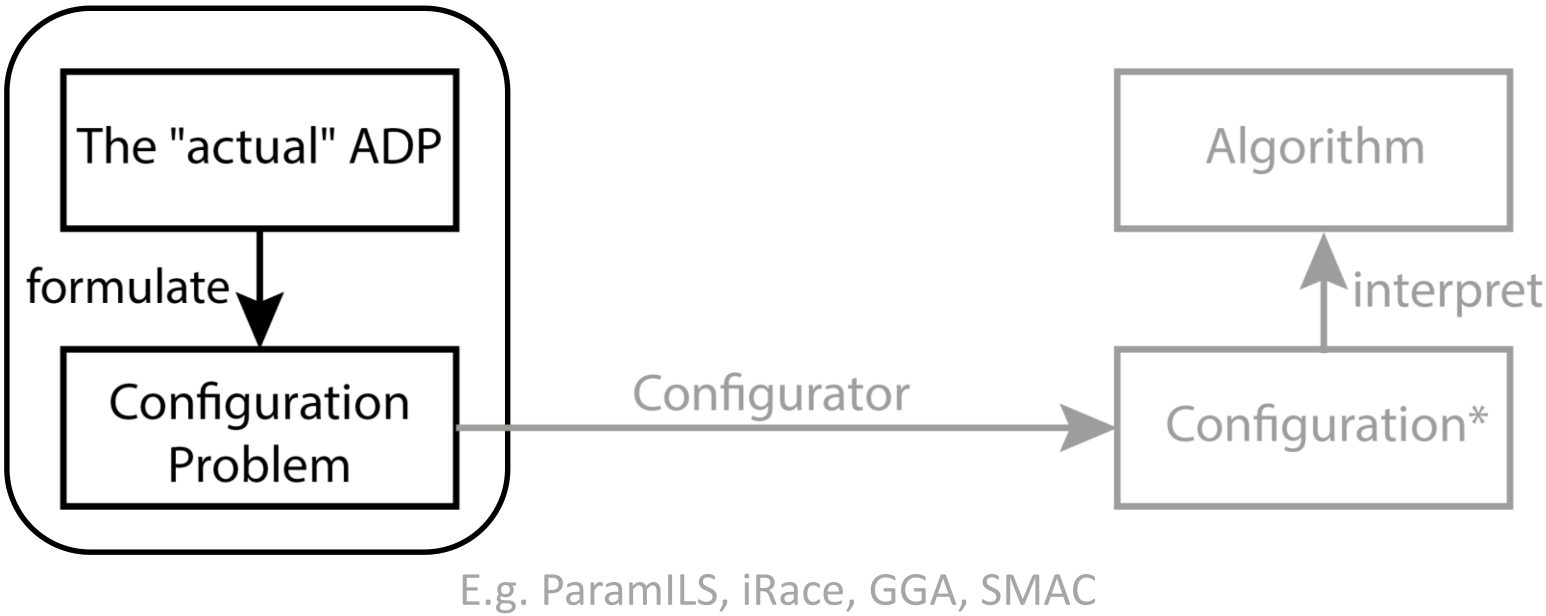
Mixed Integer Programming:
- IBM CPLEX: 2-500x speedup

Machine learning:
- Auto-Weka: Similar/better than best with default settings.

Many more: *www.prog-by-opt.net*

Limitations?

w.r.t. formulating the ADP as a Configuration Problem



Quality of the resulting design?

Question: Is it theoretically possible to always obtain the same quality of design using PbC, which solves the ADP by set-ASP reduction, as those design approaches which solve it using input-ASP or dynamic-ASP reductions?

Set-ASP $\equiv_T$ $\mathbf{input{-}ASP}$ $\equiv_T$ dynamic ASP?

For instance: Given unlimited resources. Can we, using tuners (e.g. ParamILS, iRace or SMAC), always design algorithms as good as those obtained by per instance tuners (Hydra/ISAC)? How about using parameter control?

No?

“Configurators return a single algorithm to be used on all possible inputs.”

average-case performance
dependant on input-distribution
(we must re-optimize whenever the use-case changes…)

“Portfolio builders return a portfolio of non-dominated algorithms.”

best-case performance
Input-distribution independent

Dynamic approaches: even more powerful!?

(ability to adapt to stochastic events)

Yes!

"A (dynamic) portfolio solver is just another algorithm"
➔Formulate the algorithm space of the set-ASP to include it.

Consequences:

- Discrimination of (dynamic) portfolio solvers is misguided:
  – Negative: Excluding them from competitions…
  – Positive: Free Lunch for (dynamic) portfolios…
- Upward reductions:
  – Input-ASP $\leq$ Set-ASP ($\Theta \sim$ family of selection mappings)
  – Input-ASP $\leq$ Dynamic-ASP ($\Theta \sim$ family of policies)
  ➔ **RL: Configurator $\sim$ policy search approach to MDP**
- The dynamic-ASP can be solved offline

Online > offline design?

Offline: *"first design the algorithm, afterwards use it"*

1. Solve the ADP ("training phase")
2. Use the resulting design.

Online: *"refine/design the algorithm while using it"*

- Given: A sequence of instances to be solved in order.
- Objective:
  - Minimize the cost of doing so (resource usage/quality)
  - Refine the design:
    a) After solving an instance (**cross-input learning**)
    b) While solving an instance (**within-run learning**)

Online > offline de…

When have we trained enough?

Offline: *"first design the algorithm, af…*

1. Solve the ADP ("training phase")
2. Use the resulting design.

Which training inputs?

Online: *"refine/desig… …ithm …*

Changing use case…

- Given: A seq… …o be solved in order.
- Objective:
    - Minimize the … …g so (resource usage/quality)
    - Refine the design:
        a) After solving an instance (**cross-input learning**)
        b) While solving an instance (**within-run learning**)

Online > offline design?

Offline: *"first design the algorithm, afterwards use it"*

1. Solve the ADP ("training phase") **Pure exploration**
2. Use the resulting design. **Pure exploitation**

Online: *"refine/design the algorithm while using it"*

- Given: A sequence of instances to be solved in order.
- Objective:
  - Minimize the cost of doing so (resource usage/quality)
  - Refine the design:
    a) After solving an instance (**cross-input learning**)
    b) While solving an instance (**within-run learning**)

Exploration vs. exploitation trade-off

Online > offline design?

Offline: *“first design the algorithm, afterwards use it”*

1. Solve the ADP (“training phase”)
2. Use the resulting design.
3.

Online: *“refine/design the algorithm while using it”*

- Given: A sequenc … solved in order.
- Objective:
  - Minimize the cost o … (resource usag…
  - Refine the design:
    a) After solving an instance (**cross-input learning**)
    b) While solving an instance (**within-run learning**)

adaptation
$\neq$
learning

What is π?
Learning curves!

Abuse RL…

Semi-online

In many practical settings:

- Minimize **response time** > total resource usage
- Availability of (cheap, free) **spare resources**:
  - Time (overnight, in-between requests)
  - Parallelism (unused cores, processors, computers)

Semi-online:

- Serve requests using the best known design
  ➔ **pure exploitation**
- Use spare resources to refine it ➔ **pure exploration**

Anytime ~ semi-online

Given: Anytime ADP solver (e.g. ParamILS, SMAC):

1. Start the design process in a separate thread.
2. For each request to solve x (*asynchronous*)
   a) Obtain $a_{\text{incumbent}}$ from the design process.
   b) Solve x using $a_{\text{incumbent}}$
   c) Return solution to the client.
   d) Add x to the set of training inputs (+ result of run)

   (possibly discounting to address non-stationarity)