AClib: a Benchmark Library for Algorithm Configuration

Frank Hutter, Manuel López-Ibáñez,
Chris Fawcett, Marius Lindauer,
Holger Hoos, Kevin Leyton-Brown, and Thomas Stützle

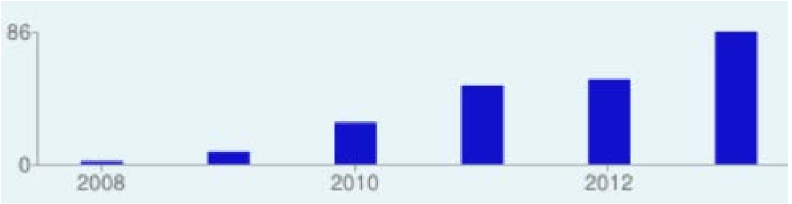
18 February 2014

Motivation

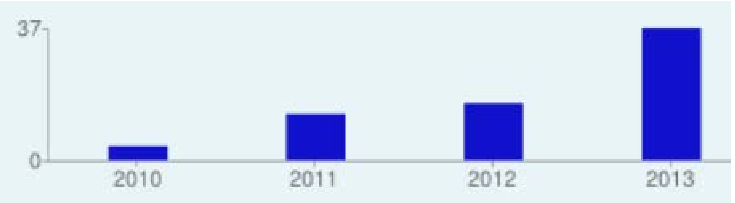
- Most heuristic algorithms have **free parameters**
  - E.g. IBM ILOG CPLEX: 76 parameters
    - Preprocessing, underlying LP solver & its parameters, types of cuts, …
- Algorithm configuration aims to **find good parameter settings automatically**
  - Eliminates most tedious part of algorithm design and end use
  - Saves development time & improves performance
  - Produces more reproducible research

Mainstream Adoption of AC Methods

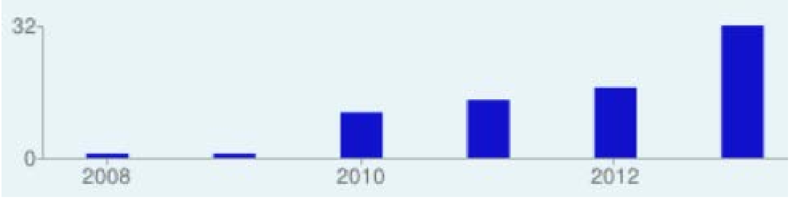
- Many **different types of algorithms**
  - Tree search, local search, metaheuristics, machine learning, …
- **Large improvements** to solvers for **many hard combinatorial problems**
  - SAT, MIP, TSP, ASP, time-tabling, AI planning, …
  - Competition winners for all of these rely on configuration tools
- Increasingly popular (citation numbers from Google scholar)



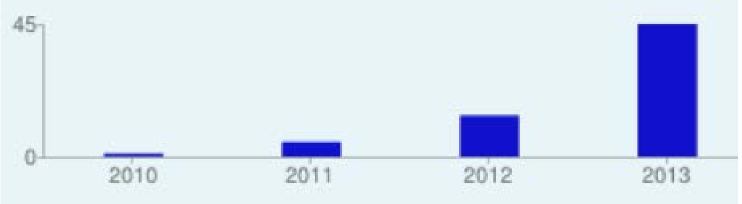
ParamILS [Hutter et al.,'09]



Iterated F-Race [Birattari et al., '10]



GGA [Ansotegui et al, '09]



SMAC [Hutter et al., '11]

Benefits of an AC Benchmark Library

- **Comparability & reproducibility**
  - Easy access to broad range of standard benchmarks
  - Reduced effort for empirical evaluation
  - More meaningful results

- **Standardization of interfaces**
  - Simplifies use of AC procedures
  - Speeds up development

The Algorithm Configuration Problem

Definition

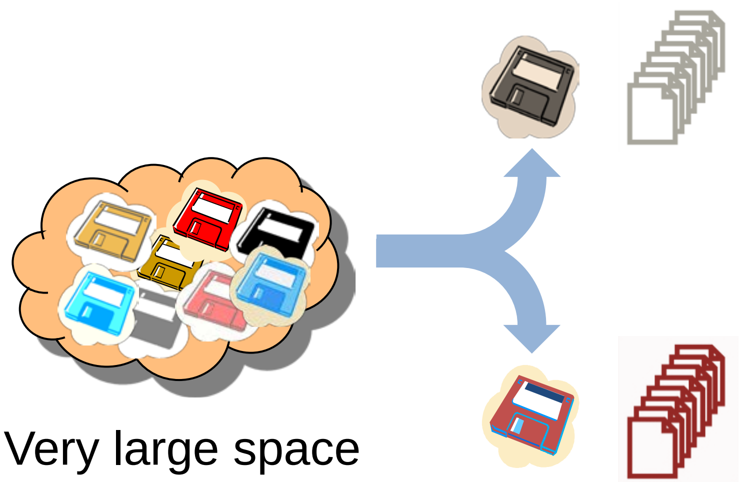
- Given:
  - Runnable algorithm *A* with configuration space $\boldsymbol{\Theta} = \Theta_1 \times \cdots \times \Theta_n$
  - Distribution D over problem instances Π
  - Performance metric $m : \boldsymbol{\Theta} \times \Pi \rightarrow \mathbb{R}$
- Find:

$$\boxed{\boldsymbol{\theta}^* \in \arg\min_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} \mathbb{E}_{\pi \sim D}[m(\boldsymbol{\theta}, \pi)]}$$

Motivation

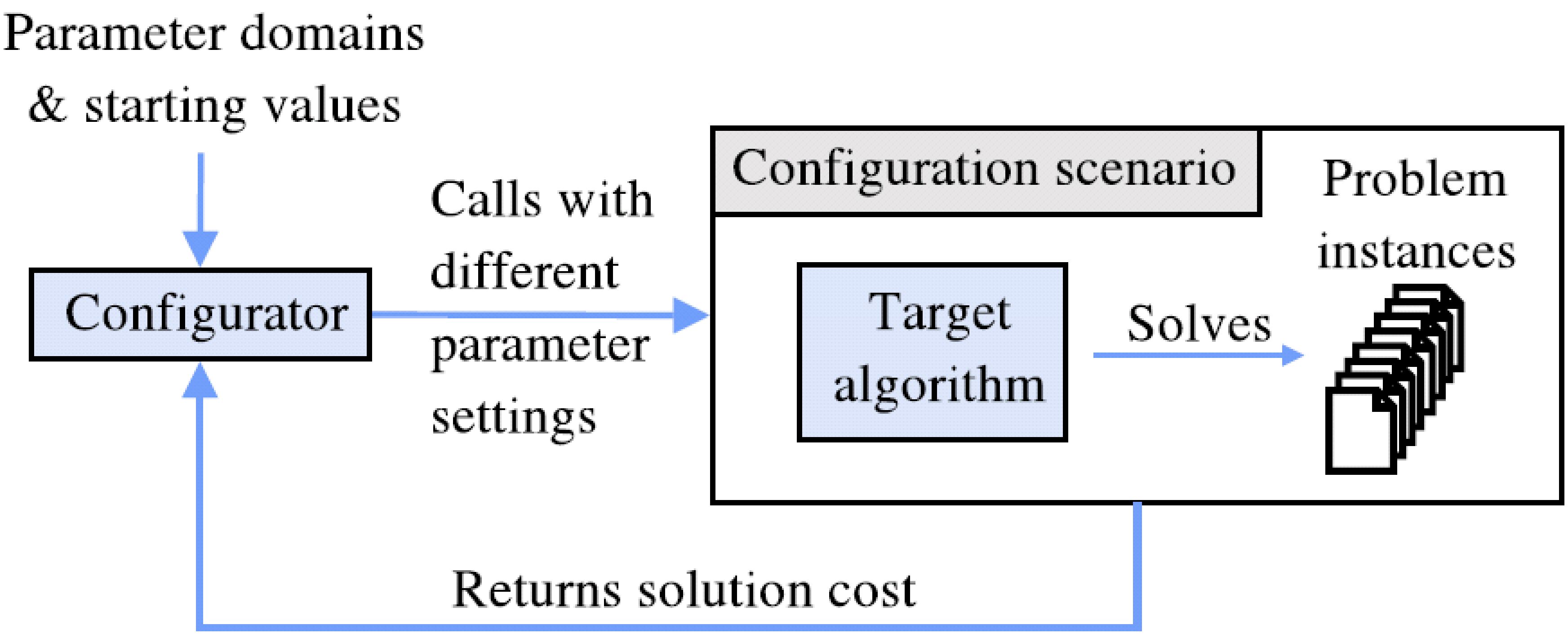
Customize versatile algorithms
for different application domains

- Fully automated improvements
- Optimize speed, accuracy, memory, energy consumption, …



Very large space
of configurations

The Algorithm Configuration Process



Methods for Algorithm Configuration

Work on numerical parameter optimization (e.g., BBOB)

- Evolutionary algorithms community, e.g., **CMA-ES** [Hansen et al, '95-present]
- Statistics & machine learning community, e.g., **EGO** [Jones et al, '98], **SPO** [Bartz-Beielstein et al, '05-present]

Early work on categorical parameters

- **Composer** [Gratch et al, '92 & '93]
- **Multi-TAC** [Minton, '93]
- **F-Race** [Birattari et al, '02]

General algorithm configuration methods

- Iterated Local Search, **ParamILS** [Hutter et al., '07 & '09]
- Genetic algorithm, **GGA** [Ansotegui et al, '09]
- **Iterated F-Race** [Birattari et al., '07-present]
- Model-based Algorithm Configuration, **SMAC** [Hutter et al., '09-present]

Algo. Configuration vs. Blackbox Optimization

Parameter types

- Continuous, integer, ordinal
- **Categorical**: finite domain, unordered, e.g., {a,b,c}
- **Conditional**: only active for some instantiations of other parameters

Optimization across a **distribution of problem instances**

$$\boxed{\boldsymbol{\theta}^* \in \arg\min_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} \mathbb{E}_{\pi \sim D}[m(\boldsymbol{\theta}, \pi)]}$$

- Stochastic Optimization
- Instances often differ widely in hardness

Budget: CPU/wall time vs. # function evaluations

- Overheads of configurator count!
- Can exploit that fast function evaluations are cheaper
- Can save time by cutting off slow runs early

AClib: Components

- **Configuration scenarios**

Problem	Solvers	#Scenarios Runtime	#Scenarios Quality	#Parameters
SAT	16 different solvers	75	0	2 – 270
MIP	CPLEX	4	4	76
ASP	Clasp	3	0	85
AI Planning	LPG & Fast Downward	20	0	45 – 66
Time-tabling	CTT	1	1	7 – 18
TSP	ACOTSP, ACOTSP-VAR	0	2	11 – 28
bTSP	MOACO	0	1	16
Machine Learning	AutoWEKA	0	21	768

- For convenience, we also include **configuration procedures**
  – So far: ParamILS, SMAC, and Iterated F-Race

AClib: Design Criteria

- **Variety**
  - Problems: decision & optimization problems, machine learning
  - Algorithm types: tree search, local search, machine learning
  - Number of parameters: 2 - 768
  - Parameter types: continuous / discrete / conditional
  - Objectives: runtime to optimality / solution quality
  - Degree of homogeneity of instances

- **Assessing different configurator components**
  - Search: which configuration to try next?
  - Racing/intensification: how many runs, which instances?
  - Capping: when to cut of a run?

AClib: Resolves Technical Challenges

- **Unified way to wrap target algorithms**
  - Built-in control of CPU time & memory
  - Reliable measurements of CPU & wall time
    - No more need to rely on target algorithm's time measurements
    - Consistent use of wall time / CPU time
- **Identical invocations of a target algorithm**
  - Callstrings are independent of the configurator
  - Otherwise systematic biases possible,
    leading to incomparable results in the literature

AClib: Contribute

- Contributing a **benchmark scenario**
  - Algorithm & its parameter description
  - Instances, Features, training/test split
  - CPU time & memory limits
  - Algoritm wrapper
    - Generates a call string given an instantiation of parameters
    - Parses the algorithm result
- Contributing a **configuration procedure**
  - Accept scenarios in AClib format
  - Basically:
    call target algorithm on the command line and get results back

Future Work

- **For you: use AClib ;-)** www.aclib.net
- **Ontology** of algorithm configuration scenarios
- **Large-scale evaluation**
  - Which configurator performs best on which types of problems?
- **Algorithm Configuration Evaluation**
  - Planned as AAAI 2015 workshop (together with Yuri Malitsky)
  - Submit configuration scenarios! (same format as in AClib)
  - Submit configurators!